261: 8th November

This week, we will take a bit of a break from numerical analysis and just concentrate on getting our 'C' programming skills up to scratch.

The '.c' File

When we write a 'C' program, it should be put in a file that has '.c' at the end of its name. Some of the editors have extra features that are invoked when they open '.c' files. For example, xemacs, can colour different parts of the program differently, depending on what they do, but it needs to know that that it is looking at a 'C' program, before it can do this.

Once we have opened a '.c' file, the next thing to do is to create a 'main' subroutine. When you run a program, it always starts from the beginning of the 'main' subroutine. For this reason, the computer expects every program to contain one. Without 'main', the computer wouldn't know where to start.

Typically the 'main' subroutine looks like this:-

```
int main() {

return 0;
}
```

The gap in between the line beginning 'int' and the line beginning 'return', is where you put all your commands.

Variables

A variable is like a storage box. You give the box a label, which is the name of the variable, and you put a value inside it. The value can be a number or a string of letters and you can change the value during the running of the program. However, whenever you refer to the label in the program, the computer interprets it as meaning the contents of the storge box. So, if the variable was called 'var1' and it contained a number and we included a command to increase 'var1' by 1, then it would infact, increase the number contained in 'var1'.

Unfortunately, you can't change the type of expression that you store in a variable, in the middle of the program. Each variable can only contain either a string of characters, or an integer number or a floating point number or an extra precise floating point number. In fact, the type of each variable has to be declared at the beginning of the subroutine in which it appears.

As an example, we will add four variables to our 'main' subroutine. The first, 'var1', is an integer, the second, 'var2', is a floating point number, the third, 'var3', is a floating point number with more significant figures than 'var2' and the fourth, 'var4', is a string, which can contain alpha-numerical characters.

```
int main() {
   int var1;
   float var2;
   double var3;
   char *var4;

   return 0;
}
```

Standard Commands

You can see that in the above piece of code, some of the lines end with a semicolon. Generally, commands that instruct the computer to do something must end with a semicolon, whilst commands that just provide structure, like the first and last line, don't need them.

Some other common commands are 'printf', 'for' and 'if'.
'printf' is the command to print on the screen. It is used in the form

```
printf(``\nThe number is %d\n'', i);
```

Effectively, the computer will print everything between the quote marks on the screen. However, the combination of characters '\ n' is interpretted as the command to move onto a new line. The combination '%d' means 'insert here the value of the variable after the comma'. If 'i' is an integer than '%d' must be used. If 'i' is a float, then '%e' or '%f' is appropriate. For a string, '%s' is used.

It is possible to print several variables in the same 'printf' command. For example, the command,

```
printf(``\nThe 1st number is %d, 2nd is %f and 3rd is %d\n'', i, j, v);
```

would print 'i', 'j' and 'v' in place of '%*d*', '%f' and '%d', respectively, where 'i' and 'v' are integers and 'j' is a float.

'for' is a command that can be used to loop over a piece of code. It is used as follows

```
for(i=0;i<10;i=i+1) {

}
```

If 'i' is declared as an integer, then this piece of code would set 'i' to zero and then execute the code between the curly brackets. Once done, it would then set 'i' to 'i'+1, which is what the third statement, in the normal brackets, means and execute the code in the curly brackets again. It would stop when 'i' is no longer less than 10, which would be after the tenth loop through the code.

The 'if' command is a way of deciding which set of instructions to execute. It's used as follows

```
if(a<b) {

}
```

If 'a' is less than 'b', then the commands between the curly brackets will be executed, otherwise, they'll be ignored.

We can add an 'else' statement to the 'if' command as follows

```
if(a<b){

}
else {

}
```

Now, if 'a' is less than 'b' the commands between the first set of curly brackets is executed, but if 'a' is not less than 'b' the commands between the second set are executed.

Subroutines

A subroutine is a way of dividing the program up into chunks of code, where each chunk performs a specific function. Every time you want to use the chunk of code, you just need to refer to the name of the subroutine.

A subroutine called "largest_num" would look like this

```
float largest_num(float a, float b) {
   float lg;
   if(a<b){
      lg = b;
   }
   else {
      lg = a;
   }
   return lg;
}
```

This subroutine takes two floating point numbers, compares then and returns whichever is the largest. To use it in the 'main' subroutine, to get the largest of two floats, 'g' and 'h', we would have to include the following line.

```
y=largest_num(g, h);
```

and y would then take the value of whichever was largest.

Compiling

To convert your '.c' file into a program that you can run, you need to compile it. However, before we can do this, we've got to clean up the file and put it into a format that the compiler will find acceptable.

Firstly, we need to include our 'function prototypes'. A function prototype is just a copy of the first line of a subroutine. If you put your subroutines after the 'main' subroutine, then you need to copy the first line and put it in a space before the 'main' subroutine. This may sound strange, but when the program is compiled, the compiler goes through the file once, from top to bottom, line by line and it's not happy about compiling commands in 'main' that use subroutines that it hasn't seen yet.

Many of the commands that we regularly use, like 'printf', are infact subroutines in their own right and their prototypes are kept in a file called 'stdio.h'. To include their prototypes, you must include the command

```
#include<stdio.h>
```

at the beginning of the file. Usually this is the first line in a '.c' file.

Finally, to compile the file, save it, go to the unix command line and type

```
gcc filename.c
```

'gcc' is the name of the compiler and it will produce a file called a.out, which is your program. You can run it by simply typing it's name (unless you're logged in on the Astro server, in which case, you'll need to type'./a.out').

Disclaimer

These notes are only intended to refresh your memory. They aren't a comprehensive lesson in 'C' programming. If any of the material here is unfamiliar, then I would recommend spending a few days having a look at one of the following textbooks and trying a few of the examples that they provide. The assessments that you will be faced with, later in the course will require a good knowledge of everything here, plus more.

A Book On C by Kelley and Pohl. Addison-Wesley Publishing. ISBN 0-8053-1677-9.

The C Programming Language by Kernighan and Ritchie. Prentice Hall. ISBN 0-13-110362-8.

The C Book by Banahan, Brady and Doran. Addison-Wesley. ISBN 0-201-54433-4.

Programming Using The C Language by Hutchison and Just. McGraw-Hill Company. ISBN 0-07-031541-8.

A First Book Of C by Bronson and Menconi. West Publishing Company. ISBN 0-314-81348-9.

Programming in ANSI C by Kochan. Hayden Books. ISBN 0-672-48408-0.

C How To Program by Deitel and Deitel. Prentice Hall. ISBN 0-13-089572-5.

Questions

Write down, on a page of A4, a program that creates nine variables to act as matrix elements. Assign to them the elements of the matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

and get the program to calculate the determinent of the matrix, before printing it out on the screen.

Write a second program that adds up all the integer numbers between 0 and 1000000 and prints the answer on the screen.