COMAL TODAY 25

COMMON COMAL™ KEYWORD CHART
page 29

BUFFER
page 46, 52

PRIME FACTORS
page 56

THE STORY OF COMAL
page 1

COMAL STANDARDS
page 34

POWER TO THE PEOPLE
page 65

PROCEDURES
page 38

DON'T TOUCH THAT FILENAME
page 62

RECORDS
page 66

LEARNING MODEL
page 60
The Story of COMAL told by Borge Christensen

Are you going to be moving around a lot during your talk, or will you be fairly stationary?

What would you prefer?

Well, I don't mind.

I promise you that I will no do Danish Folk Dancing.

Oh.

That would be a disaster. We would probably be thrown out of NATO afterwards.

Anytime you are ready.

I'm ready now. Well, I'm going to simply talk. Well, I'd like to thank you all for coming. First, excuse me if sometimes it happens that I misspeak the words. It is a foreign language to me and my stock of words could all of a sudden disappear. You'll hear me searching for a word. Please bear with my language.

I have been so surprised to see the enthusiasm in America about this language of ours. What I'm going to talk about tonight might help you understand why I am surprised because this was from the start a very humble project, in a very small town, in a very small country, where it usually is quite cold, which might be one of the reasons why we are sitting so much indoors at our computers of course.

From the start this was a very humble project, in a very small town, in a very small country, where it usually is quite cold.

It started, in fact, in ... well, I might add one or two words about myself. I am hired by her Majesty, the Queen of Denmark, to teach Mathematics at a small college near the German border. When I say that I am hired by her Majesty, the Queen, it is actually true. I have a contract with her signature on it. Now you know how small our country really is. Her majesty does not go out each night and lock the door to the kingdom, as it says in Hans Andersons fairy tales. But, she can sign contracts, personally, with some professors, at some colleges, in some positions. So, she expects me to teach Mathematics, and that's what I'm doing.

Besides that I have to give courses in Computer Studies, specifically with applications of a mathematical kind. And that was how it all came about. In 1972 we got a Data General NOVA 1200 mini-computer, which some of you might know or have heard rumors about. We were supposed to start this new subject, Computer Science. Since I had long time ago taken a course at the University (only three weeks, but they didn't know that), I was expected to take care of this new thing, and try to teach the students how to use it.

The language that came with it was called Extended BASIC. One of the first things that we experienced was that it was so crowded with bugs that even the cleaning person began to complain about it. It was real bad. It broke down ... it was a so called core share system - no disk. It was only paper tape, all of it. Paper tape and teletypes. And it broke down regularly every four hours, so we had to re-install the programs by reading the paper tapes at a speed of 10 characters per seconds. It was quite a job. I think even this one would seem fast compared to that [referring to a Commodore Disk Drive]. This is, as you know, the worlds fastest tape station.

Another thing that seemed to go wrong was the way the students performed. That was to put it mildly ... miserably. The programs, even small programs were very bad. They didn't understand much of it, and it happened again and again that they could do it much easier on their pocket calculators.

Also, for myself, I found it extremely difficult to mark their works, because it was hard to read, simply to read their programs. Twenty or thirty line programs were hard to find out what was going on. I had to draw diagrams very often, simply to find out what they had did, what methods they had applied, what approach they had used. Of course, it might have been me that was not good enough, but that is an uncomfortable idea, that you're not good enough, so after awhile, I went to a nearby university, I knew their Computer Science section was a very good one ... in fact I went there because we had to set up some exam papers, some examination problems, and I didn't know what to do about this, because I could see disaster ahead, even if I only gave them simple problems to solve in this language, this so-called language, BASIC.

Here, in the afternoon, I met a man, his name was Benedict Lofstedt. When I came in and explained to him what this was all about, he said: Well, this is not your fault. this is not the students fault. it is the language that you are using. It is in fact, not a computer language. It is an offense to the human mind.
I am a mathematician, I'm not a computer scientist. And we were quite impressed by this new technology. It cost a whole lot of money and the local newspapers had written about it. So, I was surprised that he asked me to go back and read a new book by Niklaus Wirth called Systematic Programming. In that book he presented a language called Pascal. So we looked at it and I could see at once that this was what programming ought to be about.

Now, we couldn't use that Pascal directly. As it was, it was a very good language for advanced University students and professionals. But for my students, it would be impossible.

There are one very good thing about BASIC ... it's interactive. You can type in small programs, have them run at once. You won't have to write a program using an editor, having the text stored, have the compiler load it, compile it ... going through all these phases of editing, compiling, and manipulating the operating system; that would be far too complicated for the students I had. So, we would have to maintain some of the user friendly environment there is in BASIC. I think that is the only good thing you can say about BASIC, it's the user environment. And it is important to understand that there is a difference between the language itself, and the operating environment this language is working in. We found out somehow, that by putting these names in arrays, and using tokens to address them, the whole project. Otherwise we could go ahead with what we had. No understanding at all.

We went to one of the major companies and they asked me, rather cynically by the way, unfriendly, to come back again with $50,000.

Well, I then had what you would call a really good fortune. I got two very bright students. I did machine coding at the NOVA, which was in fact a 16 bit machine, and I told them how to code, machine code. In a few weeks they became much better at that than I was, than I ever became. They were young people, and had learned it very fast.

One day one of them came up, and said: This definition of yours that Benedict and you have been doing. I think we can do that.

I was very happy and surprised. Well try it.

They worked on it for about half a year, and eventually came up with the first implementation. We had a very close co-operation. Again and again we took discussions on how to implement the different details. I can still remember the day that we tried to put in long variable names. This was a problem then. There were no long variable names. Instead of saying number, name, we could only say N or N1. Still I think there are many BASICS that only allow short variable names.

We had a huge board where we wrote down all sorts of ideas. It was a great experience. It was quite fantastic, to be in that process with these two extremely clever young boys.

But we wanted them, wanted them right away in the first version to be at least 8 characters. We wanted to allow 8 characters. At least we would then be able to give variables sensible names. Such that the programs would be much more readable. This one of the first things we were after.

We found out somehow, that by putting these names in arrays, and using tokens to address them,
The Story of COMAL (continued)

and thereby addressing their data fields, that we could do it that way. This sort of discussions were going on for half a year. We had a huge board where we wrote down all sorts of ideas. It was a great experience. It was quite fantastic, to be in that process with these two extremely clever young boys.

We poured coffee in him, and put him into a cold shower. Then he would work for 24 hours. It was just great.

I had a problem with one of them. He was a heavy drinker. Now and then he would go away, and we knew where he was. But only at a certain time when he had had enough could we pick him up. We could not persuade him to come back again until he had taken so and so much. He would then be very drunk. And we needed him, as he was our expert in debugging. Danish beer can be quite heavy, quite strong. We poured coffee in him, and put him into a cold shower. Then he would work for 24 hours. It was just great.

The first version of COMAL ... still we hadn't called it that, we had a Danish name for the project ... was running on the 5th of August, 1974. It was a five line program with a REPEAT ... UNTIL. And we had all the automatic indentation ... that was all done. We were working on the IF ... THEN ... ELSE with indentation. We pinned it on the board, and it was stolen a fortnight later, so I don't know where it is. This is quite a pity.

In February 1975, we had a version to send out. Now, you should imagine that we would send it out on a disk. We had no disk! No diskette, no hard disk, no soft disk, or anything. We sent it out on paper tapes.

I don't know if any of you have worked on a teletype.

Several people cried out Yes!

OK ... you know what it's like. Most would have a high speed reader, typically, 500 characters per second, which was quite respectable. But many of them would have to use a teletype 10 characters per second... it took one hour to load COMAL. So please don't complain over this. (God bless the poor fellow who then switched the whole thing off by some mistake).

Again and again we went around ... by the way, when we implemented, we only had paper tape. We had a fast paper tape punch and a fast paper tape reader, but we used miles on end of paper tape. Sometimes were walking around in paper tapes up to our knees. My job most of the time was to wind up ... I invented a very fine technique. I think I could even do it today if I had a chance ... wind it up very fast.

This was an experience of a lifetime, to be doing such a thing. It was absolutely fantastic.

One thing that might happen eventually, you probably know that ... I need to tell you that the whole implementation cost me $300. That was all we had, so we did it in our spare time ... we thought it was great fun. And it was. This was an experience of a lifetime, to be doing such a thing. It was absolutely fantastic.

When we were in the process of reading in a paper tape, it happened eventually that one stepped on it. And we could start all over again, because the paper tape broke. This was one of the deadly sins, I don't know how many are allowed.

We were fortunate, because most of the institutions that were of our type, (high schools, colleges, vocations schools, etc) had in fact Data General NOVAs, because one local Danish company was the distributor of that machine. We didn't have problems with such obscure machines such as Hewlett Packard or whatever that is. We had the NOVA all over. So they took it and started to use it. We were quite successful. I think it took half a year and they had stopped using the regular BASIC and turned over to COMAL.

Now, as it appeared, the major reason for doing that was the long variable names, believe it or not. Especially the students. The teachers were more hesitating, but the students took it very fast. They could now write NUMBER=5 or NAME$="Peter Sorenson".

COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716 - Page 3
The Story of COMAL (continued)

And then the IF .. THEN .. ELSE. That was one that really was bought at once. I don't know if you are aware that doing branching, especially nested branching, using GOTO is in fact very very difficult. Therefore we had the nested IF ... ELSE ... ENDIF only up to a level of four at that time, but that appeared to be enough.

And then of course, the third facility, was the named procedures. We didn't have parameters at that time. You might call it simply named subroutines. Instead of saying GOSUB 5000, we could say EXECUTE PRINTOUT. And then you would have PROC PRINTOUT ... ENDPREC. That was all there at that time.

This was all much much easier. And it was faster. This was another thing we noted and which helped us. I don't know if you are aware that when BASIC tries to find line number 5000 starting at line number 10, it will go over all the line number to try to find it. In the first COMAL we did, we put up links, such that if there was an EXECUTE (that was the keyword we used) EXECUTE PRINTOUT in say line 50, and PROC PRINTOUT would appear in line 500, there was a pointer telling line 50 to jump at once (in machine code) to line 500. That of course was much much faster than having to go sequentially through all the line numbers. It was also one of the advantages that made COMAL popular very fast.

We were well aware that we should have some fancy English looking name, otherwise no one would have it.

The first COMAL included:
- Long Variable Names
- IF .. THEN .. ELSE
- Procedures & EXECUTE

The college back home when it appeared to me, that we had been fooling around with the language ALGOL, which you probably know, ALGOThmic Language, and all of a sudden it appeared that this should be COMMON Algorithmic Language. Shortened is COMAL. That was in fact how the name was made. That's all. It is a really country name.

That was, in fact, how the name was made. That's all. It is a really country name.

We continued to improve it a little ... as I said before, we then sent it out. I was not doing much more machine coding, because the students asked me to write the documentation. That's of course a very important thing. I also wrote a text book which later became Beginning COMAL. That took 3 or 4 years before it developed into the Danish edition of that book.

This went on until around 1978, when the first microcomputers appeared. We got a problem. The problem we had to face would be the following. Imagine the Danish schools had started to buy PET or APPLE. That meant that we were sent back to BASIC again. This was to me very frightening. It was so frightening in fact that if I had been forced to teach BASIC again, I would not have taught computers at all. That's true! I was completely determined, if they were to force me to teach BASIC again, I would have said: OK, you hired me to teach Mathematics, Algebra, Statistics, and Number Theory, and I'm good at that, so that's what I'm going to do. You can take your computers somewhere else. I was absolutely determined to do that. But of course, it would be better if we could come up with a microcomputer with COMAL.

So in 1978 I defined a new version, where we put in the parameter mechanisms and the sort of things you know now, most of it at least. And we found another group in a college somewhere else that were willing to try to improve, enhance, expand COMAL into what we call COMAL 80 because it was not until 1980 when it was really finished, and it was running on a Z80 based computer, Zilog 80. So we had two reasons to call COMAL 80.

At the same time another Danish company designed and built a computer for schools. It was called the Comet. We are not always that modest in Denmark. It appeared to be a very good computer by the way. It is still used especially at technical schools.
The Story of COMAL (continued)

because it has one of those busses where you can add in extra electronics all the time to the buss. It is still very popular. That was the first computer to run a COMAL that you would recognize. If you saw programs written for that computer, you wouldn't be surprised at all. This would look very much like what you see today. What had to come would then be to make better environment, better editors, to put in extra facilities, such as graphics, sprites, and sound.

But at that time, 1980-1981, I stopped doing much about it. It was then taken over by different groups, one of which is UniComal which is one of the most talented. Maybe I should add now about the way UniComal started. One of them Mogens Kjaer, wrote a letter to me, where he criticized my definition of COMAL 80. When I saw this letter, I knew that once that he would be one of the boys that could do something about it. This was simply one of the most qualified critics that I ever had. It was very good. I adjusted my COMAL according to what he advised me to do. And then I asked him if he might do ... he had intended to do an implementation for a computer that wouldn't have any chance of going anywhere. I advised him not to do that, but instead to go for the Commodore PET. It was obviously a very good computer.

At that time, you should know, by the way, a company could not sell a computer to a Danish school if it did not have COMAL. Hewlett Packard never sold their computer since it did not have COMAL. We had given it such a strong position. The teachers wouldn't have it. The teachers refused to use BASIC. That was a very strong situation. We hold in fact a very strong position without telling anybody. Simply by giving them a language that was apparently better.

Mogens Kjaer made this version for Commodore, and this is the version now known as 0.14. After he did that, the UniComal group was set up and they then went on. From then on I have not had much to do with it. These young fellows are doing a terrific good job. They added the packages, which is probably the most important extension to COMAL that was made after the latest definition. This was the package concept that you could have packages in machine code and call them.

From then on you know more or less what the story is all about. Still a new version now is being designed for the Z80 CP/M based computers for there is an English computer called the Amstrad, which is quite popular in Europe. They expect to sell around 600,000 of them. We have a version of COMAL for that now [this is the CP/M COMAL]. It is very much like the UniComal version. There are a few improvements, but not much.

What I'm interested in now ... I don't think we should expand COMAL much more. I feel perfectly well with the version for the Commodore 64. I have 17 of them for my students. We will probably have one or two PC's only because an institution like ours must have PC's. I probably can persuade someone to use them if we get them.

A company could not sell a computer to a Danish school if it did not have COMAL.

What I'm interested in now is to see a better user environment.

What I'm interested in now is to see a better user environment. It has improved very much with FIND and CHANGE. I would like to see full screen editors, without line numbers. There is a COMAL, Mytech [Alder] COMAL, with such a very good editor, but I'm sorry to say that the COMAL is not as good as it ought to be, but that may come. There is now an international standardization committee. We have a meeting in Scotland in September. We are going to discuss, among other things, whether we should get rid of line numbers, or at least make them redundant. You can get them if you like, but you can avoid them. You are aware I expect, that line numbers are not needed after we have left the teletype. You can with a screen editor, you can move the cursor wherever you like, and insert a line by manipulating some control code. Whereas with the teletype, line numbers were quite ingenious. You couldn't get back to the place between line 10 and 20, but you could print line 15 and have it inserted between line 10 and 20. This was in fact very well made considering that it was made for teletypes. But they are, I think, gone long ago.
The Story of COMAL (continued)

If anyone had told me in 1974 that ten years later I would be standing in Los Angeles, I probably wouldn’t have believed him.

This is more or less how it was. When we started it, we ... well if anyone had told me in 1974 that 10 years later I would be standing in Los Angeles, I probably wouldn’t have believed him. Because we only wanted, as simple as that, was a tool that our students could use. Nothing pretentious or great project or anything. It appeared that our students could use it, it has developed over the years. It is therefore, as you may guess, very impressive, and makes me ... Well I don't know exactly how happy I really am, maybe I'll find out when I come back, maybe, that you really like it and are so enthusiastic about it. I think it is a good tool. It appears to be so. And I hope that many more people can get to know about it.

Well, you are welcome to ask questions and ... that's a lot of detail, but I may add that I used a slogan ... I hope you don't think I'm nasty ... I used a slogan that in the 70's I said that: It might very well be that the good God has invented the computer. But there is no doubt that his black majesty then came up with BASIC shortly thereafter [laughter] ... in order to confuse peoples minds, because that's the way he prefers to operate. [applause].

I would like to begin by taking a poll. Of all the COMAL programmer here, I would like to hear yays or nos from those who would like to see line numbers implemented for COMAL 2.0. All who would like to have line numbers ...

[interruption] What about how it is to be implemented? If you remove line numbers, how do you differentiate being in command mode and being in program mode.

That's right. Now, I think I said that it should be made redundant. But you would have to have two modes, an editing mode, and a runtime mode. In fact you can see it on the Mytech [Alder] COMAL, how it operates. There's no problems. I can demonstrate it after this meeting for whoever likes to see it. This is very easy to implement and use. But still you can use line numbers.

What would be the object of getting rid of the line numbers? What is the point? What is the big controversy of getting rid of them?

I would hate to admit that it is some academic idea. But I think it is. The idea is ... when I say full screen editor, I do not mean only the text you can see on the screen right now, but you should be able to scroll up and down. You can now in COMAL have the listing coming up, but you cannot take the cursor up and have the listing coming down again [note: this is now possible in IBM PC COMAL, AmigaCOMAL, and C128 cartridge COMAL]. So you should be able to look at the screen as simply a window, through which you pull a film either forward or backward.

But that is not the most important thing, I think I am best at giving examples because I am not a computer scientist, so this is not ... I am not one of those desktop academics that think these things out. I look at the students and try to find out what they need. And then we do it. COMAL is very pragmatic. We never took in new facilities and new structures until we felt that the students needed them. If they made a lot of mistakes, or if they had to do more work than necessary with details, then maybe an extra facility would make it much easier for them. That is in fact how the parameters were introduced. We discovered of course, that parameters would make it much easier for them, so we introduced them. But not until then.

That is why I'm a mathematician - I hate numbers. As a child, when I found out that you didn't have to use numbers, that you could use letters instead, that was it!

Now, imagine that you have a list on the screen. Say from 10 to 240. You want to insert a line between 30 and 40. No, I think I'll take a better one. You'll have a list from 610 to 700, or, my mental calculation is a little ... that is why I am a mathematician, I hate numbers. As a child when I found out that you didn't have to use numbers, that you could use letters instead, that was it!

OK, you have here some numbers from six hundred something to six hundred something else. You want to insert a line. You know then, if you have to use the line number. Very easy. Instead of saying 615, you write 15 for example. You introduce a wrong number. Later you cannot find that darn line. Finally you see it. What is doing there. It is between 10 and 20. It should be between 610 and 620. What you should be able to do, with or without line numbers, is to move your
cursor up to 620, press CTRL I - the text opens up, and you insert the line. So you get what you see. That's the whole idea. Also, if you put the cursor at 601, press CTRL D, and zip, the line disappears. This is what I am after. This is what I am trying to get the standards committee to accept at the next meeting. I know I'll have a hard time doing it because I know it is very difficult.

I got into an enormous lot of fights and arguments when I came out with the language.

That's another thing I might have told you. I got into an enormous lot of fights and arguments when I came out with the language. In fact, don't expect that people are happy and grateful, for your coming up with this. A lot of people would say: Why what's this about, we are perfectly OK with what we have.

There were articles in papers written against me, as though I was some new unknown form of Communist in disguise. It took me some years to understand why people can get mad at you when you come up with new ideas.

A programming language is not just a tool. It is something different and something more.

A programming language is not just a tool. It is something different and something more. In fact, what I was saying to some of these people ... or how I was interpreted was: You are telling us that we are using bad language. And that is certainly not a nice thing to tell people. Really! It is! Now I am much more careful not to be too tough. You think the saying comparing BASIC and the Devil was bad enough. Wow. You should have heard the arguments we used in Danish. They are absolutely untranslatable. They never should be anyway.

I hope I answered the question about the full screen.

What you are really after is a full screen editor. The line numbers are neither here nor there.

Well OK. In a 40 column screen it would give you 5 characters more. And sometimes it may prevent the wrap around. But that is minor. And whether the line numbers are going to be there or not is also of a minor concern. The important thing is that you have a full screen editor. You get what you see.

Referring to that problem, not only not being able to find the line that you were trying to insert when you mistyped it. About three days ago, I had one of those. I could not find it. Nothing at all worked. What I did was I double struck one of the keys and it typed in place of a line I had already written and everything went to pot.

Yes, that's right. If you could open up the text, you would know exactly what you were doing. You would see which line was pushed in order to make room for the next.

Well, I've seen a lot of this with my students. Coming up with a program and saying, well I had a program here 15 minutes ago, but ... I remember, by the way, one experience we had with BASIC programs, and this is a true story again. Long John was the students name. He was almost 2 meters. He was a favorite with basketball. He was also a good programmer, but he came up one day with this, not a very big program. He asked me, in Danish of course, in a fair translation, he asked me the following question: Borge, could you explain to me why this program works?

Well, John. We sat down and we found out that three lines were never executed. He jumped around them. We sorted things out ... that was in the old bad BASIC days. He was only one out of a series of similar experiences. Now, if I should put a ... I could hardly use the good English work BASIC any more, but a basic philosophy, it would be the following: Computers should be as easy to use for human beings as possible.

We should be controlling computers, not the other way around. This was what was behind us all the time when we made COMAL.

We should be controlling computers, not the other way around. This was what was behind us all the time when we made this [COMAL]. Benedict Lofsted, the first one ... he was not in the second project because he went to Greenland, and that's a long ways from Denmark to Greenland. So, I had to the second project all alone. But it was not that
difficult, because it appeared that the first definition was so well designed that we could enhance it, extend it, without violating the original idea.

This is one of the other things, I hope you agree, that it still appears that COMAL can take extensions without the original concept being ruined. Maybe that is why you like the language in fact ... is the regularity in it ... or as a computer scientist would say, the orthogonality of it. I tried to learn that word.

One problem I have as my program gets growing to some length, that you start forgetting procedure names and things like that. Is it possible to have a window or something which could list the procedures that you have going at this point?

Yes. What you are asking is a list of the symbol table, of the variable names and the procedure names, probably with some indication of where they were to be found. We have that in the new CP/M COMAL. We have a LISTSYM or LISTPROC, I forget what we called it. Note that this has nothing to do with the language. This is the environment. And that is what is what we are after now. To make it even more friendly.

Is there a possibility of having certain procedures built in? For example, rounding up or that sort of situation?

Yes, it should be. That is what they are doing with the machine codes, the packages. In fact, you can build in anything. You can make yourself a commercial package.

Also, like COMAL 2.0 has built in graphics, if you want to draw a circle, you just use the keyword circle and its parameters. In 0.14 you have to have a procedure.

In the Mytech [Alder] version ... I hope it will eventually be a good version. You can even write the code [for packages] in the language C. [AmigaCOMAL also allows this] And C is much easier to use than machine code, in case you don’t know it. And then use them as COMAL packages. That is to say, that all the procedures you write in machine code can be invoked or called as if they were normal user defined procedures.

I wanted to tell you a couple days ago, but I never got around to it. There is an environment, called artificial intelligence, developed at the University of Suffax, called PopWalk. It works in a screen oriented environment, providing you access to LISP, PROLOG, and POP11... that’s the three languages. What I thought was extremely powerful, is that you could link to it subroutines, procedures, whatever you want to call them, in a host of other languages, including Pascal, Fortran, C. That strikes me as being extremely powerful. Each of these languages have some particular advantage, for some particular applications. Maybe it’s the language you happen to know, that’s already an advantage. Otherwise the language may be well suited to doing a particular task. So, I really try to encourage the idea that COMAL should be able to access routines compiled in a wide variety of Languages.

By the way, Seymour Papert (of Logo fame) took a look at COMAL, and he liked it!

I don’t know whether that could be done, or how feasible it is, but it would be very practical to have it [in 1989 AmigaCOMAL and IBM PC COMAL 3.0 now allow this]. Of course you should be aware, that a language cannot be totally general. There are applications where even COMAL wouldn’t cope with. We had a discussion with Seymour Papert, the man who made LOGO. He put out his position, quite clearly that, ... by the way, he took a look at COMAL, and he liked it! ... and he expressed his opinion that even children should try to learn more than one programming language. It will be more like programming concepts that they will be learning about. And it appears also that children are learning these things extremely fast. The robots, which I’ve brought one or two samples with me, you can look at them later if you like, were programmed by 11 year old children. We had no problems. The teacher had problems. The children had not.

Well, it seems, from what we’ve learned today of programming languages, that there are really three classes of languages. Each of them fulfill a role that the other ones cannot.

Yes. My guess is that in the future there will only be two. I think that the LISP languages will be taken over by the Logic, the PROLOG type. I think so. The three classes you think of are
probably the algorithmic languages, the LISP kind of languages, and the logic, PROLOG, kind of languages. COMAL is of course, an algorithmic language, it's in the name. LISP languages would be LISP or LOGO, LOGO is a LISP language. The third one would be PROLOG, a language that is mentioned very often, especially when you speak about 5th generation computers, and the competition between the United States and Japan. Who will be there first with this super thing. But I think that eventually you will see that there will be two. The algorithmic languages, they will never disappear. In robotics, they will still perform. Just imagine ADA. ADA is an algorithmic language, which by the way, COMAL would be a very good kindergarten ADA. It could be an introduction to ADA. I really think that. Pascal is an algorithmic language, Algo, etc. Whereas PROLOG... and I think it is with PROLOG ... PROLOG has much the same relation to LISP, as Pascal would have to FORTRAN.

Well, the GOTO... Of course, in COMAL there are of course lots of GOTO, but they are hidden. The interpreter is doing them for you. IF, ELSE, ENDIF ... from IF you can jump to ELSE if the condition is not met. This could be an introduction to ADA. I really think that. Pascal is an algorithmic language, Algo, etc. Whereas PROLOG... and I think it is with PROLOG ... PROLOG has much the same relation to LISP, as Pascal would have to FORTRAN.

So much the better.

Of course, the trick is to get their attention. Once they see it, a little exposure to it, they immediately see that it is just the thing.

They have to see it to believe it. They must have a chance to see what COMAL is all about, to see a few examples, a few real programs.

This is probably the most important point you are indicating there. I heard one tonight who tried to introduce COMAL to some teachers somewhere. They have to see it to believe it. They must have a chance to see what COMAL is all about, to see a few examples, a few real programs, before they use it. You can give long talks, even very delicately formulated talks, about it. But there is nothing like a short example. You will see that, if you want to tell people about COMAL, you will have to show them a program in COMAL, and let them see the syntax, the error messages they get immediately, and the indentation that shows them... Let them look at an IF .. THEN .. ELSE, and REPEAT .. UNTIL, simple things, procedures can come later. As soon as they have seen that, they will know what it is about.

Now, I am running out of words. I can feel it. But don't hesitate to ask questions if you like.

Being one who has almost no experience in COMAL programming, I was interested if you could elaborate on those three levels of programming language that you just mentioned.

I could, but I am afraid it would be too technical. Do you know about LOGO and PROLOG. I would gladly do it afterwards. It is quite technical. The only thing I can say is, COMAL is a very typical algorithmic language. And if you want to know about a LISP language, the easiest way would be to look into LOGO, not at the turtle graphics! Because that has really nothing to do with LOGO. You can do turtle graphics in any language. You see it in USD Pascal for one. That was a real sensation. That he [Kenneth Bowles] used turtle graphics to train university students. And probably what made USD Pascal so popular.

COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716 - Page 9
The Story of COMAL (continued)

Whatever it would be, LOGO is using so called lists, which are more or less what mathematicians would call sets. PROLOG is totally different. It is the language that uses relations. It is more or less abstract algebra.

I have a simple question. Could you relate all the computers that you know that run COMAL now. We'll give you ten minutes.

Oh no. That would be boring.

No. I understand that there are some foreign computers. I would just like to know.

Well, in Denmark we have the old Z80 based microcomputers, which would look very much like Cromemco or IMSAI or whatever. In fact, Comet would look very much like the IMSAI, although the IMSAI used 8080 as far as I know. Then there are new ones coming up now, based on the 8086. Piccolo, and another one called Partner. More or less PC clones. And the Swedish Compis, which is the one Mytech [Alder] COMAL could be used on, also a 80186 based computer. Then the English Amstrad is one you may not know about. Apple has a COMAL in fact, MetaWin COMAL. You have never seen that. It is not very good, I would not recommend it. A new one is being implemented right now. But only if the Apple has a Z80 card (CP/M). Then there will be PC COMAL of course. And the VAX computer. And Wicat. I don't know much about that, only that there is a COMAL for it. And Mytech [Alder] claim that they have one for the Mac. They call it MacCOMAL. Whether that is true or not, I don't know, I never saw it. [It is true ... we have seen the beta test of it working]. And I hate to speak about things I have not seen. Because in this branch, rumors ...

Have you considered introducing a help facility in the COMAL environment.

I hope that COMAL will go on developing. But you should expect it to be more in the environment. But you should expect it to be more in the environment. The language as such I think is probably more or less finished. The definition is closed. There will not be added many more facilities. There are a few things, such as functions. You know that in 0.14 a function can return integers or reals, but not strings. Whereas in 2.0 functions can return strings. In Mytech [Alder] a function can return vectors, which might be an improvement in the language itself. Otherwise it would be to make it more and more user friendly.

What is the Mytech [Alder] version based on?

It is written in C. This means it can run on any computer that has a C compiler. Probably it would be very easy to do, if they haven't got one already, a 68000 version. We are trying to persuade them to put it on the Amiga, whenever that appears.

I've seen advertisements for PROMAL. Is that related to COMAL at all?

Not at all. I saw a German review of it. In Germany COMAL is very well known and is used extensively in many schools. The reviewer started to point out explicitly that PROMAL has nothing to do with COMAL.

Coming back to a point raised earlier. How about the possibility of compiling functions and procedures written in COMAL; compiling them directly into machine code. Once it works, it works forever, and then it runs a lot faster.

That would be a thing that probably anyone would like to have, a COMAL compiler. You would have something like the perfect system then.

That would of course be a thing that probably anyone would like to have, a COMAL compiler. You would have something like the perfect system then. The interpreter to develope interactively the program, to test it. As soon as the program runs as it should, you would take the program and compile it. It should be very easy to do, and I hope that someone will do it.

Let's let Borge get some rest.

Page 10 - COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716
Editor's Disk

I came home after work on Sunday night, June 4, ready to rush down and sign onto QLink for our First Sunday of the month COMAL meeting. My mail included a large envelope from Denmark. A quick glance at it set me wondering. I often get large envelopes from Denmark ... that is the main source of COMAL and its developers! However, the envelopes are always hand addressed. My address on this envelope was typed. It was from Tonder Statsseminarium ... that's Borge's School! My apprehension was confirmed, for opening the envelope I found another envelope ... the one I had just sent to Borge ... unopened with a short note attached to it. This is what the note said:

COMAL Users Group: Letter to Borge Christensen return. We are sorry to tell that Borge Christensen has died on the 13 of May.

Needless to say, that was a sad day, and a sad meeting on QLink. That envelope, with the note still attached to it is being saved, even though seeing it brings a tear to the eye.

Borge Christensen, father of COMAL, was a great man. Some of you may have been fortunate to have heard his talk at the MARCA Commodore Computer Show ... or in California, when he gave the talk whose transcript precedes this column. Borge was quite a speaker! He was interesting! Even exciting! If you were not lucky to have heard him speak, you still have a chance. You can get a video tape of his talk in California. It is a home production (not a slick commercial product). It is available from Mitch Pilchuk, 102 Evergreen St, Sterling, VA 22170 for just $15 (VHS tape only). Thanks to Norm Pollinger for taping the talk! We also filled one Amiga disk full of digitized pictures of Borge from this talk. All Amiga COMALites will want a copy of this disk.

Tribute: We are putting together a disk in tribute of Borge. If you want to contribute to it, send in your program, package or text file right away. Mark it clearly TRIBUTE submission. And, yes, that is Borge on the cover of this issue.

Directories: You don’t realize how much is on some of the disks until you look at its directory! Nearly one full page of tiny print for just the AmigaCOMAL disk!

Networks: get COMAL support on QLink, PLink and soon CompuServe. We're printing free ads for each in this issue for your info.

Challenge: Let's put some fun back into programming! A programming Challenge. Any computer! Any language! Since COMAL makes programming easier, we should have the advantage, right? Tentative schedule: 1st night of each month; 9pm Central time; COMAL Conference area; PLink and/or CIS. We need you on COMAL's team!

Easy: This issue includes virtually our whole COMAL Information Booklet! Many subscribers may not have even seen our 24 page booklet we send out free to those who send us a SASE asking for info. Now you have it (pages 12-33). Feel free to copy any of those pages for club newsletters. A disk with the text files also is available if you want to put any of the files on a local BBS or reformat them into that required by your newsletter.

AmigaCOMAL Ships: The wait is over! The price is under $100 as promised. See a sample page from the manual on page 68. We urgently need COMAL programs for Amiga ... If you convert any COMAL Today programs to Amiga, please send us a copy on disk.

New Textbook: A new 300 page textbook should be shipping by the time you get this. It applies to the newest Amiga and IBM COMALs as well as Power Driver and C64 cartridge. If you want the answers you can get a matching disk (specify what computer format!). See a sample page from the book on page 69.

Power Driver Compiler: Released as ShareWare (on Today Disk #25, plus you can download it from QLink & PLink). If David Stidolph gets enough response from the ShareWare he will do an update with even more features.

Doc Box: Only a few left. Since a minimum order is 1000, we will not stock them after we run out. They are available from office supply stores (but at a higher price).

New Laser Printer: We want our newsletter and new books to look good. Thus we are getting a new LaserJet IIIP printer with a Super Cartridge 2. This will let us print in 10 point Times Roman (this is 10 point) plus Helvetica twice the size we have now for titles. Plus it will allow better graphics! We're counting on your support in the coming months so we can pay for the printer.

Free Contributor Disks: If we use your program or article in COMAL Today or on a Today Disk, you get a free disk of your choice from the left column of the DISKS page of the order form. Just send us a note specifying what we published of yours, and what disk(s) you want in return. This includes all issues since #24.

COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716 - Page 11
What Is COMAL?

COMAL was designed with the beginner in mind, including many interactive features. It is tolerant of mistakes, yet discourages bad habits. This allows a beginner to develop skills necessary in advanced structured programming languages, such as, well, COMAL. The widely acclaimed turtle graphics are an integral part of COMAL, and have been for nearly a decade (see page 18).

Advanced programmers also pick COMAL as their language of choice. They utilize the many advanced and powerful features of COMAL that beginners do not need to even know about. However, as beginners' programming skills improve, COMAL always presents even more power and features for them to use.

Let's examine some features that attract beginning programmers and keep advanced programmers.

In direct mode, you interact directly with the computer. You type in a command and COMAL responds right away. A beginner usually starts with a PRINT statement. The PRINT statement tells the computer to display a message on the screen. This is an important command. Would you want a program to calculate your mortgage payment if it couldn't tell you the answer? In its simplest form, the PRINT statement can display a text constant. For example, type:

```
PRINT "hello"
```

and COMAL responds with:

```
hello
```

COMAL is also good with calculations. When you balance your checkbook, COMAL can help. If your starting balance is $313.76 and you wrote two checks ($250.00 and $125.98) type:

```
PRINT 313.76-250.00-125.98
```

COMAL immediately responds with:

```
-62.22
```

No wonder that check bounced. Actually, a useful program is usually harder than this. But don't worry, COMAL helps you in many ways.

COMAL provides a full screen editor to help you write programs. You don't have to load a special program to use this editor. Just type in the program lines from direct mode as described above. A program line consists of a line number followed by a statement. With its AUTO command, COMAL can provide the line number for you. You just type in the statement.

COMAL's editor doesn't sit around watching you type either. It tries to figure out what you are telling it. If it doesn't understand, it will tell you right when you enter the line! Often it offers advice on how to fix mistakes. This is when you will like COMAL's insert, delete, and cursor features. You don't retype the entire line, just the corrections.

I can hear you saying, "Ok, COMAL is friendly, but what can it really do?" The power of COMAL comes from its commands, file handling ability, and program structures.

You have already seen the PRINT command. COMAL provides several commands to deal with numeric and text data. The data can be provided to a running program from DATA statements in the program, from a file stored on disk, or by the person using the program. COMAL supports all the usual math operators such as addition, subtraction, multiplication, and division. It also knows logarithms, trig functions, two methods for generating random numbers, and a lot more. Text strings may be added together or have sub-strings extracted from them. A built-in string search even tells the position of one string within another if there is a match. Comparison operators allow strings and numbers to be sorted. Numbers can be converted to text format and back again.

COMAL supports two kinds of data files. Sequential files read data from start to finish. You can write to sequential files or read from them, but not both at the same time. Random access files are good if you store lots of records together, but only access a few at a time. You can read any record, update the information, and write it back. You don't have to start at the beginning of the file.

Normally, COMAL uses the screen for all its output. But a special option redirects PRINT statements to your printer or even a disk file. You SELECT the output location. Later, you SELECT the screen again.

Easy so far! But it is COMAL's structures that simplify programming and keep advanced users from switching to another language. COMAL has four loop structures, two conditional branching structures, named procedures and functions with parameters, and an error trapping structure. They are illustrated on page 28. AmigaCOMAL and the newest IBM PC COMAL 3.0 also include a powerful RECORD structure and POINTERS.

The loop structures execute one or more consecutive statements over and over. Most loops have a one-line short form.
The REPEAT and WHILE loops are similar. Both loop an indefinite number of times. The difference is when they decide to stop. The REPEAT loop places its test at the end of the loop. Thus the statements inside the loop will always be executed at least once. The WHILE loop places the test at the start, so it is possible to skip the statements inside altogether.

COMAL also includes the FOR loop. It counts from a starting value to the terminating value. This not only lets you specify the exact number of times the loop should execute, but also gives you a counter to use for other things.

The last loop structure is LOOP. Like REPEAT and WHILE loops, the statements inside are executed an indefinite number of times. However, it's EXIT is from the middle, not the ends.

The conditional structures choose one section of code over another based on a comparison. A multi-line IF structure has its primary test on the first line (optional ELIF secondary tests are also allowed). If the test is true, its statements are executed. Otherwise, an ELSE section is executed (if you include it). A one-line IF is also available.

The CASE structure evaluates one expression at the start (either a number or a text string). The value is compared to the values in the WHEN statements. If the main expression value matches a value in a WHEN statement, the section of code below that WHEN statement is executed. If there is no match, the section of code below the optional OTHERWISE statement is executed.

COMAL automatically indents lines within structures for you! It also capitalizes keywords and puts variable names in lower case. This makes it easy to follow the program.

COMAL also allows you to store program segments on disk. You recall them into future programs using the MERGE command. Lines are renumbered automatically to fit the new program.

COMAL is a powerful language. However, no language can include everything. That's why COMAL includes the possibility to extend itself. You can write long routines, give them names, and call them with a single command, just like a PRINT statement. These routines can be called from direct mode or a running program. These are procedures and functions.

A procedure is usually a set of statements that work well together. Information can be transferred to and from procedures through variables called parameters.

Functions are like procedures -- multi-line structures which can have parameters and are called with a single statement. The difference is that unlike a procedure, a function also returns a value (string or numeric) to the statement which called it.

Don't overlook COMAL's error handling structure. A language which permits interaction with a user should anticipate the possibility for errors which are difficult to resolve. A classic example is division by zero. Many languages give you two choices: test every expression before it has a chance to cause an error, or let the system crash. The first method has two drawbacks: it slows program execution and the test itself may cause a crash.

COMAL gives you a third choice - utilize its Error Handler! This structure can be used to minimize the effects of errors. The Handler can't create the data in a missing file in the case of a file not found error, but it can give you a second chance to put the correct disk in the disk drive. Since the Error Handler is a true structure it is easy to use. Advanced programmers can utilize it fully, complete with nested Error Handlers.

Finally COMAL can be extended and expanded indefinitely with packages. Beginners can use packages without knowing how they are created. Advanced programmers will find packages an exciting way to tailor the COMAL system to their needs. Now even beginners can try their hand at writing packages since AmigaCOMAL allows you to write a package in COMAL (or in Assembler or C).

The above description is not a complete review of COMAL, but should give you an idea of what to expect. All COMAL 2.0 implementations are compatible and include these features (see the chart on page 18). Plus each adds its own special characteristics (such as mouse, windows and speech in AmigaCOMAL). Trace commands are often included to help follow program flow. Some of these enhancements are mentioned in the separate reviews of the different implementations.

The one exception is Power Driver for the Commodore 64. It includes virtually all of Common COMAL, yet you are permitted to give away copies of it! It lacks packages, error trapping, LOOP structure, and external procedures. Also, user defined functions can only be numeric.

Permission granted to copy any or all of these COMAL information pages.
(pages 12-43)

We can provide user groups with multiple copies of these pages as a booklet for distribution at meetings.

Each of the articles in this set is also available as a text file on disk. This allows newsletter editors an easy way to include information in their newsletter, using their special formats.
How To Do It In COMAL

How to see what is on a disk:
Put the disk into the drive. Type: CAT  or  DIR

How to retrieve a SAVED program from disk:
Put the disk into the drive. Type: LOAD "name"

How to retrieve an ASCII program file:
Put the disk into the drive. Type: ENTER "name"

How to run a program that is in the computer:
Type the command: RUN

How to run a program from disk:
Put the disk into the drive. Type: RUN "name" or CHAIN "name"

How to look at a program:
If the program is not in the computer, retrieve it first. To see the program in the computer type: LIST or DISPLAY

How to store the current program to disk:
Type the command: SAVE "name"

How to list the current program to the printer:
The program must be in the computer. Type: LIST "Ip:"

How to store program lines as an ASCII file:
Example, to store procedure pause, type: LIST pause "pause.lst"

How to merge procedures into other programs:
Example, to merge pause into current program: MERGE "pause.lst"

How to renumber a program:
To restore lines numbered by 10's type: RENUM

How to add lines to the end of a program:
The program must be in the computer. Type: AUTO (now type in the lines)

How to write a new program:
Type the commands:
NEW AUTO  (now type in the program)

How to add lines in the middle of a program:
The program must be in the computer. Locate the place you wish to add a line. For example:
0040 PRINT "Welcome to my program."
0050 PRINT "Hope you enjoy it."

Use a line number between the lines. Type: 45 PRINT "Written just for you."

How to erase lines in a program:
Find the line number of the line to be erased (line 80 in this example). Type: DEL 80

You may delete a block of lines all at once. Find the first and last line numbers in the block of lines (lines 500 through 750 for this example), then issue the DEL command:
DEL 500-750

You may also delete a procedure or function by name. Type: DEL name

How to exit COMAL:
To return to the main operating system, type: BYE or BASIC  //C64

How to transfer a COMAL program:
Make sure the program is the current program in memory. Store it to disk as an ASCII file. Type: LIST "name.lst"

Transfer the file to the proper disk format for the other computer (use a conversion program, network, BBS, or direct connect cable).

Retrieve the program into the other COMAL. Type: ENTER "name.lst"

Note: Power Driver includes a substantial subset of the features.

Disk conversion programs available:
- Big Blue Reader for C64/C128 with 1571/1581 (reads/writes IBM PC and CP/M formats).
- PC Util2 (on our boot disks) and CrossDos for Amiga (reads/writes IBM PC 3.5" disks).
- Access64 and The 64 Emulator for Amiga (hook your 1541, 1571, or 1581 to the Amiga).
Common Questions and Answers

Question: What computers can run COMAL?
Answer: Commodore 64, Commodore 128, PET, CBM 8032, MS-DOS, IBM-PC and PS/2, Amiga and CP/M computers. An implementation is under way for Apple IIe/IIC. There are several more versions available in Europe, including Unix, BBC, Acorn, RC, and Compis.

Question: Do you put out a newsletter?
Answer: Our COMAL Today newsletter is the source of up to date information about COMAL. Contributions come from leading COMAL experts as well as users world-wide. Membership is not currently available, but a subscription to COMAL Today is virtually the same thing.

Question: Are your disks copy protected?
Answer: Absolutely NOT! You may make backup copies for your own use. In fact, C64 Power Driver may be given away, as can the Demo disk for CP/M COMAL. You are not allowed to give out copies of the other COMAL system disks. but you can distribute or sell your own COMAL programs.

Question: Is a COMAL compiler available?
Answer: The reason most implementations of COMAL are so last is that they are semi-compiled as you enter the program! In addition to this, most versions of COMAL include a Run Time Compiler, with many advantages. A true compiler is not yet available. However, the Run Time system turns COMAL programs into stand-alone files that can be run without the need of COMAL. Thus you can distribute your COMAL programs to every user, not just those who own COMAL.

Question: Do you recommend that students start with COMAL and progress to LOGO or BASIC?
Answer: Yes! By all means start with COMAL. But No! LOGO and BASIC are not needed once you are using COMAL. BASIC is a step backwards and will only give students bad habits to UNlearn later. And COMAL includes virtually all of BASIC, without its hassles. LOGO is usually chosen for its Turtle Graphics - and these are included in many COMALs. COMAL is the natural and official first language taught in five European countries.

Question: How about sample programs?
Answer: Over 4000 programs on are available on more than 50 different disks for the C64. Many of these are being converted to disks for the new COMAL implementations.

Question: Can a BASIC or LOGO program run under COMAL?
Answer: No. However, it can be rewritten to do so. Likewise, BASIC and LOGO cannot run a COMAL program.

Question: Is COMAL Commodore and IBM approved?
Answer: Yes! It was approved by Commodore England. Commodore Canada introduced it at the World of Commodore Show, giving away over 500 COMAL disks in two days. The COMAL Handbook was approved by Commodore USA. The COMAL Cartridge is manufactured by Commodore Denmark. IBM PC COMAL was produced by IBM Denmark.

Question: Will a Common COMAL program I write on my Commodore 64 run on IBM or Amiga?
Answer: Yes. We use a COMAL program to process orders here. It was written on a Commodore but is now running on an IBM PC compatible with 20 Meg Hard disk. We only had to change the file names and remove one short cut in substrings that Commodore allows but IBM does not.

Question: COMAL lacks the left$, right$, and mid$ functions. Can I do substrings?
Answer: Yes. To get a portion of a string just include the position of the first and last character within the string. Example:

\[
\text{\texttt{part$:=text$( start:ending)}}
\]

See page 20 for more string information.

What Do They Say?

"If languages interest you, this one is well worth a look... You may find that it's just what you have been looking for." Jim Butterfield, COMPUTE!

"COMAL was just what I was looking for." Colin Thompson, RUN

"I can recommend a better, faster, and cheaper programming language ... COMAL, the most user friendly language around." Mark Brown, INFO

"Combines some of the best features of languages like Logo, Modula, Pascal, and Ada in an easy to use format." Ahoy!

"I don't have enough space to list all the good points!" Noland Brown, Midnite Software Gazette

"COMAL is the optimal educational computing language." Jim Ventola
This database program displays brief instructions and asks for your text input. Everything you type before the last character on a line is considered data. The last character is the command. Thus what you type resembles an English sentence.

- End your line with a period, and the data is written to the file.
- End with a ? and a text search starts.
- Type just a ? for a full listing of the file.
- Type just a @ to quit.

An example RUN is included below. Text the user types is underlined, the computer response is in italics, explanations are in «small print».

**RUN**
«file is created»
> COMAL is nicer than BASIC.
«text is added to file»
> Len Lindsay is Captain COMAL.
«text is added to file»
> nicer?
COMAL is nicer than BASIC.
> Len Lindsay: 608-222-4432.
«text is added to file»
> COMAL?
COMAL is nicer than BASIC.
Len Lindsay is Captain COMAL.

> ?
freeform.dat.
Demo.
COMAL is nicer than BASIC.
Len Lindsay is Captain COMAL.
Len Lindsay: 608-222-4432.

> @
«program ends»

**Power Driver** does not have an error handler. The following replaces the last PROC in the program:

```
PROC filecheck(filename$, filenum) CLOSED
DIM d$ OF 2 // only want first two characters
OPEN FILE filenum, filename$, READ // try opening it
d$=status$ // get the status of the disk drive
CLOSE FILE filenum
IF d$="M00" THEN // "M00" (two zeros) means it exists
OPEN FILE filenum, filename$, WRITE // create it
WRITE FILE filenum: filename$, "Demo" // first record
CLOSE FILE filenum
ENDIF
ENDPROC filecheck
```

Read the first column on the next page for info on how to enter this same exact program into any COMAL programs, but /UTO will provide them for you as you type:

**PAGE** // clear the screen
PRINT "Free Form Database by Joel Rea«
PRINT "Last character is the command:"
PRINT "? -- add to file,"
PRINT "? -- search file,"
PRINT "@ -- exit program,"
DIM line$ OF 80, text$ OF 80
DIM filenames$ OF 20, command$ OF 1
filenames$="freeform.dat"; num=7 // file specs
filecheck(filenames$,num) // if file not exist, create it
REPEAT
get line from user
CASE command$ OF
WHEN "." // period
   add line to file
WHEN "?"
   display matches
OTHERWISE
   NULL // do nothing
ENDCASE
UNTIL command$="@"
END // optional end statement
//
PROC get line from user
PRINT // blank line
REPEAT
INPUT "\:" line$ // user types in something
max=LEN(line$) // length of input
UNTIL max>0 // must enter something
command$=line$(max:max) // just last character
IF max>1 THEN // more than just a command
   line$=line$(1:max-1) // remove last character
ELSE
   line$="" // only a command, no other text
ENDIF
ENDPROC get line from user
//
PROC add line to file
OPEN FILE num,filenames$,APPEND
WRITE FILE num: line$
CLOSE FILE num
ENDPROC add line to file
//
PROC display matches
OPEN FILE num,filenames$,READ
WHILE NOT EOF(num) DO
   READ FILE num: text$
   IF line$="" OR line$ IN text$ THEN PRINT text$="",
ENDWHILE
CLOSE FILE num
ENDPROC display matches
//
PROC filecheck(filenames$, filenum) CLOSED
TRAP // check if the file exists
OPEN FILE filenum,filenames$,READ // try opening it
HANLER // if file does not exist we drop down here
OPEN FILE filenum,filenames$,WRITE // create file
WRITE FILE filenum: filenames$,"Demo" // first record
ENDTRAP
CLOSE FILE filenum // close the file from either open
ENDPROC filecheck
// Power Driver does not have TRAP ..., HANLER
// replace the above procedure with the one on the left
Free Form continued...

It is easy to enter a COMAL program. COMAL helps you out every step of the way. Before you start a new program, erase any current program. Type:

```
NEW <Enter>
```

After each line, hit the «Enter» key (labeled «Return» on some computers).

COMAL programs require line numbers. However, we list them without line numbers since you can use AUTO mode to enter the program. Type:

```
AUTO
```

COMAL will respond with:

```
0010 □
```

The cursor is after the line number 10. You just type the first program line and hit «Enter». COMAL then prompts you with the next line number:

```
0020 □
```

Type each line in the program in this manner. COMAL will help you. Don't type in the spaces to indent the lines. COMAL does that automatically for you. No need to capitalize the keywords. COMAL will do that for you too. Don't type the name after ENDPROC. COMAL will add it for you later. If you make a mistake, COMAL will offer you advice.

After typing all the lines, stop AUTO mode (varies by computer):

**Power Driver:** Hit «Return»

**C64 Cart:** Press «STOP»

**Amiga:** Press «Esc»

**CP/M:** Press «Esc»

**IBM:** Type «Ctrl» + «C»

Before you try out the program store it on disk to be safe. If anything goes wrong (lightning strikes), you can reload the program from disk. Type:

```
SAVE "freeform"
```

Now run the program (see sample on the previous page). Make sure you have the program in memory before continuing!

### Routine Stealing!

This is a COMAL specialty. It is easy for you to steal routines (called procedures or functions) from other COMAL programs, and merge them into a new program you are writing!

To do this just store the routine to disk in ASCII text form (with the LIST command). To merge it with a future program use the MERGE command. As an example let's store the procedure FILECHECK:

```
PROC filecheck "filechk.lst"
```

Power Driver does not list routines by name, only by line range. Therefore, you must first find the starting and ending line number of the routine. It also uses its ENTER command to do the MERGE later, so it is best to renumber the lines into the 9000 range. Do this:

```
RENUM 9000:1
LIST 9005-9083 "filechk.lst" □
```

### Phone Book, Phase 1...

Now you are ready to try the second program, doing some routine stealing! First erase the current program. Type:

```
NEW
```

As before, the AUTO command gives you line numbers. Type:

```
AUTO
```

Now just enter the program as shown in the next column. When you get to the end, stop AUTO mode (see previous column).

Now we will merge the routine we are stealing from Free Form Database. Type:

```
MERGE "filechk.lst"
```

COMAL will automatically renumber the lines as they are merged into the program!

Power Driver does not have MERGE. It uses ENTER instead, and does not renumber the lines for you. Type:

```
ENTER "filechk.lst"
```

The program is now ready to SAVE and RUN. However, to satisfy your curiosity, list it to see the merged procedure:

```
LIST
```

### Electronic Phone Book

```
DIM name$ OF 20,phone$ OF 12
DIM filename$ := "phone.dat",num:=5://filespec
filecheck(filename$,num) // check on file
REPEAT
PRINT "e=enter f=find l=list q=quit"
CASE INKEYS OF
WHEN "e","E"
enter'no: name$ := INPUT "What name? " : name$ search'for(name$)
WHEN "f","F"
search'for(""")
WHEN "q","Q"
END // This ends the program here!
OTHERWISE // not valid key
PAGE // clear the screen
ENDCASE
UNTIL TRUE=FALSE // forever
PROC enter'name
INPUT "Enter name: ": name$
IF name$>"" THEN
WHEN "e","E"
enter'no: phone$ := INPUT "Enter phone: ": phone$ add'to'file
ENDIF
ENDPROC add'to'file
PROC search'for(search$)
OPEN FILE num,filename$,READ
WHILE NOT EOF(num) DO
READ FILE num: name$,phone$
IF search$="" or search$ in name$
PRINT name$,TAB(21),phone$
ENDIF
END WHILE
CLOSE
PRINT "Hit <return> when ready."
WHILE inkey$<>chr$(13) DO NULL
ENDPROC search'for
<stop AUTO mode here, see left side>
```

Here's a sample of what it does:

```
RUN
e=enter f=find l=list q=quit
```

```
<enter name: Captain COMAL
<Enter phone: 608-222-4432
```

```
e=enter f=find l=list q=quit
<Enter name: Emergency
<Enter phone: 911
```

```
e=enter f=find l=list q=quit
<what name? COMAL
<Enter name: Captain COMAL
<Enter phone: 608-222-4432
```

```
e=enter f=find l=list q=quit
```

COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716 - 17
TURTLE GRAPHICS

Turtle Graphics were what made LOGO popular, yet COMAL has had a faster turtle for years! These Turtle commands work on all our COMALs:

BACK : move turtle backwards
   back(<length>)
   back(50)

BACKGROUND : set screen background color
   background(<color number>)
   background(2)

CLEAR : clear the graphics screen
   clear // also can use clearscreens

DRAWTO : draw a line from current point
   drawto(<x coord>, <y coord>)
   drawto(50, 80)

FILL : fills in area with current color
   fill(<x coord>, <y coord>)
   fill(50, 80)

FORWARD : move turtle forward
   forward(<length>)
   forward(100)

GETCOLOR : returns color of specified pixel
   getcolor(<x coord>, <y coord>)
   print getcolor(50, 80)

GRAPHICSCREEN : turn on graphic screen
   graphicscreen(<mode>) //Power Driver use:
   graphicscreen(0) //SETGRAPHIC(<mode>)

HIDETURTLE : make turtle invisible
   hideturtle

HOME : put the turtle in its home position
   home

LEFT : turn turtle left
   left(<degrees>)
   left(90) // a right angle

MOVETO : move to loc without line
   moveto(<x coord>, <y coord>)
   moveto(50, 80)

PENCOLOR : set turtle drawing color
   pencolor(<color number>)
   pencolor(2)

PENDOWN : put pen down, turtle draws
   pendown

PENUP : pick pen up, turtle does not draw
   penup

PLOT : plot a point in current color
   plot(<x coord>, <y coord>)
   plot(50, 80)

PLOTTTEXT : put text on graphics screen
   plopttext(<x coord>, <y coord>, <text>)
   plopttext(0, 24, "press space to continue")

RIGHT : turn turtle right
   right(<degrees>)
   right(180) // reverse direction

SETHEADING : set turtle heading
   setheading(<degrees>)
   setheading(180)

SETXY : set turtle x, y coordinates
   setxy(<x coord>, <y coord>)
   setxy(50, 80)

SHOWTURTLE : make turtle visible
   showsturtle

textscreen //Power Driver use SETTEXT

TURTLESIZE : set turtle size (0 to 10)
   turtlesize(<size>)
   turtlesize(6)

SPIROLATERAL

The screen dump below is from AmigaCOMAL. The program (top of the next page) will run on all COMALs. Rod Graham presented this program in COMAL Today #4. He said students in his class would rather play with this program than Frogger.
Spirolateral: The Program

USE turtle // Power Driver use SETGRAPHIC(0)
hideturtle
nowrap // Power Driver omit this line
DIM sequenceS OF 10 // limit R and L sequence to 10
REPEAT
textscreen // Power Driver use SETTEXT page
PRINT "Once running, press Q to Quit"
PRINT "or any key for the next segment."
PRINT AT 4,1: "Try size=20 angle=127"
INPUT AT 7,1: "Sequence of R and L: ": sequenceS
PRINT AT 7,1: "Try size=20 angle=127"
INPUT AT 9,1: "What angle: ": angle
INPUT AT 11,1: "What size (O=Quit): ": length
IF length<=O THEN END // end it here if 0
INPUT AT 13,1: "What sequence: ": sequenceS
graphicscreen(O) // Power Driver use SETGRAPHIC
home
REPEAT
FOR seq=1 TO len(sequenceS) DO
CASE sequenceS(seq:seq) OF // one char substring
WHEN "r", "R"
right(angle)
WHEN "l", "L"
left(angle)
OTHERWISE
null // error in sequence input
ENDCASE
ENDFOR seq
UNTIL done // see the function DONE below
UNTIL TRUE=FALSE //forever, see END in the-INPUT section
FUNC done CLOSED
DIM keypressS OF 1
keypressS="" // init
WHILE keypressS<CHR$(1) DO keypressS=KEYS //look for key
IF keypressS IN "Qq" THEN RETURN TRUE //yes they're done
RETURN FALSE // all other keys
ENDFUNC done

Tree: The Program

This program by Phred Linn is taken from COMAL Today #8. It runs on all our COMALs. The screendump from AmigaCOMAL at the bottom of the previous column is not as impressive as the screen itself.

USE turtle // Power Driver use SETGRAPHIC(0)
hideturtle
graphicscreen(O) // Power Driver use: SETGRAPHIC(O)
length=44; triangle=44 // try other values for fun!
start=2*(length DIV 20)+4
back(4*start)
limb(start) // the trunk for the tree
tree(length,treeangle)
//
PROC tree(length,angle)
IF length<2 THEN RETURN // stop the recursion!
left(angle)
limb(2*(length DIV 20))
tree(length/2,RND(10,70)) //recursive call, random angle
back(length)
IF angle>25 THEN
right(angle)
limb(2*(length DIV 20))
tree(length/2,RND(10,70)) //another recursive call
back(length)
right(angle)
limb(2*(length DIV 20))
tree(length/2,RND(10,70)) //another recursive call
back(length)
ELSE
right(angle*2)
limb(2*(length DIV 20))
tree(length/2,RND(10,70)) //another recursive call
back(length)
ENDIF
left(angle)
ENDPROC tree
//
PROC limb(count)
forward(length)
right(90)
FOR x=1 TO count STEP 2 DO
forward(x)
right(89)
forward(length)
right(91)
forward(x+0.035*length)
right(91)
forward(length)
right(89)
ENDFOR x
forward(count/5)
left(90)
ENDPROC limb

C64 Power Driver Note:

Graphics commands do not require ( ) parentheses. This was considered better for younger users. Note that any command with only one parameter (such as FORWARD 100 can be issue with or without ( ). The commands with 2 or more parameters must not use ( ). Power Driver also includes many built-in Sprite commands that give you complete control over the sprites without using any PEEK or POKES.
**People Link & QLink - On-Line COMAL Support**

**Subj: string func? From: Icarus**
Can someone explain strings and string functions, particularly placement of the colons?

**Subj: strings From: Captain C**
Since BASIC doesn't have user defined string functions, they are unfamiliar to most. Compare them to a built-in function. We talk about `SPC$(5)` returning 5 spaces:

```plaintext
AS=SPC$(5)
PRINT "HI",SPC$(5),"JIM"
HI JIM
```

See the 5 spaces in the printout. And `AS` now is 5 spaces. Next we can write a string function:

```plaintext
FUNC jim$(num) CLOSED
DIM temp$ OF num$3
temp$="" // start with nothing
FOR x=1 TO num DO temp$:+"JIM"
RETURN temp$
ENDFUNC jim$
```

Now try it:

```plaintext
PRINT jim$(3)
```

JIMJIMJIM

Now, about the colons. Colons are used to specify substrings:

```plaintext
AS="BIRTHDAY"
AS(1:3) would be "BIR"
AS(2:5) would be "IRTH"
```

The colon just separates the start position from the end position.

**Subj: colons From: Comalite J**
Think of a string as a one dimensional array of characters:

```plaintext
DIM mystring$ OF 6
```

At first, it looks like this:

```plaintext
| +---+---+---+---+---+---+
| 6 | "MAX (DIMmed Length)
| 0 | "Current-Length
| +---+---+---+---+---+---+
| a | a | a | a | a | a |
| +---+---+---+---+---+---+
| 1 2 3 4 5 6 |
```

Now execute this:

```plaintext
mystring$="Joel"
```

```plaintext
| +---+---+---+---+---+---+
| 6 | "MAX-LEN
| 0 | "CUR-LEN
| +---+---+---+---+---+---+
| j | o | e | l | a | a |
| +---+---+---+---+---+---+
| 1 2 3 4 5 6 |
```

A similar shortcut is available for substrings which end with the last character. Just specify the first character followed by a colon. The next two lines are the same:

```plaintext
PRINT mystring$(5:)
PRINT mystring$(5:LEN(mystring$))
```

which, in this case, is:

```plaintext
PRINT mystring$(3:6)
```

The result is "el R".

For a one character substring, just specify the `start` and `end` as the same:

```plaintext
PRINT mystring$(3:3)
```

In COMAL, you can ASSIGN to a substring! For example, assuming `mystring$` equals "Joel R", then:

```plaintext
mystring$(2:4)="an"
```

Characters 2-4 were changed without affecting the rest of the string. Substrings are padded with spaces to the right if needed:

```plaintext
PRINT mystring$(2:4)="an"
```

```plaintext
PRINT mystring$
```

```plaintext
PRINT mystring$(2:2)="im V"
PRINT mystring$
```

```plaintext
jim V 
```

The above is an example of on-line COMAL support. In addition to message boards, there are upload / download libraries of COMAL programs and national monthly COMAL meetings: every first Sunday and second Thursday on QLink; every third Wednesday on People Link (and CIS pending). Meetings are at 9pm Central time.

**People Link - to get to our COMAL Support Section just /GO COMAL. To sign up on PLink call 1-800-524-0100. Email to CAPTAIN C.**

**Quantum Link - to get to our COMAL Support Section go to CIN; Computing Support; Programmers Workshop. To sign up on QLink call 1-800-392-8200. Email to CAPTAIN C.**

**CompuServe - to get to our COMAL Support Section ... GO AMIGAV then LN to find COMAL listed. To sign up on CIS call 1-800-848-8199.**

---

20 - COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716
COMAL is an easy to use, yet powerful language. It is standardized and available for many popular computers. COMAL has become the common ground between popular computers. The chart on page 28 shows the common keywords. The pages following the chart give more details about each keyword, including syntax and an example.

In addition to sharing common keywords, each COMAL system provides the same programming structures (see page 27).

But it goes even further than just keywords and structures. Each COMAL has a compatible environment. They all share the same look and feel. Once you know COMAL on one computer, you will feel right at home using COMAL on a different computer! They share a common friendly environment.

The built-in full screen editor makes program entry easier. If you can see it on the screen, you can edit it. COMAL checks each line you enter -- when you enter it. If it sees a mistake it tells you right away, and even attempts to help you fix it! The SCAN command checks your entire program to verify the multi-line structures. Before you run your program you will know that each line and structure is proper.

When you LIST a program, COMAL automatically indents the structures for you! It also capitalizes the keywords but shows your variable names in lower case. You get a very readable program listing -- especially if you take advantage of long variable names. You can use names as long as the screen line permits!

COMAL also helps you fine-tune your program. A TRACE facility helps you locate errors. The Error Handler allows your programs not only to catch errors, but often to correct them while the program continues running.

You should enjoy writing programs in modules (procedures and functions). You give each one a name, and pass information back and forth with it via parameters. It can be open and share all variables with the main program or be CLOSED so that all variables and data inside it are considered local. An IMPORT statement allows you access to specific variables if needed. Once you have defined a module, you can call it from direct mode -- just like any other COMAL keyword! You can even call the module from inside itself (called recursion).

Your modules are actually extending COMAL. In fact, a module can be stored on disk as an "external" procedure or function. Any program can then include that module simply by adding a one line "header". COMAL automatically gets the module from disk when it needs it.

Packages are another method of extending COMAL. They are usually written in machine code and are specific to each computer. Part of the beauty of packages is that they allow fast access to hardware dependent features. An even greater advantage is that you call them from COMAL by name, just as if they were part of COMAL itself. You can use existing packages without knowing how they were created.

Normally, the user can stop a program by pressing the break key (usually a «stop» or «esc» key). However, there are times when you want to be sure that the program is not stopped. COMAL allows the break key to be disabled and re-enabled anywhere within a program. Furthermore, it provides an ESC flag to notify the program when the user presses the break key.

Since COMAL is available on many different computers, you need a convenient method of transferring programs between them. The MERGE and ENTER commands accept a program stored as an ASCII file. Of course COMAL also provides a way to store program lines in ASCII format: the LIST command! You can LIST your program to any output location: screen, printer, disk file, or even modem.

BENCHMARK TIMINGS

We did some quick benchmark tests. The results are shown below. The programs used are referenced under the chart. COMAL is nearly always faster than BASIC. This difference is significantly more dramatic when dealing with strings. Also it seems that AmigaCOMAL is on top between the various COMAL implementations. See also page 26.
There are two current versions of COMAL for the Commodore 64. They are the most popular versions in U.S.A. and therefore the best supported and documented.

Power Driver is the souped up replacement to COMAL 0.14. It is an introductory version that almost meets the Common COMAL standard, but is lacking in a few areas. However, it is far better than other C64 languages and costs just $9.95 including a 22 lesson interactive tutorial. It includes built-in sprites and turtle graphics.

The COMAL 2.0 cartridge is a full Common COMAL with 11 built-in packages. It's capabilities amaze even professional programmers. It includes sprites, sound, turtle graphics, fonts, and more.

COMAL 2.0 has commands to support virtually every hardware feature on the C64. It has the graphics and sprite commands of Power Driver plus much more. It adds commands to draw circles and arcs. You can change numeric limits of the graphics screen for function plotting. Automatic sprite animation is supported, allowing a sprite to move and change its shape while the program is doing something else; like sound.

There are commands to play a note, and set its frequency, duration and waveform. An entire song can be played by putting the notes and durations into an array and telling COMAL to play it. The tempo will be exact because interrupts are used for timing; you don't need to write delay loops.

Several peripherals are supported. Joysticks, paddles, and lightpens can be used with built-in commands. The modem port can be selected as the default input/output device.

If you'd like to change the C64 character set, the font package makes it possible to redefine the characters. If you create a special font for a game, the font can be saved with the game. Next time it's loaded, your special font is automatically installed!

You have to know assembly language to create a package for C64 COMAL 2.0, but not to use a package. Complete documentation is available, explaining interfacing machine language to COMAL programs. COMAL 2.0 Packages, by Jesse Knight, explains how to create a package. Packages Library, by David Stidolph, on the other hand, explains how to use existing packages (no knowledge of machine language is necessary).

It's even possible to burn your favorite packages into an EPROM and plug them directly into the COMAL cartridge, ready to use every time you turn the computer on. In fact, an EPROM chip called Super Chip is available now. It adds about 100 new commands to COMAL and includes support for the C128.

Power Driver has all of the structures of Common COMAL except LOOP, packages and the Error Handler. It supports procedures and functions, but not nested or external. Also, user defined string functions are not implemented. The ability to IMPORT variables is missing, but procedures and functions do not need to be imported.

Even without the above commands, Power Driver is a powerful language which exceeds others. The graphics commands are among it's best features. The built-in turtle graphics allow you to plot points on the graphics screen or draw a line from one point in any direction for any length. You can fill in shapes and mix text with the pictures. 16 colors are supported in hi-res and multi-color modes.

If drawing pictures is not enough, try sprites. You must first define a shape (often from data statements or a file). Once a shape is defined, it can be assigned to a sprite. Then you can display it anywhere on the graphics screen in any color. It is possible to cycle through several shapes to create a cartoon. Motion can be simulated by adjusting the position of the sprite. Up to eight sprites can be shown at once and there are commands to tell if they touch each other. No peeks or pokes are needed while using sprites.

HOW DOES IT RATE?

<table>
<thead>
<tr>
<th>Overall Rating</th>
<th>A</th>
<th>Reliability</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of Use</td>
<td>A</td>
<td>Error Handling</td>
<td>A</td>
</tr>
<tr>
<td>Documentation</td>
<td>A</td>
<td>Value for Money</td>
<td>A</td>
</tr>
</tbody>
</table>

COMAL Starters Kit rated by The Book of Commodore 64 Software

<table>
<thead>
<tr>
<th>Performance</th>
<th>10</th>
<th>Reliability</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of Use</td>
<td>9</td>
<td>Documentation</td>
<td>8</td>
</tr>
</tbody>
</table>

COMAL Starters Kit rated by The Best Vic/C64 Software

(10 is highest possible rating)

Overall rating: ••••

C64 COMAL 2.0 cartridge rated by INFO magazine

(5 stars is the highest possible rating)

Overall rating: ••••

COMAL Starters Kit rated by INFO magazine
AmigaCOMAL is already my favorite COMAL. It is fully Common COMAL compatible, except LINK is performed by USE (this is actually better).

AmigaCOMAL includes all the fun of Speech and Turtle Graphics plus it has the most advanced COMAL features such as RECORDs and POINTERS, plus a more advanced implementation of packages and the error handler. It is faster than BASIC, yet easier to use and more powerful.

AmigaCOMAL has many useful features, over and above the standard friendly COMAL. TIMES and DATES are included, as is a built in TIMER. When listing a program, you can stop it, and then just by moving your cursor up to the top of the screen, cause the program to be listed backwards to see any lines you might have just missed. Also, when entering a program line, as soon as you hit «Enter» the line is checked for proper syntax. If there is an error, the cursor is put on the point of the error, and a message window pops up. If the line is OK, it is immediately Re-Listed, with proper indentation, keywords in CAPS, omitted words filled in, := converted to :=, and so on.

Packages are a way of extending the COMAL system. Until now, packages always had to be written in Machine Code or Assembler. AmigaCOMAL allows you to write packages in COMAL or C as well! Now anyone can add extensions to COMAL, written in COMAL itself.

For example, you may wish that COMAL had a FILE:EXISTS function that would check if a file exists, and a TYPE command, to type out the contents of a text file. You can add those extensions to COMAL yourself. We did those two as an example for you, and include the package on the disk so you can see how easy it was (we call the package TYPE). All you have to do is issue the command: USE TYPE and AmigaCOMAL links the commands in our TYPE package into the COMAL system. Once you do that, you can type out any text file in direct mode (or within a program) like this:

```
TYPE("filename.txt")
```

The TYPE command checks with FILE:EXISTS before trying to type out the file, so it is smart. It also can ignore the «carriage return» in files from IBM and C64 (Amiga uses only «line feed» at end of line).

There are many packages on the AmigaCOMAL system disk. To see what new commands are in a package, type:

```
USE packagename
LISTPACK packagename
```

There also is an insert mode for program editing, just like with a word processor. You also can open a blank line anywhere on the screen (nice to type in the line you wish to add to the program). Also FIND and CHANGE commands are included to make it easier to edit your program.

Other nice expanded features include WAIT. The command WAIT 5 will wait 5 seconds. The command WAIT will wait for a keypress or mouse click. Both are very handy and useful, especially to beginners. And since the Amiga is multi-tasking WAIT does not use any CPU time. AmigaCOMAL also gives you a protected input field, like that in professional software. The user is restricted to where they can type in reply to an INPUT request. Full access to subdirectories is also included, similar to DOS.

A Compiler is available as a low cost option. It allows you to turn a COMAL program into a stand-alone file that can be run on any Amiga without the need of COMAL. This allows you to distribute your programs to any Amiga user, not just those who own COMAL.

AmigaCOMAL has its own preferences file that lets you set up how much memory to allocate to the COMAL system (up to your full installed memory) and the way the windows come up. You have a Command Window and an Execute Window. They can be on separate screens, on the workbench, or even combined into one screen (like other COMALs with one screen).

You can PASS commands to AmigaDOS or open a new CLI window from inside COMAL. Plus, all the intuition, Exec, DOS, etc., routines are available to you, complete with all the structures pre-defined. Advanced programmers also have commands to Allocate and Deallocate data areas. TRACE with a Single Step mode is available to help in program debugging. And the Error Handler also includes a RETRY command, that allows the trapped statements to be executed over again.

There are three types of integer types (to be compatible with C structures). You can assign a value to every element in an array with one command. For example: READ table will read values from DATA statements into the array named table. Or give every element that same value like this: ITEM(i):="not assigned". You also can automatically determine the top and bottom indices to any array with the commands MAXINDEX and MININDEX.

AmigaCOMAL is not copy-protected. You are encouraged to make a backup copy for yourself. However, you are not permitted to give out copies to others. (See page 26 for speed comparisons.)
UniComal IBM PC COMAL 2.2 runs under MS-DOS version 2.0 or higher on machines with at least 256k of memory. It allows 64k for program storage and another 64k for variables. It supports systems with the math coprocessor. It's price just dropped to only $165 including a compiler (previously it was about $800). It comes with an extensive reference manual packaged in the standard IBM PC style mini-binder. It also includes a spiral bound tutorial manual and many demo programs on the disks (you must specify either 5½ or 3½ disks). School licenses available too.

It is a full Common COMAL (except output location specification) plus much more. It has time and date functions and supports sub-directories with commands such as CHDIR and MKDIR. It also gives the option of passing commands directly to MS-DOS. The error handler adds RETRY to allow a second chance for many operations.

It includes Hercules support in addition to the standard Graphics, Sound and System packages. The graphics package supports turtle graphics and CGA, EGA or VGA graphics screens. The system package has commands for defining the function keys, setting textwindows, converting numbers to hexadecimal or binary format, converting strings to all upper case or all lower case, and more. Other packages are available as added cost options including UniDump (support for Laser Jet / Think Jet), UniMouse, UniNet and UniMatrix.

We use UniComal IBM PC COMAL 2.2 here to run our 3000 line COMAL program to handle the order processing. We enjoy the built in text screen windowing. It makes it easy to pop up help menus. The protected input fields give us a professional feeling system. Plus it has the full screen input that we were used to with Commodore COMAL. (If you are used to C64 COMAL, you will feel right at home with IBM PC COMAL).

Every time you enter a program line, COMAL checks it for correct syntax. If it is OK, it is immediately and properly relisted. Keywords are capitalized, = is converted to :=, THEN and DO are added as needed, and so on. While in AUTO mode, COMAL starts each new line with the cursor at the proper indentation level. If an error is found in any line you enter, COMAL does not accept it and immediately informs you of the problem by placing the cursor on the point of error and printing an informative message beneath the line. Once you correct the mistake, the error message is removed from the screen and anything it overwrote is replaced (it is a non-destructive error message).

You have full control over the function keys, normal plus Shift, Ctrl, and Alt, totalling 40 keys in all. Plus you may have different definitions in edit mode and while a program is running. Yes, 80 possible keys! Thus, when a program ends, any changes it made to function keys will not affect the way they are set up for direct mode commands. There also is an option to display a small one line menu of the keys on the bottom line of the screen. This line will change as you press Ctrl, Alt or Shift to adjust the values of the function keys for that condition («shift»+«Fl» is different than just «Fl» so the display changes to reflect this).

Since this IBM PC COMAL is much more accurate than other COMALs (double precision) it includes a DIGITS command that can be used to set what level of precision you wish to have.

Full control of subdirectories is included, comparable to their DOS counter parts. Every element in an array can be set to a specified value with one command. The top and bottom limits of an array can also be determined using MAXINDEX and MININDEX.

The compiler included with the package allows you to convert any COMAL program into a stand-alone file that can be run directly from DOS without needing COMAL. This allows you to distribute your COMAL programs to any PC user, not just those who own COMAL.

UniComal also has an Object Oriented Programming version of their COMAL that was just released. COMAL 3.0 is available for IBM PC and PS/2 and has an OS/2 support option. The price is more than double that of COMAL 2.2. Both version 2.2 and 3.0 are current and supported.

COMAL 3.0 adds many new features including POINTERs and a RECORD structure. Packages are replaced by modules and can be written in COMAL, C or Assembler. Another major advance is that programs can be over 64K long, up to the limit of installed memory. However, in order to add these features, some other things were changed. Scope now is static rather than dynamic. Statements cannot be issued in direct mode (so turtle graphics cannot be interactive anymore). Thus it appears that this version is aimed specifically at professional programmers. Version 2.2 is also good for professionals, but is great for home programmers and schools too.

Until AmigaCOMAL was released, IBM PC COMAL was the most extensive and fastest COMAL available. Now it's a close contest between the two. You'll be pleased with either! See the benchmark results on page 26.
CP/M COMAL 2.10 runs under CP/M version 2.2 or higher. We tested it on a Commodore C128, Epson, and Kaypro. It is a complete Common COMAL, and has several enhancements as well.

In addition to the usual full screen editor, you can insert a line between two other lines, delete the current line, or erase a line from the cursor position to the end of the line. An INSTALL program on the COMAL system disk lets you assign the keys to do these things if your computer does not already have keys defined for these tasks.

Two important commands included in CP/M COMAL are TRACE and SYMBOLS. TRACE allows you to single step through a program. As each line is executed, it is listed to a user defined device along with a dump of all intermediate calculations. SYMBOLS prints the names of all active variables and their current type.

CP/M COMAL allows more than one program to be in memory at a time. They can't call each other or share variables, but the EXTERNAL section makes an ideal scratch pad. Just type the command: EXTERNAL, press «return», and you are in a new program area. To get back again, type the command: MAIN.

CP/M COMAL allows the programmer access to the machine at several levels. There are commands to store or retrieve a value directly from each of the CP/M registers. You also may CALL a machine language routine starting at any address. In addition, you may include in-line machine code to execute short, relocatable routines. Advanced applications should use packages, which support parameter passing with procedures and functions called by name.

There also is a special graphics package for C128 computers running under CP/M. Everyone should be able to use the package. A DOC file is included on the disk for added information.

A Run Time compiler is a low cost option that takes a CP/M COMAL program and produces a .COM file which can be executed directly from CP/M mode. You can give the compiled program to anyone, not just COMAL owners.

A CP/M COMAL demo system is available. It includes all the features of the full COMAL system except SAVE, ENTER, and MERGE. You can load a COMAL program, list and alter it, or type in your own programs. You can list your programs to disk or printer for use in word processors and newsletters. But you can't retrieve them since the ENTER and MERGE commands are disabled. Try before you buy.

PET COMAL - Yes, we even have COMAL 0.14 for the Commodore PET 32K computer! This disk loaded system is a subset of Common COMAL and does not leave much free memory.
System Dependent Variations

Any programmer who has worked on more than one computer system will tell you about subtle differences among them. As standard as COMAL is, it also must deal with system variations. Anyone programming on one computer for personal use does not have to worry about this. However, those who use several systems should make a point of learning the differences. For example, graphics were omitted from the COMAL standard because of the incompatibility of the screen and color resolution on various computers. However, there is a common set of graphics/turtle keywords (see page 18).

One difference comes with devices and filenames. COMAL usually supports the naming conventions of the specific computer system. Commodore filenames can contain as many as 16 characters, while IBM and CP/M names are restricted to eight characters plus a three character extension. IBM and Amiga also support subdirectories and time/date stamping of files. Commodore disk drive names use numbers; CP/M and IBM use letters; Amiga uses words such as DF0. All COMALs call the screen and printer DS: and LE, except IBM, which uses the name CON; for the screen and several names for the printers.

Another difference concerns data storage. Most computers can use 5¼ or 3½ inch disks, yet each computer has its own method of storing information on that disk. Thus an IBM PC cannot read a disk written on a Commodore 1541 drive. However, this is not the only problem. There are ways to transfer data between different computer systems (including modem transfers). But even if the computers can share data, there still is one more factor to consider.

Different methods are used to represent data and programs. The user never needs to understand this, that is one of COMAL's jobs. What is important to the user is knowing that COMAL has two methods for storing programs and data on disk. One uses compact files for quick access. WRITE FILE and SAVE create these efficient files. They can only be understood by the version of COMAL that wrote them. COMAL also supports ASCII text files which can be read by most computer software including the other versions of COMAL. LIST, DISPLAY, and PRINT FILE produce ASCII files.

A debate has been going on for years concerning the null string and the IN operator. For example, what should "" IN ""abc"" return? The COMAL standard originally stated that the result was to be TRUE (true has a value of 1). A slight enhancement on this theme returns the length of the second string plus one. This value is accepted as true, and it allows a program to differentiate between the null string and a string matching the first character in the second string. Now, Common COMAL specifies that the null string is not part of any string. Thus "" IN ""text"" will always be FALSE.

A similar situation involves KEYS$. What should it return if no key is pressed? Power Driver and C64 COMAL cartridge return CHR$(0); other COMALs return "" (null string).

Another situation involves PRINTing several items on one line. There are 3 common ways to do this:

- with no space between the items
- with one space between items
- using a preset tab zone system

If you do not use the ZONE statement to set up your own tab zone intervals all COMALs will be identical: a comma (,) gives no space between items and a semicolon (;) gives one space.

However, the tab character has changed from the comma to the semicolon, and the default zone changed to match. IBM and AmigaCOMAL use the new standard. CP/M and Commodore COMALs use the original method.

BENCHMARK TIMINGS

Here are some benchmark run times from page 21 converted into bar charts.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>A-C</th>
<th>A-B</th>
<th>I-C</th>
<th>I-B</th>
<th>C-C</th>
<th>C-P</th>
<th>C-B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alhs</td>
<td>3.1</td>
<td>12.8</td>
<td>10.2</td>
<td>14.4</td>
<td>25.5</td>
<td>111.0</td>
<td>115.7</td>
</tr>
<tr>
<td>Sieve</td>
<td>6.0</td>
<td>16.0</td>
<td>8.6</td>
<td>39.8</td>
<td>25.0</td>
<td>67.0</td>
<td>157.4</td>
</tr>
<tr>
<td>Substring Replacement</td>
<td>0.2</td>
<td>5.1</td>
<td>0.3</td>
<td>9.7</td>
<td>1.0</td>
<td>2.1</td>
<td>12.3</td>
</tr>
</tbody>
</table>

A-C=AmigaCOMAL; A-B=AmigaBASIC; I-C=IBM COMAL 2.2; I-B=IBM BASIC; C-C=C64 2.0; C-P=PowerDriver; C-B=C64 BASIC
Base model Amiga 500 and IBM PC were used for the tests.
COMAL LOOPS

REPEAT
 «statement block»
UNTIL «condition»

WHILE «condition»
 «statement block»
ENDWHILE

LOOP
 «statement block»
EXIT [WHEN «condition»] «statement block»
ENDLOOP

FOR «var»:=«start» TO «end» DO «statement block» ENDFOR «var»
(STEP «amount» is an option)

COMAL DECISIONS

CASE «selector» OF
 WHEN «choice list» «statement block»
[WHEN «choice list» «statement block»]
[OTHERWISE «statement block»]
ENDCASE

IF «condition» THEN «statement block»
(ELIF «condition” THEN «statement block»)
[ELSE «statement block”]
ENDIF

COMAL ERROR HANDLER

TRAP «statement block»
(REPORT)
HANDLER «statement block»
(REPORT, ERR, ERRTEXT$)
(extension: RETRY, ERRFILE)
ENDTRAP

PROC «name»(«parameters») CLOSED
{IMPORT «var list»}
«block»
{RETURN}
ENDPROC «name»

FUNC «name» («parameters») CLOSED
{IMPORT «var list»}
«block»
{RETURN «value»}
ENDFUNC «name»

PROC «name»(«parameters») CLOSED
IMPORT «var list»
«block»
{RETURN}
ENDPROC «name»

PROC slow’print(text$)
FOR x:=1 TO LEN(text$) DO
PRINT text$(x:x),
ENDFOR x
ENDPROC slow’print

PROC just’lastS(text$) CLOSED
IF text$="" THEN RETURN ""
max=LEN(text$)
RETURN text$(max:max)
ENDPROC just’lastS

COMAL TODAY #25, 5501 Groveland Terrace, Madison, WI 53716 - 27
<table>
<thead>
<tr>
<th>keyword</th>
<th>cl i c a p 6 2 b p m o</th>
<th>keyword</th>
<th>cl i c a p 6 2 b p m o</th>
<th>keyword</th>
<th>cl i c a p 6 2 b p m o</th>
</tr>
</thead>
<tbody>
<tr>
<td>//</td>
<td>comment</td>
<td>eod</td>
<td></td>
<td>page</td>
<td></td>
</tr>
<tr>
<td>abs</td>
<td></td>
<td>eof</td>
<td></td>
<td>pass</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td></td>
<td>err</td>
<td></td>
<td>pi</td>
<td></td>
</tr>
<tr>
<td>append</td>
<td></td>
<td>errtext$</td>
<td></td>
<td>print</td>
<td></td>
</tr>
<tr>
<td>at</td>
<td></td>
<td>esc</td>
<td></td>
<td>proc</td>
<td></td>
</tr>
<tr>
<td>atn</td>
<td></td>
<td>exec</td>
<td></td>
<td>random</td>
<td></td>
</tr>
<tr>
<td>auto</td>
<td></td>
<td>exit</td>
<td></td>
<td>randomize</td>
<td></td>
</tr>
<tr>
<td>bitand</td>
<td></td>
<td>exp</td>
<td></td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>bitor</td>
<td></td>
<td>external</td>
<td></td>
<td>ref</td>
<td></td>
</tr>
<tr>
<td>bitxor</td>
<td></td>
<td>false</td>
<td></td>
<td>rename</td>
<td></td>
</tr>
<tr>
<td>bye</td>
<td>? ?</td>
<td>file</td>
<td></td>
<td>renum</td>
<td></td>
</tr>
<tr>
<td>case</td>
<td></td>
<td>find</td>
<td></td>
<td>repeat</td>
<td></td>
</tr>
<tr>
<td>cat / dir</td>
<td></td>
<td>for</td>
<td></td>
<td>report</td>
<td></td>
</tr>
<tr>
<td>chain</td>
<td></td>
<td>free</td>
<td></td>
<td>restore</td>
<td></td>
</tr>
<tr>
<td>change</td>
<td></td>
<td>func</td>
<td></td>
<td>return</td>
<td></td>
</tr>
<tr>
<td>chr$</td>
<td></td>
<td>get$</td>
<td></td>
<td>rnd</td>
<td></td>
</tr>
<tr>
<td>close</td>
<td></td>
<td>goto</td>
<td></td>
<td>run</td>
<td></td>
</tr>
<tr>
<td>closed</td>
<td></td>
<td>handler</td>
<td></td>
<td>save</td>
<td></td>
</tr>
<tr>
<td>con</td>
<td></td>
<td>if</td>
<td></td>
<td>scan</td>
<td></td>
</tr>
<tr>
<td>cos</td>
<td></td>
<td>import</td>
<td></td>
<td>select</td>
<td></td>
</tr>
<tr>
<td>create</td>
<td></td>
<td>in</td>
<td></td>
<td>output</td>
<td></td>
</tr>
<tr>
<td>curcol</td>
<td></td>
<td>inkey$</td>
<td></td>
<td>sgn</td>
<td></td>
</tr>
<tr>
<td>currow</td>
<td></td>
<td>input</td>
<td></td>
<td>sin</td>
<td></td>
</tr>
<tr>
<td>cursor</td>
<td></td>
<td>int</td>
<td></td>
<td>size</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td>key$</td>
<td></td>
<td>spc$</td>
<td></td>
</tr>
<tr>
<td>del</td>
<td></td>
<td>label</td>
<td></td>
<td>sgn</td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td></td>
<td>len</td>
<td></td>
<td>sin</td>
<td></td>
</tr>
<tr>
<td>dir / cat</td>
<td></td>
<td>let (assign)</td>
<td></td>
<td>size</td>
<td></td>
</tr>
<tr>
<td>discard</td>
<td></td>
<td>link</td>
<td></td>
<td>spc$</td>
<td></td>
</tr>
<tr>
<td>display</td>
<td>? ?</td>
<td>list</td>
<td></td>
<td>sgn</td>
<td></td>
</tr>
<tr>
<td>div</td>
<td></td>
<td>load</td>
<td></td>
<td>sin</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
<td>log</td>
<td></td>
<td>size</td>
<td></td>
</tr>
<tr>
<td>elif</td>
<td></td>
<td>loop</td>
<td></td>
<td>spc$</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
<td>main</td>
<td></td>
<td>sgn</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
<td>merge</td>
<td></td>
<td>spc$</td>
<td></td>
</tr>
<tr>
<td>endcase</td>
<td></td>
<td>mod</td>
<td></td>
<td>sgn</td>
<td></td>
</tr>
<tr>
<td>endfor / next</td>
<td></td>
<td>new</td>
<td></td>
<td>sin</td>
<td></td>
</tr>
<tr>
<td>endfunc</td>
<td></td>
<td>next / endfor</td>
<td></td>
<td>size</td>
<td></td>
</tr>
<tr>
<td>endif</td>
<td></td>
<td>not</td>
<td></td>
<td>spc$</td>
<td></td>
</tr>
<tr>
<td>endloop</td>
<td></td>
<td>null</td>
<td></td>
<td>sgn</td>
<td></td>
</tr>
<tr>
<td>endproc</td>
<td></td>
<td>of</td>
<td></td>
<td>sin</td>
<td></td>
</tr>
<tr>
<td>endtrap</td>
<td></td>
<td>open</td>
<td></td>
<td>size</td>
<td></td>
</tr>
<tr>
<td>endwhile</td>
<td></td>
<td>or</td>
<td></td>
<td>spc$</td>
<td></td>
</tr>
<tr>
<td>enter</td>
<td></td>
<td>ord</td>
<td></td>
<td>sgn</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>otherwise</td>
<td></td>
<td>sin</td>
<td></td>
</tr>
</tbody>
</table>

**c64 = C64 COMAL 2.0 cart**
**128 = C128 COMAL 2.0 cart**
**ibm = IBM PC COMAL 2.2**

**cpm = CP/M COMAL 2.1**
**ami = AmigaCOMAL**
**pow = Power Driver for C64**

---

- = fully implemented
* = implemented in a package
+ = partially implemented
? = syntax/other difference
- = not implemented
COMMON COMAL -- Keyword Syntax and Examples

// : allows comments in a program
// anything typed here

ABS : gives the absolute value
ABS(«numeric expression»)
PRINT ABS(standard number)

AND : logical AND
«expression» AND «expression»
IF number>0 AND number<100 THEN

APPEND : start at end of file for writing
OPEN [FILE] «file#» , «filename» , APPEND
OPEN FILE 2 , "test.dat" , APPEND

AT : begin at specified location
PRINT AT «row», «col»: 【print list» 【mark】
INPUT AT «row» , «col»: 【len. promp. var list» 【marks】
PRINT AT 1.1: "Section number."; num;
PRINT AT 0.8: USING 
INPUT AT 0.9; "?": reply$; /0=stay on same row
INPUT AT 5.0: "": address$; /0=stay in same col

ATN : arc tangent
ATN(«numeric expression»)
PRINT ATN(num1+num2)

AUTO : automatic line numbering
AUTO [start line»] 【increment】
AUTO 9000
AUTO
AUTO 531,1

BITAND : bitwise AND
«argument» BITAND «argument»
show(bnum BITAND %00001000)

BITOR : bitwise OR
«argument» BITOR «argument»
PRINT (bnum BITOR flag)

BITXOR : bitwise XOR
«argument» BITXOR «argument»
bnum=(num1+num2) BITXOR %10000000

BYE : exit COMAL (C64, C128 use BASIC)
BYE

CASE : multiple choice decisions
CASE «control expression» [OF]
CASE reply$ OF
CASE choice OF

CAT : gives disk directory, see also DIR
CAT [«filename»]
CAT

CHAIN : load & run program on disk
CHAIN «filename»
CHAIN "menu"

CHANGE : change text
CHANGE "«old text»","«new text»"
CHANGE "zz","print'report"

CHR$: gives the character specified
CHR$(«numeric expression»)
PRINT CHR$(num)

CLOSE : closes files
CLOSE [[FILE] «filename»]
CLOSE FILE 2
CLOSE

CLOSED : all proc/func variables local
PROC «procname»[(params)] [CLOSED]
FUNC «funcname»[(params)] [CLOSED]
PROC newpage(header$) CLOSED
FUNC gcd(n1,n2) CLOSED

CON : continue program execution after STOP
CON

COS : cosine
COS(«numeric expression»)
PRINT COS(number)

CREATE : create RANDOM file on disk
CREATE «filename»,«# records»,«record size»
CREATE "names",128.300

CURCOL : returns the cursor column position
CURCOL
column:=CURCOL

CURROW : returns the cursor row position
CURROW
row:=CURROW

CURSOR : positions the cursor
CURSOR «row», «column»
CURSOR 0,1 // 0=stay in same row

DATA : provides data for a READ
DATA «value», «value»
DATA "Sam",134,"Fred",22,"Gloria",45

DEL : deletes lines
DEL «range»
DEL 460-530
DEL pause

DELETE : deletes a file from disk
DELETE «filename»
DELETE "test5.dat"
DIM : reserve string/numeric array space
DIM «string var» OF «max char»
DIM «str array»(«index») OF «max char»
DIM «array name»(«index»)
DIM name$ OF 30
DIM players$(1:4) OF 10
DIM scores(min:max), table(10,2)

DIR : display directory of disk, see CAT
    DIR [«filename»]
    DIR

DISCARD : discards packages
    DISCARD

DISPLAY : display program without line numbers
    DISPLAY [«range»] [TO] [«filename»]
    DISPLAY "names.dsp"
    DISPLAY init

DIV : division with integer answer
    «dividend» DIV «divisor»
    result=guess DIV count

ELIF : starts an "else if" condition
    ELIF «expression» [THEN]
    ELIF reply$ IN "YyNn" THEN

ELSE : alternative in IF structure
    ELSE

END : halt program (& show message)
    END [«message»]
    END "All Done."

ENDCASE : end of CASE structure
    ENDCASE

ENDFOR : end of FOR structure
    ENDFOR [«control variable»]
    ENDFOR sides

ENDFUNC : end of function
    ENDFUNC [«function name»]
    ENDFUNC even

ENDIF : end of IF structure
    ENDFI

ENDLOOP : end of LOOP structure
    ENDDO

ENDPROC : end of procedure
    ENDPROC [«procedure name»]
    ENDPROC show'item

ENDTRAP : end of TRAP structure
    ENDTRAP

ENDWHILE : end of WHILE structure
    ENDFWHILE

ENTER : retrieve ASCII program lines
    ENTER [«filename»]
    ENTER "test5.lst"

EOF : End Of File flag
    EOF
    WHILE NOT EOF DO

ERR : returns current error number
    ERR
    CASE err OF

ERRTEXT$ : returns error message
    ERRTEXT$
    PRINT ERRTEXT$

ESC : escape key pressed flag
    ESC
    EXIT WHEN ESC

EXEC : execute a procedure
    EXEC[«procname»][«parameter»,«parameter»])
    show'item(number)

EXIT : exit the LOOP structure
    EXIT [WHEN «condition»]
    EXIT WHEN errors>3

EXP : natural log e to n
    EXP[«numeric expression»]
    PRINT EXP(number)

EXTERNAL : external proc/func
    PROC «name»[«parms»][EXTERNAL «file»]
    FUNC «name»[«parms»][EXTERNAL «file»]
    FUNC rec'size(name$) EXTERNAL "rec.ext"
    PROC instructions EXTERNAL "instruct.ext"

FALSE : predefined value equal to 0
    FALSE
    ok:=FALSE

FIND : finds text in a program
    FIND "<text string>"
    FIND " PROC "

FOR : start of FOR loop structure
    FOR «var»:=«start» TO «end» [STEP «#»] DO [«statement»]
    FOR x:=10 TO 1 STEP -1 DO PRINT x

FREE : returns amount of free program memory
    FREE
    PRINT FREE
FUNC : start of a multi-line function
FUNC «name»[«parm»)] [CLOSED]
FUNC «name»[«parm»] EXTERNAL «filename»
FUNC call'answered(msg$)
FUNC file'exists(name$) CLOSED

GET$ : returns # of characters from open file
GET$[«filenum»][«# of characters»)
text$=GET$(2,16)

GOTO : go to line after specified label
GOTO «label name»
GOTO jail

HANDLER : lines executed if error occurs
HANDLER

IF : start of conditional IF structure
IF «condition» THEN [«statement»
IF reply$ IN "yYnN" THEN

IMPORT : import into CLOSED proc/func
IMPORT «identifier» [«identifier»]
IMPORT running'total

IN : locate string1 within string2
«string1» IN «string2»
IF guess$ IN word$ THEN winner

INKEY$: : wait and return the key pressed
INKEY$
repSYM$=INKEY$

INPUT : input from keyboard or file
INPUT FILE «file#»[«rec#»]: «var list»
INPUT AT «row»,«col»,«len»: «var list»
INPUT FILE 2: text$
INPUT AT 5,2,10: "Zip Code: ": zip'code,
INPUT "Enter age---": age

INT : nearest integer (less than or equal)
INT[«numeric expression»
tally:+INT(number)

KEY$: : scans keyboard & returns key typed
KEY$ UNTIL KEY$="c"

LABEL : assign label name to the line
[LABEL] «label name»:

LEN : gives the length of a string
LEN[«string expression»

LINK : loads a package from disk
LINK «filename»
LINK "mouse"

LIST : list program lines
LIST [«range»] [TO] [«filename»]
LIST header
LIST pause "pause.lst"
LIST "myprog.lst"

LOAD : load a program from disk
LOAD «filename»
LOAD "menu"

LOG : natural logarithm of n
LOG[«numeric expression»)
PRINT LOG(number);

LOOP : start of LOOP structure
LOOP

MAIN : leave external to main program
MAIN

MERGE : merge program lines from disk file
MERGE [«line#»[«increment»] «filename»
MERGE "readrec.Ist"
MERGE 831.1 "setflags.Ist"

MOD : remainder of division (modula)
«dividend» MOD «divisor»
color=number mod 16

NEW : clears program from memory
NEW

NEXT : converted to ENDFOR, see ENDFOR

NOT : logical NOT
NOT «condition»
IF NOT ok THEN

NULL : does nothing
NULL

OPEN : open a file
OPEN [FILE] «file#»,«filename»,«type»
OPEN FILE 2,"scores.dat",READ
OPEN FILE 7,"subs.ran",RANDOM 88

OR : logical OR
«condition» OR «condition»
IF reply$<"a" OR reply$>"z" THEN

ORD : ASCII (ordinal) value of character
ORD[«string expression»
a=ORD("a")

OTHERWISE : default for CASE
OTHERWISE

PAGE : clears screen / formfeed
PAGE

« » means user supplied, [ ] means optional, { } means several | means choice -- don't type the «» [ ] { } or |
PASS: pass a command to operating system
PASS «operating system command»
PASS "dir"

PI: value of pi
PI
PRINT "Value of PI is"; PI

PRINT: print items to screen/printer/file
PRINT [AT «row», «col»]: [USING «format»]: «list»
PRINT [FILE «#», «rec»]: [USING «format»]: «list»
PRINT FILE 2: text$
PRINT AT 9,1: USING "$###.##": money
PRINT "The winner is"; winner$

PROC: start of multi-line procedure
PROC «name»[( «parm»)]: [CLOSED]
PROC «name» [(«parm»)]: [EXTERNAL «file»]
PROC readrec(number)
PROC convert(REF text$) CLOSED

RANDOM: random access disk file
OPEN FILE «file#»; «filename»; RANDOM «len»
OPEN FILE 2; "subs"; RANDOM 88

RANDOMIZE: seeds random number generator
RANDOMIZE [«seed»]
RANDOMIZE

READ: read data from DATA line or file
READ name$; age
READ FILE 2; record: name$; adr$; city$; st$
OPEN FILE 3; in$; READ

REF: parm var used in reference (alias)
REF «var»
PROC alter(REF text$) CLOSED

RENAME: rename a disk file
RENAME «old filename»; «new filename»
RENAME "temp"; "final"

RENUM: renumber program
RENUM [«target start»]; [«increment»]
RENUM 9000,1

REPEAT: start of REPEAT structure
REPEAT

REPORT: part of ERROR HANDLER
REPORT [«error code»]
REPORT 99

RESTORE: reuse DATA with READ
RESTORE [«labels»]
RESTORE month$; names

RETURN: returns value of a function
RETURN [«value»]
RETURN TRUE
RETURN (text$)

RND: random number
RND [«start num», «end num»]
dice=RND(1,6)+RND(1,6)

RUN: run program in memory or on disk
RUN [«filename»]
RUN "DMC"

SAVE: store program to disk
SAVE «filename»
SAVE "thewhale"

SCAN: scan for correct structures
SCAN

SELECT: choose output location
SELECT [OUTPUT] «type»
SELECT OUTPUT "lp:" // line printer
SELECT OUTPUT "ds:" // data screen

SGN: -1 if neg, 0 if 0, 1 if pos
SGN(«numeric expression»)
flag=SGN(number)

SIN: gives sine
SIN(«numeric expression»)
plot(SIN(num), y)

SIZE: report on free memory
SIZE

SPC$: returns # of spaces specified
SPC$(«number of spaces»)
PRINT SPC$(39)

SQR: gives square root
SQR(«numeric expression»)
root=SQR(number)

STEP: increment FOR loop by this amount
FOR «var»:=«start» TO «end» STEP «increment» [DO]
FOR x=1 TO max STEP 2 DO

STOP: halt program execution
STOP [«message»]
STOP "now inside PROC remove'blank"

STR$: converts number into string
STR$(«number»)
zip$=STR$(number)

TAB: move cursor to specified column
TAB(«column number»)
PRINT TAB(col), name$

TAN: gives tangent
TAN(«numeric expression»)
PRINT TAN(number)

THEN: part of IF structure. See IF and ELIF
C64 COMAL 2.0 and 1581 Drive

Dear Len: I am finally enjoying my COMAL disks (I have about 20) since I recently purchased my first disk drive (a 1581). I passed on my chance to get a drive last year in order to purchase my COMAL 2.0 cartridge. I'll never regret that. I love COMAL 2.0! It is everything I ever wanted in a computer language. COMAL 2.0 with the 1581 disk drive is an almost unbeatable combination on my C64. They work beautifully together and the 1581 has the high density storage I need for certain database applications while COMAL 2.0 gives me the power to access the same. You can put a lot of programs on one 1581 disk! I placed 242 COMAL 0.14 files including COMAL 0.14 itself on one disk! - Randy Ragle, DeQueen, AR

Amiga, Benchmarks, 64 Emulator

Dear Len: Having tried BASIC again on an Amiga, I missed the many advantages of COMAL. I could hardly wait to demonstrate COMAL to my local Amiga club. Converting from C64 COMAL programs to AmigaCOMAL was easy. After LISTing the programs to disk, I used the Transfer Software included with ReadySoft's The 64 Emulator 2. This program provides the necessary conversion from PETASCII to ASCII during the transfer, and works with a 1541 or 1571 drive. - Bert Denaci, Massapequa Pk, NY

We got two other letters/disks from Bert. He ran some benchmark programs on both his Amiga 500 and Amiga 2500 (remember, he did an article on benchmark testing back in COMAL Today #12). He has this to say: "It is noted that, except for the Trig test, AmigaCOMAL on the Amiga 500 is equal to or faster than AmigaBASIC on the Amiga 2500 with its Accelerator board." Bert also submitted his 3D airplane program (issue #13) in AmigaCOMAL. It will be on our first AmigaCOMAL programs disk.

In conclusion, Bert says, "I am pleased to have AmigaCOMAL. It is like getting an old friend back, and now I can convert COMAL 2.0 and 0.14 programs to the Amiga."

Less Technical

Dear Mr. Lindsay: I have enjoyed reading your magazine and will continue to be a subscriber. I would like you to know that I am disappointed that the last couple issues were very technical. What I enjoyed most about your magazine was how you demonstrated the COMAL language. - R Daniel Gamble, Metairie, LA

[Hopefully this issue will do the trick. We try to please both those who want tons of tech talk, and the general users]
by Svend Daugaard Pedersen (AmigaCOMAL developer)

I read with interest Joel Rea's article in COMAL Today #24 about a new Multilevel COMAL standard. As the programmer of AmigaCOMAL, I certainly have some comments as well as my own idea of a new COMAL standard (COMAL90?).

It is necessary to include some sort of a structured data type.

First some comments. Joel has realized (along with many others) that it is necessary to include some sort of a structured data type. The idea of reusing the keyword DIM is natural, although the problem with introducing new keywords will almost be eliminated if my proposal is followed (more on that later). The virtual memory file is a nice idea. Also the AT clause in the DIM statements is useful but it cannot replace a pointer variable. The very useful data types lists and trees cannot be constructed using the AT clause.

I agree that pointer variables are awful. They pull a programming language down to a lower level (in fact quite close the machine language level). But I have not seen any replacements for them, yet.

The WITH statement would be very handy but unfortunately it will be very difficult to implement in a line oriented programming language like COMAL. I think it only would be possible at the cost of speed or the cost of the great flexibility in which COMAL handles different types.

A variable without a type identifier (and the dollar sign is the only one) is a floating point variable unless it has been declared to be of another type in a special statement. In fact, this is not a new idea. In the existing COMALs a "floating point variable" may be used as a name for a function, a procedure, an array and even a label. But it must be declared before it is used in this way.

The declaration of integer variables should be done in a DIM statement:

- DIM alfa, beta OF LONG
- DIM i, j, k OF USHORT, a, b OF BYTE

It is more difficult with the structured data type. First we have to declare the type:

- RECORD person'type
  - FIELD nameS OF 20
  - FIELD addressS OF 30
  - FIELD age OF UBYTE
- ENDBRIDGE person'type

Then the variables may be declared:

- DIM person OF person'type
- DIM person'array(100) OF person'type

Note that I have not used the word DIM in the definition of the type. This is because it is only a definition of the inner structure of the type. No data field is created.

A function and a procedure declaration should look like:

The problem is that an AmigaCOMAL program using all the new things is very unpleasant to look at (and Joel's new COMAL will have the same problem). All the different special characters ($, #, %, !, & and @ etc.) are not very nice to look at (Borge once said that it looks as if the program has got measles).

My idea is that we go back to the original COMAL80 standard from 1979. In this COMAL there are only two data types: strings and numbers (floating point). A string variable is identified by a dollar sign. But what about the integers, the structured data type and whatever else you can think of. Should we drop that? Of course not!

In fact, this is not a new idea.

A variable without a type identifier (and the dollar sign is the only one) is a floating point variable unless it has been declared to be of another type in a special statement. In fact, this is not a new idea. In the existing COMALs a "floating point variable" may be used as a name for a function, a procedure, an array and even a label. But it must be declared before it is used in this way.

The declaration of integer variables should be done in a DIM statement:

- DIM alfa, beta OF LONG
- DIM i, j, k OF USHORT, a, b OF BYTE

As a COMAL, the new standard will be rather close to Amiga COMAL, as well as most other COMALs.

Now to my ideas of a new standard. You may think this must be close to my AmigaCOMAL. Of course it will. As a COMAL, the new standard will be rather close to AmigaCOMAL, as well as most other COMALs. But I think some quite new ideas have to be introduced.

Having finished the implementation of the AmigaCOMAL in which I have included structured data types, pointers and three different integer types (byte, short and long), I realized that I had went too far, although all the extensions were made in the "spirit of COMAL" (I discussed every extension with Borge Christensen).
FUNC euler(m,n:USHORT):USHORT CLOSED

and

PROC read'field(REF p:person\text{type}, line)

The word OF could be used instead of ':' but I think the lines would grow too long. Note the parameter \text{line}. This is a floating point parameter since the default type for this name is float.

And now to my next idea (please get yourself a drink if you have weak nerves!). I think we should make COMAL case sensitive!

I think we should make COMAL case sensitive.

Why? Well, Joel has touched on the problem. New keywords might be in conflict with variable names in existing programs. But this is not the only case where you have the problem. I am sure you have tried to find a name for an output procedure. All the names you can think of are reserved: print, list, output, out, write, display. In fact it is easier to be a Dane. All reserved words are English so the Danish words are all free (but I can assure you that this is the only advantage we Danes have in working with computers. Can you imagine the problems we have with national letters having the same ASCII codes as ]/[(/)?)

Keywords should all be written with upper case letters. Names may be written with upper case letters as well as lower case and two names such as Fred and fred are different. In this way all the names for the output procedure listed above may be used. Another advantage is, that the naming style known from Modula-2 and C (at least C on the Amiga) can be used: \text{PersonType} instead of \text{person\text{type}. I think it is more readable.}

My intention was that all the routines should have the same names and use exactly the same parameters as the ones used in C.

I had a problem when I wrote the standard packages used to call system routines in the Amiga from AmigaCOMAL. My intention was that all the routines should have the same names and use exactly the same parameters as the ones used in C. Using the same parameters caused no problem but some of the names were reserved words in AmigaCOMAL (for instance Write, Close and Read). My solution was to add the prefix pck to all names so that you have to write pckopenscreen instead of \text{OpenScreen} and pckioerr instead of \text{IoErr}. With a case sensitive COMAL the more readable names could have been used.

I look forward to see the reaction to my proposals.

References:

Common COMAL - Problem Areas, Len Lindsay, \textit{COMAL Today} #24, page 5

Common COMAL - Definition and Test Functions, \textit{COMAL Today} #24, page 21

Proposed COMAL Multilevel Standard, Joel Rea, \textit{COMAL Today} #24, page 47

Guest Editorial, Robert Ross, \textit{COMAL Today} #18, page 4

COMAL Kernal, \textit{COMAL Today} #17, page 40

How To Use The Kernal, Richard Bain, \textit{COMAL Today} #17, page 55

COMAL Standards Meeting, Brian Grainger, \textit{COMAL Today} #15, page 62

Scope Rules, Richard Bain, \textit{COMAL Today} #14, page 60

COMAL Standards Conference / Graphics Kernal, \textit{COMAL Today} #7, page 8

UPDATE To CP/M COMAL KEYS

James Synnamon sends us this update:

\begin{verbatim}
KEY  Result
CTRL-Q Opens up a blank line above the line that the cursor is on. The cursor moves to the start of the blank line.
CTRL-V Pushes all characters to the right of the cursors position one space to the right, opening up a space under the cursor.
ESC Moves the cursor to the start of the next line (not the same as hitting \texttt{«Return}}.
\end{verbatim}

[AmigaCOMAL does this too]
AmigaCOMAL has all the COMAL Kernal and Common COMAL keywords and structures. In addition it has significant extensions! It is very fast (see the benchmark programs results in this issue). It is totally “Amigatized”.

Plus all that, it seems to be a next generation of COMAL by expanding the language. It seems just as powerful as the new IBM PC COMAL 3.0, yet it is not nearly as complex! Power to the people!

AmigaCOMAL Features

- PRINT AT and INPUT AT let you position the starting location on the screen.
- PRINT USING gives you formatting control of numeric output.
- AmigaCOMAL support section on People Link and CompuServe national networks. Includes Download Libraries, Q&A Message base, Official notices and monthly conferences.
- Turtle Graphics are included (Logo compatible).
- Record structures! They can be nested. A new record structure can use an existing model.
- System structures! All the structures necessary to interface directly with AmigaDOS, Exec, Intuition, etc. are predefined for you.
- Pointers are available. You can construct lists and trees using these pointers!
- ALLOCATE and DEALLOCATE for data areas.
- Compiler is available. Make any COMAL program a stand-alone program that can be started directly from AmigaDOS (by CLI or WorkBench Icon). The icon is created automatically for you. You may distribute your compiled programs as you wish (no royalty to pay to AmigaCOMAL authors).
- Pass parameters to a compiled COMAL program or to AmigaCOMAL itself (tool types from its icon in WorkBench, or the tail end of the command line in CLI).
- List all the packages currently in "use"... or list all the new commands added by any package via the LISTPACK command.
- TRACE and SINGLE STEP mode is available for program debugging.
- TIMES and DATES are included as well as TIMER.
- Lines are Re-Listed immediately as you enter a program! Omitted words are inserted, keywords capitalized, = converted to :=, spacing corrected, indented to proper level, etc.
- Execute and Command Windows can be separate or combined into one Window.
- Scroll a program listing back up the screen if you just miss a line.

A Compiler is available!

Here is just a sampling of AmigaCOMAL features:

- Fully Common COMAL compatible!
- Multi-line IF and CASE decision structures.
- Multi-line FOR, REPEAT, WHILE and LOOP loop structures, including LOOP x TIMES.
- Super flexible multi-line Error Handler structure, complete with RETRY option.
- User defined procedures and functions, with parameter passing, recursion and LOCAL variables.
- IMPORT, EXPORT and GLOBAL statement for complete flexibility with CLOSED procedures and functions.
- Test out a procedure from direct mode!
- Procedures and functions can be external to your program. A running program can execute them directly from disk.
- Store procedures and functions on disk. Merge them into future programs with the MERGE command.
- Instant syntax error help! If a program line is not correct, an error message window pops up with helpful information about what is wrong, and the cursor is placed at the point of error.
- Automatic backup of files. Original file is not overwritten if you store a new file with the same name. The original file is renamed with the extension .backup tacked onto its name.
- Protected INPUT fields. Your programs will have the professional look.
AmigaCOMAL Features (continued)

- Write packages in COMAL! Finally I can write packages myself. Packages also may be written in C, Modula2, or Assembler.

- SIGNAL is used by the system to allow any package instant access to import events occurring, such as USE, DISCARD, NEW, RUN, END, Error Abend, etc.

- Pass the "name" of a procedure or function as a parameter.

- Assign values to a whole array with one statement. Example:

  ```
  READ table(,) //reads values from DATA statements
  ```

- DIM can be used with normal variables.

- Numbers can be represented in binary or hex as well as decimal.

- SELECT INPUT and SELECT OUTPUT are joined by SELECT INOUT.

- READWRITE is an added mode when opening files.

- MININDEX and MAXINDEX return the top and bottom index to an array.

- Includes Function Keys, Drop Down Menus, Mouse, and Window/Screen control.

- Has an Insert mode.

- Insert a blank line anywhere on the screen.

- Remove a line from anywhere on the screen.

- Has Amiga Key short cuts to commands (Run, Step, Pause, Continue, Stop,...)

- Install program gives you flexibility. Interlace screen or not, hi-res or not, default font, etc.

- Amount of memory to allocate to COMAL work space is limited only by memory available.

- Memory Window can be constantly open, always showing the free memory available to COMAL.

- Directory and Subdirectory control with CHDIR or CD, MADEDIR or MKDIR, DIR$, DIR, CAT, UNIT and UNIT$.

- Sequential and Random Access files in either ASCII or binary format.

- PASS command to open a new CLI window or to pass a command to AmigaDOS.

- WAIT command to put COMAL program in a waiting state (for a set time or until a key or mouse button is pressed). This does not tie up the CPU and is needed for proper multi-tasking.

- Three types of integer variables for compatibility with C routines.

- ROUND is built in (plus the standard INT).

- Repeat a string "function". Example:

  ```
  PRINT 5"a"
  aaaa.
  ```

- Screen and color controls.

- Messages package gives full access to this Amiga capability.

- Event handling / exceptions allow you to monitor exceptions. This allows a type of interrupt routine.

- Direct interface to Amiga libraries, such as Exec_library, Dos_library, Intuition_library, Graphics_library, Layers_library, Diskfont_library, Icon_library, Potgo_library.

- Devices package with its own predefined record structures.

- Windows and Screens packages allow you open and control your own windows or screens.

- Gfx, RastPort and View packages with predefined structures for your use.

- Three different graphics packages to choose from. Turtle has Logo compatible graphics. PCGraphics is compatible with Commodore 64 and IBM PC COMAL graphics. Graphics is the standard package.

- Speech package that makes it easy to have your Amiga talk back.

All this and more as they say.

COMALs time may finally have come. AmigaCOMAL has arrived (under $100) and IBM PC COMAL 2.2 has dropped in price from $800 to only $165 (including the compiler). Disk loaded Power Driver for the Commodore 64 is now only $9.95 and includes a 22 lesson interactive on disk tutorial.
by Captain COMAL

I developed this exercise to demonstrate procedures in AmigaCOMAL. However, it will apply to other COMAL 2.0 systems (even to Power Driver for much of it), except for the more advanced parts. Only AmigaCOMAL can do the complete exercise.

The first thing we will do is write a procedure. We will make mistakes on purpose to see the error messages. Before you start writing a new program, make sure you clear out the program area of any previous program. Type:

Next we want to start up the automatic line numbering mode. COMAL will provide the line numbers for us:

Now just type in the procedure as follows. I will intersperse comments and suggestions. Only type the one that assigns rvsonS ..• the one to match your computer! Do not type the leading //.

PROC bar(nameS,max,num) CLOSED
<
Start a procedure

width=80

II
width output screen, C64 users set to 40
rvsonS=chrS(16)irvsoffS=chrS(16)llreverse field Amiga

rvson$=CHR$(18)irvsoffS=CHR$(19)

II
reverse on IBM

rvson$=CHR$(29)irvsoffS=CHR$(29)

II
reverse on CP/M

PRINT name$irvsonS,

II i
gives 1 space, gives no space

Note: we are still in AUTO mode, and COMAL has prompted you with the next line number (0060). However, we forgot to put in the STEP for the FOR loop, so we need to go up and add it! All COMALs will let you cursor up and then over to one space after the word num ... then add this:

FOR x:=0 TO num STEP max/(width*.8) ENTER

All COMALs have an insert mode, so for the above addition, you could turn on insert mode and type the addition in front of the word DO, however, it is not necessary, since COMAL will add the word DO for you anyway!

Amiga users also can use the mouse to position the cursor. To add the above you could have pointed the mouse to the D in the DO, and clicked the left mouse button to put the cursor there.

After adding the STEP to the previous line, we are still in AUTO mode, so we can continue with our procedure (remember, with reverse field mode, a space is printed as a box):

PRINT ":", // print one space, note comma at end
ENDWHILE // this is wrong, but type it anyway
PRINT rvsoff$, // turn printing back to normal
PRINT USING "###.":nun
ENDPROC //don't type in the proc name, COMAL adds it later

Now stop AUTO mode. The way to do this varies with the computer:

STOP AUTO MODE

Amiga: «Esc»
IBM: «Ctrl» + «Break»
C64 2.0: «Run/Stop»
Power Driver: «Return» on blank line
CP/M «Esc»

You should save your work now. Save your programs often. If anything should ever go wrong, you will be able to get back most of your work that way:

SAVE "temp"

Note: type the next line in several steps to illustrate the syntax error messages you get immediately. The specific error messages will vary between COMALs, but all will give you immediate error messages and help. I will list the messages given by AmigaCOMAL. The final correct line is shown after these intermediate lines. Remember, we are still in AUTO mode this whole time:

FOR «enter»
(variable expected)
FOR x «enter» <=== use lots of extra spaces
{: expected)
FOR x «enter»
[illegal type] <=== need a number, got a blank
FOR x=0 «enter»
[TO expected)]
FOR x=to «enter» <=== no space after 0
[illegal type]
FOR x=to «enter» <=== need a number again
FOR x=to «enter» <=== use lots of extra spaces
All COMALs except C64 will now relist the line like this:
FOR x=0 TO num DO
Note that most COMALs (not C64) will tack on a filename extension for you. CP/M and Amiga COMALs uses .sav while IBM uses .cml.

COMAL is very helpful. It checks each line as you type it for errors. We demonstrated how it caught errors in lines entered. COMAL also can check your whole program to verify if all the structures are correct. Remember, we typed ENDWHILE to end the FOR loop. Watch COMAL catch this:

```
COMAL caught the error and advises you of it:
ENDFOR expected (0070)
```

The line number (0070) will vary depending on how you typed in the procedure. The message will vary between COMALs, but will be similar. It is a helpful message. It is telling us that it expected to find an ENDFOR statement in line 70, but did not! Let's fix that line. But let's type NEXT instead:

```
LIST 70
0070 ENDFOR
```

Now, cursor up and change the line to this:

```
0070 NEXT
```

As soon as you hit «enter» Amiga, IBM and CP/M COMAL will immediately re-list the line properly (including the indentation):

```
0070 ENDFOR
```

C64 note: it also changes NEXT into ENDFOR, but does not re-list the changed line.

After correcting that line, have COMAL scan it for errors again:

```
SCAN
```

The cursor returned right under SCAN, with no error messages. That means all is OK.

Now, save your work again:

```
SAVE "bar"
```

Now, take advantage of one of COMAL's features! Test out the procedure from direct mode. Type:

```
bar("test",25.5)
```

You should have a bar graph similar to this:

```
test ______ 5.0
```

It works! Oh, before we forget, list your procedure to see how COMAL added the procedure name after ENDPROC for you ... and added the FOR loop variable after ENDFOR too:

```
PROC bar(nameS,max,nlJII) CLOSED
width:=80 // width output screen, C64 users set to 40
rvsonS:=chrS(16);rvsoffS:=chrS(16)//reverse field Amiga
PRINT nameS;rvsonS, // gives 1 space , gives no space
FOR x:=0 TO num STEP max/(width*0.8) Q.Q
PRINT "", // print one space, note comma at end
ENDFOR x
PRINT rvsoffS, // turn printing back to normal
PRINT USING "##I#.fI":run
ENDPROC bar
```

Notice all the things COMAL does for you. It capitalized all the keywords and put your variable names in lower case (regardless of how you typed them). It indents lines properly. It removes extra spaces typed in a line (except those with quotes). It changed the = into := (computer science people like having the colon in front of the = when it is used for assignment). It added the word DO at the end of the FOR statement. It changed NEXT into ENDFOR and added the variable name. It added the procedure name after ENDPROC. All in all, it makes you look good (as a programmer that is).

You have already saved the procedure to disk. That is how you store programs on disk, so you can LOAD them back any time.

Now, let's prepare for the future and store the procedure in ASCII form, so it can be merged into future programs. Type:

```
LIST "bar"
```

COMAL (except C64) will tack on the .lst filename extension for you.
Procedures – Multiple Views (continued)

Power Driver note: you should renumber procedures to have line numbers over 9000 before you list them to disk, as Power Driver does not renumber lines as it merges them later. Do this:

RENUM 9000,1
LIST "bar"

While we are at it, let’s store the procedure in a form that can be used as an external procedure:

SAVE "bar.ext"

Power Driver note: external is not implemented in Power Driver.

AmigaCOMAL has one more option for us ... we can turn this procedure into a package! Just add this one line:

0001 EXPORT bare,,)

The commas inside the () indicate how many parameters are being passed. Use the same number of commas as you used when typing the PROC header statement.

Now, just save it as a package:

SAVE "bar.pck"

You just created your very own package! Applause!

Now you can erase the procedure and start the demo program:

NEW

Use AUTO mode to type in the program:

AUTO
page // clears the screen
DIM test$(1:4) OF 20 // set u p a titles string array
// 1:4 means elements 1 through 4, 20 means 20 chars each
RESTORE TITLES // restore next data pointer
READ test$(t) // READ values for the WHOLE array at once

NOTE: The above line to read in values for the whole array at one time is a short cut that works with IBM and AmigaCOMAL. C64 COMAL has to read one element at a time like this:

FOR z:=1 TO 4 DO READ test$(z)

Power Driver note: RESTORE does not allow you to specify the point to restore to, it always restores to the first data item. You must make sure the DATA is in the correct order, and in this program it is. Just omit the DATA 0,"junk" line below. Omit the RESTORE lines as well.

Still in AUTO mode, continue typing:

RESTORE AMIGA // restore next data pointer again
FOR x:=1 TO 4 DO // notice var x again, but no conflict
PRINT "^H >>>>":test$(x)
READ comal,basic // read two numbers
max=comal // find the largest of the two numbers
IF basic>max THEN max=basic // is it larger?
bar("COMAL",max,comal) // draw first bar
bar("BASIC",max,basic) // draw second bar
PRINT // blank line
ENDFOR // no need to type the X, COMAL adds it for you
PRINT "Press a key to continue...

Here we must emphasize one aspect of the Amiga. It is multi-tasking. It can do more than one thing at a time. AmigaCOMAL provides a proper way to wait for a keypress so that other tasks are not slowed down: WAIT. Wait will wait for a keypress or right mouse button. You also may include a number after it, specifying the number of seconds to wait. Example: WAIT 5.

So, Amiga users, still in AUTO mode, type this line next:

WAIT

Other COMAL users type this line instead:

WHILE KEYS="" DO NULL

Note: Power Driver sees no keypress as a CHR$(0) rather than a null string. It would use the following line:

WHILE KEYS=CHR$(0) DO NULL

To allow for Power Driver, you could use the following line with every COMAL:

WHILE KEYS<CHR$(1) DO NULL

AmigaCOMAL provides a proper way to wait for a keypress so that other tasks are not slowed down:

WAIT

Now, still in AUTO mode, finish the program:

DATA 0,"junk" // this will be skipped over by RESTORE
TITLES:
DATA "this Benchmark"
DATA "Calc Benchmark"
DATA "Sieve Benchmark"
DATA "Substring Benchmark"
AMIGA:
DATA 3.1,12.8,7.2,16.9,6.6,16.0,0.2,5.1 // Amiga

Stop the AUTO mode here (see note above on how to do that). The last line contains the time in seconds for benchmark tests run on an Amiga 500 comparing BASIC and COMAL. The data will be properly read and used on IBM and C64, however, it will be Amiga comparisons! To have the
comparisons be applicable to the computer being used, replace the last line with one of these:

DATA 10.2, 14.4, 32.8, 55.6, 8.6, 39.8, 0.3, 9.7 //IBM 2.2
DATA 25.5, 115.7, 72.9, 106.4, 28.0, 137.4, 1.0, 12.3 //C64 2

Of course, Amiga users can type in the data for IBM or C64 too, if they want to see the comparison for those computers.

Now, save your work before we continue:

SAVE "temp2"

That is the whole program ... except for the bar procedure! However, we can tack on that procedure easily now, in one of three different ways!

We could just merge the procedure into this program so the program includes everything:

MERGE "bar"

Power Driver uses the ENTER command instead of MERGE:
ENTER "bar"

You now have everything in one place. Have COMAL check it all out for you:

SCAN

Now, save the demo program:

SAVE "bardemo"

Try the program:

RUN

If you forgot what the bar procedure looked like, just list it. You can tell COMAL to list any procedure you want. Just specify the procedure name like this:

LIST bar

Power Driver does not allow listing procedures by name.

So far you have seen some of the flexibility of procedures. You have tested them from direct mode. You have merged them into a future program. You have listed them individually. But there is more to show you yet!

Sorry, but we must leave Power Driver users behind now.

COMAL 2.0 allows you to execute procedures directly from disk! The procedure itself does not have to be part of the running program. All you need is a special procedure header with an EXTERNAL specification at its end.

Let's demonstrate this right now. First list the bar procedure:

LIST bar

Now, delete the procedure:

DEL bar

Now put the cursor on the C of the word CLOSED at the end of the PROC header line that is still displayed on your screen. Amiga users can just point the mouse there and click the left mouse button. Others (and Amiga users too if they want) can use the cursor up and cursor right keys to get there.

Now type over the word CLOSED. Type this:

PROC bar(name$, max, num) EXTERNAL "bar.ext"

IBM and Amiga users do not have to specify the .ext filename extension, as COMAL will tack it on if you omit it.

List the program to verify that the procedure is indeed gone, and only the header remains:

LIST

Now, you can see that the program will still run, using the procedure it needs right from disk!

RUN

Next we will show how an EXTERNAL procedure can have a different name. The name of the procedure as saved to disk can be different than the name given in the procedure header that utilizes it! To do this, we will write a bar procedure with a slightly different look:

NEW AUTO
PROC alternative(n$, max, num) CLOSED
DIM bar$ OF 27 // will work on all screens
bar$=3*"1234567890" // string multiplication/see note
PRINT n$;bar$(1:(num/max)*27);
PRINT USING "###;#": num
ENDPROC // COMAL will add the name for us later

COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716 - Page 41
O84 COMAL does not have the string multiplication extension. Use these two lines instead to assign bar its value:
```
bar$="**
FOR x=1 TO 3 DO bar$="1234567890"
```
Have COMAL check out the procedure:
```
SCAN
```
Now make sure it works. Test it from direct mode:
```
alternative("test",23,9)
```
Now, we want to save it on disk with the same name as the original bar external procedure. Since you don't want to lose your original procedure, rename that file first:
```
RENAME "bar.ext","bar1.ext"
```
Now you can save our new procedure on disk as an external procedure:
```
SAVE "bar.ext"
```
Now, run the bardemo program we previously saved to disk (unchanged mind you):
```
RUN "bardemo"
```
```
>>> Aha! Benchmark
COMAL 123456 3.1
BASIC 123456789012345678901234567 12.8
```
```
>>> Calc Benchmark
COMAL 12345678901 7.2
BASIC 123456789012345678901234567 16.9
```
```
>>> Steve Benchmark
COMAL 12345678901 6.6
BASIC 123456789012345678901234567 16.0
```
```
>>> Substring Benchmark
COMAL 1 0.2
BASIC 123456789012345678901234567 5.1
```
```
Now we leave all the COMALs behind except AmigaCOMAL. (IBM COMAL 3.0 also can do this, but with a more complex process, so it will be covered in a future article.) We will show how easy it is to access the bar procedure in the bar package we created earlier. First, delete the bar procedure header line:
```
DEL bar
```
Then just add this line at the beginning of the program:
```
1 USE bar
```
The program will run just like before, except now you are using a procedure from a package!

References:
Listerine, Will Bow / R. Hughes, *COMAL Today* #24, page 64
Reveal PROCs, Dick Klingens, *COMAL Today* #18, page 60
Easy Instructions, Captain COMAL, *COMAL Today* #16, page 24
Introduction to Procedures, David Stidolph, *COMAL Today* #15, page 36
Procedures and Functions - Our Collection, *COMAL Today* #15, page 40
Merger, David Stidolph, *COMAL Today* #15, page 50
Program Construction, Captain COMAL, *COMAL Today* #15, page 51
Keywords, Procedures, Packages, Richard Bain, *COMAL Today* #14, page 32
Listerine, Will Bow, *COMAL Today* #14, page 22
COMAL 2.0 External Procedures, Captain COMAL, *COMAL Today* #7, page 27
Automated PROC & FUNC Librarian, Kevin Quiggle, *COMAL Today* #6, page 75
Pass an Array as a Parameter, Captain COMAL, *COMAL Today* #4, page 48
Function as a Parameter / Local and Global, *COMAL Today* #2, page 32

Now you see how external procedures can be updated and all programs that use them will always use the newest versions ... as long as you keep the same filenames and number and type of parameters. Packages offer this same advantage, but they were out of reach to the normal COMAL programmer ... until now.

Page 42 - COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716
People/Link's AmigaZone Club enters its fourth year with more goodies than ever to offer the Amiga enthusiast. This page lists just some of the features that make the AmigaZone your BEST online resource for Amiga news, information, help and entertainment.

**NEW COMAL CLUB!!**

Our notice areas are a hotbed of informative and interesting conversations. Need help? Post a notice and chances are your problem will be solved by the next time you log in. Talk to industry experts; software and hardware vendors and technical reps from dozens of Amiga-supporting companies like Gold Disk, Lattice, WordPerfect, Micro Systems Software, Microlusions, New Horizons, ASDG, and dozens more.

For the Amiga, Plink is by far the largest and most active network...offering something for everyone. (Its) public domain library (is) one of the best around with thousands of files ready for downloading. Of the four networks discussed...Plink has the lowest signup fee and some of the lowest hourly rates. Plink offers a lot for your money.

Lou Wallace, Amiga World, May 1989

**The Conferences**

The AmigaZone's live conferences are bigger and better than ever. Special weekly themed get-togethers discuss MIDI and Amiga Music, Graphics and Video, Games, Developers' Issues, and Hardware. Chat with your friends and with industry leaders. You never know who might show up and join in the fun. You can tune into our live AFlexx class with Bill Hawes and learn from the master. Of course on Sunday nights there's our acclaimed "AmigaMania" conference where you'll often find up to live dozen Amigoids at once chatting away with each other. And we've added a couple new twists:

**Prizes!**

We're giving away all kinds of free prizes like disk magazines, software, and other goodies. Just show up at our Sunday night Conferences and you could go home a winner! We've got decks of cards and a special terminal program called CardComm which displays them on your Amiga's screen. How about a few hands of poker or blackjack with your friends?

**The Library**

Over 3000 files (and growing every day). Over 120 Megabytes of freely-distributable software for your Amiga. The Zone's library is unsurpassed anywhere. Super-fast downloading with WXmodem, available in many popular terminal programs. All files are Sysop-tested for your peace of mind. If you want to share your own creations, uploading is always FREE during non-prime usage times.

This page was produced on an Amiga with Professional Page, eclips, Interchange and sweat. Plink logo by Allen Hastings. Text and Layout by Harv Laser. (Plink: CBM*HARV)
Power Driver Variable Storage
by David Warman

While writing the Power Driver BUFFER routines, I uncovered some information on how variables are stored in memory. It is sketchy, but may provide a starting point for anyone wishing to investigate further.

<table>
<thead>
<tr>
<th>Pointer Memory Area Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$38-39: COMAL program</td>
</tr>
<tr>
<td>$3C-3D: name table/$3E-3F=end of table:</td>
</tr>
<tr>
<td>$3A-3B: variable location table</td>
</tr>
<tr>
<td>:free memory</td>
</tr>
<tr>
<td>$40-41: variables</td>
</tr>
<tr>
<td>$42-43: DIM'ed variables</td>
</tr>
<tr>
<td>$44-45: top of memory</td>
</tr>
</tbody>
</table>

COMAL program is simply the lines that make up the program, without the name table or variables.

Name table is a list of the names of all the variables, PROCs, and FUNCs used in the program, including all typing errors and any old names that may have been changed. Once a name has been defined, it stays in the name table and is SAVEd along with the program. The only way to get rid of an unused name is to LIST the program to disk, then ENTER it back. This process builds up a new name table, consisting of only the current names. The format of the name table is a length byte followed by the ASCII representation of the name. The end of the name table is marked by a zero byte. Example, if a program has two variables, min and max, the name table would look like this:

3 77 73 78 3 77 65 88 0
Len in Len max end
Len = Length of name (3 in each case)

The variable location table is arranged in groups of five bytes that provide information on each name. These 5-byte groups are in the same order as the names in the name table. The first byte determines what the name is: integer variable, string array, simple real, PROC, etc. The numbers that represent each type are listed in COMAL Today #9, page 68, in the PROC printtype. This is for 2.0, but the values appear to be the same in Power Driver. The second and third bytes are the lo-byte and hi-byte, respectively, of the exact memory location of the variable. I haven't yet determined the purpose of the fourth and fifth bytes.

After the variable location table is an area of free memory. Its size is returned by FREE and SIZE.

Variables are stored at the top of memory, first the simple nondimensioned variables, then just before end of memory, the dimensioned strings & arrays.

Reference:
Change Drive - Power Driver, COMAL Today #24, page 67
Power Driver Turtle Parameters, R. Hughes, COMAL Today #24, page 62
Power Driver Memory Location, R. Hughes, COMAL Today #24, page 58
Show COMAL Name Table, Mike Lawrence, COMAL Today #9, page 68
COMAL 0.14 Memory Map, COMAL Today #6, page 28

Power Turtle Functions
by Leo R. Reed

While reading my issue of COMAL Today #24, I noticed an article Power Driver Turtle Parameters by R. Hughes on page 62. His FUNCtions x'pos and y'pos may suffice while the turtle is in the visible area of the viewing screen. They are not sufficient for keeping track of the turtle when it is off screen. Also, note that programs that are compiled need different PEEKs than those running under the Power Driver interpreter!

Power Driver Interpreter FUNCtions:

```lisp
FUNC xcor CLOSED
   hi#=PEEK(28742) // for Power Driver
   RETURN ((hi# * 256 + PEEK(18127))-hi# DIV 128) * 65536
ENDFUNC xcor

//
FUNC ycor CLOSED
   hi#=PEEK(28746) // for Power Driver
   RETURN (hi# * 256 + PEEK(18127)-hi# DIV 128) * 65536
   IF hi#>128 OR (hi#=128 AND lo#>199) THEN
      RETURN 256 * (256-hi#) + 199 - lo#
   ELIF hi#<=128 THEN
      RETURN (256 * (256 - hi#) - lo# + 199) - 65536
   ENDIF
ENDFUNC ycor
```

Power Driver Compiler FUNCtions:

```lisp
FUNC xcor CLOSED
   hi#=PEEK(18126) // for Power Driver COMPILER
   RETURN ((hi# * 256 + PEEK(18127))-hi# DIV 128) * 65536
ENDFUNC xcor

//
FUNC ycor CLOSED
   hi# = PEEK(18130) // for Power Driver COMPILER
   RETURN (hi# * 256 + PEEK(18131) * 65536
   IF hi#>128 OR (hi#=128 AND lo#>199) THEN
      RETURN 256 * (256-hi#) + 199 - lo#
   ELIF hi#<128 THEN
      RETURN (256 * (256 - hi#) - lo# + 199) - 65536
   ENDIF
ENDFUNC ycor
```

Page 44 - COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716
COMAL: The Next Generation

Two new COMAL implementations represent the next generation of COMAL: IBM PC COMAL 3.0 from UniComal and AmigaCOMAL from ComWare. Though we have not received a formal announcement, it seems that the AmigaCOMAL developers have joined together with UniComal (who also were the developers of the C64 and C128 COMAL cartridges). With all those brilliant programmers working together, the future of COMAL seems secure!

UniComal's goals are to continuously develop and maintain UniComal program development tools so that users are assured access to the newest programming techniques, rapid execution speed and a high level of user friendliness. Their new IBM PC COMAL 3.0 is an expression of these goals. Two years of development time has been invested in the 3.0 implementation. The result was a program development tool with many intelligent facilities that can only be obtained otherwise by combining the features of several other programming languages.

Likewise, AmigaCOMAL has been under development since the Amiga was first released. It was created with guidance from Borge Christensen, the father of COMAL. While it includes many advanced features, it maintains what has become a trademark of COMAL. It is totally user friendly!

Both AmigaCOMAL and IBM PC COMAL 3.0 have many similar advancements. It is easy to see that they have been cooperating with each other, even while working as two separate companies!

Record Structures! On the Amiga these are implemented through a RECORD structure. IBM uses a STRUC structure. Both allow you to refer to all the variables in the data structure as one thing ... or to refer to individual variables in the structure separately.

Packages may be written in COMAL, C or Assembler! Packages have always been easy to use by all, even the beginner! Now it is possible for anyone to write their own package too. Packages (called modules in IBM 3.0) may be written in COMAL itself. A user can find out what new commands are added by any package with two easy steps:

1. First USE the package (module):

   USE "packagename"

2. Next, ask for the list of new commands:

   LISTPACK "packagename" // AmigaCOMAL & IBM 2.2
   INTERFACE "packagename" // IBM 3.0

Pointers and pointer variables have been introduced in these next generation COMALs. It is now possible to define dynamic data structures to take advantage of the efficient memory usage which this entails (such as lists and trees).

Both Amiga and IBM 3.0 allow the programmer to utilize as much memory as needed (within the limit of available memory in the computer). You also may declare arrays which are larger than 64K.

Both also can handle external events. Example: execute a procedure each time a device requests service. It is also possible to designate the priority of events. This facility is very important to the efficiency and updating of a program in connection with control tasks, data logging, etc.

IBM PC COMAL 3.0 has also expanded the PRINT USING statement so that text can easily be printed together with numbers in various formats. The INPUT statement also was extended with additional termination options. Full network support is also available as an option.

Both Amiga and IBM implementations were produced in Denmark. Thus it is easy to see why they went out of their way to support the national differences, in particular the different character sets (NLS).

The new Profile Concept gives IBM PC programmers the ability to easily configure version 3.0 according to their special requirements. AmigaCOMAL utilizes a Preferences file for the same capability, and includes an INSTALL program that makes creating the preferences file easy.

In addition to the facilities and features mentioned above, a number of other improvements have been made. For example: several program structures have been improved (LOOP «number» TIMES), the full screen editor has been made even easier to use (with full mouse controls on the Amiga), improved string handling (string multiplication), better program structure checking, improved commands, and much more.

Other programming languages have been improving in the past years. However, COMAL remains a jump ahead of them with these next generation implementations.

Special IBM note: Both versions 2.2 and 3.0 of IBM PC COMAL are current and supported. Version 2.2 has dropped in price to just $165 including its compiler. Special school licenses are available. It is still ideal for the non-professional programmer! See the benchmark times on page 21.
Power Driver BUFFER

by David Warman

I like writing programs in Power Driver since they can be used by people who do not have the 2.0 cartridge, and can even be compiled and distributed to people without any form of COMAL, possibly bringing in some new COMAL users in the process. However, one drawback of Power Driver, even though it's a great improvement over COMAL 0.14, is the 15K memory limit. I often feel restricted by this limitation, especially after being used to the cartridge. It occurred to me that I might be able to store data in the 8K area under the Kernel ROM at $E000, so I set about learning how to do this. The result is the machine language program called "ml.buffer/pwr".

I'll refer to this memory area as a BUFFER, since it is similar to COMAL 2.0's "pkg.buffer", which utilizes the memory under the cartridge. This routine does not increase program memory; you still have 15K of program space. It is a special area of memory, like BASIC's $C000 area; it can be thought of as a relative file in memory. You can use it to store text or data. Another feature of the BUFFER is that any data in it survives from one program to another. Therefore, data can be placed in the BUFFER by one program, and retrieved by a program that is CHAINed from the first one.

The program "buffer/pwr.procs" contains all the necessary PROCs to install the ML and transfer data between the BUFFER and the program. The first three procedures set up the machine language. If you need more memory for your program, you can put these in a loader program that sets up the buffer, then CHAINs your main program.

init'buffer

This PROC calls the other two PROCs.

load'obj(«filename$»)

Example: load'obj("ml.buffer/pwr")

Loads the ML program "ml.buffer/pwr". The ML resides at $E000, and handles the transfer of data between the COMAL program and the BUFFER.

poke'irq

POKEs a short ML routine into the top part of the cassette buffer. It is relocatable, but if the location is changed, the SYS address in the read/write PROCs must also be changed. The routine in the cassette buffer prepares to call the BUFFER handling code at $E000 and is executed each time a call is made to one of the BUFFER read/write PROCs.

Buffer Data Transfer via BTEXT$:

Transfer of data between a COMAL program and the BUFFER is via a string variable which must be named btext$. The reason is explained below.

Before write'buf or read'buf are called, the string btext$ should be DIM'ed to a length sufficient to hold the maximum length of data you want to transfer to or from the BUFFER at one time.

write'buf(«loc»)

Example: write'buf(15)

Copies the entire contents of btext$ into the BUFFER, starting at the location passed as the parameter (15 in the example). The location can range from 0 to about 7600.

read'buf(«loc», «length»)

Example: read'buf(0,100)

Copies the contents of the BUFFER into btext$, starting at location «loc» and proceeding for «length» characters. The example reads from BUFFER locations 0-99 and stores those characters into the variable btext$. If «length» is greater than the DIMensioned length of btext$, btext$ will be filled and the extra characters will be ignored (truncated).

fill'buf(«loc», «length», «fill'char»)

Example: fill'buf(50,10,32)

This PROC will fill the BUFFER from «loc» to «loc»+«length»-1 with the ASCII value specified in «fill'char». The example puts spaces in BUFFER locations 50-59, since CHR$(32) is a space.

file'to'buffer(«file$», «bufpos», «reclen»)

Example: file'to'buffer("test.dat",10,40)

This PROC will fill the BUFFER from «loc» to «loc»+«length»-1 with the ASCII value specified in «fill'char». The example puts spaces in BUFFER locations 50-59, since CHR$(32) is a space.

Optionally, to save memory, you can combine all the BUFFER read/write/fill PROCs into one PROCedure, as in PROC buffer in "buffer/pwr.procs". For the type parameter, use 0 to read from the BUFFER, 1 to write to it, and 255 to fill it.

file'to'buffer(«file$», «bufpos», «reclen»)

Example: file'to'buffer("test.dat",10,40)

Loads string data from a SEQ file into the BUFFER. «bufpos» is the BUFFER location at which you want the file to start loading, and «reclen» is the length of each string.
buffer'to'file("file$","bufpos"», "recen"», "recs")

Example: buffer'to'file("temp.dat", 10, 40, 100)

Saves the contents of the BUFFER into a SEQ file. "recs" is the total number of records to be written to disk; the other parameters are the same as for file'to'buffer.

As mentioned above, all data is passed to and from the BUFFER through a string called btext$. In writing this program, I had to do some research into how Power Driver stores variables in memory. The 0.14 memory map was of some help in finding the name tables, but it didn't describe just how variables are stored. I was unable to determine exactly how parameter variables operate, so I settled for using a string with a constant name that the machine language program could always find.

When a call is made to one of the BUFFER read/write PROCedures, the values of the parameters are first POKEd into locations in the cassette buffer. Then the COMAL PROC executes a SYS to address 992, also in the cassette buffer, which prepares to jump to the BUFFER-handling routine under the Kernal at $E000.

Normally, data cannot be read from under the Kernal; first the interrupts must be disabled, then the Kernal must be banked out so that the RAM underneath it can be read. The code in the cassette buffer takes care of all this, then jumps to $E000 to perform the actual read/write. After this is done, the program jumps back to the cassette buffer where the Kernal is banked back in, the interrupts are re-enabled, and control is returned to the COMAL program.

This version has some restrictions which I hope to solve in a future version.

- The variable btext$ cannot be passed as a parameter to any PROC which executes calls to the BUFFER. Parameter variables are handled differently than other variables, and I haven't yet been able to figure out how they work.

- Care must be taken not to write data past the end of the BUFFER. The program performs checks to make sure data is not written outside the BUFFER, but no error is communicated to the user; the action is simply aborted. The exact size of the BUFFER area is 8192 bytes, but the ML uses 577 bytes, leaving 7615 bytes (BUFFER locations 0-7614). In addition, there is a Kernal routine which writes data to a section of RAM high in the BUFFER area. From what I've been told, this only occurs on a warm or cold start, which writes data to $FD30-$FD4F. So, pressing «RUN/STOP» + «RESTORE» may destroy 32 bytes of data. If you want to avoid this area, don't use BUFFER locations 6895-6926.

- Since this routine uses the same area as the graphicscreen, it is incompatible with programs that use graphics.

- Programs that use the BUFFER cannot be compiled. This is because the name table pointers that the ML reads get moved when the program is compiled, and I haven't been able to find where they are moved to.

If and when I solve any of these problems, I'll release an updated version. I hope this program proves useful!

Note: This is version 1.1 of "ml.buffer/pwr". An earlier version had an abbreviated fill feature; it could only fill the entire BUFFER with zeroes. You can find the version number of these and future versions with this program:

```vbnet
DIM version$ OF 10
OPEN FILE 5,"ml.buffer/pwr"
version$=GETS$(5,10)
CLOSE 5
PRINT version$(7:9)
```

SETUP BUFFER

In order to use the BUFFER procedures described above, you must have the Machine Language (ML) in memory! This ML is contained in the file "ml.buffer/pwr" on Today Disk 25, and is loaded by the PROC load'obj in the program just discussed. If you wish to type in the BUFFER program, you also must type in the SETUP BUFFER program, which is a substitute for the ML file you do not have.

SETUP BUFFER will read the machine language from DATA statements and POKE it into memory. Once the ML is in memory it will stay there until the power is turned off or the graphic screen is activated, so this program is only needed to set up the ML. You can LOAD or CHAIN other programs after RUNning this one.

If you use this program to set up the ML, you should delete the PROC load'obj from "buffer/pwr.procs", as well as the program line in PROC init'buffer that calls it.

Reference:

Text Package, Dick Klingens, COMAL Today #11, page 63
Text Package, Dick Klingens, Package Library Volume 2, page 51
**Power Driver BUFFER**

**BUFFER/PWR.PROCS**

PROC init'buffer
   //This PROC must be called before BUFFER commands can be used. The file "ml.buffer/pwr" must be on the disk.
   //You must first run the program SETUP BUFFER.
   DIM btext$ OF 100 //this number is the max transfer size
   poke'irq
   //omit the next line if you are using SETUP BUFFER
   load'obj("ml.buffer/pwr")
ENDPROC init'buffer

PROC poke'irq
   //This routine is relocatable
   //it prepares to call the BUFFER handling code at $E000
   FOR address=992 TO 1009 DO
      READ byte
      POKE address, byte
   END FOR address
   DATA 120,120
   DATA 165,165
   DATA 41,253
   DATA 133,133
   DATA 32,0,224
   DATA 165,165
   DATA 9,2
   DATA 133,133
   DATA 88
   DATA 96
ENDPROC poke'irq

PROC load'obj(name$)
   //omit if using SETUP BUFFER
   POKE 157,0
   POKE 858,169
   POKE 859,0
   POKE 860,76
   POKE 861,213
   POKE 862,255
   OPEN FILE 78,name$+"p",READ
   SYS 858
ENDPROC load'obj

II-.---. Method 1 ----.---.-
II.(Separate PROC for each function)
PROC read'buf(loc,length)
   POKE 1010,0 //read
   POKE 1011,loc MOD 256
   POKE 1012,loc DIV 256
   POKE 1013,length MOD 256
   POKE 1014,length DIV 256
   SYS 992
ENDPROC read'buf

PROC write'buf(loc)
   POKE 1010,1 //write
   POKE 1011,loc MOD 256
   POKE 1012,loc DIV 256
   SYS 992
ENDPROC write'buf

II-.---. Method 2 ---.---.-
II.(One PROC for read, write & fill)
II.Use dummy values for un-needed parameters.
II.Example calls:
   //buffer(1,30,0) //read
   //buffer(127,0,write',0) //write
   //No length parm needed;
   //Entire string "btext$" is stored
   //Assumes write$=1
   //buffer(5,20,255,32) //fill
PROC buffer(loc,length,type,fill'char)
   POKE 1010,type //read,write,or clear
   POKE 1011,loc MOD 256
   POKE 1012,loc DIV 256
   POKE 1013,length MOD 256
   POKE 1014,length DIV 256
   SYS 992
ENDPROC buffer

II.Use the following PROCs to load a SEQ file into the BUFFER, and to save the BUFFER to a file.
II.
PROC file'to'buffer(filename$,bufpos,record'len)
   IF file'exists(filename$) THEN
      OPEN FILE 11,filename$,READ
      WHILE NOT EOF(11) DO
         read'buf(bufpos)
         bufpos:+record'len
      ENDWHILE
   ELSE
      PRINT STATUS$ ENDIF
   CLOSE FILE 11
ENDPROC file'to'buffer

PROC buffer'to'file(filename$,bufpos,record'len,records)
   PASS "s0:+filename$
   OPEN FILE 11,filename$,WRITE
   FOR i#:1 TO records DO
      write'buf(bufpos)
      bufpos:+record'len
   ENDFOR i#
   CLOSE FILE 11
ENDPROC buffer'to'file
Power Driver BUFFER Demo

This program demonstrates the use of the BUFFER commands. Do not type the PROC load'obj if you use SETUP BUFFER, and omit the line in init'buffer that calls it.

// by David Warman - Aug. 23, 1989
init
title
demo

// PROC init
DIM reply$ OF 1, btext$ OF 2000
DIM text$ OF 40
BACKGROUND 0
BORDER 0
TRAP ESC-
ENDPROC init

// PROC demo
PAGE
PRINT AT 2, 1: "Reply ""y"" if you haven't loaded"
PRINT "the ML since turning on the power."
INPUT AT1, 1: "Load ML?": reply$
IF reply$ = "y" THEN init'buffer
PAGE
PRINT Throughout this demo, use these
PRINT "commands:" PRINT
PRINT "w: write INPUT'ed text to buffer"
PRINT "r: read text from buffer and display"
PRINT "f: fill buffer"
PRINT "STOP: to end demo"
REPEAT
reply$ = INKEY$
CASE reply$ OF
WHEN "w"
PRINT "Enter text: (two lines max)"
INPUT ">": btext$
PRINT "Written ... "
ENDIF
ENDPROC demo

PROC init'buffer
poke'irq

// omit next line if you used SETUP BUFFER
load'obj("ml.buffer/pwr")
ENDPROC init'buffer

COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716 - Page 49
// PROC poke'irq
FOR address:=992 TO 1009 DO
READ byte
POKE address,byte
ENDFOR address
DATA 120 // sei
DATA 165,1 // ldi $01
DATA 41,253 // and #$fd
DATA 32,0,224 // jsr $e000
DATA 165,1 // ldi $01
DATA 9,2 // ora #$02
DATA 133,1 // sta $01
DATA 08 // cli
DATA 96 // rts
ENDPROC poke'irq
//
//omit the following procedure if you use
//the SETUP BUFFER program
PROC load'obj(name$)
POKE 157,0
POKE 858,169
POKE 859,0
POKE 860,76
POKE 861,213
POKE 862,255
OPEN FILE 78,name$+,p,READ
SYS 858
CLOSE FILE 78
ENDPROC load'obj
// PROC read'buf(loc,length)
POKE 1010,0 //read
POKE 1011,loc MOD 256
POKE 1012,loc DIV 256
POKE 1013,length MOD 256
POKE 1014,length DIV 256
SYS 992
ENDPROC read'buf
// PROC write'buf(loc)
POKE 1010,1 //write
POKE 1011,loc MOD 256
POKE 1012,loc DIV 256
SYS 992
ENDPROC write'buf
// PROC fill'buf(loc,length,fill'char)
//fill BUFFER with <fill'char>
POKE 1010,255 //fill
POKE 1011,loc MOD 256
POKE 1012,loc DIV 256
POKE 1013,length MOD 256
POKE 1014,length DIV 256
POKE 1015,fill'char
SYS 992
ENDPROC fill'buf
// PROC title
PAGE
PRINT "The entire string <btext$> is stored"
PRINT "in the BUFFER, starting at BUFFER"
PRINT "location 1."
wait
PAGE
PRINT "Read buffer into <btext$>.
PRINT " read'buf(75,100)"
PRINT " The contents of the BUFFER, starting"
PRINT " at position 75 and proceeding for 100"
PRINT " characters, are copied into <btext$>."
PRINT " If <btext$> is DIM'ed less than 100"
PRINT " bytes, it will be filled to its"
PRINT " maximum length."
PRINT "Fill BUFFER."
PRINT " fill'buf(76,25,32)"
PRINT " Fills BUFFER locations 76-100 with"
PRINT " spaces. The first parameter specifies"
PRINT " the first BUFFER location to fill, the"
PRINT " second parameter is the number of"
PRINT " locations to fill, and the last"
PRINT " parameter is the ASCII value of the"
PRINT " character you want to fill the BUFFER"
PRINT " with."
wait
ENDPROC title
// PROC wait
PRINT AT 25,6:"Press any key to continue...",
WHILE KEYS<>CHRS(0) DO NULL
WHILE KEY$=CHRS(0) DO NULL
ENDPROC wait
// PROC viewfile(filename$)
PAGE
PRINT "Looking at file:";filename$
PRINT
IF file'exists(filename$) THEN
   done:=FALSE
   TRAP ESC-
   OPEN FILE 2,filename$,READ
   WHILE NOT (EOF(2) OR ESC OR done) DO
      INPUT FILE 2: text$
      PRINT text$
      shift'wait
   ENDWHILE
   CLOSE
   INPUT "Press <return> to continue: ": reply$
   TRAP ESC+
ELSE
ENDIF
ENDPROC viewfile
// PROC shift'wait
shift'flag:=653; ok:=PEEK(shift'flag)
WHILE NOT ok DO
   ok:=PEEK(shift'flag)
   IF ESC THEN ok:=TRUE; done:=TRUE
   PRINT "press shift"+CHR$(145)
   PRINT ""+CHR$(145)
ENDWHILE
ENDPROC shift'wait
// FUNC file'exists(filename$) CLOSED
DIM ds$ OF 2
OPEN FILE 79,filename$,READ
ds$:=STATUS$
CLOSE FILE 79
RETURN ds$="00"
ENDIF
JOIN THE ON-LINE COMMODORE® USER GROUP.

Imagine being part of a nationwide on-line user group. With new QuantumLink, you can instantly exchange ideas, information and software with Commodore users everywhere, and participate in live discussions with Commodore experts.

And you can participate in conferences held by Len Lindsay, access COMAL public domain programs, and have your questions answered by other Comalites. You can even share your public domain COMAL programs with others.

These are just a few of the hundreds of features available. If you already have a modem, you can register on-line for a free software kit and trial subscription. Hook up and call 800-833-9400. If you need a modem, call QuantumLink Customer Service at 800-392-8200.
Textbuffer Package

by David Warman (QLink = DavidW57)

This package for C64 COMAL 2.0 provides up to approximately 22K of memory for use by COMAL programs. This extra memory doesn't mean that programs can be 52K bytes in length (22K plus the normal 30K), or that arrays can be larger. It is a special area of memory that is accessed by the PROCedures and FUNCTIONs in the package. If you are familiar with either the text or buffer package, then you know how textbuffer works.

The text package, first published in COMAL Today #11, allows you to use the 16K area of RAM under the COMAL cartridge to store text (or numeric data converted to string format). The drawback is that it only allows you to read or write data starting at the beginning of the file and proceeding straight through to the end, similar to working with a SEQ file on disk. This is fine for many applications, but sometimes it is necessary to access data in a non-linear manner.

The buffer package uses the same 16K area, but gives you instant access to any location within the buffer, similar to a random access file. As far as I know, the buffer package was only included on the Packages Library Volume 1 disk, so many COMAL users may be unaware of it. (In addition, it was LINKed to the program "singedrive'copy" on Cartridge Demo Disk #1 and I've used it in some of my programs on COMAL Today disks.)

My textbuffer package is similar to the buffer package, but offers some improvements. In addition to using the 16K of RAM under the cartridge, this package also gives you the option to use the 8K area of memory under the Kernal ROM and lets you save the data in the buffer along with a program. Below is a description of all the commands in the package. (Note: this is the package version of the Power Driver program "ml.buffer /pwr", which uses the RAM under the Kernal - and also is in this issue.)

All the non-string parameters in the following commands are integers, except those in the definebuf PROCedure which are real numbers.

readbuf("text$", "loc", "len")

This will read «len» bytes from the buffer, starting at buffer location «loc». Buffer locations are numbered starting with zero. The contents of the buffer will be placed in the string variable passed in the "text$" parameter. If the string is not DIM'ed high enough to hold the number of bytes specified, it will be filled to its maximum DIMensioned length.

writebuf("text$", "loc")

This command will write the contents of «text$» to the buffer, starting at location «loc». If an attempt is made to read or write past the end of the buffer, a TRAPpable error will be generated.

fillbuf("loc", "len", "num")

Example: fillbuf(0, 100, 32)

This command fills the buffer from location «loc» to «loc»+«len»-1 with the number «num». The example fills the first 100 bytes of the buffer with chr$(32), which is a space.

definebuf("strt", "end", "use'hi", "hi'strt", "hi'enci")

Example: definebuf(ml'end+1, 49151, true, Se000, $ffff)

This command sets up the actual memory locations that the buffer will use, and is generally executed only once, if at all. If you don't use this command the default values will be used.

I included this option in an effort to make textbuffer work with other packages. The text and buffer packages both use the entire area under the cartridge from $8009 to $fff, making it impossible to use them with most other packages, since nearly all packages use this same RAM area. (It may be possible to use text and buffer in conjunction with other packages that reside high in the undercartridge-RAM, as long as you don't fill the buffer too full!)

The textbuffer code itself begins at $8009, but I included the source code so that it can be relocated if necessary, if you have the right assembler. For example, if you are using another package which starts at $9000 and goes to $9500, you can assemble textbuffer at $9501. Then you must decide where to put the buffer itself; that's what the parameters control.

It needs to be explained here that the buffer is in two non-contiguous parts, one under the cartridge between $8009 and $fff, which I'll call the lower buffer, and the other part - the upper buffer - under the Kernal ROM from $e0000 to $fff. However, it functions as if the two parts were one consecutive memory block.

The lower part of the buffer can be placed anywhere between $8009 and $fff, as long as it doesn't overwrite the ML - a TRAPpable error will
be produced if you attempt this. In fact, the buffer area can even be before the ML. The start and end points are indicated by the memory addresses passed in the parameters «strt» and «end». The third parameter should be TRUE or FALSE, indicating whether or not you want to use the high part of the buffer. If you don't use it, the remaining parameters are irrelevant.

If you want to use the high part of the buffer, the parameters «hi'strt» and «hi'end» can be used to put it anywhere from $e000 to $ffff, inclusive. The only reason I can think of to not use this whole area is that there is a Kernel routine which (unintentionally?) writes to a few locations near the end of this area. I think the only time this happens is during a warm or cold reset, but if you want to avoid any trouble, set the top of the upper buffer below $fd30, since locations $fd30 to $fd4f may be altered. Also, if you use high RAM, you won't be able to use the graphic screen, since it occupies the same memory locations.

ml'start

Returns the start address of the textbuffer package. It is really only useful for determining where to place the buffer area, to avoid overlap.

ml'end

Returns the end address of the ML. The statement:

```
definebuf(ml'end+1,49151,TRUE,$e000,$ffff)
```

would place the lower buffer immediately after the ML. Note: these parameters are the default settings.

buf'start

Returns the start address of the lower buffer.

buf'end

Returns the end address of the lower buffer.

hibuf'start

Returns start address of the high part of the buffer (usually $e000, or 57344).

hibuf'end

Returns the end address of the high buffer.

keepbuf(«int»)

Example: keepbuf(TRUE)

The parameter should be TRUE or FALSE, telling COMAL whether or not to save the buffer contents along with the program and the package. When set to TRUE, you can save the text in the buffer along with a program, without having to load the buffer from a file each time. There are two restrictions. First, the buffer area must be after the ML in order to be saved, although it doesn't have to be immediately after it; everything from the beginning of the ML to the end of the buffer will be saved with the program. Second, only the lower buffer will be saved, not the part after $e000. In addition, the buffer won't be saved if the package is ROMmed (see below). The default for this command is FALSE.

rombuf(«int»)

Example: rombuf(FALSE)

If the parameter is TRUE, the textbuffer package will be ROMmed, meaning that it won't be saved with the program; it will stay in the computer's memory. In this way, you can LOAD one program with the textbuffer package LINKed to it, then, after executing rombuf(TRUE), you can CHAIN to other programs that don't have the package LINKed, thus saving disk space and reducing loading time. You only need a USE textbuffer statement in the CHAINed program to use the textbuffer commands. The default setting for this command is FALSE.

version'textbuffer$

Returns the version number of the package, along with additional information.

I also included on the disk a version of the package that is assembled at $a000, to provide more versatility for those who don't have the proper assembler.

TEXTBUFFER demo program

This simple program demonstrates how to use the textbuffer package by allowing the user to enter text into the buffer and then read it back. Note that the textbuffer package that is LINKed to this program is assembled at $a000, rather than $8009. This is so those without an assembler can de-LINK the package and have two versions, giving more flexibility in using it with other packages.

References:

Text Package, Dick Klingens, COMAL Today #11, page 63
De-Link a Package, Dick Klingens, COMAL Today #11, page 65
De-Link Package, Dick Klingens, COMAL Today #18, page 64
Buffer Package, Packages Library Volume 1
Programs which read and write disk data files are easily created using COMAL. Let's make it even easier by showing you a simple way to structure your data and at the same time make it compatible with other languages and all versions of COMAL. However, first we will review some data processing terms dealing with the storage of information in computer files.

A field is the smallest unit of data that has meaning in describing information. A group of related fields, treated as a unit, is called a record. Similar records are grouped into a file. One field (or more) may be designated as a key used for sequencing the file or for locating records. Records may be fixed or variable in length. Using fixed lengths is less flexible and more wasteful of disk space. Files may be either RANDOM (directly read) or sequential (read serially). We will be using sequential files here. COMAL supports two types of sequential files:

Binary files (recommended by the COMAL Handbook) are created by the WRITE FILE statement. Each record is preceded by a count of how many characters are in the record. The READ FILE statement is used to read these records. Binary files are only readable by the specific implementation of COMAL that wrote them.

ASCII files are created using the PRINT FILE statement and each record is followed by a delimiter (such as a comma). The INPUT FILE statement is used to read these records. ASCII files can be read by most computers.

Since we want the data files generated here to be useful to other languages and data processing programs we will use ASCII files.

We will develop a method of maintaining several data fields, each of variable length, that are combined into one string and written to disk. Note that other languages break up string INPUT using comma's as the delimiter between parts. COMAL will read in the entire string with one INPUT FILE statement. Your program can break the fields up into their separate parts.

Here are two examples of data that we could be enter into our sample data file:

- Jones, John, 123 1st St, Alpha City, WI, 56789, 555-1212
- Smith, Steve, 456 2nd Street, Beta Town, MI, 54321, 222-4432

Notice each field does not necessarily start in the same position within the record. The following two procedures will combine different fields into one string and separate them into a single string.

Note: FIELD is a keyword in some COMALs, so we use item$.

```comal
PROC separate(source$, REF item$(1)) CLOSED
// Separate fields divided by commas in source$ into the
// string array: item$(1)
index:=0
WHILE "," IN source$ DO
  comma:=" ," IN source$
  index:+1
  item$(index)]:=source$(1:comma-1)
  source$]:=source$(comma+1:LEI(source$))
ENDWHILE
ENDPROC separate

PROC combine(REF source$, REF item$(1), num'items) CLOSED
// Combine array elements into a single string with each
// field separated by commas
source$]:=item$(1)
FOR index:=2 TO num'items DO
  source$]:=source$+" ," +item$(index)
ENDFOR index
ENDPROC combine
```

The program listed at the end of this article uses a string array to keep the relevant information on an individual, and stores that information in an ASCII disk file. This information could be read in by another data base program (even one written in BASIC) or any other COMAL. Each element of the string array keeps part of the data on an individual:

- Array element 1: Name
- Array element 2: Address
- Array element 3: City
- Array element 4: State
- Array element 5: Zip Code
- Array element 6: Phone Number

In case of disaster, always keep several old generations of your data and backup the entire disk frequently. Comments or suggestions may be EMAiled to JeffreyS5 on QLink.

This information could be read in by another data base program, even one written in BASIC.

```comal
DIM fS(6) OF 40, all'data$ OF 240, choice$ OF 1
REPEAT
  menu
  CASE choice$ OF
    WHEN "A", "a"
      add entry
    WHEN "L", "l"
      print entries
    WHEN "Q", "q"
      quit
    OTHERWISE
      NULL
  END CASE
UNTIL TRUE=FALSE // forever, see quit option
PROC separate(source$, REF item$(1)) CLOSED
```

Page 54 - COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716
// Separate fields divided by commas in source$ into the
// string array: itemS()

// string array index:=0
WHILE "," IN source$ DO
  comma:="," IN source$
  index:=index+1
  itemS(index):=source$(1:comma-1)
  source$:=source$(comma+1:LEN(source$))
ENDWHILE
ENDPROC separate

PROC combine(REF source$, REF itemS(), num'items) CLOSED
  // Combine array elements into a single string with each
  // field separated by commas
  source$:=itemS(1)
  FOR index:=2 TO num'items DO
    source$:=source$+","+itemS(index)
  ENDFOR index
ENDPROC combine

// PROC combine(REF source$, REF itemS(), num'items) CLOSED
// Combine array elements into a single string with each
// field separated by commas
// source$:=itemS(1)
// FOR index:=2 TO num'items DO
//   source$:=source$+","+itemS(index)
// ENDFOR index
// ENDPROC combine

PROC add'entry
  INPUT "Name: ": f$(1)
  INPUT "Address": f$(2)
  INPUT "City": f$(3)
  INPUT "State": f$(4)
  INPUT "Zip code": f$(5)
  INPUT "Phone ": f$(6)
  combine(add'data$, f$(6))
ENDPROC add'entry

PROC menu
PAGE
PRINT AT 2,1: "Simple data base"
PRINT AT 6,1: "A) Add entry"
PRINT AT 8,1: "L) List entries"
PRINT AT 10,1: "Q) Quit"
REPEAT
  INPUT AT 14,1,1: "Choice": choice$
UNTIL choice$ IN "AaLlQq"
ENDPROC menu

PROC print'entries
OPEN FILE 2,"sample.dat",READ
WHILE NOT EOF(2) DO
  separate(all'data$, f$(6))
  PRINT "Name: ": f$(1)
  PRINT "Address": f$(2)
  PRINT "City": f$(3)
  PRINT "State": f$(4)
  PRINT "Zip code": f$(5)
  PRINT "Phone ": f$(6)
ENDWHILE
CLOSE FILE 2
ENDPROC print'entries

PROC quit
PAGEx
PRINT "Done."x
END
ENDPROC quit

Tech Talk
by Robert Ross

While I rarely use GOTO, I hope it isn't entirely expunged, unless other statements are added for graceful loop exits or control. How about allowing EXIT in WHILE, REPEAT and FOR loops? [AmigaCOMAL does allow this.] Or perhaps something like EXITFOR name? [I like that idea. It could be extended to EXITWHILE and EXITREPEAT too.] And something like CONTINUE (as in C) to go on at the start of the next iteration of a FOR, WHILE, REPEAT or LOOP loop. Makes GOTO look good, right?

And though not really a part of COMAL, how about a requested editor feature? This is from one who doesn't like to often type "#$%&^" (or ). At least in C64 2.0, a name table is constructed as lines are entered. The type of a variable name may change as a program is written, but how about keeping track in the name table of the last type specified while editing and using that as a default type for subsequent lines? 

Interesting that you bring up a typeless variable now. Check Svend Pedersen's article on page 34 for similar thoughts.

C64 2.0 Technical Disk

Robert Ross has done extensive and highly advanced work with COMAL 2.0. He has submitted a disk with programs we cannot thoroughly test, covering material too advanced for most readers. This includes:

- A patch for the META package (see COMAL Today #10, page 64).
- Character conversion functions package.
- Source to Sizzle fast loader.
- Interrupt driven timer for C64 2.0.

Anyone wanting a copy of these programs and associated text files on disk should send $2 to cover our costs and request the Robert Ross Technical Disk.
Prime Factoring Methods

by Bill Inhelder

In the days before microcomputers, the task of factoring a nine digit number into primes presented a substantial challenge. We used programmable calculators such as the Olivetti Programma P101. With a 24 by 18 inch footprint and a weight of about 40 pounds, they were immense by today’s standards. Programming used a symbolic code and output was hard copy on 4 inch paper rolls. Efficient algorithms using minimal memory had to be designed to carry out tasks. Even so, the task of factoring a large number into primes could take as long as an hour! The same algorithm translated into BASIC and run on a microcomputer reduces the elapsed time by a factor of 6 (even more with COMAL).

The purpose of this article is to present the original algorithm ("primes") together with various improvements in order to create COMAL programs of increasing efficiency.

The original algorithm begins by dividing a given natural number by the following divisors: 2, 3, 5, 7, 9, ..., that is, 2 and the set of odd numbers. If division produces a zero remainder, that divisor is a factor and is printed out. Each time a prime factor is found, the original number is replaced by the new quotient thus reducing the size of the number and speeding up the search for new prime factors. The algorithm must then test the new quotient for the same prime factor since repeated prime factors can occur. Once the test fails, the algorithm tests for divisibility by the next odd number. The process halts when the divisor exceeds the quotient. The dividend is then printed out as the final prime factor. Note that not all consecutive odd numbers will be primes. For example, 9 is not prime; however, since all prime factors of 3 have already been removed from the number, the odd number 9 will not appear as a factor. Of course this slows down the process of finding remaining prime factors since 9, 15, 21, 27, etc will be tested unnecessarily.

"Primes" adapts the algorithm to the structure of COMAL and uses repeated calls of the procedure pr_factors. My first attempt at implementing the algorithm involved recursive calls of the procedure. This worked fine for numbers which factored, but promptly blew up with an out of memory error for a large prime number. It was an excellent lesson of the dangers of recursive calls of a procedure.

"Primes2" is a literal translation of the original algorithm into COMAL using four GOTO's. This was done to determine which program was more efficient (see chart of run times at the end of this article).

In "primes2" the original algorithm is improved by adding a conditional statement which eliminates odd divisors which are divisible by 3. Thus 9, 15, 21, 27, etc are skipped.

There still remains odd factors which are divisible by 5, 7, 11, etc. The ideal would be to use only the set of prime factors. This set would have to be generated and read into the program from an external data file. The algorithm requires that the set of prime factors be numbers less than sqrt(999999999) or 31622. The integral log formula for approximating the number of primes in the interval 1 to 31622 gives an answer of about 3433 primes.

Writing a program to speed up the process of factoring a number into its primes is now dependent on finding a way to generate the 3400 primes which will serve as possible divisors. The method of the Sieve of Eratosthenes is reasonably efficient, except that it can be used only for primes less than about 15,000 (see "sieve"). It requires an integer array which takes up almost all the computer memory (on a C64) when dimensioned up to 15,000. Further since the sieve algorithm is such that the array cannot be partitioned, this means that the primes from 15,000 to 31,622 must be generated by other means.

Program "upper" does this by using primes from 2 through 179 as divisors for all odd numbers in the interval 15000-31622. Running time for the program is over 35 minutes (on a C64). The program "sieve" writes the first group of primes to the sequential file, "primes.dat", while program "upper" appends the second group of primes to that file (ultimately totals 3403 primes).

Finally, "primes3" reads in the sequential file of primes for use as potential divisors. This results in the most efficient program.

The largest prime less than 999,999,999 is 999,999,937. The worst case composite: 998,812,807.
Prime Factoring Methods (continued)

branches are less time consuming than conditional branches. Eliminating divisors which are multiples of 3 saved some time in "primes". My disk drive loads the prime factor data file in 1 minute (far less on IBM and Amiga). Once loaded, many numbers can be factored at surprising speed. Remember, the above table represents the worst case. Most composite numbers are factored into primes in a matter of seconds. A number is identified as prime in the program if the only factor printed out is the number itself.

Try your telephone and social security numbers to see if they are prime and, if not, what their prime factors are. Can you find the second largest prime?

Programs "sieve" and "upper" are a two program set that creates the file "primes.dat". You only need them if you wish to create a new file on another disk. (Or if you do not have Today Disk #25).

To get a list of the first 3403 primes, printout the sequential file, "primes.dat".

A more efficient sieve algorithm, which differs from the classical one, can be found in COMAL Today #13 on page 36. However, unlike the classical algorithm, it is not at all clear why it works! Generally I avoid using algorithms I can't explain. This algorithm generates twice the number of primes using the same amount of memory as the classical one and is twice as fast. Unfortunately it still does not generate enough primes to factor the largest natural numbers COMAL or BASIC are capable of handling.

References:
Prime Factorization, Steve Kortendick, COMAL Today #13, page 75
1000 Primes Revisited, COMAL Today #13, page 36.

Program Notes

These programs will run on all COMALS, if they have enough available memory. AmigaCOMAL can run them with its memory set at 64K. However, since AmigaCOMAL has three types of integers, the "sieve" program integer array should use sieve% rather than sieve#. The # type integer is too large and uses up more memory. You may use it, but then you then should set AmigaCOMAL's memory size to at least 96K.

As you are aware, COMAL can time how long a program takes to run. All COMALS except C64 use timer to count seconds. C64 uses time to count jiffies (1/60th of a second). Depending on your computer, use these lines to set and read the timer:

TIE O // C64
PRINT USING "Time: ###.## secs."; TIME/60

TIE 0 // IBM and Amiga
PRINT USING "Time: ###.## secs."; TIMER

Also, IBM and Amiga have an easier way to read in values for every element in an array. For example, primes3 can replace 5 lines with this:

READ FILE 2: p#()

Sieve can use this:

sieve#:1 = I

Upper Primes can use this:

READ d()

Of course, these are just short cuts for Amiga and IBM. They can use the longer methods too! It is your choice.

SIEVE WRITER

// Using the sieve method, primes less than 15,000 are generated and stored in data file: primes.dat // Program by Bill Inhelder (program 1 of 2)
TIME O // C64
//TIE 0 // Amiga / IBM
PAGE
PRINT "Working..."
DIM sieve%(15000)
FOR i:=1 TO 15000 DO sieve%(i):=1 // *see shortcut*
k:=2
WHILE k<SQR(15000) DO
FOR i:=2*k TO 15000 STEP k DO sieve%(i):=0
k:=k+1
ENDWHILE
count:=1
OPEN FILE 2,"primes.dat",WRITE
FOR i#:2 TO 15000 DO
IF sieve%(i#)=1 THEN
WRITE FILE 2: i#
ENDF:
ENDFOR i#
CLOSE FILE 2
PRINT USING "Time: ###.## secs."; TIME/60 // C64
//PRINT USING "Time: ###.## secs."; TIMER // Amiga / IBM

UPPER PRIMES WRITER (run sieve first)

// Using primes 2 through 179, primes from 15001 to 31622 are generated and appended to data file: primes.dat // Program by Bill Inhelder (program 2 of 2)
TIME 0 // C64
//TIE 0 // Amiga / IBM
PAGE
DIM d(40), p%(1650)
count:=1
FOR i:=1 TO 40 DO READ d(i) // *see shortcut*
PRINT AT 12,2: "Calculating Primes From 15000 to 31643" FOR i:=15001 TO 31643 STEP 2 DO finish:=FALSE
Prime Factoring Methods (continued)

\[ j:=1 \]
\[ k:=0 \]
\[ \text{WHILE NOT finish DO} \]
\[ \quad \text{pr'factors} \]
\[ \text{ENDWHILE} \]
\[ \text{ENDFOR i} \]
\[ \text{PRINT AT 14,9: "Writing Primes To File"} \]
\[ \text{OPEN FILE 2, "primes.dat", APPEND} \]
\[ \text{FOR i:=1 TO 1649 DO WRITE FILE 2: pf(i)} \]
\[ \text{CLOSE FILE 2} \]
\[ \text{PRINT AT 16,12: "Writing Completed"} \]
\[ \text{PRINT USING "Time: ##f.## sees."}: \text{TIME/60} \]
\[ \text{DATA 3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59} \]
\[ \text{DATA 61,67,71,73,79,83,89,97,101,103,107,109,113} \]
\[ \text{DATA 127,131,137,139,149,151,157,163,167,173,179} \]

PROC pr'factors
\[ q:=i \text{ DIV d(j)} \]
\[ \text{IF q=0 THEN} \]
\[ \quad \text{finish:=TRUE} \]
\[ \text{ELSE} \]
\[ r:=i \text{ MOD d(j)} \]
\[ \text{IF k=1 AND d(j)>q THEN} \]
\[ \quad \text{pf(count):=i} \]
\[ \quad \text{count:=count+1} \]
\[ \quad \text{finish:=TRUE} \]
\[ \text{ELSE} \]
\[ \text{IF r<>0 AND d(j)<i THEN} \]
\[ \quad k:=1 \]
\[ \quad j:=j+1 \]
\[ \text{ELSE} \]
\[ \text{finish:=TRUE} \]
\[ \text{ENDIF} \]
\[ \text{ENDIF} \]
\[ \text{ENDIF} \]
\[ \text{ENDPROC pr'factors} \]

PRIMES (original)

// primes - including GOTO's
PAGE // program by Bill Inhelder
branch1:
PRINT "Type in a whole number >=2 and less" 
INPUT "than 999999999": n
TIME 0 // C64
//TIME 0 // Amiga / IBM
k:=0; d:=2
branch2:
d:=d+1
IF d>3 THEN d:=d+1
branch3:
q:=n \text{ DIV d} 
IF q=0 THEN 
PRINT 
GOTO branch1 
ENDIF
r:=n \text{ MOD d} 
IF k=1 AND d=q THEN
PRINT n 
GOTO branch1 
ELSE
PRINT "Type in a whole number >=2 and less" 
INPUT "than 999999999": n
UNTIL n=2 AND n<999999999 
TIME 0 // C64 
//TIME 0 // Amiga / IBM 
k:=0; d:=2 
// set divisor initially to 2 
WHILE NOT finish DO 
pr'factors 
ENDWHILE
PRINT USING "Time: ##f.## sees."}: \text{TIME/60} 
\text{DATA 3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59} 
\text{DATA 61,67,71,73,79,83,89,97,101,103,107,109,113} 
\text{DATA 127,131,137,139,149,151,157,163,167,173,179} 

PROC pr'factors
q:=i \text{ DIV d(j)} 
\text{IF q=0 THEN} 
\quad \text{finish:=TRUE} 
\text{ELSE} 
r:=i \text{ MOD d(j)} 
\text{IF k=1 AND d(j)>q THEN} 
\quad \text{pf(count):=i} 
\quad \text{count:=count+1} 
\quad \text{finish:=TRUE} 
\text{ELSE} 
\text{IF r<>0 AND d(j)<i THEN} 
\quad k:=1 
\quad d:=d+1 
\quad IF d>3 THEN d:=d+1 
\text{ELSE} 
\quad PRINT d; 
\quad \text{replace dividend with new quotient} 
\quad k:=0 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDPROC pr'factors} 

PRIMES2 (improved version)

// primes2 - improved version of prime'factors made by 
// eliminating trial divisors which are multiples of 3 
PAGE // program by Bill Inhelder
again:=TRUE
WHILE again DO 
finish:=FALSE 
REPEAT 
\text{PRINT "Type in a whole number >=2 and less"} 
INPUT "than 999999999": n
UNTIL n=2 AND n<999999999 
TIME 0 // C64 
//TIME 0 // Amiga / IBM 
k:=0; d:=2 
// set divisor initially to 2 
WHILE NOT finish DO 
pr'factors 
ENDWHILE 
PRINT USING "Time: ##f.## sees."}: \text{TIME/60} 
\text{DATA 3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59} 
\text{DATA 61,67,71,73,79,83,89,97,101,103,107,109,113} 
\text{DATA 127,131,137,139,149,151,157,163,167,173,179} 

PROC pr'factors
q:=n \text{ DIV d} 
\text{IF q=0 THEN} 
\quad \text{finish:=TRUE} 
\quad \text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{ENDIF} 
\text{FINISH} 
\text{AGAIN} 
\text{REPEAT}
Prime Factoring Methods (continued)

```plaintext
finish:=FALSE
REPEAT
  PRINT "Type in a whole number >=2 and less"
  INPUT "than 999999999": n
UNTIL n>=2 AND n<999999999
TIME 0 // C64
// TIMER 0 // Amiga / IBM
K:=0; d:=2
WHILE NOT finish DO
  PRINT factors
ENDWHILE
PRINT USING "Time: ###.# secs.": TIME/60
// C64
// TIMER 0 // Amiga / IBM
PROC print factors
  q:=n DIV d
  IF q=0 THEN
    PRINT
    finish:=TRUE
  ELSE
    r:=n MOD pN(j)
    IF k=1 AND d>q THEN
      PRINT n
      finish:=TRUE
    ELSE
      IF r<>0 AND d<n THEN
        j:+1
        k:=1
      ELSE
        PRINT d;
        n:=q
        k:=0
      ENDIF
    ENDIF
  ENDPROC print factors
RUN TIMES
Sieve·Writer
A2  43.7
A3  81.2
A2 110.5
C2 243.3

Upper·Writer
A2  278.6
A3 n/a did not work
A2 1104.0
C2 2177.0

primes (all 4 versions)
A2-0  36.5
A2-1  44.1
A2-2  33.0
A2-3  10.1
A3-0 126.4
A3-1 131.2
A3-2 126.1
A3-3  32.3
C2-0 165.0
C2-1 179.6
C2-2 163.2
C2-3  43.1

Base model Amiga 500 and IBM PC clone used for the tests.
A2=AmigaCOMAL
A2=AmigaCOMAL
A3=IBM 3.0
C2=C64 2.0
```

COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716 - Page 59
Psychologists tell us that when learning occurs it often manifests itself in a modification of behavior. Positive reinforcement can strengthen learned behavior while negative reinforcement can undo learned behavior and lead to alternative behavior. The program learning model is a graphic simulation of a rat's ability to learn and to modify its behavior. A rat is located in a compartment containing only water. Once a day the door to a corridor opens leading to 3 feeding stalls each of which contains food. When the rat enters a stall to feed, the doors to the other two stalls close. In this way the rat receives a one-day supply of food.

The program graphically illustrates the feeding habit of a randomly selected rat over a 20 day period. Data is kept on the frequency with which the stalls are used. Following the initial observation period, the program allows the user to direct the placement of food into only one stall. The rat's task is to find the food before starvation causes death. This description belies the complexity of the decision making process. To illustrate the difficulty, let us consider the problem from a human standpoint. Suppose a prisoner is locked in a room with only water. There are 3 buttons on a wall and a slot which dispenses food. Each day the prisoner presses one of the buttons and receives a ration of food for that day. Once a button is pressed for that day the other buttons are inoperative. After some experimenting the prisoner discovers that it does not matter which button is pressed to obtain food for the day. No one button produces a better quality or choice of food.

Suppose the prisoner is more likely to press the right hand button than any of the others. Now one day he presses the right hand button and receives no food. Pressing the other buttons does no good. He figures that someone simply forgot to put the food in the compartment. Consequently he presses the right hand button the next day and still receives no food. Hunger and possibly panic make him reason that one or both the other buttons will provide food. On the third day he presses the middle button and still receives no food. Logic now tells him that, if there is any food at all, the left hand button will produce food. His reasoning will be correct provided the food is not moved from the left compartment (where it was for 3 days) back to the right compartment. Nothing is certain when a three-fold choice exists and there is an arbitrary positioning of the food.

The simulation of a rat's reasoning ability should not necessarily duplicate human reasoning ability. What sort of algorithm might simulate a rat's ability to solve the task of obtaining food? The algorithm should not produce an absolute either/or logical action but rather one which is probability based. We need an algorithm where decisions are based upon the probability of a random number falling within a specified interval. Specifically consider the interval of real numbers between 0 and 1. Select two random numbers in that interval and designate the smaller, the lower bound, and the larger, the upper bound. Suppose the lower bound is .3 and the upper bound .9. If a random number between 0 and 1 is selected, there is greater probability that it will lie in the middle interval rather than the right hand interval. This rat will select the middle stall for food more often than the left stall and much more often than the right stall. By moving the lower and upper bounds we can influence the rat's behavior in a probabilistic sense.

The algorithm used in the program to increase or decrease the bounds makes use of the rat's intelligence (rat I.Q.). Two iterative equations accomplish the task of increasing and decreasing the bounds:

\[
\text{newbound} = \text{oldbound} \times (1 - \text{smart}) + \text{smart} \\
\text{newbound} = \text{oldbound} \times (1 - \text{smart})
\]

The first equation increases the bound at a decreasing rate permitting the upper bound to approach but not exceed 1. The second equation decreases the bound at a decreasing rate permitting the lower bound to approach but not be less than 0. The rat's intelligence is assigned randomly and varies from .15 to .45 representing R.I.Qs ranging from 70 to 130. To make the algorithm function properly, upper and lower bounds must be swapped when the lower bound exceeds the upper bound. In addition the rat must be provided with a memory which recalls the previous result of its action. Rats with low R.I.Qs will take longer to find the food and may die of starvation before succeeding.
Learning Model (continued)

If the rat is right oriented select the left or middle stalls for food and keep it there until the rat finds it consistently. The more often the rat receives positive reinforcement (food), the more consistently will the rat go to that stall. Even so, occasionally the rat will revert to an old (incorrect) habit. After the rat has learned the location of the food, change it to another location and keep it there until it finds it consistently. Finally move the food around from stall to stall each day to see if you can exceed the frustration tolerance of the rat or starve it to death. In either case, if you succeed, the program will halt. Press Q to exit program. If you wish, rerun the program for a different rat.

Those who are interested in observing the changes in the bounds with respect to the placement of food insert

PRINT lo;hi;stall$ before the ENDLOOP statement in the procedure path'selection. The bounds and food location will appear on the textscreen after exiting the program.

Reference:

Challenge

I have a programming challenge for those interested in Learning Machines. It is one thing to write a program that has instructions on what to do in response to all possible situations. It is another thing to write a program that learns; that will not do what it has been told to do, but what it has learned to do.

Martin Gardner designed the game HEXPAWN as a challenge for a simple learning program. He correctly assumed that most readers would not wish to construct a learning machine that involved hundreds of possibilities. HEXPAWN has only 24 possible situations. The game is trivial, but does give you an easy basis for a learning program.

The rules of the game are simple. The game board is a 3 by 3 checkerboard. Each side has three pieces (white for one team, black for the other). These pieces (called pawns) are placed on the players end row at the start of the game, leaving the middle row blank.

There are only two possible moves:

- move one of your pawns forward one square if that square is empty.

- capture opponent pawn by moving onto its square diagonally to the left or to the right. The captured pawn is removed from the board.

The moves are similar to those of a pawn in chess.

There are three ways to win:

- Advance one of your pawns to the third row.

- Capture all your opponents' pawns.

- Put your opponent into a position where there is no possible move.

Players take turns, moving one pawn at a time. A draw is not possible.

Construct your program so that it always makes the second move (you play first). This makes the program easier for you.

You now have enough information to write the program. However, additional explanations complete with good illustrations are available in the book The Unexpected Hanging and Other Mathematical Diversions by Martin Gardner beginning on page 90. This is a Fireside book published by Simon and Schuster.

Send in your program entries now. More hints next issue (don't want to spoil your fun).
Don't Touch That Filename!!

by Len Lindsay

Don't you hate it when someone changes the name of your program on the disk (I know, we do it here to submissions). Well, I will let the secret out.

AmigaCOMAL is a next generation COMAL. It includes many new features such as two built-in keywords: argnum and argarray$. These give you information about how the program was started and any parameters passed to it. Another new feature allows you to write packages in COMAL itself.

So, I wrote a package that you should find interesting. I call it proginfo. It adds three new functions to AmigaCOMAL:

```
ProgramName$
ProgramDir$
ProgramDevice$
```

These will tell you three things about the program running: it's filename, the directory it was in, and the device it was on.

This is generally useful only for a compiled program. Run the package from AmigaCOMAL and you will only find out where AmigaCOMAL was when it was started.

Two Tips

After writing this package, I'd like to point out two items for those who will be involved in similar projects. This may seem like plain common sense to you, but it had me going in circles.

AmigaCOMAL checks to see if a package is already "loaded" when you issue a USE command. If so, it just uses that, and does not reload the package. This is great, since it saves disk access if you use a package over and over. However, it is a hinderance when developing a package. For example, I created the package and a quick little program to test it. I found a problem, so I LOADed the package, modified it, then SAVEd the package with the same name so I could run my test program unchanged. I ran the test program. Hmmmm. The problem was still there. I tried several other "fixes". None worked. Then it occurred to me that I was updating the disk package, but not the one AmigaCOMAL was using ... it kept using the original since it was "loaded" already.

Solution: issue a DISCARD command before you test your package revisions.

Next I found that argarray$(0) considers the current directory the ROOT when a program is started (except if you give a full device & directory when starting it from the CLI). Therefore, using argarray$(0) to find the program device and directory would not always work. I had to modify my package to find the program device and directory from the contents of DIR$ when a full path was not returned by argarray$(0). If the package does not find a ":" in the name, it assumes that it is not a full path. Thus my functions to return the device and directory are in two parts, analyzing one of two possible path strings.

Silly Test Program

I wrote a sample silly program that can be used to test out the package. It expects that it's file name will be SeeCOMAL. Try renaming it ... then run it. Also note that the program checks for certain tooltypes to be set if it was started by clicking on it's icon in WorkBench. In this example, it checks if they are COMAL Users Group's name and address. To pass the test, they must be exactly as specified in the three DATA statements (an interesting way to make sure your copyright notice stays with the program in its icon tooltypes).

The program, icon and package are on Amiga Today Disk #25.

The program, icon (with tooltypes set) and package are on Amiga Today Disk #25. For those who don't get that disk, here is a step by step procedure to follow to see the program and package at work:

1) Type in the sample program and save it on disk:

```
SAVE "SeeCOMAL"
```

Note that AmigaCOMAL will create a matching icon for the COMAL program. This is not the icon we will be modifying later!

2) Type in the package (it is listed after the program). Save it on disk:

```
SAVE "ProgInfo.pck"
```

3) Now you can test your program. The results will not be as interesting as after you compile it, but you will find out if all is running OK:

```
LOAD "SeeCOMAL"
RUN
```

4) Once your program is running OK, you are ready to compile it (this step requires the optional compiler available as part of the AmigaCOMAL Developers Option). Compile the
Don't Touch That Filename!! (continued)

Program SeeCOMAL (see your documentation that came with the compiler). Note that the compiler will create a matching icon for the compiled program. This is the icon we will be modifying next!

5) Go into WorkBench. Click on the disk you just stored the program on. You should see the icon for the program, unless you have it in a subdirectory. In that case you must have drawer icons for the subdirectories leading to the compiled COMAL program (or just copy it to the root of the disk).

6) Click once on the programs icon. This selects it. Then hold down the right button on the mouse. This activates the WorkBench menus. Slide over to the left, and then down to choose INFO from the menu (highlight INFO then let go of the mouse button). The INFO screen should come up for the program.

7) Put in three tool types. Click on the ADD gadget to add them. Here are the three lines you should add as tooltypes:

COMAL Users Group USA Ltd
5501 Groveland Terrace
Madison, WI 53716

8) Click on the SAVE gadget. You now are ready for the test.

9) Double click on the icon for the program. It should start up and give you this report (the info provided in the report will of course be based on the filename you used, it's subdirectory and tooltypes set>:

Program name is "test-proginfo"
Hey, you changed it!

It is in subdirectory SeeComalDir/on device ComalToday25:

The program was started from Workbench with no tooltypes.

Hey! Who deleted my tooltypes!?@^

This message was brought to you by a COMAL program from Captain COMAL!

Press a key or click your mouse now...
«click right mouse button»

Test Program (to be compiled first)

// seeecomal or test-proginfo by Len Lindsay
USE PROGINFO // use the package named proginfo
page // clear the screen

DIM mytools(3) OF 40 // three type tool names
READ mytools() // read them all in at once
DATA "COMAL Users Group USA Ltd"
DATA "5501 Groveland Terrace"
DATA "Madison, WI 53716"
PRINT AT 2,1: "Program name is:" programname$ compare(programname$,"SeeCOMAL")
CURSOR 5,1 // ready for next print
IF programdir$="" THEN
  PRINT "It is in the root directory"
ELSE
  PRINT "It is in subdirectory:" programdir$
ENDIF
IF programdevice$="" THEN
  PRINT "on your current device ("$,dir$,")".
ELSE
  PRINT "on device:" programdevice$
ENDIF
CURSOR 9,1 // ready for next print
PRINT "The program was started from":
PRINT chr$(18),argarrayS(-1),chr$(18)
IF argnum<1 THEN
  PRINT chr$(18),"Hey!
PRINT "Who deleted my tooltypes!?"
ELSE
  PRINT "with these tool types set:
FOR x:=1 TO argnum DO
  PRINT chr$(20);argarrayS(x);chr$(20)
  END FOR x
END IF
PRINT chr$(20);argarrayS(x),mytool$(x)
PRINT
END IF
ELSE
  IF argnum<1 THEN
    PRINT chr$(19),"Hey!
PRINT "Who deleted my tooltypes!?"
ELSE
  PRINT "with these tool types set:
FOR x:=1 TO argnum DO
  PRINT chr$(20);argarrayS(x),chr$(20)
  compare(argarrayS(x),mytool$(x))
  END FOR x
END IF
ENDIF
PRINT
PRINT chr$(20);chr$(19);chr$(16),chr$(16)
PRINT "Hey, you changed it!";chr$(16),
ENDIF // chr$(16)=reverse
ENDPROC compare

Next comes the listing of the package. This package is written in COMAL, so you can type it in yourself! Enter it just like any program. However, when you save it, end the file name with .2£k and viola, you have created a package!

PACKAGE: ProgInfo

EXPORT programname$ // name of program file
EXPORT programdir$ // name of directory it was in
EXPORT programdevice$ // name of device it was on
FUNC programname$ CLOSED
ps:=argarray$@O) // full name: device:directory/name
// next use the IN operator to get a substring
ps:=ps("/" IN ps)+1)
// next check the subdirectories
WHILE "/" IN ps DO ps:=ps("/" IN ps)+1:
RETURN ps
ENDFUNC programname$

FUNC programdir$ CLOSED
IF ":" IN argarray$@O) THEN \\ a full path
RETURN dirdir$(argarray$@O)
ELSE
RETURN dirdir$(dir$) // use dir$ for path
ENDIF
ENDFUNC programdir$

FUNC dirdir$(pS) CLOSED // this func is NOT exported
TRAP
in case there is no ":" in pS
pS:=pS("/" IN pS)+1:
HANDLER
NULL // don't change pS
ENDTRAP
dS:="" // init to null in case no directories
WHILE "/" IN pS DO // as long as "/" is in the name...
dS:=ps(l:"/" IN ps}) // concatenate the string
ps:=ps("/" IN ps)+1:
ENDWHILE
RETURN dS
ENDFUNC dirdir$

FUNC programdevices$ CLOSED
IF ":" IN argarray$@O) THEN // a full path
RETURN device devices$(argarray$@O)
ELSE
RETURN device devices$(dir$) // use dir$ for path
ENDIF
ENDFUNC programdevices$

FUNC device devices$(pS) CLOSED // this func is NOT exported
IF ":" IN pS THEN // substring IN search
RETURN ps(l:"/" IN pS)) // substring with IN search
ELSE
RETURN ""
ENDIF
ENDFUNC device devices$

Note: this package and program was on PLink for AmigaCOMAL
users to download over a month before publication here. Ah, the
benefits of instant access on-line support. •
Power To The People

by Jesse Knight

[Just a few days after Jesse received AmigaCOMAL he had solved one of the annoying quirks ... when a program ends, the execute screen stays in front. The command screen is not returned to the front. If you would like the command screen to automatically pop to the front, this package will do it for you.]

This small package utilizes the signal routine. As you can see in the listing, it only needs one line! If it senses a signal number 7, 8 or 9 it calls the commandtofront procedure, which pushes the command window to the front. Section 4.2 of your manual explains the meanings of the signal values. Here are the meanings for the three we are catching:

7 - Program end by program error.
8 - Program end.
9 - Program stop that may be CONTInued.

It is easy to set up this package. Just use AUTO mode to type it in as if it were a program. Then use the SAVE command to store it on disk as a package. It is a good idea to keep it in the packages subdirectory, so this command should do the trick:

SAVE "#packages/CFront.pck"

If you have COMAL installeed on a hard drive, you will need the correct path to the packages directory. One sure way to store it properly is to change directories so that packages is the current directory with a command similar to:

CD "#packages"

Then you can just use a normal SAVE command with the .pck filename extension:

SAVE "CFront.pck"

This gives a package with the name CFront. It could just as easily have been called CommandToFront, but CFront is much less typing.

Now, just issue a USE command to start it up:

USE CFront

If you would like it to start each time you start COMAL, add a tooltype to the AmigaCOMAL icon:

USE CFront

Double click on the AmigaCOMAL icon to start COMAL with CFront active. You can see how AmigaCOMAL gives you power to change even its operating environment! Power to the people.

Since this is a package, it may at times become "discarded". In that case just issue the USE command again.

USE SYSTEM
USE INTUITION_LIBRARY
EXPORT commandtofront
GLOBAL pckwindowtofront
GLOBAL comalstructa
GLOBAL commandtofront

PROC commandtofront CLOSED
  pckwindowtofront(comalstructa.command_ioi.window#)
ENDPROC commandtofront

PROC signal(s) CLOSED
  IF s>=7 AND s<=9 THEN commandtofront
ENDPROC signal

SYSTEM Fix

[Jesse also found the following problem ... which is easily corrected.]

Section 4.4.1.2 of the manual lists the io_struct@ record structure. Appendix D.1 also lists this structure. However, Appendix D.1 lists nine more items at the end of the record definition! This is the correct version. However, looking at the SYSTEM package on the original AmigaCOMAL disk we find the io_struct@ record structure is also missing those fields.

You can see the power of AmigaCOMAL when I tell you how easy it is to correct the SYSTEM package. Just add the missing lines and save it back to disk! Do this:

CD "/packages" /use path to your packages directory
LOAD "system.ock"
LIST 90-160
0090 RECORD io_structa
  0100 FIELD screen#
  0110 FIELD screentype%, screendepth%, screenwidth%, screenheight% // wrap line
  0120 FIELD windoW#
  0130 FIELD windowdepth%, charnoX, lineno%
  0150 FIELD gzzxoff%, gzzyoff%
  0155 FIELD windowwidthX, windowheightX
  0160 ENDRECORD io_structa
AUTO 151.1
0151 field fontid#, fontheightX, fontwidth%, fontbase%
0152 field virtual#
0153 cursor!, softstyle!
0154 menuhd#
0155 menubytes#
0156 «Esc»
SAVE "system.ock"

That is all there is to it. The SYSTEM package is written in COMAL, and it is easy to modify COMAL coded packages. Note that only the earliest AmigaCOMAL disks had the missing fields. The master disk was corrected immediately. ■
by David Stidolph

AmigaCOMAL not only implements Common COMAL, but extends it with many new features. This article focuses on one of those new features: RECORDS. A record is a set of data elements tied together under the same name. An example of this is a simple data base to keep track of a person's name, address and telephone number. In any other COMAL you might use the following variables:

```
DIM last'name$ OF 20
DIM first'name$ OF 15
DIM init$ OF 1
DIM address$ OF 25
DIM city$ OF 20
DIM state$ OF 2
DIM zip$ OF 5
DIM phone'num$ OF 15
```

This would work, but to pass a person's information to a procedure would require eight parameters -- a lot of typing at the very least. In AmigaCOMAL you could group all these variables into one record. For example:

```
RECORD type'person
  FIELD last'name$ OF 20
  FIELD first'name$ OF 15
  FIELD init$ OF 1
  FIELD address$ OF 25
  FIELD city$ OF 20
  FIELD state$ OF 2
  FIELD zip$ OF 5
  FIELD phone'num$ OF 15
END RECORD type'person
```

This may be a bit more typing initially, but let's suppose you need a procedure to enter information about a new person, but want to pass the variables as parameters because you want several different persons data in memory at once (the old person, the new person and matching data for comparisons). Under most COMAL's you would have a procedure declaration like this:

```
PROC input'data(REF ln$,REF fn$,REF in$,REF ad$,REF ci$,REF st$,REF zip$,REF ph$) //wrap line
```

Besides shortening the names so they would fit on a line, you would have to keep constant track of not only the names of the variables, but the order to pass them to the procedure. If you need several different versions of the same data, the names would start to get very long (like DIM esensioning a string):

```
DIM personal OF type'person
DIM old'person OF type'person
DIM new'person OF type'person
```

All of these records have the same named variables within them, but each of those variables could hold a different value. `personal` is different from `old'person`. Besides making parameter passing easier, records can be assigned the values of other records if they are of the same type. For example, you may wish to make a new person the current one, but first change the current one to be the old person:

```
old'person := personal
personal := new'person
```

Records can really help your programming, and on the Amiga they are absolutely necessary.

---

Notice that you don't have to remember the order of parameters. To refer to any one of the variables kept in `personal` you use the following method:

```
personal.last'name$
```

`personal`, as a prefix to the variable informs COMAL that this is a record variable, and specifies the specific record (`personal`). After that comes the actual variable you want to access. In order to use `personal` you must first define it (like DIMensioning a string):

```
DIM personal OF type'person
```

This tells COMAL that you want a variable record named `personal` and to look at the definition `type'person` for the different type of variables and their names. Another advantage is that you could create several different records based on `type'person` and make your program both shorter and more readable:

```
DIM personal OF type'person
DIM old'person OF type'person
DIM new'person OF type'person
```

Records can really help your programming, and on the Amiga they are absolutely necessary.
PROC init'program
max'records:=20
TRAP
OPEN FILE 1,"demo.rnd",RANDOM varsie(type'personQ)
HANDLER
CREATE "demo.rnd",max'records,varsie(type'personQ)
OPEN FILE 1,"demo.rnd",RANDOM varsie(type'personQ)

// Amiga COMAL automatically initializes variables
// to null values, so we don't have to.
FOR record'nurn:=1 TO max'records DO
WRITE FILE 1,record'nurn: personQ
ENDFOR record'nurn
ENDTRAP
ENDPROC init'program

PROC menu(REF choice$) CLOSED
DIM keystroke$ OF 1
PRINT "Main Menu"
PRINT
PRINT "1) Add person"
PRINT "2) Delete person"
PRINT "3) Find person"
PRINT "4) Quit program"
PRINT
REPEAT
PRINT "Your choice:"
choice$:=inkey$
UNTIL choice$ IN "1234adfqADFQ"
PRINT
PRINT
ENDPROC menu

PROC do'menu(choice$)
CASE choice$ OF
WHEN "1","a","A"
input'person(personGl)
record'nurn:=next'free
IF record'nurn=0 THEN
PRINT "No more records available!"
prompt'reTURN
ELSE
personQ.valid!:=true
WRITE FILE 1,record'nurn: personQ
ENDIF
WHEN "2","d","D"
FOR record'nurn:=1 TO max'records DO
READ FILE 1,record'nurn: personQ
IF personQ.valid! THEN
PRINT "(name:"
dispIay'person(personGl)
IF wants'deleted THEN
personQ.valid!:=false
WRITE FILE 1,record'nurn: personQ
ENDIF
ENDFOR record'nurn
WHEN "3","f","F"
INPUT "Enter characters to match: ": match$ 
FOR record'nurn:=1 TO max'records DO
READ FILE 1,record'nurn: personQ
IF personQ.valid! THEN
IF match$="" OR match$ IN combineS(personQ) THEN
PRINT "Can be reached at:" phone$ 
ENDIF
ENDIF
ENDFOR record'nurn
OTHERWISE
NULL // ignored
ENDPROC do'menu

PROC quit
CLOSE FILE 2
END "Done."
ENDPROC quit

PROC display'person(personQ)
PRINT "name:"
PRINT personQ.first'nameS:personQ.initalS
PRINT "Sex: ": personQ.sexS,""
IF personQ.married THEN
PRINT "is married"
ELSE
PRINT "is not married"
ENDIF
PRINT "Phone number: ": personQ.phone$ 
PRINT
ENDPROC display'person

FUNC combineS(personQ) CLOSED
RETURN personQ.last'nameS+personQ.first'nameS
ENDPROC combine

PROC input'person(REF personQ)
INPUT "Last name: ": persona.last'nameS
INPUT "First name: ": personQ.first'nameS
INPUT "Middle initial: ": personQ.initalS
INPUT "Sex: ": personQ sexe$ 
REPEAT
 INPUT "Married (y/N): ": confirm$ 
UNTIL confirm$ IN "yynN" 
personQ.married:=(confirm$ IN "yY")
INPUT "Number of children: ": personQ.num'of'children 
INPUT "Phone number: ": personQ.phone$ 
ENDPROC input'person

PROC press'reTURN CLOSED
PRINT "Press RETURN to continue" 
WHILE keystroke$<>chrS(13) DO NULL
ENDPROC press'reTURN

FUNC next'free CLOSED
IMPORT type'personQ,max'records
DIM personQ OF type'personQ
FOR record'nurn:=1 TO max'records DO
READ FILE 1,record'nurn: personQ
IF NOT personQ.valid! THEN RETURN record'nurn
ENDFOR record'nurn
RETURN 0
ENDPROC next'free

PROC wants'deleted CLOSED
IMPORT type'personQ,max'records
DIM personQ OF type'personQ
FOR record'nurn:=1 TO max'records DO
READ FILE 1,record'nurn: personQ
IF NOT personQ.valid! THEN RETURN record'nurn
ENDIF
RETURN 0
ENDPROC wants'deleted

ENDCOMAL
You can get a list of all the routines in a specified package by using the command:

```
LISTPACK <package name> [[TO] file name]
```

**Example:** To get a printer listing of all the routines in the graphics package you would type:

```
LISTPACK Graphics TO "lp:"
```

When a package is read into RAM from disk it will stay in RAM until all packages are removed by the command:

```
DISCARD
```

### 4.2 Programming Packages

A package may be written in AmigaCOMAL (COMAL package) or it may be written in assembler, C or another suitable compiler (machine coded package).

In this section we will describe how to develop COMAL coded packages. Developing machine coded packages will be discussed in the Development Manual (a separate, added cost option).

A COMAL coded package is simply a normal AmigaCOMAL program with some EXPORT statements added at the beginning and then saved to disk with the SAVE command, specifying a file name that ends with .pkg.

**Example:** The following package contains only one procedure, rand, which is an improved version of the rnd function in AmigaCOMAL (for further details see BYTE March 1987 page 127).

```
0010 // Random package
0020
0030 EXPORT rand
0040
0050 x:=1; y:=10000; z:=3000 // Seeds
0060
0070 FUNC rand CLOSED
0080 x:=171*(x MOD 177)-2*(x DIV 177) // First generator
0090 y:=172*(y MOD 176)-35*(y DIV 176) // Second generator
0100 z:=170*(z MOD 178)-63*(z DIV 178) // Third generator
0110 temp:=x/30269.0+y/30307.0+z/30323.0 // Combine
0120 RETURN temp-int(temp)
0130 ENDFUNC rand
```

AmigaCOMAL, 5501 Groveland, Madison, WI 53716 - 83
## DISK DIRECTORIES

### COMMODORE 64

**Today Disk 22**

**Front**

- bootslow
- fastboot
- ml.sizzle
- power driver
- hi
- menu

<table>
<thead>
<tr>
<th>power driver</th>
<th>system files</th>
</tr>
</thead>
<tbody>
<tr>
<td>next are text</td>
<td>files read by</td>
</tr>
<tr>
<td>the menu prog</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>notes.txt</th>
<th>graphics.txt</th>
<th>sprites.txt</th>
<th>powerkeyword.txt</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt'maze.pwr</td>
</tr>
<tr>
<td>animal fun</td>
</tr>
<tr>
<td>chi-square.pwr</td>
</tr>
<tr>
<td>creatormaker.pwr</td>
</tr>
<tr>
<td>display bam.pwr</td>
</tr>
<tr>
<td>football quiz</td>
</tr>
<tr>
<td>image'lines.1st</td>
</tr>
<tr>
<td>maze.pwr</td>
</tr>
<tr>
<td>multident.pwr</td>
</tr>
<tr>
<td>music</td>
</tr>
<tr>
<td>unscratch.pwr</td>
</tr>
<tr>
<td>walking boots</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data files</th>
</tr>
</thead>
<tbody>
<tr>
<td>fortress.song</td>
</tr>
<tr>
<td>joyful.song</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sprite files !</th>
</tr>
</thead>
<tbody>
<tr>
<td>imag.26animals</td>
</tr>
<tr>
<td>imag.ant</td>
</tr>
<tr>
<td>imag.bear</td>
</tr>
<tr>
<td>imag.cat</td>
</tr>
<tr>
<td>imag.dog</td>
</tr>
<tr>
<td>imag.elk</td>
</tr>
<tr>
<td>imag.fox</td>
</tr>
<tr>
<td>imag.goat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>func.digits</td>
</tr>
<tr>
<td>func.value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sprite images</th>
</tr>
</thead>
<tbody>
<tr>
<td>for use with</td>
</tr>
<tr>
<td>read file</td>
</tr>
</tbody>
</table>

---

**Today Disk 22**

**Back**

<table>
<thead>
<tr>
<th>hi</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>2.0 programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>animal fun</td>
</tr>
<tr>
<td>bootlang syne</td>
</tr>
<tr>
<td>buffer editor</td>
</tr>
<tr>
<td>c128 hi-res cube</td>
</tr>
<tr>
<td>chi-square</td>
</tr>
<tr>
<td>convertimag-shap</td>
</tr>
<tr>
<td>creatormaker</td>
</tr>
<tr>
<td>dir designer</td>
</tr>
<tr>
<td>image'lines.1st</td>
</tr>
<tr>
<td>iterative (blas)</td>
</tr>
<tr>
<td>mouse maze</td>
</tr>
<tr>
<td>mouse maze'alt</td>
</tr>
<tr>
<td>multident</td>
</tr>
<tr>
<td>music</td>
</tr>
<tr>
<td>partitionaid1581</td>
</tr>
<tr>
<td>playscore'rose</td>
</tr>
<tr>
<td>reorder dir</td>
</tr>
<tr>
<td>sprite creator</td>
</tr>
<tr>
<td>walking boots</td>
</tr>
<tr>
<td>write'playscore</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data files</th>
</tr>
</thead>
<tbody>
<tr>
<td>dat.rose</td>
</tr>
<tr>
<td>fortress.song</td>
</tr>
<tr>
<td>joyful.song</td>
</tr>
<tr>
<td>txt.dir help</td>
</tr>
<tr>
<td>txt.buffer help</td>
</tr>
<tr>
<td>txt.buffer inst</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>func.digits</td>
</tr>
<tr>
<td>func.value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sprite images</th>
</tr>
</thead>
<tbody>
<tr>
<td>for use with</td>
</tr>
<tr>
<td>read file</td>
</tr>
</tbody>
</table>

---

**Attention!**

- all the needed power driver files are combined to 1 self-disolving arc file below

- this file may be uploaded to networks and local bbs's

- after download place a blank formatted disk in the drive & issue the run command: run it will write the full files for the power driver system onto the disk

- the file is on qlink in our comal section

- please don't give copies of this disk out

---

**DISK DIRECTORIES**

**Today Disk 22**

**Back**

---

**Notes**

- copyright 1988 |
| comal users |
| group usa ltd |
| 5501 groveland |
| madison, wi |
| 53716 |

---

**DISK DIRECTORIES**

**Today Disk 22**

**Back**

---

**Notes**

- copyright 1988 |
| comal users |
| group usa ltd |
| 5501 groveland |
| madison, wi |
| 53716 |

---

**DISK DIRECTORIES**

**Today Disk 22**

**Back**

---

**Notes**

- copyright 1988 |
| comal users |
| group usa ltd |
| 5501 groveland |
| madison, wi |
| 53716 |

---

**DISK DIRECTORIES**

**Today Disk 22**

**Back**

---

**Notes**

- copyright 1988 |
| comal users |
| group usa ltd |
| 5501 groveland |
| madison, wi |
| 53716 |

---

**DISK DIRECTORIES**

**Today Disk 22**

**Back**

---

**Notes**

- copyright 1988 |
| comal users |
| group usa ltd |
| 5501 groveland |
| madison, wi |
| 53716 |

---

**DISK DIRECTORIES**

**Today Disk 22**

**Back**

---

**Notes**

- copyright 1988 |
| comal users |
| group usa ltd |
| 5501 groveland |
| madison, wi |
| 53716 |
DISK DIRECTORIES

shap.xenopus
shap.yak
shap.zebra

-----
[please do not]
copy this disk
-----
copyright 1988
comal users
group usa ltd
5501 groveland
madison, wi
53716
608-222-4432
-----

Today Disk 23
Front

bootslow
fastboot
ml.sizzle
power driver
hi

power driver
programs

bbca color book
cocs color book
computer
cowboy
cryptograms.pwr
dribble
hearts
high'card
tower

data files

dat.cryptograms2
A-room'descs.ran
A-seqdata.dat
A-help.txt

procedure
prompt.proc

| programs that were listed to disk. use the commands: |
| new |

| enter | to retrieve |
| the programs |
| the programs will run under both 2.0 cart & power driver |
| more lst files on back of the disk for power driver |
| a-h/make'ran.lst |
| a-h/make'seq.lst |
| allegan-high.lst |
| envelope-prt.lst |
| utility |
| expr-eval.lst |
| use the next 2 files to set screen colors that work with both 2.0 cart & power driver |
| toplines1-7.1st |
| textcolors.lst |

copyright 1988
comal users
group usa ltd

5501 groveland
madison, wi
53711

please do not give out copies of this disk

Today Disk 23
Back

boot for 2.0
bannerprinter2.0
cryptograms2.0
demo/calc2.0
diff'limtd'agg
freecatalogdb2.0
graphsideways2.0
hark.xmas.song
param'graph2.0
stringsets2.0

data files

1985 kwh.grf
1986 kwh.grf
1987 kwh.grf
1988 kwh.grf
average kwh.grf
average temp.grf
dat.cryptograms1
db.freecats
dif'lim.hrg
docorwho.ran
message1.txt
nana dollars.grf
nana sales.grf
whoyears.dat

| new | enter |
| programs |
| proc/funcs |
| func.get'input |
| func.value |
| proc.calculator |
| proc.prompt |
| program files |

今天的盘24

bootslow
fastboot
ml.sizzle
power driver
hi

| power driver | programs |
| d-read'write.pwr | filemaster.pwr |
| listerine.pwr |

| 2.0 cartridge | programs |
| fandsgrapher | xyz.words! |
| xyz.puzzle |
| comal-fortran | translator |

| fortran.1st |
| pics demo |
| pkg.pics |
| pic a dragon |
| hrg.trek |
| listed to disk |

| 2.0 cart only |
| who-edit-2.0.1st |

use enter to retrieve them
2.0 check the
// comments
for notes

Page 70 - COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716
<table>
<thead>
<tr>
<th>DISK DIRECTORIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>chaos.lst</td>
</tr>
<tr>
<td>fracpattern1.lst</td>
</tr>
<tr>
<td>fracpattern2.lst</td>
</tr>
<tr>
<td>fracpattern3.lst</td>
</tr>
<tr>
<td>fracpattern4.lst</td>
</tr>
<tr>
<td>fracpattern5.lst</td>
</tr>
<tr>
<td>fracpattern6.lst</td>
</tr>
<tr>
<td>fracpattern7.lst</td>
</tr>
<tr>
<td>mastermind.lst</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>chaos</td>
</tr>
<tr>
<td>fracpattern1</td>
</tr>
<tr>
<td>fracpattern2</td>
</tr>
<tr>
<td>fracpattern3</td>
</tr>
<tr>
<td>fracpattern4</td>
</tr>
<tr>
<td>fracpattern5</td>
</tr>
<tr>
<td>fracpattern6</td>
</tr>
<tr>
<td>fracpattern7</td>
</tr>
<tr>
<td>mastermind</td>
</tr>
</tbody>
</table>

---

### IBM PC COMAL 2.2 Disk 2

- **CO87** BAT
- **COH87** BAT
- **COMAL87** EXE
- **GRAPHV87** PKG
- **GRAPHH87** PKG
- **COMAL87** RUN
- **TEST** DOC
- **TEST** ASM
- **TEST** PKG
- **TEMPLATE** ASM
- **COMAL** SYM
- **READ** ME
- **COMAL** RME
- **GRAPHV** RME
- **COMALC** RME
- **HERCULES** DOC

**17 Files**

---

### IBM PC COMAL 2.2 Disk 3

- **<DIR> SUPP**
- **SOUND** DOC
- **SCALES** CML

---

### IBM PC COMAL Special Series 1

- **TESTWIND** LST
- **WINDOWS** LST
- **CALENDAR** LST
- **MAGICSQR** LST
- **NIM** LST
- **NUMTOWOR** LST
- **HANOI** LST
- **DATABASE** LST
- **STARTREK** RND
- **DOCWHO** RND
- **FILEIO** LST
- **MULTIDEN** LST
- **ABS** LST
- **AND** LST
### DISK DIRECTORIES

| AT    | LST | ATN | LST | BITAND | LST | BITOR | LST | BITXOR | LST | CASE | LST | CHR | LST | CLOSED | LST | COS | LST | CURCOL | LST | CURROW | LST | CURSOR | LST | DATA | LST | DIM | LST | DIV | LST | DO | LST | ELIF | LST | ELSE | LST | ENDCASE | LST | ENDFOR | LST | ENDFUNC | LST | ENDOOP | LST | ENDTRAP | LST | ENDO WHILE | LST | EOD | LST | ERR | LST | ERRTEXT | LST | ESC | LST | EXEC | LST | EXIT | LST | EXP | LST | EXTERNAL | LST | FALSE | LST | FOR | LST | FREE | LST | FUNC | LST | GOTO | LST | HANDLER | LST | IF | LST | IMPORT | LST | IN | LST | INT | LST | LEN | LST | LOG | LST | LOOP | LST | MOD | LST | NOT | LST | NULL | LST | OF | LST | OR | LST | ORD | LST | OTHERWISE | LST | PI | LST | PROC | LST | RANDOMIZE | LST | READ | LST | REF | LST | REPEAT | LST | REPORT | LST | RESTORE | LST | RETURN | LST | RND | LST | SGN | LST | SIN | LST | SPC | LST | SQR | LST | STEP | LST | STR | LST | TAB | LST | TAN | LST | TEST | LST | THEN | LST | TO | LST | TRAP | LST | TRUE | LST | UNIT | LST | UNTIL | LST | VAL | LST | WHEN | LST | WHILE | LST | ZONE | LST | COMMON | LST | README 6 | LST | 9 Files | | README 6 | LST | 9 Files | | IBM PC COMAL | 3.0 Supplemental Programs | | ACKER | LST | | ARC | LST | | BEAR | LST | | BOUNCE VGA | LST | | BOX | LST | | BUSINESS | LST | | CALENDAR | LST | | CALEXT | LST | | CALMOD1 | LST | | CALMOD1 MOD | LST | | CALMOD2 | LST | | CALMOD2 MOD | LST | | CMY | LST | | COLORS | LST | | COMPLEX | LST | | COMPLEX2 | LST | | DIFF | LST | | DIFF$ | FNC | | DIFF$ | LST | | DIFFEXT | LST | | EVAL | LST | | FIND | LST | | FRAC | LST | | FRAC STR | LST | | FRACEXT | LST | | FRACS | LST | | FRACTION | LST | | GAUSS | LST | | HANOI | LST | | HLS | LST | | INPUT | LST | | LIFE | LST | | LIST | LST | | MANDEL | LST | | MANDEL MD6 | LST | | PAINT | LST | | PERMUTE | LST | | POLYGON | LST | | POLYGON2 | LST | | PRIM | LST | | PRIMES | LST | | QUEUE | LST | | QUICK | LST | | RADIX | LST | | RGB | LST | | SCALES | LST | | SIEVE | LST | | 15 Files | | IBM PC COMAL | 3.0 System 1 | | COMAL | EXE | | COMAL87 | EXE | | COMAL RUN | LST | | COMAL87 RUN | LST | | COMC | EXE | | COMC LST | LST | | PROTECT | EXE | | PROTECT MSG | MSG | | RUNUS | MSG | | RUNGR | MSG | | RUNDK | MSG | | CFIND | LST | | CFIND CML | CML | | CFIND | ARF | | ARG | LST | | COMALC | RME | | README 5 | LST | | 15 Files | | IBM PC COMAL | 3.0 System 2 | | GRAPHICS EXE | LST | | IBM PC COMAL | 3.0 System 1 | | COMAL | EXE | | COMAL87 | EXE | | COMAL RUN | LST | | COMAL87 RUN | LST | | COMC | EXE | | COMC LST | LST | | PROTECT | EXE | | PROTECT MSG | MSG | | RUNUS | MSG | | RUNGR | MSG | | RUNDK | MSG | | CFIND | LST | | CFIND CML | CML | | CFIND | ARF | | ARG | LST | | COMALC | RME | | README 5 | LST | | 15 Files | | IBM PC COMAL | 3.0 System 1 | | COMAL | EXE | | COMAL87 | EXE | | COMAL RUN | LST | | COMAL87 RUN | LST | | COMC | EXE | | COMC LST | LST | | PROTECT | EXE | | PROTECT MSG | MSG | | RUNUS | MSG | | RUNGR | MSG | | RUNDK | MSG | | CFIND | LST | | CFIND CML | CML | | CFIND | ARF | | ARG | LST | | COMALC | RME | | README 5 | LST | | 15 Files | | IBM PC COMAL | 3.0 System 1 | | COMAL | EXE | | COMAL87 | EXE | | COMAL RUN | LST | | COMAL87 RUN | LST | | COMC | EXE | | COMC LST | LST | | PROTECT | EXE | | PROTECT MSG | MSG | | RUNUS | MSG | | RUNGR | MSG | | RUNDK | MSG | | CFIND | LST | | CFIND CML | CML | | CFIND | ARF | | ARG | LST | | COMALC | RME | | README 5 | LST | | 15 Files | | IBM PC COMAL | 3.0 System 1 | | COMAL | EXE | | COMAL87 | EXE | | COMAL RUN | LST | | COMAL87 RUN | LST | | COMC | EXE | | COMC LST | LST | | PROTECT | EXE | | PROTECT MSG | MSG | | RUNUS | MSG | | RUNGR | MSG | | RUNDK | MSG | | CFIND | LST | | CFIND CML | CML | | CFIND | ARF | | ARG | LST | | COMALC | RME | | README 5 | LST | | 15 Files | | IBM PC COMAL | 3.0 SCOM | | SCOM | EXE | | SCOMUS | MSG | | SCOMGR | MSG | | SCOMDK | MSG | | COM DEMO LST | LST | | COM DEMO HLP | LST | | EXAMPLE1 | LST | | SCOM | RME | | README 1 | LST | | 15 Files | | IBM PC COMAL | 3.0 SCOM | | SCOM | EXE | | SCOMUS | MSG | | SCOMGR | MSG | | SCOMDK | MSG | | COM DEMO LST | LST | | COM DEMO HLP | LST | | EXAMPLE1 | LST | | SCOM | RME | | README 1 | LST | | 15 Files | | IBM PC COMAL | 3.0 SCOM | | SCOM | EXE | | SCOMUS | MSG | | SCOMGR | MSG | | SCOMDK | MSG | | COM DEMO LST | LST | | COM DEMO HLP | LST | | EXAMPLE1 | LST | | SCOM | RME | | README 1 | LST | | 15 Files | | IBM PC COMAL | 3.0 SCOM | | SCOM | EXE | | SCOMUS | MSG | | SCOMGR | MSG | | SCOMDK | MSG | | COM DEMO LST | LST | | COM DEMO HLP | LST | | EXAMPLE1 | LST | | SCOM | RME | | README 1 | LST | | 15 Files | | IBM PC COMAL | 3.0 SCOM | | SCOM | EXE | | SCOMUS | MSG | | SCOMGR | MSG | | SCOMDK | MSG | | COM DEMO LST | LST | | COM DEMO HLP | LST | | EXAMPLE1 | LST | | SCOM | RME | | README 1 | LST | | 15 Files |
The largest group of Amiga™ users in the world shares its problems and solutions online every day in CompuServe’s Amiga Forums. And you can join them.

Whether you’re an Amiga novice or a professional user in broadcasting, film special effects, animation, or music production, you’ll find support from thousands of Amiga users and nearly every third-party Amiga software and hardware vendor.

Looking for a solid CAD program? Want to make the most of your Amiga’s multitasking capabilities? Ask somebody who’s been through it all. There’s no better way to get more out of your Amiga.

To join CompuServe, see your computer dealer.
To order direct or for more information, call 800 848-8199. If you’re already a member, type GO AMIGA at any ! prompt.

Make Amigos With Other Amigas.
DISK DIRECTORIES

WorkBench
Boot
Note: you need WShell, A:Rexx & FACC II to get all the files.

C
AddMem
Alias
Arc
Ask
Assign
Avail
BaseName
Binddrivers
booz
Break
CD
ChangeTask
Cap
ConMan
Copy
Date
Delete
Dir
DiskChange
Dropbuf
E
EndCLI
EndSkip
Eval
Exec10
Execute
Fault
FF
FileNote
flush
GetEnv
hi
History
IconX
Info
Install
Join
List
LoadLib
LoadMB
Lock
MakeDir
morerows
Mount
Move
PatchMB
Path
Protect
Push
QMouse
Queue
Read
Relabel
Rename
Resi
rexxmast
Run
RunBack
RunWSH
rx
rxc
rlxib
mathiee.doub. lib
mathiee.doubtrans. lib
mathtrans. library
rexxarplib.library
rexxsupport.library
rexxsys.lib.library
screensh. library
translator.library
version.library
wshell.library

READERS

C
MuchMore
Ctype-0
CLI-Only
CLS
PlusCR
StripCR
StripFL
Ctype
Ctype-2
Ctype-3
Includes
DoseqAes.ASM
Header.ASM
InteqAes.ASM
Macro.ASM
Startup.ASM
SysEqAes.ASM
PlusCR.s
ReadMe2
StripCR.s
StripFL.s
Distribution
FullView
FDView
FDView.doc
FVCpress

Less
less
less.doc
less.help
makefile
README
ReadMe.doc
src
ch.c
command.c
funs.c
help.c
input.c
io.c
less.h
line.c
main.c
option.c
os.c
output.c
position.c
position.h
prim.c
print.c
prompt.c
screen.c
signal.c
ttyin.c
version.c

SYSTEM

ARun
Dimouse
Echo
Else
EndIf
Failat
IF
Lab
Play
Prompt
Quit
Resident

BOOT2

BooTDocs
morerows.doc
pprefs.doc
QMouse.doc
UtilDocs
WB1.3.2-LICENSE
WB1.3.2-ReadMe

$DPAT
PCD
SPAT
ScreenX
ScreenX.docs
System
booz
CMD
GraphicDump
InstallPrinter
MergeMem
NofastMem
PrintFiles
Unarc
Unzip

COMAL Today #25, 5501 Groveland Terrace, Madison, WI 53716 - Page 75
DISK DIRECTORIES

MouseReader.doc
MuchMore
ReadMe
Print
OView
QView
ReadAll
ReadAll
read
Strings
TextDisplay
TextDisplay
TextFormat
TextFormat
Ty
README
ty
מוקד
Type
View
POSTER
View
WordCount
Wordcount
Wordcount.c

Text Editors

ANSIEd
Demo
Demo.doc
EASTER.ANSI
FILE.ANSI
FLAG.ANSI
GRAPE.ANSI
MAIN.ANSI
MB.ANSI
Samples.doc
Avi
Avi.doc
POSTER
c
less
MuchMore
c dysnusEd.demo
cedefaults
cedefaults.asm
cedefaults.c
cedemo
cedhelp.hlp
cedmacros
ReadMe
Distribution
 Hed
hned
ReadME

Jed
Jet
status
type
manual
Version
wait

ARP/\ DOCS

arp13
.doc
info
arp.readme
Arpinstal1
AshMan
arpcodes
addbuffers
arrun
ask
assign
break
cd
changeaskpriority
copy
date
delete
dir
diskchange
echo
failat
filenote
fixarp5.sav
if
info
install
join
lab
list
loadlib
makedir
mount
path
prompt
protect
quit
Readme.arp
relabel
rename
resident
search
set
setdate
skip
sort
stack

CreatePort
CreateTaskResult
CreateStr
DeletePort
DeleteStr
DosAllocMem
DosFreeMem
EscapeString
FileRequest
FindCLI
FindFirst
FindNext
FindTaskResult
FPrint
FreeAccess
FreeAnchorChannel
FreeDOSList
FreeTaskResult
FreeTaskResList
FreeTrackedException
GADS
GetAccess
GetDevInfo
GetEnv
GetTracker
InitStdPacket
LDiv
LMod
LMult
LoadPrg
ObtainResidentPath
PathName
PatternMatch
Pread
Print
Puts
Qsort
ReadLine
ReleaseResidentRemResidentPrg
RListAlloc
SendPacket
SetEnv
StapleStr
Strcmp
Strncmp
StrtoStaple
SyncRun
TackOn
Toupper
TRAK_ID
UnloadPrg

NEW MANUAL

ArpAllocFreq
ArpOpenLibrary
AsyncRun_ADDEND
FileRequest_ADDEND
FindFirst_ADDEND
GetEnv_ADDEND
GetKeywordIndex
PatternMatch_ADDEND
Print
StapleStr_ADDEND
SyncRun_ADDEND

OLD MANUAL

AddADevs
AddAKnode
AddResidentPrg
ArpAlloc
ArpAllocMem
ArpDupLock
ArpExit
ArpLock
ArpOpen
Assign
AsyncRun
Atol
BaseName
BtoCStr
CheckAbort
CheckBreak
CheckSumPrg
CloseWindowSafe
CompareLock

req.msb
request-ia89
req.ml
req1.ml
req2.ml
req3.ml
arreq
jlib
ml_loader
post
request-now8
arp.bmp
exec.bmp
FileReq
read.me

Notes

Amiga COMAL disk may soon be expanded into a two disk set.

The BOOT disk set now consists of three disks. You must send us your WorkBench disk when you order to prove you own WorkBench. You also must get WShell, since most of the system is built around it. It is the best shell out! If you do not get FACC II and ARExx, some listed files will not be on your disk that are shown in the directory.

I think I have all the DOC files back with the programs, but if I missed any, please let me know. It may turn into a four disk set. Some included programs are shareware.

UtilMaster is shareware. I am testing the next release and I am fascinating. Please register your copy so he will release the next version!

Text Readers & Editors let you try many types to find one you like best. Many are shareware.

We desperately need some COMAL Demo programs for a Demo disk! Send your submissions in as soon as you can.

We also want to put together a supplemental packages disk. If you write your own packages, consider sharing them with other COMALists. You can submit them on disk, or via People Link or CompuServe.

Page 76 - COMAL Today #25, 3501 Groveland Terrace, Madison, WI 53716
ORDER FORM
Subscriber #

Name: _________________________________
Street: _________________________________
City/St/Zip: ___________________________
Visa/MC#: ______________________________
Exp Date: _____________________________
Signature: ______________________________

Nov '89 - Prices subject to change without notice

TO ORDER:
- Fill in subscriber# / address (above)
- Check [x] each item you want to order
- Add up items/shipping/handling (fill in below)
- Send check/money order or charge it (above)
- Charge orders may call 608-222-4432
- or Mail to: COMAL Users Group, U.S.A., Ltd.
  5501 Groveland Terrace, Madison, WI 53716

SUBSCRIPTIONS:
Expired subscribers must renew before they may
order at subscriber prices (renewal starts with the
issue where you left off). New subscribers can
order at the same time as subscribing.

- How many issues? $4 per issue (max 5)
  (Canada/APO add $1 per issue, 1st Class)
- How many disks? $9 per disk (max 5) (C64)

$4.95 each for backissues; circle ones wanted:
1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20/21 22 23 24
$9.95 each, C64 COMAL Today Disks; circle to order:
1/2 3/4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20/21 22 23 24

Notice: All orders must be prepaid in US dollars.
Minimum order $10. Minimum shipping is $3 (does
not apply to subscription only orders); Canada,
APO & 1st Class add $1 more per newsletter and
book. Newsletter is published as time permits (no
set schedule); size and format varies. Cancelled
subscriptions receive no money back. Orders
accepted from Canada and USA only. Prices shown
are subscriber prices and reflect a $2 discount.
Allow 2 weeks for checks to clear. $15 charge for
checks not honored. Prepaid PO's from companies
accepted only if no conditions of sale apply.
Wisconsin residents add 5% sales tax and if your
county charges a county tax, include it as well.

ENTER TOTALS HERE:
Item Total ($10 minimum): $
Ship Total ($3 minimum): $
Wisconsin tax: $
Grand Total enclosed: $

BACKISSUES - COMAL Today

Price per issue: $4.95 (no extra shipping other than min)
(circle numbers on the left; matching disks are also available)
1 - Original first issues still available (not
reprints). Only a few copies left.
2 - PRINT FILE versus WRITE FILE;
Recreation; Stack overflow; Functions/Parameters
3 - sold out
4 - Graphics screen dump in COMAL;
spirolateral; Dodge 'em; Music; arrays /parameters
issues 1-4 Spiral bound: (see COMAL Yesterday/book)
5 - How COMAL statements are stored; strings;
Target; Inventory; Pitfall Harry
6 - Memory maps; 2.0 & 0.14 compared; Shadow
letters; Draw molecule; Function keys
7 - COMAL Standards meeting; Book reviews;
Function keys; External proc
8 - Sprites; Soundex; Forest; Sound; Fun print;
Recreation; Font sprites; Depreciation
9 - Modems; Ascii codes; TRAP; Function keys;
Metamorphose sprite; Bitmaps; Sizzle; C64 cart
internal structure & token table; Rod the
Roadman; Strings; Icon Editor
10 - Sorts; Font Editor; Draw; Walker; Phone
Database; Missing Letters; Designer; Compressed
bitmaps; Meta; Compare disk files; Easy Curves
11 - 2.0 keywords chart; Disk directories; Pop
Over; Graphics Editor System; Ram disk
12 - Cart package keywords chart; Rabbit; 3D
Fractals; Cart schematics; 2 column printing; Pic
Finder; Benchmarks; Free form database; Transfer
programs from 0.14 to 2.0; Kelly's Beach
Index to issues 1-12, 4,848 entries, 56 pp; (see books)
13 - Superchip keywords chart; Sprites; Wheel of
Fortune; Sets; Benchmarks; BASIC into COMAL;
Encryption; C128 package; Kastle
14 - Outliner; Listerine; Scope; Stacks; CASE
statement; Calendar; Multi-directory; Modems
15 - Over 60 0.14 Proc/Func listed!; Dr Who
database; Program construction; Wheel of Fortune
16 - Smart file reader; Text input window;
Magic squares; Easy instructions; Read & Run;
Learn subtraction; NIM; Tiny directory; Sorts
17 - COMAL Kernal in full; Ram expander;
Sprites; Calculate PI; 0.14 graphic/sprite chart
18 - Reversi; Poetry; Hammurabi; Puzzle; Zip
Zone; Fractal geometry / Mandelbrot; stacks
19 - Power Driver keywords chart; Sample book
pages; Coloring book; Rotating 3D
20/21 - Files; Black box; Best of 1-19
22 - Walking sprites; Animals; Maze; Music
23 - Message board; Programming; Graphing
24 - Common COMAL; Picture Package; Chaos
Disks are $9.95 each. Unless the disk label specifically states that you may give out copies, our disks may not be copied or placed into club disk libraries. Choose from the disks below:

[ ] Amiga Today Disk 25
[ ] Amiga Borge Christensen digitized pics
[ ] Amiga Text Readers
[ ] Amiga Text Editors

[ ] IBM Today Disk 25
[ ] IBM Special Series Disk #1
[ ] IBM Special Series Disk #2 (Test System)

[ ] CP/M COMAL Demo Disk ($5)

[ ] Beginning COMAL disk
[ ] Today INDEX disk (2 disks count as 1)
[ ] Games Disk #1 (0.14 & 2.0)
[ ] Modem Disk (0.14 & 2.0)
[ ] Article text files disk

Today Disks:
[ ] Today Disk (one disk type--circle choices):
  1 2 3 4 5 20 & 21 24 25
[ ] Today Disk (double sided -- circle choices):
  6 7 8 9 10 11 12 13 14 15 16 17 18 19 22 23

0.14 Disks:
[ ] NEW: Sid Siferlein disk set (3 disks count as 1)
[ ] Data Base Disk 0.14
[ ] NEW: Power Driver Tutorial Disk
[ ] Auto Run Demo Disk
[ ] Paradise Disk
[ ] Best of COMAL
[ ] Bricks Tutorial (2 disks count as 1)
[ ] Utility Disk 1
[ ] Slide Show disk (circle which): 1 2
[ ] Spanish COMAL
[ ] User Group 0.14 disks (circle numbers):
  1 2 3 4 5 6 7 8 9 10 12

2.0 Disks:
[ ] NEW: Robert Ross Technical Disk ($2)
[ ] NEW: Ajax School disk set (2 disks count as 1)
[ ] NEW: Norquay School disk
[ ] Data Base Disk 2.0
[ ] Superchip Programs disk
[ ] Read & Run
[ ] Math & Science
[ ] Typing disk (2 disks count as 1)
[ ] Cart Demo (circle which): 1 2 3 4
[ ] 2.0 user disks (circle choices): 11 13 14 15

$ = these disks assume you have the book

Note: Some disks may be supplied on the back side of another disk. Disk format is Commodore 1541 unless specified otherwise. We replace any defective disk at no charge if you return the disk with a note explaining what is wrong with it. Some disks are being reduplicated and appropriately relabeled.

$10.95 Sprite Pak
Two disk set. Huge collection of sprite images, sprite editors, viewers, and other sprite programs. For 0.14 and 2.0.

$12.95 Font Pak
Three disk set. Collection of many different character sets (fonts) for use with 0.14 and 2.0 including special font editors!

$14.95 Graphics Pak
Five disk set. Picture heaven. Includes Slide Show, Picture Compactor, Graphics Editor and lots of pictures (normal and compacted)

$29.95 Sprite, Font & Graphics Pak
All ten disks mentioned above!

$9.95 C128 CP/M Graphics
Graphics package on disk for use with CP/M COMAL on the C128. Includes turtle graphics and preliminary Font package.

$10.95 Guitar Disks
Three 0.14 disk set. Teaches guitar by playing songs while displaying the chords and words.

$14.95 Cart Demo Disks
Four disks full of programs demonstrating the many features of the C64 2.0 cartridge.

$10.95 Shareware Disks
Three disk set. Includes a full HazMat system (Hazardous Materials), an Expert System, Finger Print system, Traffic Calc, and a BBS program.

$14.95 Superchip On Disk
All the commands of Super Chip (but not the Auto Start feature) disk loaded.

$24.95 Super Chip Source Code
Full source code with minimal comments. Customize your own Super Chip. Add commands. Remove the ones you don’t need.

$14.95 2.0 User Group Disks
Four disks set for the C64 COMAL cart.

$29.95 0.14 User Group Disks
Twelve disks (User Group disk 9 is a newsletter on disk system, double disk).

$29.95 European Disk Set
Twelve 2.0 disk set. Find out what COMALites in Europe are doing.
**Systems Order Form**

**C64 - Disk Loaded**

- **Power Driver Complete** - all of the items below, Power Box, Starter Kit, and keyboard overlay. $49.95 + $6 shipping
- **Power Driver** interpreter and 20 lesson tutorial (with turtle tutor) on disk. $9.95
- **Power Box** - includes Power Driver interpreter and compiler, 3 utility disks, a toolbox of about 250 procedures and functions and Power Driver note pages. $29.95 + $4 ship
- **Starter Kit** - 12 issues of *COMAL Today*, 56 page index, 2 books and 5 disks! Over 1,000 pages! $29.95 + $4 ship
- **C64 Keyboard Overlay** - excellent condensed command reference. $3.95 + $1 shipping
- **Doc Box** binder/slipcase option ... $7.95

**C64 - Cartridge**

- **COMAL 2.0 Cart Complete** - The C64 cartridge plus all the options below (except Superchip) ... Tutorials, references, packages, applications ... $179.95 + $7 ship
- **COMAL 2.0 Cart Deluxe** - 64K cartridge; 3 reference books: 2.0 Keywords, Cart Graphics & Sound, Common COMAL Reference plus 4 demo disks. $124.95 + $3 ship
- **COMAL 2.0 Cart** - 64K cartridge with empty spot for up to 32K EPROM. Plain, no documentation. $99.95
- **Deluxe Option** - three books: 2.0 Keywords, Cart Graphics & Sound, Common COMAL Reference; 4 cart demo disks. $29.95 + $4 ship
- **Packages Option** - three books: COMAL Collage, 3 Programs In Detail, Graph Paper. $36.95 + $4 ship
- **Applications/Tutorial Option** - three books: COMAL Collage, 3 Programs In Detail, Graph Paper. $36.95 + $4 ship
- **Super Chip** - plug in chip for C64 cart; adds about 100 commands to the cartridge. $24.95
- **Doc Box** binder/slipcase option ... $7.95

**CP/M Systems** (includes C128 CP/M mode)

- **CP/M COMAL 2.0** - Full COMAL system disk plus the DEMO disk, packed in a Doc Box with manual. Works in C128 CP/M mode. $39.95 + $4 ship
- **Compiler Option** (RUNTIME system). $5.95
- **C128 Graphics Option** - Package disk $9.95
- **CP/M Package Guide Option** - Reference on how to write packages $14.95 + $1 ship
- **Common Comal Reference option** - $16.95 + $3 ship

**IBM PC & compatibles**

- **UniComal IBM PC COMAL 2.2** - Full fast system, with compiler, graphics, reference, tutorial. $165 +$6 ship
- **Option:** SCOM $145 (serial communication)
- **Option:** UniDump $45 (for laser printers)
- **Option:** UniMatrix $165 (matrix package)
- **Option:** UniMouse $45 (software mouse support)
- **Common Comal Reference option** - $18.95 + $3 ship

**Amiga (all models)**

- **AmigaCOMAL** - Full fast system, graphics, speech, exceptions, windows, screens, devices. $99.95 + $3 ship
- **Compiler/Developer Option** - $34.95 + $1 ship

**Amiga Boot Disks** (3 disk set)

All installed, ready to use, many added utilities (requires WShell purchase -- also you must send a copy of your WorkBench disk to us) $9.95
- **WSHELL** - $40 (required for Boot Disks)
- **AREXX** - $40 (optional for Boot Disks)
- **FACC II** - $28 (optional floppy speedup for boot disks)
- **Amiga Today Disk 25** - $9.95
- **Borge Christensen digitized pics disk** - $9.95
- **Text Editors disk** - $9.95
- **Text Readers disk** - $9.95

**COMAL Information Booklets**

<=How many? (20¢ each)

* = subject to customs/ship variations/availability
□ = while supplier last (out of print)
BOOKS ORDER FORM

[ ] $2.95 C64 COMAL 2.0 Keywords
#4 best seller for Feb 88 lists all the keywords built into the cartridge (including all 11 built-in packages) in alphabetic order complete with syntax and example. +$1 ship

[ ] $16.95 Beginning COMAL
#8 best seller by Borge Christensen
333 pages - General Textbook
Beginners text book, elementary school level, written by the founder of COMAL. This book is an easy reading text. You should find Borge has a good writing style, with a definite European flair. +$3 ship

[ ] $3.95 COMAL From A to Z
#1 all time best seller by Borge Christensen
64 pages - Mini 0.14 Reference book
Written by the founder of COMAL. +$1 ship

[ ] $3.95 COMAL Workbook
#1 best seller for Feb 88 by Gordon Shigley
69 pages - 0.14 Tutorial Workbook; Companion to the Tutorial Disk, great for beginners, full sized fill in the blank style. +$1 ship

[ ] Tutorial Disk Option: $9.95

[ ] $3.95 Index - COMAL Today 1-12
#9 best seller by Kevin Quiggle
52 page, 4,848 entry index to COMAL Today. The back issues of COMAL Today are a treasure trove of COMAL information and programs! This index is your key. +$1 ship

[ ] Index Disk Option: $9.95

[ ] $16.95 Common COMAL Reference
#3 best seller for Feb 88 by Len Lindsay
238 page detailed cross reference to the COMAL implementations in the USA (formerly COMAL Cross Reference) Covers: C64 COMAL 2.0, CI28 COMAL 2.0, CP/M COMAL 2.10, and UniComal IBM PC COMAL. +$3 ship

[ ] $19.95 COMAL Yesterday Issues 1-4 of COMAL Today - Spiral bound +$3 ship

[ ] $12.95 Library of Funcs/Procs
#1 best seller Dec 87 by Kevin Quiggle
80 pgs, over 100 0.14 procs/funcs, with disk. +$1 ship

[ ] $4.95 Cart Graphics & Sound
#6 best seller by Captain COMAL
64 pages - 2.0 packages reference guide to all the commands in the 11 built in cartridge packages. +$1 ship

[ ] $14.95 COMAL 2.0 Packages
#7 all time best seller by Jesse Knight
108 pgs with disk - package reference. How to write a package in Machine Code; includes C64 comsymb & supermon. For advanced users. +$2 ship

[ ] $14.95 Package Library Vol 1
compiled by David Stidolph
76 pages with disk - package collection
17 packages ready to use, many with source code, plus the Smooth Scroll Editor! +$1 ship

[ ] $14.95 Package Library Vol 2
67 pages with disk - package collection
24 example packages ready to use, most with source code, plus Disassembler, Re-Linker, De-Linker, Package Maker, Package Lister, and more. (includes windows). +$1 ship

[ ] $12.95 3 Programs in Detail
82 pages with disk by Doug Bittinger
Three 2.0 application programs explained: Blackbook (name/address system), Home Accountant, and BBS. +$2 ship

[ ] $12.95 Graph Paper
52 pages with disk by Garrett Hughes
Function graphing system for COMAL 2.0. The program can't be LISTed. Includes a version for the Commodore Mouse. +$1 ship

[ ] $12.95 COMAL Quick /Utility 2 & 3
#2 best seller Dec 87 by Jesse Knight
20 pgs with 2 disks, fast loading COMAL 0.14, printer programs, utility programs. +$1 ship

[ ] $14.95 CP/M COMAL Package Guide
The guide to making your own packages for CP/M COMAL by Richard Bain - 76 pages - advanced package reference. +$1 ship

[ ] $24.95 Intro to COMAL Advance Orders
Textbook for all COMALs by J William Leary and Len Lindsay. Approx. 300 pages. +$3 ship

[ ] Programs disk option: $5.95 Check format needed: ___ Amiga ___ IBM ___ C64 (double disk)

[ ] $16.95 Foundations with COMAL
sold out

db = Doc Box pages
+ = subject to custom/ship variations/availability
# = while supplies last (out of print)
AmigaCOMAL CONTROL Codes / CHR$(x) Results

<table>
<thead>
<tr>
<th>Ctrl char$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1 Erase to end of line</td>
</tr>
<tr>
<td>B</td>
<td>2 Erase to end of screen</td>
</tr>
<tr>
<td>C</td>
<td>3 Repeat find or change // Cursor ON</td>
</tr>
<tr>
<td>D</td>
<td>4 ?? // Cursor OFF</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>7 Flash Screen</td>
</tr>
<tr>
<td>H</td>
<td>8 Destructive backspace</td>
</tr>
<tr>
<td>I</td>
<td>9 Right tab (every 8th position)</td>
</tr>
<tr>
<td>J</td>
<td>10 Linefeed</td>
</tr>
<tr>
<td>K</td>
<td>11 Left tab (every 8th position)</td>
</tr>
<tr>
<td>L</td>
<td>12 Clearscreen &amp; home cursor</td>
</tr>
<tr>
<td>M</td>
<td>13 Carriage return</td>
</tr>
<tr>
<td>N</td>
<td>14 Insert line &amp; scroll rest of lines down</td>
</tr>
<tr>
<td>O</td>
<td>15 Delete line &amp; scroll rest of lines up</td>
</tr>
<tr>
<td>P</td>
<td>16 Reverse field toggle</td>
</tr>
<tr>
<td>Q</td>
<td>17 Extra half bright toggle</td>
</tr>
<tr>
<td>R</td>
<td>18 Underline toggle</td>
</tr>
<tr>
<td>S</td>
<td>19 Italic toggle</td>
</tr>
<tr>
<td>T</td>
<td>20 Bold toggle</td>
</tr>
<tr>
<td>U</td>
<td>21 Cursor to start of line</td>
</tr>
<tr>
<td>V</td>
<td>22 Cursor to end of line</td>
</tr>
<tr>
<td>W</td>
<td>23 Cursor to top of window</td>
</tr>
<tr>
<td>X</td>
<td>24 Cursor to bottom of window</td>
</tr>
<tr>
<td>Y</td>
<td>25 Delete character</td>
</tr>
<tr>
<td>Z</td>
<td>26 Insert character</td>
</tr>
</tbody>
</table>

Pen Color

Choose one of 32 colors by printing a chr$(x) where x is a value between 128-159 (hex $80-$9F).

Example:

```
PRINT chr$(80+3) // sets color to 3
```

Print a chr$(17) before and after the color code to shift to the next 32 colors in Extra Half Bright mode (some early Amiga models may not have this mode).

Example:

```
PRINT chr$(17),chr$(130),chr$(17) //color to 35
```

Background Color

Print a chr$(16) before and after the color code to set the background to the specified color.

Example:

```
PRINT chr$(16),chr$(130),chr$(16)//background 3
```

Bit Planes

You must have installed AmigaCOMAL with enough bit planes for the screen to see all the colors. It takes 6 bit planes to display all 64 colors in Extra Half Bright mode.

How to Enter Programs

Line numbers are required for your benefit in editing a program (but are irrelevant to a running program). We usually omit line numbers when listing a program. It's up to YOU to provide line numbers. Of course, COMAL can do it for you. Follow these steps to enter a COMAL program:

1) Enter command: NEW

2) Enter command: AUTO

3) Type in the program.

4) When done, stop the AUTO mode:

Power Driver: Hit "return" on blank line
PET COMAL 0.14: Hit "return" on blank line
C64 2.0 Cart: Hit "stop" key
C128 2.0 Cart: Hit "stop" key
CP/M COMAL 2.10: Hit "esc" key
AmigaCOMAL: Hit "esc" key
IBM PC COMAL: "control" + "C"

You may use both UPPER and lower case letters. COMAL automatically makes keywords UPPER case and variable names lower case.

You don't have to type leading spaces in a line. They are listed only to emphasize structures, and COMAL will insert them for you. You DO have to type a space between keywords in the program.

Long program lines: If a complete program line will not fit on one line in our listing, we continue it on the next line and add //wrapline at the end. You must type it as one continuous line.

Variable names, procedure names, and function names can be a combination of:

```
abcdefghijklmnopqrstuvwxyz0123456789+-_`
```

On the C64/128, the «left arrow» key (upper left corner of keyboard) is valid. COMAL 2.0 converts it into an underline. A «British pound» £ symbol is used in place of the \ backslash.