First Year Computing (C++ Course)
Dr John Joss Chinn

Week 3.

Arrays, Vectors, Nested Loops

Arrays

In the source file `io1.cpp`, which we looked at in week 1 (page 11), we used the numerical variables a and `b,` for integers, and `c,d` and e as floats. These variables could only take one value at a time. It is often useful in computer programming to be able to use a variable which takes several values at once. Take a look at the following program.

```
//array1.cpp
#include<iostream.h>
#include<conio>

void main()
{
   int i=0;
   float x[5]={12.3,13.5,-1.06,93.67,-12.5};

   while(i<5)
   {
      cout<<i<<"\t"<<x[i]<<endl;
      i++;
   }
   getch();
}
```

In the line `float x[5]={12.3,13.5,-1.06,93.67,-12.5};` we have defined an ARRAY, called x. This is simply a variable which takes several values. In this instance we have `x[0]=12.3,x[1]=13.5,x[2]=-1.06, x[3]=93.67` and `x[4]=-12.5`. Note that in computing, counting starts from zero. So although we define `x[5],` to take five values, the array goes from `x[0]` to `x[4].` Try not to let this confuse you. The `while` loop in this program is used simply to print out these ELEMENTS of the array to the screen. We could specify the array as follows, `x[]={12.3,13.5,-1.06,93.67,-12.5};` i.e. without specifying its size. If you just specify the actual elements of the array then the compiler ‘knows’ how many elements there are.

A small point is that it is a good idea to make as many relevant comments in the program as you feel you need. It has been said that it is actually good practice to have as much text in the form of comments as actual source code. For example, put the name of the source file in the program, `//array1.cpp` .

This next program specifies an empty array. There is a little `for` loop routine to get the user to enter the elements.

```
//array2.cpp
#include<iostream.h>
#include<conio.h>

void main()
{
   int i;
   float x[5];

   cout<<"Type in Five numbers for the Array\n\n";

   for(i=0;i<5;i++)  //Computers start counting from zero
   {
     cin>>x[i];
   }

   cout<<endl;

   /*The program remembers each one,
     They are not overwritten*/

   for(i=0;i<5;i++)
   {
     cout<<x[i]<<" ";
   }

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

As with the program `array1.cpp`, and the `for` and `while` loop programs which we looked at last week, this program uses an integer variable, `i`, just to increment through the elements. The first `for` loop reads in the elements and the second `for` loop just prints them to the screen. The point is that each element is remembered, it is not overwritten.

Vectors

You may need to use the idea of arrays when you read in columns of data from an external file. We shall not be looking at reading and writing to external files until week 8. However you may imagine that one may, for instance, wish to read in each entry from a column of data into an array element. So the whole column of data forms the array. You may not know beforehand how many entries there may be in a column of data (if your program were needed to read in several thousand different data files, for instance). So if we were to use arrays then we have the limitation that we do not know beforehand what size to dimension it. One way around this is to use the idea of VECTORS. A vector, in this sense, is a ‘dynamic’ array. It can be ‘grown’ by adding elements to it. The program `vector1.cpp,` over the page, uses this notion of vectors.

The program `vector1.cpp` has two vectors, `x` and `y`. The `for` loop routine creates ten elements for each vector, for `i` between zero and nine. I

have used `x[i] =i*i;` and `y[i]=pow(i,3);` for no particular good reason other than to create some interesting values for the elements. The things to notice in this program are the following:-
(1) You need to include the header `#include<vector.h>`
(2) The actual vectors are defined like this, `vector<float> y;`
(3) The line `y.push_back(i);` is the bit which 'grows' the vector to take more and more values, each time through the loop

```
//vector1.cpp
#include<iostream>
#include<vector.h>
#include<conio>
#include<math>

void main()
{
          vector<float> y;
           vector<float> x;
           int i;

           for(i=0;i<10;i++)      //this loop creates the elements
           {
              x.push_back(i);     //'grows' the vector
               y.push_back(i);
               x[i] =i*i;         //create element
              y[i]=pow(i,3);
           }

            for(i=0;i<10;i++)     //this loop prints the elements
           {
                cout << i <<"\t"<< x[i] <<"\t"<< y[i]<<endl;
            }

          getch();
}
```

Nested Loops

Over the page, the program `nest1.cpp` uses the idea of nested loops. That is you can use one loop inside of another loop. This program just prints out, to the screen:-

```
0, 0   0,1   0,2   0,3   0,4
1, 0   1,1   1,2   1,3   1,4
2, 0   2,1   2,2   2,3   2,4
3, 0   3,1   3,2   3,3   3,4
```

These are simply the incremental number of `i` and `j`. You can use a hierarchy of several loops nested inside one and other. We shall look at this idea again in future weeks.

```
//nest1.cpp
#include<iostream.h>
#include<conio.h>

void main()
{
     int i,j;

     for(i=0;i<4;i++)
     {
          for(j=0;j<5;j++)
          {
               cout<<i<<", "<<j<<"\t";
          }
          cout<<endl;
     }

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

Arrays and Nested Loops

Consider the system of simultaneous equations:-

$$\begin{gathered} x_1 - 3x_2 + 6x_3 + 2x_4 = 39, \\ -2x_1 + 4x_2 - 7x_3 - 3x_4 = -48, \\ 5x_1 + 2x_2 + x_3 + 4x_4 = 37, \\ 3x_1 + 2x_2 + 6x_3 + 7x_4 = 67 \end{gathered}$$

To solve this system of simultaneous equations, for x_1, x_2, x_3 and x_4, on a computer we would first need to put them in the form of an ‘augmented matrix’:-

$$\begin{pmatrix} 1 & -3 & 6 & 2 & 39 \\ -2 & 4 & -7 & -3 & -48 \\ 5 & 2 & 1 & 4 & 37 \\ 3 & 2 & 6 & 7 & 67 \end{pmatrix}$$

The program, `matrix1.cpp`, below, uses a ‘two-dimensional array’, `A[4][5]`, to define this matrix. There are two nested `for` loops in this program simply to print the elements to the screen. We will be looking at actually solving this type of system of simultaneous equations (by Gaussian Elimination with Partial Pivoting) in week 6.

```
//matrix1.cpp
#include<iostream>
#include<conio>

/*Either use '.h' in the include lines
  or use the following line*/
using namespace std;

void main()
{
      int i,j;

      //augmented matrix
      float A[4][5]={ 1, -3, 6,  2,  39,
                     -2,  4,-7, -3, -48,
                      5,  2, 1,  4,  37,
                      3,  2, 6,  7,  67};

      //just print out the augmented matrix
      for(i=0;i<4;i++)
      {
            for(j=0;j<5;j++)
           {
            cout<<A[i][j]<<"\t";
           }
           cout<<endl;
      }

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

Running Borland C++ Builder Version 4.0 on the machines in GB/B4

Some people where experiencing problems running Borland C++ Builder version 5.02. We don’t really need to use the latest version of Borland C++ Builder for the work that we are doing. Borland C++ Builder version 4.0 is only a couple of years old (1999) and seems to be much more stable. To run it use the following:-

- START→PROGRAMS→DEPARTMENTAL SOFTWARE→MECHANICAL ENGINEERING→BORLAND→BORLAND C++ BUILDER 4→C++ BUIDER 4

- The ‘Information’ Window comes up. Click on ‘No’

- Close the ‘Form 1’ Window

- Close the ‘Unit1.cpp’ window (Say ‘No’ to saving it)

- Go to FILE→NEW→CONSOL WIZARD (icon)
  (check Window type ‘console’, execution type ‘EXE’)

click on ‘Finnish’. (Say ‘No’ to save changes)

- Go to FILE→SAVE PROJECT AS. Change to c:\work\filename.bpr (filename is the name of your project. `.bpr` probably stands for “Borland Project”)

- Type in your source code

- Type CTL F9 to compile
  (takes a while!)

- Type F9 to run

Be sure to copy your source files (`filname.cpp`) from c:\work to your floppy drive, for safe keeping.
(Try to remember to bring a floppy disk in with you for the tutorials. I may have one or two old ones knocking around, if you forget!)

I include the following in case you are really stuck!

Using Borland C++ v3.1 in GB/B4

We have been experiencing some problems running Borland C++ Builder v5.02, in George Begg Room B4. There is an older, more stable, version of the Borland C++ compiler on the machines in GB/B4. This is Borland C++ v3.1. This is plenty good enough for the simple console applications that we shall be creating.

(1) To run this software, go to
START→PROGRAMS→WINDOWS NT EXPLORER

(2) It is a good idea, at this point, to go to **VIEW→OPTIONS** and <u>uncheck</u> the **“Hide file Extensions”** box.

(3) In Explorer, go to **m:\Borlandc\Bin** and click on **BCW.EXE**
This will open up the Borland C++ compiler, v3.1.

(4) When it comes up with “cannot find BWCCENG.DLL”, just click close.

(5) When the Borland C++ Compiler is open, go to **OPTIONS→DIRECTORIES**.
In the output directory type **c:\work**.

(6) Then, just type in your code.

(7) Make sure that you use the **old C++ type headers**, e.g. iostream.h and conio.h (i.e. with the **.h** extension)

(8) To compile press **ALT F9**.

(9) To run press **CTL F9**. Click on OK on all the little windows that come up.

(10) An output window opens. You will need to close this manually before re-editing your code and recompiling it.

(11) One other little point. When you run your application and you just want to stop it, for any reason, then just press **CTL C**.

Tutorial

(1) In array2.cpp alter the line `for(i=0;i<5;i++)` in both the first and second `for` loop to read `for(i=0;i<10;i++)`. So what happens when you try to read in too many elements, when you have already specified the size of the array as just `x[5]`?

(2) In the program `vector1.cpp` make the program ‘grow’ twenty elements.

(3) Change the value of the elements, in `vector1.cpp` from `x[i] =i*i;` and `y[i]=pow(i,3);` to elements that you devise yourself.

(4) Once you are familiar with the notion of nested loops and that of arrays. Try to adapt the program `matrix1.cpp` to:-

- (a) Add two matrices
- (b) Subtract two matrices
- (c) Multiply two matrices

For this last task you will need to be able to add elements of an array. You may find the little program over the page, `sum1.cpp,` useful for some ideas here.

```
//sum1.cpp
//Find the sum of men's heights
//Find their mean height
#include<iostream.h>
#include<conio.h>

void main()
{
	int i;
	float height[]={68,62,72,70,66,68};
	float sum_h,mean_h;
	sum_h=0;

	//read in all of the elements and add them

	for(i=0;i<6;i++)
	{
		sum_h=sum_h+height[i];
	}

	mean_h=sum_h/6;  //Find the mean height

	cout<<"sum of heights "<<sum_h<<endl;
	cout<<"mean height "<<mean_h<<endl;

	getch();
}
```