First Year Computing (C++ Course)

Dr John Joss Chinn

Week 5.

Functions, Introduction to Gaussian Elimination

Functions

The idea of functions, in C and C++, is to break the source code up into manageable portions. This hasn't really been a problem for us so far for we have only used quite small programs. However, in practice, many commercial C++ programs run to hundreds or even thousands of lines of code. Breaking it up in into functions it allows one to deal with individual portions of the code. This makes it much easier to both write the code initially and for testing for errors in the code ("debugging"). Functions are the equivalent to subroutines in BASIC and FORTRAN.

There are different types of functions. Their general form is as follows

```
return_variable_type function_name(sent_variable_types)
```

This is going to be clearer by looking at some examples. The first program below, `func1.cpp` , has the 'function declaration' `void func1(void)`. This means that the function is not sent a variable to work on, hence its argument is declared `void`, and it does not return anything, hence it is itself declared `void`, in other words its return variable type is `void`.

```
//func1.cpp
#include<iostream>
#include<conio.h>

void func1(void);     //function declaration

void main()
{
     func1();   //call to the function
     getch();
}

void func1(void)//the function
{
     cout<<"Hello World!\n";
}
```

If the argument of a function is void then you don't really need to mention it at all. Take a look at this very silly example, below. This has two functions and hence two function declarations of the form

```
void func1();
```

The program still only prints `Hello World!` To the screen.

```
//func2.cpp
#include<iostream>
#include<conio.h>

void func1();   //function declaration
void func2();   //function declaration

void main()
{
     func1();   //call to the function
     func2();   //call to the function
     getch();
}

void func1()    //the function
{
     cout<<"Hello ";
}

void func2()    //the function
{
     cout<<"World!\n";
}
```

The next program, `//func3.cpp`, does not return anything to the main function, hence it is declared void. However it is sent a real value by the main function, hence its argument is declared `double`.

```
//func3.cpp
#include<iostream>
#include<conio.h>
#include<math.h>

void radianz(double x);     /*function does not return a value
                              hence it is declared 'void'*/
void main()
{
   double a;

   cout<<"Type an angle in degrees"<<endl;
   cin>>a;
   radianz(a);             //call to function
   getch();
}

void radianz(double x)           //function
{
   double PI=3.14159;
   cout<<x<<" degrees is "<< PI*x/180 <<" radians"<<endl;
}
```

```
Type an angle in degrees
20
20 degrees is 0.349066 radians
```

The following function, `func4.cpp`, is both sent a value, from the `main` function, and returns a value to the `main` function. Notice that the `main` function uses the variables a and b, whereas the function `radianz` uses the variables x and y. This really shows that it is unimportant what you call the variables as long as they are of the same type (`float` or `int` or whatever). You could equally well use x and y in `main` as well as x and y in `radianz`. However the compiler 'sees' them as completely different variables.

```
//func4.cpp
#include<iostream>
#include<conio.h>
#include<math.h>

double radianz(double x);     //function declaration

void main()
{
   double a,b;

   cout<<"Type an angle in degrees"<<endl;
   cin>>a;
   b=radianz(a);           //call to function
   cout<<a<<" degrees is "<< b <<" radians"<<endl;
   getch();
}

double radianz(double x)         //function
{
      double y;
      double PI=3.14159;

      y=PI*x/180;
      return y;
}
```

This function has exactly the same output as `func3.cpp`, above. Because the variables a and b, and y, are declared within the functions `main` and `radianz`, respectively, they are known as *local* variables. If the main function included a reference to y, for instance, `cout<<y<<endl;` then the compiler would show an error `"'y' undeclared identifier"`, or similar. Conversely, if the function `radianz` contained a reference to a or b, then the compiler would again show this error.

The program below, `func5.cpp`, uses the same variable name, v, in both of the functions, `here` and `there`, called from `main`. The point here is that even though both functions use the variable v, neither function 'knows' that the other function is using this variable because the variable is declared *locally*, in each case.

```
//func5.cpp
#include<iostream>
#include<conio.h>
```

```
void here(void);      //function declaration
void there(void);     //function declaration

/*these functions RETURN nothing hence they are declared as
'void'
  they are SENT nothing, hence their arguments are also
'void'*/

void main()
{
   here();            //call to a function
   there();           //call to a function
   getch();
}

void here(void)       //function #1
{
   int v;             //local variable;
   v=6*5;
   cout<<"The value of v here is "<< v <<endl;
}

void there(void)      //function #2
{
   int v;             //another local variable;
   v+=5;              //add 5 to whatever v is
 /* as v has not been initialised then it will
take on any arbitrary number, plus 5 */

   cout<<"The value of v here is "<< v <<endl;
}
```

```
The value of v here is 30
The value of v here is -858993455
```

This brings us to the question of *global* variables. Global variables may be seen by any function within the program. It is best to avoid using them if possible as they may inadvertently be altered by some part of the program where you wouldn't want them altered. However `func6.cpp` below uses the idea of global variables.

```
//func6.cpp
#include<iostream>
#include<conio.h>
#include<math.h>

void radianz(double a);     //function declaration
double y;                  //global variable

void main()
{
   double x;
```

```
   cout<<"Type an angle in degrees"<<endl;
   cin>>x;
   radianz(x);             //call to function

   cout<<"here's y again "<<y<<endl;
   getch();
}

void radianz(double x)          //function
{
      double PI=3.14159;
      y=PI*x/180;
      cout<<x<<" degrees is "<< y <<" radians"<<endl;
}
```

Notice here that y is declared as a global variable, i.e. outside of either the `main` or the function `radianz`. Therefore *both* the main and the sub function can 'see' y. Typical use of this program is shown below.

```
Type an angle in degrees
30
30 degrees is 0.523598 radians
here's y again 0.523598
```

Using functions is good programming practice as it really does help you to design your program and very much helps you to debug it. Ideally the main function should just be the 'boss' program and get all the other functions to do the work.

Introduction to Gaussian Elimination with Partial Pivoting

Continuing with our work toward the assignment we will look a little more deeply into solving a system of simultaneous equations using the method of Gaussian Elimination with Partial Pivoting. This week we will try the method by hand. Next week we will think through how to write a C++ program to perform all of the necessary operations. Consider the system of equations below

$$\begin{aligned} 2x_1 + 8x_2 - 25x_3 &= 472 \\ 5x_1 - 20x_2 + x_3 &= -156 \\ 10x_1 + 2x_2 + 5x_3 &= -24 \end{aligned}$$

The augmented matrix for this set of equations is the following

$$\begin{pmatrix} 2 & 8 & -25 & 472 \\ 5 & -20 & 1 & -156 \\ 10 & 2 & 5 & -24 \end{pmatrix} = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{pmatrix}$$

The solution of these equations is $x_1 = 4$, $x_2 = 8$ and $x_3 = -16$. We therefore want to end up with a matrix which looks like the following

$$\begin{pmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & -16 \end{pmatrix}.$$

We assume, for the moment, that we do not know this solution and need to solve for it. The first thing that we do is find the row with the largest element in column 0, this is the first 'pivot'. This is row 2, and the pivot is 10. We then interchange the rows 0 and 2:-

$$\begin{pmatrix} 2 & 8 & -25 & 472 \\ 5 & -20 & 1 & -156 \\ 10 & 2 & 5 & -24 \end{pmatrix} \longrightarrow \begin{pmatrix} 10 & 2 & 5 & -24 \\ 5 & -20 & 1 & -156 \\ 2 & 8 & -25 & 472 \end{pmatrix}$$

Next we divide the whole of row 0 by the pivot to get

$$\begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 5 & -20 & 1 & -156 \\ 2 & 8 & -25 & 472 \end{pmatrix}$$

In this way we always divide a small number by a larger number. This reduces error. We can now use this new row 0 to eliminate $a_{1,0}$ and $a_{2,0}$ from the column 0. Multiply the whole of row 0 by $-a_{1,0}$, i.e. 5, to get

$$\begin{pmatrix} -5 & 1 & -2.5 & 12 \\ 5 & -20 & 1 & -156 \\ 2 & 8 & -25 & 472 \end{pmatrix}$$

Now add the new row 0 to row 1, also reinstate row 0. This gives

$$\begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & -21 & -1.5 & -144 \\ 2 & 8 & -25 & 472 \end{pmatrix}$$

Next, multiply row 0 by $-a_{0,2}$, i.e. 2, and add this to row 2 to get

$$\begin{pmatrix} -2 & -0.4 & -1 & 4.8 \\ 0 & -21 & -1.5 & -144 \\ 2 & 8 & -25 & 472 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & -21 & -1.5 & -144 \\ 0 & 7.6 & -26 & 476.8 \end{pmatrix}$$

We have now completed column 0.

We can now leave row 0 and column 0 alone and, in effect, we work on the ‘reduced matrix’:-

$$\begin{pmatrix} -21 & -1.5 & -144 \\ 7.6 & -26 & 476.8 \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{pmatrix}$$

However for clarity, and because I want to work in a similar way to the computer, I will leave in row 0 and column 0.

$$\begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & -21 & -1.5 & -144 \\ 0 & 7.6 & -26 & 476.8 \end{pmatrix}$$

The next thing to do is to find the pivot element in column 1. Well the ‘largest’ element is $a_{1,1}$ = -21. This is already in the right row. We can then divide row 1 by –21 to give

$$\begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & 1 & 0.0714 & 6.8571 \\ 0 & 7.6 & -26 & 476.8 \end{pmatrix}$$

Now we can use row 1 to eliminate $a_{2,1}$, in the bottom row. We multiply row 1 through by $–a_{2,1}$ = -7.6 and add this to row 2 to get

$$\begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & -7.6 & -0.5428 & -52.1143 \\ 0 & 7.6 & -26 & 476.8 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & 1 & 0.0714 & 6.8571 \\ 0 & 0 & -26.5428 & 424.6857 \end{pmatrix}$$

This completes column 1.

Now we have the further reduced matrix

$$\begin{pmatrix} 0.0714 & 6.8571 \\ -26.5428 & 424.6857 \end{pmatrix} = \begin{pmatrix} a_{1,2} & a_{1,3} \\ a_{2,2} & a_{2,3} \end{pmatrix}.$$

This time the pivotal element is on the bottom, the only one that is left. This is $a_{2,2}$ = -26.5428. We therefore divide row 2 by $a_{2,2}$ to get

$$\begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & 1 & 0.0714 & 6.8571 \\ 0 & 0 & 1 & -16 \end{pmatrix} = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{pmatrix}$$

This has completed the first part of the problem. We will need some C++ code to do this. The matrix is now in ‘upper tri-diagonal’ form

The next part of the operation would require another, different piece, of C++ coding. This is called ‘back substitution’. We want to eliminate elements $a_{1,2}$ and also $a_{0,1}$ and $a_{0,2}$. We begin by multiplying row 2 by $-a_{1,2}$, i.e. –0.714, and then adding this to row 1 to get

$$\begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & 1 & 0.0714 & 6.8571 \\ 0 & 0 & -0.0714 & 1.1428 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & -16 \end{pmatrix}$$

Next we multiply row 2 by $-a_{0,2}$, i.e. –0.5, and add this to row 0 to get

$$\begin{pmatrix} 1 & 0.2 & 0.5 & -2.4 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & -0.5 & 8 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 0.2 & 0 & 5.6 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & -16 \end{pmatrix}.$$

This completes the back substitution for column 2.

Next we need to eliminate element $a_{0,1}$. We multiply row 1 by $-a_{0,1}$ and add this to row 0 to get

$$\begin{pmatrix} 1 & 0.2 & 0 & 5.6 \\ 0 & -0.2 & 0 & -1.6 \\ 0 & 0 & 1 & -16 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & -16 \end{pmatrix}.$$

This is then the desired result.

Tutorial

(1) Type in the programs, containing functions, from this weeks handout. Be sure that you understand how they work. Ask the tutors if you need help in understanding.

(2) Fill in the code in the functions `double func(double x_old)` and `double deriv(double x_old)` in the program below. Note that this is adapted from `newton1.cpp` from last week.

```
//newton2.cpp
#include<iostream>
#include<conio.h>
#include<math.h>

double func(double x_old);
double deriv(double x_old);

void main()
{
      double error=1e-6;
      double x_old=1e6;   //must initialise it
      double x_new=1;

      while(fabs(x_new-x_old)>error)
      {
            x_old=x_new;
            x_new=x_old-func(x_old)/deriv(x_old);
            cout<<"x_old\t"<<x_old<<"\tx_new\t"<<x_new<<endl;
      }

      cout<<x_new<<endl;
      getch();
}

double func(double x_old)
{
      fill here
}

double deriv(double x_old)
{
      fill here
}
```

(3) Solve the following set of simultaneous equations by Gaussian elimination with partial pivoting, by hand

$$\begin{aligned} x_1 - 5x_2 - 2x_3 &= -27 \\ -2x_1 + 4x_2 + x_3 &= 18 \\ 10x_1 + 2x_2 + 8x_3 &= 34 \end{aligned}$$

(answer:- $x_1 = 5$, $x_2 = 8$, $x_3 = -4$).