SQLUG

SCOTTISH QL USERS GROUP

December 2000

NUMBER 128

THE SCOTTISH QL USERS GROUP NEWSLETTER

It came as something of a shock to realise that this is the last newsletter of the twentieth century. The media have been remarkably quiet about that compared to all the fuss we had to put up with last year. So this one is a collector's item, and for that so are the other 127, with pearls of wisdom scattered throughout.
I have had a verbal report of QL 2000 from John but so far have not received the November QUANTA so have not seen anything in print. I gather the meeting on our future went much as might have been expected, with everybody wanting things to go on much as they are but without any clear idea how this may be achieved.
You need to be careful what you say to the QUANTA editor. I wrote to him about the quirks of Outlook Express with one or two pleasantries and I find I have top billing in QUANTA as "News from Scotland".

LAST MEETING AT THORNHILL COMMUNITY HALL SUNDAY 12TH NOVEMBER

We were able to get a glimpse of the machine Giles is putting together for George at this meeting. It consists of an Aurora, Qubide and Super Gold Card, with hard disk and two HD floppies, all put together in a Thor case. It looks very neat and professional, as well as being impressive. All the interfaces are fixed into the back plate. It hasn't been passed as finished by Giles though. There is something he is not satisfied with but I~m not sure what it is. My new super duper boxed kit distinguished itself by regularly crashing. It wasn't until I took the cladding off the case that it behaved with any reliability. Either it was over heating or something was being distorted. Back to the drawing board. George was playing with his sprite generation program in preparation for writing an article and manual about it. He showed us how to put in data to get the sort of sprite you want.

This month we have an article by John Sadler which I think is part of a series. He hasn't actually promised more than one but does mention a sequel during the course of the article.

PROGRAMMING TECHNIQUES, MODERN LANGUAGES
JOHN SADLER

If you have been watching BBC2 programmes on computing you will know they have been laying emphasis on object orienting languages such as C++, scheme or Java. These languages group the data types and the functions together into classes. In the same way that you access structures in C by declaring an instance of the structure with a name and you only access the contents through the instance name, in C++ you can only access the data through functions using the instance name. This has the tremendous advantage that the same variable name is not so easily used for different instances of that data type, so eliminating a common bug in

large programs. Furthermore as the data is only accessed by functions, the implementation of the data type can easily be changed without having to change lines throughout the suite of files of the program. Classes can also inherit properties from other classes so enabling one to build up a series of complicated classes from their component classes.
Object orientated languages usually allow overloading, that is, where different versions of a function are allowed which carry out basically the same operation, but the various versions of the function are differentiated by the arguments passed to that function. They also allow the creation of polymorphic classes, that is, classes which can have different forms. Perhaps you will have noticed that in Superbasic or C all variables, arrays etc. have to be of one type and hence it is very difficult to write functions which can be applied to different types such as integers and characters. Most object orientated languages except C++ have garbage collection. This means that unlike C you do not allocate memory for data and the executable program automatically deallocates the memory of the objects not now used by the program when it runs short of space. Hence a programmer does not have to concern himself with memory leaks (that is the program does not keep on consuming more and more memory as it runs.)
There is another group of modern languages called functional languages. These languages allow the creation of user data types rather similar to ‘typedef and structures in C and the data is only changed by functions which can be polymorphic. This means that you cannot use assignments like "x = x + 3", you have to create a function plus three or create a function plus one and apply it thrice. They use lists instead of arrays and have a lot of built in list processing functions. They also have garbage collection so there is no allocation of memory. This makes their programs particularly easy to follow in principle. It is easy to understand what the program is doing, although at the minute I find them difficult to write. The other great advantage is that it is relatively easy to prove that they actually do what is intended and in fact the language is used for writing programs for checking proofs.
So what is the point of all this? We can try and adopt some of these ideas to make our programs clearer and less likely to contain errors. In C we can create a separate file for each data type or structure and the functions for accessing and acting on the data. We can build more complicated data types from simpler ones and so inherit their properties. We can ensure that we only assign, alter or retrieve data using the associated functions and hence avoid unwanted side effects. By these means our programs should be clearer and more easily understood and hence easier to write, debug and amend. However the disadvantage of these techniques is that because of the number of nested function calls, these programs run slower, but this is outweighed by the ease of writing more easily understood reliable programs. In a following article the application of these techniques will be attempted with the implementation of strongly typed objects. which are only created, and accessed by functions so ensuring that the data cannot be accidentally corrupted. These objects could be used to implement polymorphic lists, trees and other data structures, so that there is no need to write a different version for type of data.

NEXT MEETING
THORNHILL COMMUNITY HALL, SUNDAY 10th DECEMBER
I 1.30am - 4.30pm

Come to the last meeting of the twentieth century, although don’t worry if you can’t make it as we intend to go on meeting into the twenty first century. Our meeting is on the usual second Sunday of the month, and held in our new venue. I have not published a map for some time, but by now I hardly think it necessary. Any one who has not attended a meeting at Thomhill can contact me beforehand for directions if you can’t find your early copies of SQLUG.