Sorcerer Users' Group (Toronto) Newsletter
200 Balsam Ave., Toronto, Ont., M4E 3C3
Volume 1 Number 7
September 1980

EDITOR'S TURN

The first thing I'd like to mention this month is that the Sorcerer's User's Group meeting for September has been cancelled. Unfortunately, I am unable to hold the meeting at my house this month because we are doing extensive renovations and re-decorating. Duncan has applied to the Board of Education again this year in an effort to acquire a meeting place but we have not had any replies to date. Hopefully, we will have everything arranged by the next meeting and will get back to you as quickly as possible.

Last month I left you all hanging on the edge of your seats wondering what in the world I was talking about. I'm sure I have wetted your cerebral tastebuds as to what this 'C' language is all about and what it stands for. Yes, 'C' is the first letter of the word computer and it sounds like a large body of water but it is neither. Read on and I'm sure you will 'C' what I'm talking about (all puns intended).

'C' is a very powerful and flexible, top-down, structured programming language which has a very rich set of expression operators, almost no limitations or restrictions as to style and programming capabilities (apart from the limitations of the processor you are using), and a set of functions which is limited only by your imagination. You are not restricted to a pre-defined set of functions as you are with BASIC, FORTRAN, COBOL and in many ways, even PASCAL, or most interpreters and compilers for that matter. You are not held back by some programme designer's lack of insight or thoughtlessness for not including certain features in a language because he doesn't need them. You see, with 'C' you write whatever you need yourself, from the most primitive processor related functions to the most sophisticated routines whatever they may be. You write them the way you want them to function, to do exactly what you require, the way you want them done. Except for the very basic primitives, a wealth of which are almost always included already with any 'C' package, are written in 'C'. All the routines that you write are automatically reduced by the compiler to a set of independant, relocatable modules which can be incorporated into any programme you write from thereon simply by calling its name as a subroutine and passing the correct 'type' of arguments (more about types in a minute). Once written, it will never have to be programmed again. It is possible to eventually write complex routines as easily as the following:

```
night_time( )
{
     stove (off);
     doors (locked);
     porch_light (on);
     house_lights (off);
     clock (set_alarm, 7.00);

     while (time < 7.00)  monitor (alarms, fire, burglar);

     kettle (on);
     doors (unlock);
     porch_light (off);
     house_lights (on);

     if (time > 7.05)  clock (stop_alarm);
     while (time < 7.15)  wait( );
     ring (alarms, fire, burglar);
     tip (master_bed);
}
```

This programme, as you can probably see, will, at the press of a button, secure the house for you at night time as you go to bed. It will check all the alarms all night long as well. At 7:00 am it will ring the clock, open the house back up, turn on the lights and set the kettle boiling. If you are not up by 7:15, look out. The programme assumes of course, that you have all the necessary hardware controlling devices hooked up to the system.

'C' has a good set of internal control operators which include such commands as:

while...., do...while, if...., if...else...if, for...., goto, switch...CASE...default, return...., break, continue, etc.

It also has a very specific set of variable types which allow maximum efficiency of memory requirements and speed while being easy to programme. They include types such as:

int, char, float, double, long, short, unsigned, auto, extern, register and static.

Types is a major subject category of 'C' and I don't want to get too deep into this here, but, basically, a variable (string or numeric) can be short or long int or double integer or floating point. They can appear as signed numbers or unsigned numbers. Automatic variables allow a variable to appear temporarily for a particular routine and disappear upon exit (more on this later). Variables can also be declared as external which makes that a global variable to the entire programme. Register variables increase speed and throughput by keeping that variable in the processors registers as much as possible (handy if a particular variable is going to be heavily used).

'C' will also allow you to predefine and manage complex storage allocators (super variables) called structures and unions. The beauty of these declarations is that the compiler (and ultimately the programme) will keep track

of these types and work properly with them. If you increment memory pointers, let's say by 1 here, then the programme increments by the proper type. That is, if you increment an integer, the programme knows to increment the pointers by 2. If you increment a double, then the pointers are incremented by four and if you increment a structure that is fifty-seven bytes long then the pointers will automatically be incremented by fifty-seven. Given a structure that has the form:

```
struct record
{
      char last_name[16];
      char first_name[16];
      char address[48];
      int  age;
      int  height;
      double    birthdate;
      double    weight;
} person[20];
```

the programme would know that the information record of person 2 would be exactly 92 bytes away from that of person 1. To obtain any datum in a particular structure you simply point to the item. For example, if I wanted to assign my personal information to the record and I am #5 in the list, it would look like:

```
person[5]->last_name = "Row";
person[5]->first_name = "John";
person[5]->address = "200 Balsam Ave, Toronto, Ont.";
person[5]->age = 28;
person[5]->height = 5.8;
person[5]->birthdate = 100252;
person[5]->weight = (none of your business)
```

Of course, you would write a programme so that this information would be entered from the keyboard or some other input device and not from programme constants.

'C' also keeps track of global variables and local variables. That is, you can use a variable globally which allows it to be altered by any portion of the programme or locally, which locks out any programme module from changing that variable unless it is assigned to that module or its address is given to it specifically. A variable can be so local that it is only seen by a single expression within a module. This means that if you discover that a single WHILE or FOR loop needs a counter which is not required elsewhere, then you can just open a variable by any name (as long as it is not a global name) and use it even if that name is used as another local variable elsewhere, without worry. Good riddance to the BASIC problem of trying to dream up new variable names that I haven't used elsewhere in the programme without making it unreadable. You know, names like Z79A3J.

In 'C' a variable or function name can be any length but only the first eight characters are significant. This allows you to write programmes that are so easily readable, they are almost self documenting.

'C' is fully recursive to any depth.

'C' supports a multitude of processor functions as well as text definitions and substitutions, n-level nested source file inclusion, buffered I/O, etc, etc.

Another major feature of 'C' is that wherever you can have a simple statement or expression, you can have a complex set of statements or expressions by using the braces {...} which reduces it, as far as the programme and the compiler are concerned, to a simple expression.

When you get 'C', it usually comes with a large set of library commands already made for your system to perform extensive console I/O, disk file I/O, buffered I/O, character diddling, etc, etc. If you don't like the way certain functions are handled, you just adjust them to your liking.

'C' has two styles, an expanded style and an abbreviated style. In the expanded style, the programmes are easy to read and to follow. In the abbreviated style, they are slightly more cryptic but operate a little faster and take up a little less space. When you have some experience with 'C', you will find the abbreviated style as easy to read as the expanded style.

'C' puts out 8080 machine language code programmes (if you are using an 8080 or Z80 processor) and it does not require an interpreter. Thus, it runs much faster than any interpreter and even faster than most compilers. Because of its design, it is easier to read than BASIC, FORTRAN, COBOL and APL. Because of its top-down structure, it really is much easier to write programmes in 'C' than it is in BASIC, FORTRAN, COBOL or APL.

The statement I really want to make in this editorial apart from the review of the 'C' language, however, is that the language you choose to use on your system depends largely on your needs. Some languages are more suited (and dedicated) for some purposes than they are for others. If the language you are using feels comfortable to you and does everything you need, then there is very little reason to change. On the other hand, if you presently own 9 interpreters, 2 pre-compilers, 3 compilers, 6 debuggers and a host of assemblers and you are still looking around for more, then none of them will be of any use to you because you can't possibly learn enough and become experienced enough with any one of them to really write the programmes you want. Even the best programmers can't understand and programme well in more than 2 or 3 languages. The idea is to assess what your needs are as best you can, select the language that best fills your needs and stick with it until you can sit down, quickly code up some small programmes, enter them and run them with no errors. Until you reach that point, 90% of your programming effort is spent trying to understand the language and only 10% is actually spent on the logic of your programme.

Having assessed my needs, machine independant flexibility for experimental programming, faster speed than interpreters can provide, independant programmes, that is, programmes that do not require interpreters or run-time modules that has all the RAM and require the user to have the same model and version number, I discovered that 'C' fills all my needs rather well. I am very pleased.

Due to the length of this editorial, there is no SOFTWARE column this month.

GENERAL NEWS

-as you probably all know already, EXIDY has sold their data products division, namely, anything to do with SORCERER and microcomputers, to a company called RAMOTEK. The new company supposedly is heavy into graphics and controls and are rumoured to manufacture many peripheral controlling devices. It is also rumoured that they are ready to invest large quantities of the big American green stuff. Let's hope for the best and see what comes out in the next 6-8 months.

ADVERTISEMENTS

FOR SALE - IDS-440 Paper Tiger with 2K buffer and graphics options. Asking $1,250.00 or best reasonable offer. Call Tony Bagshaw at 881-1532 after 6 pm.