SESSION 1-6: FILE HANDLING

1-6.1 Using Files

During program execution, data entered at the keyboard for processing are lost once the program runs to completion or when there is power interruption without an uninterrupted power supply (UPS) to take over supply of power. Similarly, the results or outputs to the screen are only temporal and will be lost together with the input data. To ensure that data and output are not lost when programs run to completion or when there is power interruption input data can be stored in a file and output from a program written to a file on a storage device. One advantage of having the input data stored on a disk is that if the input data were to be entered from the keyboard and the program has to be run a number of times with the same input the user will have to enter the same data a couple of times while with files there is no need to enter data into a file for more than once. We shall first look at sequential files and then random files.

SEQUENTIAL FILES

A sequential file is one in which records are stored in a sequential order and are also accessed in the same order as they were stored. To access a record in a sequential, one will have to begin the search with the first record and then sequentially towards the end to locate the required record. Because of the way records are located in a sequential, accessing records are extremely slow especially if the file has a lot of records and with the records being searched for close to the end of the file. One way of speeding file handling with respect to sequential files is to read and store the file records into arrays. The problem with this is that if the file has a number of records then one may run out of the computer's RAM.

C++ has two basic classes to handle files and these are ifstream (**i**nput **f**ile **stream**) and ofstream (**o**utput **f**ile **stream**). To use these one must include the header file fstream.h to handle the file input (reading from files) and file output (writing to files). The syntax for declaring a file for input is

```
ifstream  inputFile;
```

and for output is

```
ofstream outputFile;
```

where the *inputFile* and the *outputFile* are user defined stream variables. These input and output streams must be connected to a file. Connecting a stream to a file simply means opening the file and this is achieved by using the open function. The syntax for using the open function to open a file is as follows:

```
inputFile.open("inputFileName");
```

where the *inputFile* is the stream that connects the input file. The *inputFileName* is the name of the input file as it appears on a storage device. The input file must be a text file. The extension of the file must be specified as part of the input filename. For example, to open the file input.dat for input in a program, the following statements must be included in the program. The input.dat is the external name of the file in the default directory, that is the name of the file as it is on a disk or a storage device.

```
ifstream  inputFile;
inputFile.open("input.dat");
```

Assuming the input.dat file is in the directory **c:\cplusplus\myprogs** then you will need the following statements:

```
ifstream  inputFile;
inputFile.open("c:/cplusplus/myprogs/input.dat");
```

Note the use of forward slash instead of a backslash. It must be noted that all files to be used in a program must first be opened before they can be used. If a program uses different input files at different times then a variable can be used to receive a file name from the keyboard at every run of the program.

The default mode for opening a file using the constructor of the ofstream is either to create the file if it does not exist or to delete everything in the file if it does exist and whether or not it contains data. It is possible to include a second argument to specify how the file should be handled. The second argument may be any of the following:

ios::app -- Opens the file and allows data to be appended or data additions at the end
ios::ate -- Opens the file and allows additions anywhere
ios::binary- - Causes the file to be opened in binary mode
ios::in- - specifies that a file is capable of input
ios::out- - specifies that the file is capable of output
ios::trunc -- Deletes everything in the file if it exist
ios::nocreate -- Does not open file if the file must be created
ios::noreplace -- Does not open file if the file already exists

The following program allows more data to be added to the output file whenever the program is run without first emptying the file. Such an output file mode can be very useful if you should have different input files but then you prefer all your output to be in the same output file. The bolded line shows how you can open a file for append. Don't worry if at this stage you don't understand some of the program lines as they will become clear in due course.

```
#include<iostream>
#include<fstream>
#include<cstdlib>
```

```
#include<iomanip>
using namespace std;
ifstream input;
ofstream output;
int main()
{
 int num, sum=0;
 input.open("input.dat");

 if(input.fail())
 {
        cout <<"Input file opening failed\n";
        system("pause");
        exit(1);
 }

 output.open("output.dat",ios::app);//open output file for append/ additions
 output.setf(ios::right); //set numbers to be right justified

 while (input.eof())
 {
   input >>num;
   sum=sum+num;
   output.width(10);//width for the variable num
   output<<num;
   output.width(15);//width for the variable sum
   output <<sum <<endl;
 }
 input.close();
 return 0;
}
```

CHECKING IF A FILE WAS OPENED SUCCESSFULLY

It is always good to check whether a file opened with the open function opened successfully. This will prevent a program from halting unexpectedly. For example, if an attempt is made to open a file for input and there is no such file on disk, the compiler will not be able to locate the file to open. In C++ you may not receive any message that the file could not be opened but unexpected results such as wrong output may be obtained. It is therefore advisable to follow all open statements

with a test for each open statement as to whether a file was opened successfully or not. This is achieved by using the fail() function. The syntax is as follows:

InputFileVariable.fail();

Where the *InputFileVariable* is the variable associated with the input file. The above statement will be true if the file is opened successfully and false if otherwise. For example, to test if the input file, input.dat was opened successfully or not the following statements may be included in the program:

```
ifstream input;
input.open("input.dat);
if (input.fail())
{
  cout <<"Input file could not be opened";
   system("pause");
   exit(1)
}
```

The exit(1) statement causes program execution to end immediately so that the program will not continue to produce an undesired results or outputs. The above code will display the message "input file could not be opened" and then exit only when the value returned by the fail() function is 0. The exit takes any integer as an argument but generally 1 and 0 are used. The 1 is used when there is an error and 0 for other reasons. Note that the exit function is defined in the stdlib.h header file hence to use the exit function in a program you must include the header file, stdlib.h.

READING OR WRITING TO A FILE

Once a file has been opened for input, records or data can be read from the file. Similarly for an opened output file, data can be written into the file. To read from a file the syntax is as follows:

InputFileStreamVariable >> inputVariables;

where the InputFileStreamVariable is the variable name the input file is connected to. The inputVariables are the list of input variables. The list of input variables is specified in the same way as you do with the cin stream. For example, to read data into the variables mark1, mark2 and mark3, the input statement can be written as follows if the input file stream variable is fcin:

fcin >> mark1 >> mark2 >> mark2;

To write to an output file the syntax is as follows:

OutputFileStreamVariable << outputVariables;

where the *OututFileStreamVariable* is the variable name the output file is connected to. The *outputVariables* are the list of output variables. The output variables are specified in the same way as you do with the cout stream. For example, to write data stored in the variables mark1, mark2 and mark3, the output statement can be written as follows if the output file stream variable is fcout:

fcout << mark1 << mark2 << mark2;

In the above example, the output items will be written without spaces between them hence to ensure that they are separated by spaces you must include spaces in the output statement. For example, if char sp[5]=“ “ is included in a program before the output statement then we can write the output statement as follows so that spaces will be inserted between the output data:

```
fcout >> mark1 >>sp  >> mark2 >> sp  >> mark2;
```

Note that to force each output list to be written on a separate line, you must include ">>endl" or the “\n” in the output statement.

TESTING FOR END OF FILE

Since a program can be used to process input files of different number of records at different times, you must guard against reading a data when there is nothing in the file to be read, that is when all input data in the file have been read. In C++, this can be done by using the while statement or the eof() function. The eof() function is best used when the input is a text. We will consider the while statement first. The format is

```
while (ThereIsDataToRead)
{
   loopStatements;
}
```

where ThereIsDataToRead is an input statement connected with the input file. For example, assume the input file is connected to the variable input and only one integer variable, number is used to read and sum all data (numbers) in the input file, then we can code the while statement for checking the end of the input file as follows:

```
while (input>>number)
   sum=sum+number;
```

The following program allows a user to enter a file name from the keyboard and then reads in all the input data. For each input data read, it adds to the previous input and then writes the input read and the cumulative sum into an output file. It uses the while statement to illustrate how it can be used to check for the end of file.

```
#include<iostream>
#include<fstream>
#include<cstdlib>
using namespace std;
ifstream input;
ofstream output;
int main()
{
```

```
  char filename[13];/* assumes file names will be at most 12 characters including
the file extension*/
  int num, sum=0;
  cout <<"Please enter the name of the file to process ";
  cin >>filename;
  input.open(filename);
  if(input.fail())
  {
          cout <<"Input file opening failed\n";
           system("pause");
          exit(1);
  }
  output.open("output.dat");
  while (input>>num) //while not the end of the file read a number
  {
          sum=sum+num; //add number read to sum
          output<<num <<"   " <<sum <<endl; //write to output file
  }
  input.close();
  output.close();
  return 0;
}
```

Program run: (there is no file tryme.dat in the default file directory)

```
Please enter the name of the file to process tryme.dat
Input file opening failed
Press any key to continue . . .
```

The above output is obtained displayed to the screen when the computer is unable to locate the file tryme.day.

As mentioned above, we can also use the eof() function to test for the end of an input file. The syntax is

```
InputStreamVariable.eof();
```

Where *InputStreamVariable* is the name of the input stream variable. The *eof()* function is used with the while, do while or the if statement. The following program which performs the same function as the one above uses the *eof()* to check for end of file.

```
#include<iostream>
#include<fstream>
#include<cstdlib>
```

```
using namespace std;
ifstream input;
ofstream output;
int main()
{
 int num, sum=0;
 input.open("input.dat");

 if(input.fail())
 {
        cout <<"Input file opening failed\n";
        system("pause");
        exit(1);
 }

 output.open("output.dat");

 while (input.eof())
 {
        input >>num;
        sum=sum+num;
        output<<num <<"   " <<sum <<endl;
 }
 input.close();
 output.close();
 return 0;
}
```

FORMATTING OUTPUT

Output formatting is done using the formatting flags for the setf. The setf is an abbreviation for set flag. A flag on the other hand is an instruction designed to set something either on or off. Output format is simply the layout of a program's output. This basically involves specifying the number of decimal places required, the amount of spaces between output items, etc. Unless you specify how your output should be, your output will usually not be in the format that you may like. It is therefore necessary to format your output with stream functions. To use a setf formatting flag the syntax is as follows:

```
outputFileVariable.setf(flag);
```

where *outputFileVariable* is the variable assign to the output file.

Table 6.1 shows the formatting flags for the setf.

Table 6.1 Formatting Flags for setf function

Flag	Meaning	By default is
Ios::fixed	If set prevents floating-point numbers to be written in their e-notations. Once set affects all subsequent floating point numbers.	Not set
Ios::scientific	If set causes floating-point numbers to be written in their scientific forms. Once set affects all subsequent floating point numbers.	Not set
Ios::showpoint	If set then a decimal point and trailing zeros are shown for floating point numbers. Once set affects all subsequent floating point numbers.	Not set
Ios::showpos	If set causes a positive integer value to have the plus sign (+) as a prefix. Once set affects all subsequent floating point numbers.	Not set
Ios::right	If set causes numbers to be written right justified within the set width using the width function. Thus, a number with fewer digits than the set width will have spaces before the number to make up the set width.	Set
Ios::left	If set causes numbers to be written left justified within the set width using the width function. Thus, a number with fewer digits than the set width will have spaces after the number to make up the set width.	Not set

An example of how to set numbers to scientific is as follows:

```
fcout.setf(ios::scientific);
```

where fcout is an output file variable.

The following can also be used in formatting output file streams. Note that they are not flags hence they are not used with the setf.

1. The **precision** function: This is used to set the number of decimal places required. The syntax is as follows:

```
OutputStreamVariable.precision(integer);
```

Where *OutputStreamVariable* is the name of the output stream variable. The *integer* is an integer value that specifies the number of decimal places required. An integer value of 2 means floating point numbers should be to two decimal places.

Example output.precision(5);

2. The **width** function: This is used to specify the width of numbers. The syntax is as follows:

OutputStreamVariable.width(integer);

Where the *integer* defines the width of the output numbers. Note that the width field starts from the current position of the cursor. Please note that the width set affects only the variable immediately following the width function hence each variable should have its own width if that is needed.

Example: output.width(10);

The following program shows how some of the setf flags and the width function are used in a program to set justification and field width respectively.

```
#include<iostream>
#include<fstream>
#include<cstdlib>
using namespace std;
ifstream input;
ofstream output;
int main()
{
 int num, sum=0;
 input.open("input.dat");
if(input.fail())
 {
          cout <<"Input file opening failed\n";
          system("pause");
          exit(1);
 }

 output.open("output.dat");
 output.setf(ios::left); //set numbers to be left justified


 while (input.eof())
 {
          input >>num;
          sum=sum+num;
   output.width(10);//width for the variable num
   output<<num;
   output.width(15);//width for the variable sum
```

```
    output <<sum <<endl;
  }
  input.close();
 output.close();
  return 0;
}
```

CLOSING FILES

All files used in a program must be closed before the program terminates as you can see in the above example. Closing a file disconnects a stream from the file. By default, the C++ compiler closes all files when a program runs to successful or normal completion even if you do not use the close() function to close the files, however it is a good practice to always close all your input and output files. Note that if your program does not run to completion then all your files will remain opened and your files can easily be corrupted. Another important reason for closing a file is that when a file is first opened for output and output data have been written to the file, you can only use this output file as input for another process if only it is closed and reopened for input. To close a file after use, the syntax is

```
FileStreamVariable.close();
```

where *FileStreamVariable* is the variable name of the file stream. To close a file having input as the file variable, we will code input.close();

1-6.2 File Variables as Function Arguments

File variables can be used as function arguments if functions are required to read from and/or write to file. Generally, to use input and output file in a function, the function header must appear as follows:

ReturnDataType functionName(ifstream inputFileVariable, ofstream OutputFileVariable);

Where the inputFileVariable is the name of the input file variable and OutputFileVariable is the name of the output file variable. For example, if input and output file variables are input and output then a function header of a function that is expected to use these files may appear as follows where *processFile* is the name of the function:

```
void processFile(ifstream input, ofstream& output);
```

The following program illustrates how a function can read from an input file and then output results to an output file. Note the use of the & in the output file variable. The function is expected to write to the output file hence it is going to modify the file content.

Question: Assume that you have just completed school and have been appointed as the head of the IT department of a newly established car manufacturing company at the head office. The company has a branch in each of the 10 regions in the country to sell car manufactured at the head

office. Each branch sends in their sales transaction at the end of every month to the head office. The head office requires a program to process the data and to generate results in the following output format:

	Quantity of cars sold									
Bra	1	2	3	4	5	6	7	8	9	Total
==	=	=	=	=	=	=	=	=	=	==========
1	X	x	x	x	x	x	X	x	x	
4	X	x	x	x	x	x	X	x	x	
7	X	x	x	x	x	x	X	x	x	
2	X	x	x	x	x	x	X	x	x	
3	X	x	x	x	x	x	X	x	x	
5	X	x	x	x	x	x	X	x	x	
6	X	x	x	x	x	x	X	x	x	
10	X	x	x	x	x	x	X	x	x	
9	X	x	x	x	x	x	X	x	x	
8	X	x	x	x	x	x	X	x	x	
==	=	=	=	=	=	=	=	=	=	==========
Tot	X	x	x	x	x	x	X	x	x	

The above problem basically involves the following processes:

i. Read the input data from file and store them in an array so as to avoid having to open and read the input file a couple of times

ii. Compute the row and the column totals. The column and the row totals should preferably be stored after the last column and row of the array used in step (i) respectively. It will be easier to ensure that the column totals are value of the cars and not the quantity.

iii. Since the output should be sorted in order of branch that sold the highest amount (in terms of money) of cars, the array must be sorted based on the column totals.

iv. The entire matrix including the row and column totals should be printed using the output format given.

Program listing

```
#include<iostream>
#include<fstream>
#include<iomanip>
using namespace std;
const int rows=12,columns=11;
//The program uses subscripts from 1 for branches and car type
//extra locations have been added to cater for column and row totals
void getInput(ifstream input); // read input data into array
void ComputeRowTotals(double xx[][columns]);//compute row totals
void ComputeColumnTotals(double xx[][columns]);//compute column totals
```

```
void SortRows(double xx[][columns]);//sort rows based on Total amount column
void StoreOutput(double xx[][columns], ofstream output);//write to output file
double x[rows][columns]; //array to store input data
char line[26]="==========================";//for underlining
const double cost = 21596500;//factor of cost of a car

int main()
{
	ifstream input; //defines input file variable
	ofstream output; //defines output file variable
	input.open("input.dat"); //opens input file input.dat
	output.open("output.dat");//opens output file output.dat
	cout <<"Reading input data into memory.....\n";
	getInput(input);
	cout <<"Computing row totals......\n";
	ComputeRowTotals(x);
	cout <<"Computing column totals....\n";
	ComputeColumnTotals(x);
	cout <<"Sorting output data as required.....\n";
	SortRows(x);
	cout <<"Writing output to output.dat file ...\n";
	StoreOutput(x,output);
	cout <<"All processes completed now. Thanks\n\n";
	return 0;
}

void getInput(ifstream input)
{
  int a,b,c;
	while (input >>a >>b >>c)//while not end of file read in car details
	{
		x[a][b]=double(c);
		x[a][0]=a;
	}
}

void ComputeRowTotals(double xx[][columns])
{
	int i,j;
	double sum,sum2=0;
	 for(i=1;i<rows;i++)
	 {
		sum=0;
		for (j=1;j<columns-1;j++)
```

```
                    sum=sum +cost*j*xx[i][j];
                xx[i][columns-1]=sum/1000000.0; //will give totals in millions
                sum2=sum2+xx[i][columns-1];
        }
        xx[rows-1][columns-1]=sum2;
}


void ComputeColumnTotals(double xx[][columns])
{
        int i,j;
        double sum;
        for(j=1;j<rows;j++)
        {
                sum=0;
                for (i=1;i<columns;i++)
                        sum=sum +xx[i][j];
                xx[rows-1][j]=sum;
        }

}

void SortRows(double xx[][columns])
{
        int passes,comps,k,n=columns-1;
        double temp;
        for(passes=1;passes<columns;passes++)
                for(comps=1;comps<columns-passes;comps++)
                {
                        if(xx[comps][n]>xx[comps+1][n])
                                for(k=0;k<=n;k++)
                                {
                                        temp=xx[comps+1][k];
                                        xx[comps+1][k]=xx[comps][k];
                                        xx[comps][k]=temp;
                                }
                }
}

void StoreOutput(double xx[][columns], ofstream output)
{
        int i,branch,carType;
        output <<"\t\t\tAmount (in cedis) sold for car type\n";
        output <<"Branch \t";
```

```
            for(i=1;i<columns-1;i++)
                    output <<"\t" <<i;
            output <<"   Total Amount(Million)\n";
            output <<line <<line <<line  <<line <<endl;
            for(branch=1;branch<columns;branch++)
            {output <<"\t" <<x[branch][0] <<"\t";
                    for(carType=1;carType<columns;carType++)
                            output <<x[branch][carType] <<"\t";

             output <<endl;
            }
            output <<line <<line <<line  <<line <<endl;
            output <<"\t\t";
            for(carType=1;carType<columns;carType++)
               output <<x[rows-1][carType] <<"\t";
input.close();
output.lose();
}
```

Input data:

1 5 20	2 5 20	3 5 20	5 5 20	4 5 20
1 4 15	2 4 15	3 4 15	5 4 15	4 4 15
1 6 25	2 6 25	3 6 25	5 6 25	4 6 25
1 3 50	2 3 50	3 3 50	5 3 50	4 3 50
1 2 20	2 2 20	3 2 20	5 2 20	4 2 20
1 1 5	2 1 5	3 1 5	5 1 5	4 1 50
1 9 9	2 9 9	3 9 9	5 9 9	4 9 9
1 8 1	2 8 10	3 8 1	5 8 18	4 8 1
1 7 1	2 7 15	3 7 1	5 7 1	4 7 1
9 5 20	7 5 20	8 5 20	10 5 20	6 5 20
9 4 15	7 4 15	8 4 15	10 4 15	6 4 15
9 6 25	7 6 25	8 6 25	10 6 25	6 6 25
9 3 50	7 3 50	8 3 50	10 3 50	6 3 50
9 2 20	7 2 20	8 2 20	10 2 20	6 2 20
9 1 5	7 1 15	8 1 25	10 1 35	6 1 65
9 9 9	7 9 9	8 9 9	10 9 9	6 9 9
9 8 41	7 8 11	8 8 21	10 8 31	6 8 1
9 7 1	7 7 1	8 7 1	10 7 1	6 7 1

Program output

	Amount (in cedis) sold for car type									
Branch	1	2	3	4	5	6	7	8	9	Total Amount (Million)
1	5	20	50	15	20	25	1	1	9	12979.5
3	5	20	50	15	20	25	1	1	9	12979.5
4	50	20	50	15	20	25	1	1	9	13951.3
6	65	20	50	15	20	25	1	1	9	14275.3
7	15	20	50	15	20	25	1	11	9	14923.2
5	5	20	50	15	20	25	1	18	9	15916.6
2	5	20	50	15	20	25	15	10	9	16650.9
8	25	20	50	15	20	25	1	21	9	16866.9
10	35	20	50	15	20	25	1	31	9	18810.6
9	5	20	50	15	20	25	1	41	9	19890.4
	215	200	500	150	200	250	24	136	90	157244

Question: Consider an input file, infile.dat that contains pairs of data points (x_i and y_i) on each row. Write a C++ program to read the data points and calculate the equation of the line. Your output should be of the form $y = ax + b$, that is the line of best fit where *a* and *b* are constants. The values for *a* and *b* can be obtained using the following relations:

$$a = \frac{\sum_{i=1}^{n} y_i - b\sum_{i=1}^{n} x_i}{n}$$

$$b = \frac{\sum_{i=1}^{n} x_i y_i - \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n}}{\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}}$$

where

$\sum_{i=1}^{n} x_i y_i$ = sum of the product $x_1y_1 + x_2y_2 + \ldots + x_ny_n$

$\sum_{i=1}^{n} x_i$ = sum of x-values=$x_1 + x_2 + \ldots x_n$

$\sum_{i=1}^{n} y_i$ = sum of y-values = $y_1 + y_2 + \ldots + y_n$

$\sum_{i=1}^{n} x_i^2$ = sum of squares of x-values = $x_1^2 + x_2^2 + \ldots + x_n^2$

Solution:

Program listing

```
#include<iostream>
#include<fstream>
#include<cmath>
using namespace std;
void Process (ifstream input, ofstream output);
int main()
{
        ifstream input;
        ofstream output;
        input.open("infile.dat");
        output.open("outfile.dat");
       Process(input,output);
        input.close();
        output.close();
        return 0;
}

void Process (ifstream input, ofstream output)
{
        int n=0;
        float x,y, a, b;
        float sumx=0,sumxy=0,sumxsqd=0,sumy=0,corr;
        output <<"\tX-Values\tY-Values\n" ;
        while (input >> x >> y) //while not the end of file
        {
                sumx=sumx+x;
                sumy=sumy+y;
                sumxsqd=sumxsqd+x*x;
                sumxy=sumxy+x*y;
                output <<"\t" <<x <<"\t\t" <<y <<endl;
                n++;
        }
         b=(sumxy-sumx*sumy/n)/(sumxsqd-pow(sumx,2)/n);
```

```
        a=(sumy-b*sumx)/n;

        output <<"\n\nThe equation of the line is y = " <<a <<"x + " <<b;
        cout <<"Computations completed, ";
}
```

Input file content:

```
 1  1.5
2.5   2
 4   4
 6   4
 8   5
 9   7
11   8
15  10
```

Output file content

X-Values	Y-Values
1	1.5
2.5	2
4	4
6	4
8	5
9	7
11	8
15	10

The equation of the line is y = 0.778363x + 0.624303

Question: Talk and Talk More (TTM) mobile phone service provider requires a program to compute the cost of national and international calls. National calls are of two types namely same network and other network. The cost of the calls is determined as follows:

(i) Any call started between 7am and 7pm, Monday through Friday, is charged as follows:
- 1200 and 2400 cedis per minute for same and other network respectively
- 8,200 cedis for international calls.

(ii) Any call started before 7am or after 7pm, Monday through Friday, is charged as follows:
- 800 and 1400 cedis per minute for same and other network respectively
- 5,200 cedis for international calls.

(iii) Any call started on Saturday or Sunday is free, 600 and 4000 cedis respectively for same network, different network and international calls.

The days of the week are coded 1 through 7 for Monday through Sunday respectively. Also the same network, different network and international calls have the codes 1, 2 and 3 respectively.

You are required to write a program that will accept as input the code for the day and the code for the type of the call, that is same network, other network or international. In addition, your program should accept for each call the length of the call in minutes and the time a call started. The time will be input in 24-hour notation so that time 1.48 is input as 1348. You are required to use at least one function in your program. The output should be directed to file 'output.dat'

Program listing

```
#include<iostream>
#include<fstream>
#include<iomanip>
using namespace std;
ofstream fcout; //fcout declared as output file variable

void weekday(); //Computes Monday to Friday calls
void weekend(); //Computes Weekend calls
void writeDetails(char peak[15]);//Writes output for a call made to 'output.dat'
double total=0, cost;
int CallLength,CallStart,CallType,day;
char line[23]="----------------------";//for underlining
int main()
{
        fcout.open("output.dat");
        fcout <<"\t\tList of Phone Calls for Today\n";
        fcout<<"Type\tStart\tlength of Call\tDay of Call\t\t     Cost" <<endl;
        fcout <<line <<line <<line <<line <<endl;
        do
        {
                do
                {
                cout<<"\nPlease enter a valid time call started in 24-hr format ";
                 cout <<"\nDon't type leading zeros, 1.00am should be entered as 100";
                cin>>CallStart;
                } while(CallStart<0||CallStart>2400);
          cout<<"Enter the length of call ";
          cin>>CallLength;
          do
          {
            cout<<"Enter valid type of call:1, 2 or 3 ";
```

```
            cin>>CallType;
          } while(CallType<1||CallType>3);
          do
          {
            cout<<"Enter day of Call,1-7, 1 for Monday, etc ";
            cin>>day;
          }while(day<1||day>7);

          if(day<6)
                    weekday();
          else
                    weekend();

        }while (CallLength!=1);
        fcout <<endl <<line <<line <<line <<line;
        fcout <<"\n\tTotal\t\t\t\t\t\t";
        fcout <<setprecision(2) <<setfill('*') <<setw(15)<<total;
        fcout <<endl <<line <<line <<line <<line <<endl;
        return 0;
}

void weekday()
{
        if(CallStart>700&&CallStart<1900) //test if calls were made during the day
                if(CallType==1)
                        cost=1200.0*CallLength;
                else if(CallType==2)
                        cost=2400.0*CallLength;
                else
                        cost=8200.0*CallLength;
        else //call was made on a week end
                if(CallType==1)
                        cost=800.0*CallLength;
                else if(CallType==2)
                        cost=1400.0*CallLength;
                else
                        cost=5200.0*CallLength;
        total=total+cost;
        writeDetails("Week Day");
}


void weekend()
{
```

```
        if(CallType==2)
                cost=600.0*CallLength;
        else if(CallType==3)
                cost=4000.0*CallLength;
        writeDetails("Week End");
        total=total+cost;
}

void writeDetails(char peak[15])
{
 fcout.setf(ios::fixed);
 fcout <<CallType <<"\t"  <<setw(4) <<setfill('0')<<CallStart;
 fcout <<"\t\t" <<setw(3) <<CallLength;
 fcout <<"\t" <<peak <<"\t\t" <<setprecision(2) <<setfill('*') <<setw(15) <<cost;
 fcout <<endl;
}
```

Output from program with test data

List of Phone Calls

Type	Start	length of Call	Day of Call	Cost
1	0700	010	Week Day	********8000.00
3	0750	010	Week Day	*******82000.00
2	0810	005	Week Day	*******12000.00
3	0900	006	Week Day	*******49200.00
1	0933	012	Week Day	*******14400.00
3	1105	015	Week Day	******123000.00
1	1321	011	Week Day	*******13200.00
1	1440	050	Week Day	*******60000.00
1	1603	020	Week Day	*******24000.00
1	1800	030	Week Day	*******36000.00
3	2005	010	Week End	*******40000.00
2	2100	100	Week End	*******60000.00
2	0448	010	Week Day	*******14000.00
2	1000	001	Week Day	********2400.00
	Total			******538200.00

Question: Secret Agents in a certain country normally communicate to each other in writing by coding the message. A secret message written in codes has been received. The message consists of all the 26 alphabets and some special characters. Special characters are to be left unchanged. The letters to be decoded and their respective codes are as follows:

LETTER A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
CODES J A M E S B N H Y F R O C Q U P K W I D G L T V X Z

Write a program to decode a given message and output the original message. All messages end with a full stop and each message contain only one full stop.

Program listing:

```
#include<iostream>
#include<cstring>
#include<cctype>
using namespace std;
int main()
{   int i,j,k=-1;
  char EnglishChar[]="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  char codes[]=      "JAMESBNHYFROCQUPKWIDGLTVXZ";
  char mess[1000];//a message length is assumed to be at most 1000 characters

  do //read characters until the full stop is encountered
  {
    k++;
   cin.get(mess[k]);
   } while (mess[k]!='.');
  for(i=0;i<k;i++)
  {
    if(!isalpha(mess[i]))
       cout <<mess[i];
    else
        for(j=0;j<26;j++)
         if(toupper(mess[i])==codes[j])
            cout <<EnglishChar[j];
                                  //converts lower case characters to upper
  }                                       //for comparisons.
  return 0;
}
```

If the above program is run with the message "UGW BJDHSW, OSD DHX TYOO AS EUQS, JCSQ!!." the output will be "OUR FATHER, LET THY WILL BE DONE, AMEN!!"

1-6.3 Input and Output File Member Functions

The following member functions are used with files. To use any of these functions you must include the header file fstream.h in the program. Again, to use any of the following functions you must prefix it with the name of the stream variable followed by the dot. For example, if an input

file variable is infile then to use the open function you will have to write infile.open("rootName.ext");

Form of function call	Description
bad()	returns true(1) if the file is corrupted
close()	closes or disconnects the stream from the file it is connected to
eof()	returns true if it is the end of the file and false if otherwise
fail()	returns true if a file open operation was unsuccessful.
get(x)	Reads one character from the input file stream and assign to x. note that it does not skip whitespace such as the space.
getline(s,n)	Reads a string of n-1 characters from the input file stream and assign to s.
open("rootname.ext")	connects the file with the name rootname.ext (eg input.dat) to the stream named by the input or output stream.
peek(x)	Reads a character from the input stream and assign to x. however, it does not remove the character from the input stream and therefore allows the next input statement to read the same character.
precision(n)	use to define the number of decimal digits, n to be printed for a floating point number where *n* is an integer.
put(x)	use to write the character stored in x to the output stream variable.
putback(x)	use to write a character stored in x back to the input stream to be read as the next input character from the same stream.
setf(flag)	to set a particular flag on for output to the output file. example outfile.setf(io::fixed) will prevent floating point numbers to be displayed using the e notation. see above for the other flags.
unsetf(flag)	use to unset a flag that has been set using the setf.

width(n)	use to set the field width of the next output to the output stream variable to n where n is an integer.

Question: The Kwame Nkrumah University of Science and Technology wants a computerised system for marking students multiple choice questions at the end of each semester. Each exam paper will consist of 50 objective questions. Students' answers will usually be entered into a file to serve as input to the program. The content of each file will contain a student's index number, first name, last name and the answer string. The answer to each question may either be A, B, C, D or E. An F means a student did not answer a particular question. Without using user defined functions, write a program which first reads in the Examiner's answer string and then the students' answer strings and output the student's index number, last name, first name and exams mark under appropriate headers. Assume no exams paper will be taken by more than 500 students.

Program listing

```
#include <iostream>
#include <cstdlib>
#include<fstream>
using namespace std;
char correct[51];
char answer[100][51];
char ID[500][20], fname[500][20], sname[500][20];
int main()
{    ifstream fcin;
     ofstream fcout;
     fcin.open("c:/minput.dat");
     if(fcin.fail())
       cout <<"Input file failed while opening..... " <<endl;
     fcout.open("c:/moutput.dat");
     int i,cnt=0;
          fcin >>correct;
          cout<<correct <<endl;
//read rcords and store in arrays
     while(fcin>>ID[cnt] >>fname[cnt] >>sname[cnt] >>answer[cnt])
          {
               cout <<"Record " <<cnt+1 <<" has been read" <<endl;
               cnt++;
          }
// write output headers and then output lines
      fcout<<"Index No. \tLast Name\t\tFirst Name\t\tExams Mark" <<endl;
     for(i=0;i<cnt;i++)
     {
```

```
        int sum=0;
        for(int j=0;j<50;j++)
          if (answer[i][j]==correct[j])
            sum=sum+1;
        fcout <<ID[i] <<"\t\t" <<sname[i] <<"\t\t\t" <<fname[i] <<"\t\t\t"
              <<sum <<endl;
      }
      system("PAUSE");
      fcin.close();
      fcout.close();
      return 0;
}
```

Sample input file content: Please note that between index numbers, first name, last name and the answer string there is at least one blank space between them. This is for those who want to test their solution with the same input. The first line is the lecturer's answers to the questions. Note also that each answer string is exactly 50 characters long.

```
ABDCBBDECCAADABCDECADEBDEAACBBCEDDDAEBABBCCDECCEED
2240803 James Arhin  ABBCDEDBEABACEDEEEDBBACCDESSDEDBBCDECCEDDEDAASDEDD
2240805 Kwesi  Mensah   CEECDEDBEABDDEDEEEDBDDECDESSDEDBAAAECCEDCEDDABDEDA
2240806 Kingsley Mawusi   BBBDECCBEABACEDEABDBBACCDESSDEDBBCDECCEDDEDAASDEDA
2240808 Lovelace  Hope  ACCCDEDBEABACEDCCEDBBACCDESSDCCBBCDECCECCEDAASDEDB
2240810 Julia  Roberts  ABDCBBDECCAADABCDECADECDDESSDDCDBCDECCEDDEDAASCDDA
2240811 Pierre Pierro  ACDCBBDECCACDABCDECACEBDEAACCBCEDDDAECABBCCDECCCED
2240812 Joe   Kambikambi   ABDCBBDECCAADBBCDECBDEBDEAACBBCEDDDBEBABBCCDBCCEED
2240814 Peter Wyganjo  ABDCBEDECCAADEBCDECADEBEEAACBBCEEDEAEBEBECCDECCEED
2240815 David Mugo   ACDCBCDEDCDADADCDECDDEBCEAACBBCEDDDAEBABBCCDECCEED
2240816 Araba Biwowofie  ABDCBBDECCAADABCDECADEBDEAACBBCEDDDAEBABBCCDECCEED
2240817 Elizabeth Taylor   ABDCBBDECCAADABCEECEDEBDEAEEBBCEDDDAEBABBCCDECCEED
2240818 Mary Mensah   ABDCBBDECCAADABCDECADEFDEAACBFCEDDDAEBABBCFDECCEED
2240819 Magi Odoom  DBDDBBDECCAAFABCDECADEBDEADCBBCEDDDADBABBCCDECCEDD
2240822 Obaa Panyin  AAACBBDECAAADABCDECAAABDEAACBBCEAADDAEBABBCCDECDED
2240823 Kwesi Kyeremanteng  ABDCBBDECCAADABCDECADEBDEAACBBCEDDDAEBABBCCDECCEED
```

Program run based on the above input file.

Index No.	Last Name	First Name	Exams Mark
2240803	Arhin	James	9
2240805	Mensah	Kwesi	7
2240806	Mawusi	Kingsley	4
2240808	Hope	Lovelace	9
2240810	Roberts	Julia	26
2240811	Pierro	Pierre	44
2240812	Kambikambi	Joe	46
2240814	Wyganjo	Peter	43
2240815	Mugo	David	43
2240816	Biwowofie	Araba	50

2240817	Taylor	Elizabeth	46
2240818	Mensah	Mary	47
2240819	Odoom	Magi	44
2240822	Panyin	Obaa	33
2240823	Kwesi	Kyeremanteng	50

RANDOM FILES

There is really not much difference between a sequential and a random file in C++. The main difference is not physical but rather the method used in accessing records and updating them. As mentioned earlier, sequential file requires that to search for a record you always have to start with the first record and search sequentially through to the end of the file to locate the required record. For a random file, it is possible to position the file pointer directly at the required record thus making record retrieval far faster than sequential file irrespective of where the record appears in the file. One other difference between a random and a sequential file in C++ is that, for a sequential file, the records may have variable length but for a random file the records must have fixed length. Thus for a random file, each record takes the same amount of disk space as such with random files one usually waste some disk space but this is usually compensated for by the speed with which records can be retrieved. One advantage of having a fixed record length is that it is possible to determine the amount of space required to store the records in advance and it is also possible to calculate the number of records if the total disk space used by the file is known.

Opening a random file

The way a random file is opened is same as that of a sequential file. As such the open statement is used with the same format, that is

```
randomFileVariable.open("filename.dat",mode);
```

where *randomFileVariable* is the name of the variable the file is connected to. The mode is also one or more of the following:

ios::app -- Opens the file and allows data to be appended or data additions at the end
ios::ate -- Opens the file and allows additions anywhere
ios::binary- - Causes the file to be opened in binary mode
ios::in- - specifies that a file is capable of input
ios::out- - specifies that the file is capable of output
ios::trunc -- Deletes everything in the file if it exist
ios::nocreate -- Does not open file if the file must be created
ios::noreplace -- Does not open file if the file already exists

Thus, if we want to create a random file by name *employee.dat* connected to the variable *empFile* then we can use the following line of code to open the file.

```
empFile.open("c:/employee.dat",ios::out);
```

To open the above file so that we can create the file and then update the records in it we can use the following statement:

```
empFile.open("c:/employee.dat",ios::out|ios::in);
```

The modes *in* and *out* indicate that we can read and also write to the file.

CHECKING IF A FILE OPENS SUCCESSFULLY

Just like sequential file, there is the need to check if a file was opened successfully especially if it is being opened as an output file or for append. The check is done in the same way as sequential file. Thus, to check if the *empFile* opens successfully or not, we can use the following statements

```
if(empFile.fail())
{
    cout<<"\n…cannot open file……\n";
    exit(1);
}
```

READING FROM A FILE

To read from a random file the *seekg*() function is used. The format of this function is as follows:

```
fileVariable.seekg(num_bytes, origin);
```

where *fileVariable* is the name of the file variable. The *num_bytes* is an integer value that specifies the number of bytes to skips in the file. Note that in C++ the number of bytes are not read but simply skipped. The *origin* is used to specify where to begin the skipping specified by the num_bytes. The origin can be one of the three values as follows:

Origin	equivalent	Description
SEEK_SET	ios::beg	Beginning of file
SEEK_CUR	ios::cur	Current record position
SEEK_END	ios::end	End of of file

The values of origin are defined in the header <cstdio>, thus one must include this header in his program if random files are used. The ios::beg, ios::cur and ios::end on the other hand are defined in the <fstream>.

C++ has other helpful I/O functions that enable programmers to perform more powerful I/O disk access. Each of these functions is contained in the fstream header.

read(array,count): This is a buffered I/O function that allows the data specified by count to be read into the array pointer specified by the array. One advantage of this function is that a single read() function can be used to read in much data.

write(array,count): This is also a buffered I/O function that allows the data specified by count to be written from the array pointer specified by the array. One advantage of this function is that a single write() function can be used to write much data.

remove("filename"). This function is used to erase a file on disk. If the operation is carried out successfully a 0 (zero) is return and -1 if an error occurs.

1-6.4 User Defined Header Files

So far we have been using 'inbuilt' or pre-defined header files such as **iostream.h** in almost all our programs. C++ allow programmers to write their own user defined headers that can be included in any program they intend to use the content of the header files. It will be a good practice for programmers to write their own header files so that all user defined functions, classes, structures, etc can be stored in this file such that they will not have to either code the same functions in different programs or cut and paste functions from one program to another. Writing a header file is as easy as writing any program. The way a header file is written is similar to the way any program in C++ is written. The content of a header file can be the same as the content of a program but without the main function and its block. For example, for any given program, if the main and its block are deleted, the file can be saved as a header file to be used in different programs if the need be. User defined header files should have the extension .h. In including a pre-defined header function we usually use the directive such as

```
#include<iostream.h>
```

The syntax for including a header file is similar to that of a pre-defined header file except that the symbols <and > are replaced with " and " respectively. Thus, the syntax for including a user defined header file is as follows:

```
#include "headerFile.h"
```

where *headerFile* is the name of the user defined header file. Note that a programmer's header file is treated similar to other user created files and hence why the name is usually enclosed in quotes.

We will now create a small user defined header file and write a small program to access the content of the header file.

Content of header file(myfunctions.h): We want the content of our header file to contain three user defined functions that we intend to use in a program. The functions will be called to return factorial of a given integer value, to return the average of two given integer values and also to return array elements sorted in ascending order. We shall call these functions factorial, average2 and Sort1DArray respectively. We will also call the header file myfunctions.h and its content may be as follows:

```
int factorial(int n);
double average2(int x, int y);
void Sort1DArray(int x[], int n);
```

```
int factorial(int n)
{
 int fact;
  if(n<0)
   {
    cout <<"Illegal call to factorial function....";
    return -1;
   }
  else
  {
    fact=1;
    for(int i=2;i<=n;i++)
     fact*=i;
  }
  return fact;
}


double average2(int x, int y)
{
return (double(x)+y)/2.0;
}

void Sort1DArray(int x[], int n)
{
  int j,i,temp;
  for(i=0;i<n;i++)
  for(j=0;j<n-i;j++)
    if (x[j]>x[j+1])
      {
        temp=x[j+1];
        x[j+1]=x[j];
        x[j]=temp;
      }
}
```

Program listing:

```
#include <iostream>
#include <cstdlib>
```

```
#include "myfunctions.h"
using namespace std;
int main()
{    int i,x[10],n;
     cout <<"Please Enter value of n: ";
     cin >>n;
     for(i=0;i<n;i++)
     {
      cout <<"Enter x[" <<i+1 <<"]: ";
      cin >>x[i];
     }
     Sort1DArray(x,n);
     cout <<"The sorted list will be as follows" <<endl;
     for(i=0;i<n;i++)
     cout << x[i] <<"\t";
     cout <<endl;
     cout <<"\nThe average of the 1st and last items (after sorting) is "
          <<average2(x[0],x[n-1]) <<endl;
     cout <<"\nThe factorial of " <<x[0] <<" is "
          <<factorial(x[0]) <<endl;
    system("PAUSE");
    return 0;
}
```

Program run:

```
Please Enter value of n: 10
Enter x[1]: 20
Enter x[2]: 8
Enter x[3]: 17
Enter x[4]: 90
Enter x[5]: 34
Enter x[6]: 7
Enter x[7]: 120
Enter x[8]: 110
Enter x[9]: 65
Enter x[10]: 23
The sorted list will be as follows
7    8    17    20    23    34    65    90    110    120

The average of the 1st and last items (after sorting) is 63.5
```

The factorial of 7 is 5040
Press any key to continue . . .

Self Assessment 1-6

1. Briefly distinguish between a sequential and random files

2. Explain how files can be used in a program

3. What is the difference between an identifier and a file variable?

4. Explain how you can create your own user defined header with atypical example.

SESSION 2-6: POINTERS

2-6.1 Using Pointers

Pointers are very useful in Computer programming. They are constructs that give programmers more control of the Computer's memory. A pointer points to the memory address of a variable. A computer's memory is made up of a number of small addressable units called bits. Eight of these form a byte. Every byte in memory has an integer value as it memory address. Memory addresses are sequentially numbered starting at zero and through to the maximum amount of memory that has been installed in the computer. If the amount of RAM installed on a computer is 64MB, then the computer will have the bytes in memory numbered 0, 1, 2, up to the power of 2 closest to 64,000,000. Note that 1MB is about 2^{20} = 1,048,576. During program execution, data and program instructions are stored in the RAM hence each data and instruction is identified by its unique memory address. Pointers have a number of uses and notable among these are for working with arrays, especially char arrays, for dynamic memory allocation of data structures and for linking data structures together. One advantage in using pointers is that they enable you to manipulate the address of variables without knowing their addresses in advance.

Though pointers are very useful care must be taken when using them as they can make a lot of things go wrong.

In fact, we have already make use of pointers when we treated user defined functions that are call by reference. For example, a user defined function with a function prototype such as **swap(int& A, int& B);** the type of the parameters A and B is not int but rather a pointer-to-int. In this case, when the swap function is called, it is the address of the variables that are passed or copied into the variables A and B.

USING A POINTER

Before a pointer can be used it has to be declared to point to a particular data type. Generally, pointers are given names in the same way as identifiers. To make things a bit easier, in all our programming examples, we will prefix variable names with the character p as pointer names. For example, if we want a pointer to point to a variable say Height, then we will use the name pHeight as the name of the pointer. Pointers are declared by placing an asterisk (*) just after the data type of the pointer. The general syntax for declaring a pointer is as follows:

```
dataType* pointerName;
```

where *dataType* is a data type and *pointerName* is the name of the pointer being declared. Pointers are generally initialised to null. For example,

```
int* px;
```

declares the variable px to be a pointer of integer type. Note that it is perfectly right to write the above example as int *px; or int * px;.

There are two major pointer operators and these are the ampersand sign (&) and the asterisk sign (*). With pointers, the * is called the indirection operator. The & is used to obtain the address of something be it data or variable. The * is used to get the content of a pointer. The operators + and – can also be used for pointers for addition and subtraction respectively.

Once a pointer has been declared, it can be used to store the address of a data or variable. The general form of storing address in a pointer is as follows:

```
pointerName = &variable;
```

where *pointerName* is the name of the pointer and variable is the name of the variable or the data whose address is to be stored.

```
Example:        xPointer = &x;
```

where *xPointer* is the name of the pointer and x is a variable. The xPointer must have been declared previously as a pointer to be of the same data type as the variable x. Thus, if x is an integer variable,

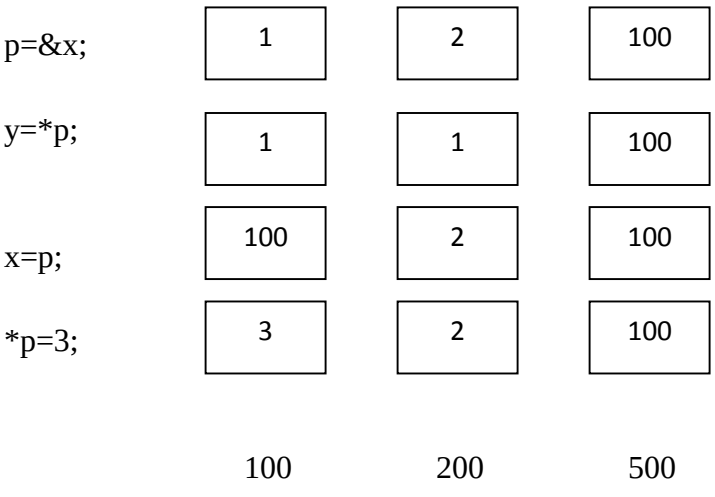
then pointers to point to x must also be integers. The above statement will cause the address of x to be assigned to the pointer, xpointer.

A pointer can also be used to get indirect access to the data stored at the address that it points to. To do so, the following syntax is used:

variable = *pointer;

where *variable* is the name of the variable to store the data being pointed to by the pointer. For example, Age =*pAge; means that the compiler should retrieve the data pointed to by the pointer pAge and store the retrieved data in the variable Age. Let us consider the following diagram to understand the idea of pointers. Let us assume that in a program we have the following declarations. We shall assume that x, y and p will have the memory addresses 100, 200 and 500 respectively. In the figure below, each rectangular box represent a memory unit. The value at the bottom left of a box gives the address of the memory while the value in a box gives the content of the memory unit.

```
int x,y;
int *p;
x=1; y=2;
```



Each of the following points explains what happens when each of the statement in front of boxes on any particular row is executed. Now, after the execution of x = 1and y = 2, with these values stored at address 100 and 200 respectively,

- the next statement (p = &x) tells the compiler to store the address of the variable x in p. Since x is at address 100, p will have the value 100, that is p is now pointing to x. This means that for line 1, only the content of address 500 will change to the address of x, that is to 100.
- The next statement, y = *p tells the compiler to retrieve the data pointed to by p and store the value in y. This means that the data 1 will now be the value of y and hence the content of the second box on line 2 will be one.
- The next statement (x = p) also tells the compiler to assign the current value of p to x. Note that before the execution of this statement, the value of p is 100 hence x will now have the value 100.
- The statement *p = 3 means that the memory address p points to should be changed so that p will now point to 3. Since before executing this statement p has the value of 100, it implies that the content of memory address 100 should be changed to 3.

Generally, when we have a variable x to have its value as 30 then x++ or ++x will increase the value in x by 1. However, this is not so with pointers. If a pointer p points to the memory address 100 then *p++ or ++*p will increase the value of pointer by the number of bytes used by the data type of the pointer. For example, a float number is stored using 4 bytes. Thus, p++ will increase the current address in p by 4. This means p will now points to address 104 and not 101. The right way of saying the value of a pointer has been increased by 4 is to say the pointer has been moved by four bytes.

2-6.2 Dynamic Arrays

So far we have used arrays declared to be of a fixed size. During program executions such arrays cannot have their size change. This tends to be a problem as one may not be too sure of exactly how many storage locations may be required. In most cases, programmers tend to declare large array size whereby most of them go wasted as they are never used. It is possible in C++ to use the new operator to create dynamic arrays. They are called dynamic arrays because they are created and destroyed during program execution or while a program is running. Note that it is not only arrays that can be created as dynamic. Strictly speaking there is nothing like static arrays. In C++ such arrays are referred to as automatic arrays this is because their dynamic properties are controlled automatically for the programmer. They are created and destroyed automatically when a function or a program runs to completion.

To declare a dynamic array (one dimensional)

(i) you must first define a pointer type using the typedef. The syntax of the typedef is as follows:

typedef pointerType* pointerName;

where *pointerType* is the same as the type of variable it will point to. The *pointerName* is the name of the pointer.

(ii) You must also define a pointer variable to point to the dynamic array in memory. This will also serve as the name of the dynamic memory. To do so the syntax is as follows:

pointerName ArrayName;

(iii) The dynamic array is then created using the new operator as follows:

ArrayName= new ArrayType[size];

where the *ArrayType* is the same as the *pointerType* in (ii). The size of the dynamic array is given in square brackets as shown above.

It is always advisable to check that the dynamic array was created successfully. If for example, one computer runs out of memory then it will not be possible for the dynamic array to be created and hence why it is necessary to check. If the creation was not successful then array name used in (iii) will be NULL. Once a dynamic array has been created it is used like any other ordinary array. Its elements are referenced or accessed in the same manner as ordinary arrays. After a dynamic array has been used, it must be destroyed. Destroying a dynamic array simply means releasing the memory allocated so that the memory can be reused if the need be. The delete operator is used in destroying dynamic arrays. The syntax is as follows:

delete [] ArrayName;

where *ArrayName* is the name of the dynamic array.

Let us consider the following example:

```
typedef  int* xPointer;
xPointer x;
x = new int[10];
if(x==NULL)
  {cout <<"Unable to create array, insufficient memory"
    exit(1)
  }
  :
delete [] x;
```

The above statements define the pointer XPointer. A pointer variable x is then declared. Next the x is allocated 10 memory locations. A check is made using the if statement to determine whether the dynamic array creation was successful or not. If the creation was unsuccessful a message is displayed to the effect. Finally, the dynamic array, x is deleted after use.

MULTIDIMENSION DYNAMIC ARRAYS

Under this topic we shall discuss only two dimensional arrays as most programmers generally use either one dimension or two dimensional arrays. We have already seen how to declare a one dimensional array. The way a two dimensional dynamic array is declared is not much different from the way a one dimensional dynamic array. For simplicity, we shall use the following format in declaring a two dimensional dynamic arrays. Lets assume that the variables rows and columns have been used to accept the number of rows and columns of a matrix and that we now want to declare rows by columns array. Assume the rows and columns are of data type int and the array to be declared is A, then we can use the following statements in declaring the array A.

```
int (*A)[cols];
 A=new int[rows][cols];
```

The main problem of the above declarations is that if you are using the Bloodshed Dev C++ compiler then everything is fine but then if you are using other C++ compilers such as the Visual C++, then you are required to declare the cols as a constant before it is used. Let us consider the following program.

```
#include <iostream>
#include <cstdlib>
using namespace std;
#define print cout <<
#define input cin>>
int main()
{
 int rows,cols,i,j;
 print "Please enter the number of rows in first matrix: ";
 input rows;
 print "Please enter the number of columns:  ";
 input cols;
 int (*A)[cols];
 A=new int[rows][cols];
 for(i=0;i<rows;i++)
  for(j=0;j<cols;j++)
    {
      print "Enter A[" <<i+1 <<"][" <<j+1 <<"]: ";
      input A[i][j];
    }
print "output\n";
 for(i=0;i<rows;i++)
  {
  for(j=0;j<cols;j++)
  {
      print A[i][j]*5 <<"\t";
```

```
 }
 cout <<endl;
 }
system("PAUSE");
return 0;
}
```

The above program works perfectly well when using the Bloodshed C++compiler. The user is first prompted for the number of rows and columns of the matrix. The values entered are then used to declare the array. Thus, the above program will enable a user to enter the number of rows and columns for any arrays, allow the array elements to be entered and then output 5 times the array elements. A typical program run is as follows:

```
Please enter the number of rows in first matrix: 3
Please enter the number of columns: 4
Enter A[1][1]: 1
Enter A[1][2]: 2
Enter A[1][3]: 3
Enter A[1][4]: 4
Enter A[2][1]: 5
Enter A[2][2]: 6
Enter A[2][3]: 7
Enter A[2][4]: 8
Enter A[3][1]: 9
Enter A[3][2]: 0
Enter A[3][3]: 2
Enter A[3][4]: 4

output
5       10      15      20
25      30      35      40
45      0       10      20
Press any key to continue . . .
```

If the same above program is run using the Visual C++ compiler, the following error messages will be displayed.

```
--------------------Configuration: dynamicArrays - Win32 Debug--------------------
Compiling...
dynamicArrays.cpp
c:\myc++book\dynamicarrays.cpp(12) : error C2057: expected constant expression
```

```
c:\myc++book\dynamicarrays.cpp(12) : error C2466: cannot allocate an array of constant
size 0
c:\myc++book\dynamicarrays.cpp(13) : error C2540: non-constant expression as array
bound
c:\myc++book\dynamicarrays.cpp(13) : error C2440: '=' : cannot convert from 'int (*)[1]'
to 'int (*)[]'
        Types pointed to are unrelated; conversion requires reinterpret_cast, C-style cast or
function-style cast
c:\myc++book\dynamicarrays.cpp(18) : error C2036: 'int (*)[]' : unknown size
c:\myc++book\dynamicarrays.cpp(25) : error C2036: 'int (*)[]' : unknown size
Error executing cl.exe.

dynamicArrays.obj - 6 error(s), 0 warning(s)
```

As can be seen from the error messages, the problem is with line 12 (the bolded line in the above program). If we rewrite the program as follows in introducing the bolded line the program works as exactly as above, hence you need to check with your compiler as to whether the number of columns in a dynamic array should be constant or not.

Program listing

```
#include <iostream>
#include <cstdlib>
using namespace std;
#define print cout <<
#define input cin>>
int main()
{
 int rows,cols, i,j;
 const int colos = 100;
 print "Please enter the number of rows in first matrix: ";
 input rows;
 print "please enter the number of columns: ";
 input cols;
 int (*A)[colos];
 A=new int[rows][colos];
 for(i=0;i<rows;i++)
  for(j=0;j<cols;j++)
    {
      print "Enter A[" <<i+1 <<"][" <<j+1 <<"]: ";
      input A[i][j];
```

```
   }
print "output\n";
for(i=0;i<rows;i++)
 {
 for(j=0;j<cols;j++)
 {
    print A[i][j]*5 <<"\t";
 }
 cout <<endl;
 }
system("PAUSE");
return 0;
}
```

Program run

```
Please enter the number of rows in first matrix: 3
please enter the number of columns: 3
Enter A[1][1]: 1
Enter A[1][2]: 2
Enter A[1][3]: 3
Enter A[2][1]: 4
Enter A[2][2]: 5
Enter A[2][3]: 6
Enter A[3][1]: 7
Enter A[3][2]: 8
Enter A[3][3]: 9
output
5       10      15
20      25      30
35      40      45
Press any key to continue . . .
```

2-6.3 Pointers and Functions

In a functional definition, whenever an array is passed as an argument it is actually a pointer that is passed. Passing an array actually involving passing the base address of the array and not the array elements themselves. Generally, if x is the name of an array and PolyNom is the name of a function, then it is legal to call the function PolyNom by passing the array x as follows:

```
void PolyNom(int x[]);
```

Now, instead of passing the array name, we can pass the base address. As such, the above statement can be written as follows:

```
void PolyNom(int *x);
```

The following program illustrates how the base address of an array is passed to a function to access the array elements. The program is just to compute the standard deviation of 10 numbers to be entered at the keyboard.

```
#include<iostream>
#include<cmath>
using namespace std;
void stdv(int y[], int m);
int main()
{
        int x[10],* xp;
        int n=10;
        xp=x;//same as xp=&x[0];
        for(int i=0;i<n;i++)
                cin>>x[i];
        stdv(xp,n); //same as stdv(x,n);
        return 0;
}

void stdv(int* y, int m) //same as stdv(int y[],int m);
{
 double mean,sum=0;
 for(int i=0;i<m;i++)
        sum=sum+y[i];
 mean=sum/m; sum=0;
 for(i=0;i<m;i++)
   sum=sum+pow(y[i]-mean,2);
 cout <<"The standard deviation of the above numbers is "
         <<sqrt(sum/m) <<endl;
}
```

2-6.4 Pointers and Arrays

An array name in C++ is in fact a constant pointer that is initialised to this base address hence one can say an array name by itself is an address. When an array is declared the compiler allocates enough memory locations for the array elements. If for example, the element at location zero of the array has the storage or memory address to be 1000, then for a system like mine that uses four

bytes to store an int variable, the remaining array elements will be stored at locations 1004, 1008, 1012 etc. Rather than using a subscript to access the array elements, it is possible to use a pointer for the same purpose. To move a pointer from one array element to another, all that is needed is to increase the pointer value by one and not by the number of bytes that are used to store each value of the array. Let us consider the following program:

```
#include<iostream>
using namespace std;
int main()
{
        int x[10], * xp;
        int n=10;
        for(int i=0;i<n;i++)
                cin>>x[i];
        for(xp=&x[0];xp<=&x[9];xp++)
                cout <<xp <<"\t" <<*xp <<endl;
        cout <<endl;
        return 0;
}
```

The above program declares an array, x of size 10. A subscript i is used to accept the elements of the array from the keyboard using the first for statement. The second loop first stores the address of the array elements in the pointer xp, it then display the address at which the element was stored and the value stored at the address. If the above program is keyed and run, the following output will be obtained.

Input :10 20 30 4 50 6 70 21 100 16

Output:

```
0x0012FF50      10
0x0012FF54      20
0x0012FF58      30
0x0012FF5C       4
0x0012FF60      50
0x0012FF64      6
0x0012FF68      70
0x0012FF6C      21
0x0012FF70      100
0x0012FF74      16

Press any key to continue
```

As can be seen from the output, the address of an element is 4 bytes more than that of its predecessor. The address 0x0012FF50 is the base address of the array. The addresses are given in hexadecimal, which is base 16. Please note that xp=x and xp=x[0] are equivalent and will store the base address of the array to xp. Note that running the above program a number of times may not necessarily give the same memory address as the allocation of the base address is more or less done at random.

Even though pointers and array names are treated very similar, there is one basic difference and that is array name is a constant pointer and not a variable hence if x is an array name then expressions such as x=xp and x++ are illegal as attempts are being made to alter the constant value while for a pointer xp the expressions xp=a and xp++ are legal. In simple terms, array names cannot change in a program while pointer values can or be modified.

POINTERS TO POINTERS

The C++ language allows a pointer to point to another pointer just as a pointer points to a memory location. To do this one must add (*) to each higher level of a pointer. For example, if p is a pointer to a variable x, then p is usually declared using a single asterisks as *p, however if q is to point to p then q is at a higher level than p hence q will be declared as **q. Let us consider a typical example by assuming the following program:

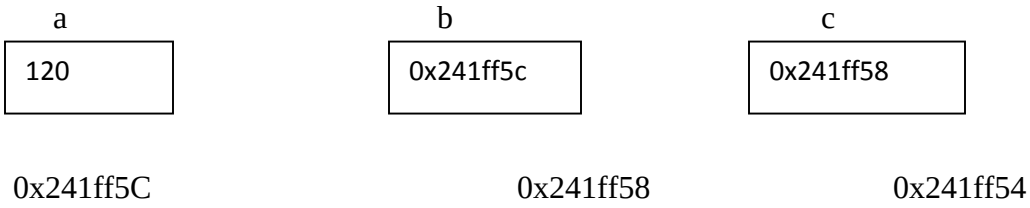
Program listing

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    int a;
    int *b;
    int **c;
    a=120;
    b=&a;
    c=&b;
    cout<<"The number stored in a is " <<a <<"." <<endl;
    cout <<"The address of a stored in b is " <<b <<endl;
    cout<<" and the address of  (b)  " <<b
       <<" is " <<c <<"\nand c is stored at address " <<&c <<endl;
    system("PAUSE");
    return 0;
}
```

Program run

The number stored in **a** is 120.
The address of a stored in **b** is 0x241ff5c
and the address of (**b**) 0x241ff5c is 0x241ff58
and **c** is stored at address 0x241ff54
Press any key to continue . . .

The above information in the Computer's memory will look like as follows:



Each of the above boxes represents a memory cell or location. The value below the boxes are the addresses at which the items a, b and c respectively are stored. The values within the boxes specify the value stored at any particular memory location. Note that when you run the above program on your PC, your memory address may not be the same as mine. Even running the same program on my PC at a later date may not give me the same memory addresses as above since the above memory locations may have been used by another program also running at the same time.

POINTERS AND STRUCTURES

The C++ compiler allows structures to be pointed by pointers. This ability enables structures such as links, stacks and queues to be implemented easily in C++. The pointer must be declared as a pointer to the structure. Let us consider the following example:

```
struct commodity
{
        char name[40];
        char desc[40];
        int quantity;
        int cost;
};
```

For the above structure commodity, we can define a pointer to point to the structure as follows if commp is the name of the pointer and that goods is of the type commodity.

```
commodity goods;
commodity * commp;
```

We can initialise the pointer to point to goods as follows:

commp = &goods;

We can then use the reference operator (->) usually used exclusively with pointers to structures and pointers to classes. The use of -> allows us to avoid the use of parenthesis in referencing the members of a structure. For example, to access or refer to the field name, we can write goods->name which is equivalent to (*goods.name). The goods->name is interpreted as the member name name of the structure goods. The following table throws more light on the use of -> and parenthesis with structures.

Expression	Meaning	Equivalent to
goods.name	Member, name of structure goods	goods->name
*goods.name	Data pointed to by member, name of structure goods	*(goods.name)
*(goods.name)	Data pointed to by member, name of structure goods.	*goods.name
goods->name	Member, name of structure goods.	goods.name

To use the getline member function to accept data from the keyboard into the member name, the required statement is as follows:

cin.getline(goods->name,40)

and for using just cin we have

cin>>goods->name;

POINTERS AND CLASSES

Just as structures can be pointed to by pointers, classes can also be pointed to by pointers. Similarly, we can refer directly to members of a class by using the reference operator, ->. Let us consider the following class definition for cars.

```
class car
{
    Private:
        char make[20];
        char model[20];
        int year;
```

```
        own ownerName[40];
        char RegNo[10];
        insure insurers[40];
    public:
      void getCarInfo();
      void storeCarData();
      void ProcessCarInfo();
    };
```

If we define a variable commercial to be of the type car and pointer to car to be commerp, that is

```
car commercial;
car *commerp;
commerp=&commercial;
```

then we can refer to the member variables or elements make and year for example as follows:

```
commerp->make;
commerp->year;
```

The member function are also referenced in the same manner. For example, to call the member function getCarInfo() will be given as follows:

```
Commerp->getCarInfor();
```

NAMESPACES

Namespaces provide a means of defining a set of global classes, objects and/or functions under a common name. Normally programs written by different people do have name clashes when they are combined. The use of namespaces in C++ help to resolve problems like this. The syntax for declaring a namespace is as follows:

```
namespace identifier
{
  Namespace-definition;
}
```

where *identifier* is any valid identifier and namespace-definition is the definition of the namespace. For example, the following defines a namespace called TrustMe.

```
namespace TrustMe
{
  int  CostPrice, SellingPrice;
}
```

Once a namespace is defined, its members can be accessed by using the scope resolution operator. For example, to access the variables CostPrice and SellingPrice outside the TrustMe namespace will be as follows:

```
TrustMe::CostPrice;
```

TrustMe::SellingPrice;

Let us consider the following program:

```
#include <iostream>
#include <cstdlib>
using namespace std;
namespace toffee
{
  double price=1500.00;
}
namespace chocolate
{
  double price = 2500.05;
}
int main()
{
    cout <<"5 boxes of toffee and 2 boxes of chocolates sell at ";
    cout <<(5*toffee::price+2*chocolate::price) <<endl;
    system("PAUSE");
    return 0;
}
```

The above defines two global variables price. Note that the price under namespace toffee is different from the one under chocolate. The price is used under namespace to initialise the variable to the selling price of the item (in this case representing the name spaces toffee and chocolate). To use the price under the namespace it was defined, there is no need to use the scope resolution operator. It is only used when a namespace variable is being used outside the namespace under which it was declared. As mentioned above, namespace members are global.

'USING NAMESPACE'

The namespace can be preceded by the reserve word using in a function so that objects and functions can be accessed directly. The general form of using namespace is as follows:

```
using namespace nameSpace;
```

where nameSpace is the name of the namespace whose members are to be accessed directly. Let us illustrate this by way of example:
program example:

```
#include <iostream>
#include <cstdlib>
```

```
using namespace std;
namespace toffee
{
  double price=1500.00;
}
namespace chocolate
{
  double price = 2500.05;
}
int main()
{
    cout <<"5 boxes of toffee and 2 boxes of chocolates sell at ";
    using namespace toffee;
    cout <<(5*price+2*chocolate::price) <<endl;
    system("PAUSE");
    return 0;
}
```

program output

```
5 boxes of toffee and 2 boxes of chocolates sell at 12500.1
Press any key to continue . . .
```

Comments

In the above program, the bolded line instructs the compiler to assume references to the member price without a prefix of a namespace to be that of the namespace toffee. Since the bolded price is not preceded with the name of any namespace, it is assumed to be that of toffee hence the value 12500.1 (5*1500+2*2500.05). Note that there can be as many using namespace as may be needed but care must be taken to ensure that the correct member is used. In the above example removing the chocolate:: from the prefixed price will give the result 10,500 (that is 5***1500**+2***1500**).

When a using namespace, mine is used in a program and another namespace, yours is used after mine and assuming that both mine and yours have a member by name church, then the compiler will indicate an error when the church is used without being preceded with any namespace. In the following program, you may think its correct and will work but this is not the case. The compiler will indicate that the use of 'price' is ambiguous. This is because by using the using namespace for both toffee and chocolate we are trying to tell the compiler that their members can be referenced directly. However, since they both have the same member, price there is the need to precede each of them with its namespace.

```
#include <iostream>
#include <cstdlib>
using namespace std;
```

```
namespace toffee
{
  double price=1500.00;
}
namespace chocolate
{
  double price = 2500.05;
}
int main()
{
     cout <<"The output when 'using namespace toffee'  " <<endl;
     cout <<"5 boxes of toffee and 2 boxes of chocolates sell at ";
     using namespace toffee;
     cout <<(5*price+2*price) <<endl;
     using namespace chocolate;
     cout <<"The output when 'using namespace chocolate'  " <<endl;
     cout <<"5 boxes of toffee and 2 boxes of chocolates sell at ";
     cout <<(5*price+2*price) <<endl;
     system("PAUSE");
     return 0;
}
```

The above program can only compile when the prices in the bolded line are preceded with the appropriate namespace such as

```
cout <<(5*chocolate::price+2*toffee::price) <<endl;
```

ALIAS NAMESPACES

C++ provides the possibility of defining alternative means of existing namespaces. The syntax is as follows:

```
namespace newName = existingNamespace;
```

where *newName* is the alternative name of the existing namespace. The *existingNamespace* is the name of the existing namespace.

TEMPLATES

Function templates allow one to create generic functions capable of accepting any type of data and also returning results of any data type. This makes it possible to eliminate function overloading. I hope you do remember functions overloading where you can have two or more functions with the same name, identical definitions and just a minor differences in the number of arguments or parameters and the data types. Thus, the use of templates eliminate the need to have two or more

functions having the same name in a program. Templates are identified by the reserve word *template*. There are two main forms of defining a template and these are as follows:

template <class identifier> template_definition;

or template <typeName identifier> template_definition;

where identifier will later be used as a valid data type in declaring other variables. The template_definition are the necessary declarations and instructions that make up the template definition. To call a function the syntax is as follows:

template_function <dataType> (parameters);

where *dataType* is a valid predefined or user defined data type. *Parameters* are the list of parameters to be passed on to the template. The parameters in the parameter list should be separated by commas.

To throw more light on templates let us consider the following overloading functions:

```
void swap(int a, int b)
    {
        int temp;
        temp=b;
        b=a;
        a=temp
    }

void swap(double a,double b)
    {
        double  temp;
        temp=b;
        b=a;
        a=temp
    }

void swap(char  a,char b)

    {
        char temp;
        temp=b;
        b=a;
        a=temp
    }
```

If you look at the three functions, they are very identical. The only difference between them is that one is called when we want to interchange the content of two integer variables, the other when we

want to interchange content of two double variables and the last one is called when we want to swap the content of two char variables. Instead of defining three different functions, we can use template and define just one function which we can call when we want to swap integers, doubles, chars or even variables of other data type. For example we can define a template for the above functions as follows:

```
template <class anyType>
void  swap(anyType a, anyType b)
{
        anyType temp;
        temp= b;
        b=a;
        a=temp;
}
```

You can include the above template in any program that you will have to swap the content of two variables exactly as above and will work without any modification. Please note that to use the above function to swap the content of two variables in any program, the two variables must be of the same data type else an error or warning message may be displayed. Note also that to use a template function in a program, the template prototype and the template header must begin with the line

```
template <class typeParameter>
```

where the *typeParameter* is to be used in the body of the template function like any other data type. The class in the above defines the type of the parameter. When the swap template is called, the compiler will notice the type of the parameters, it will then use the template to create a function definition with the type parameter anyType replaced with data type of the passed parameters.

Let us consider the following program.

```
#include<iostream>
using namespace std;
template <class MyType>
MyType MyType1(MyType a, MyType b, MyType c)
{
        if (a>b&&b>c)
                return (a+b);
        else if (a>b&&b<c)
          return (a+c);
        else
        return (b+c);
}
```

```
int main()
{
        int x=2,y=4,z=8;
        double t=1.2,d=2.5,e=1.9;
        float m=4.0,n=11.3,p=0.08;
        int k=MyType1<int> (t,d,e);
        double f =MyType1<double>(x,y,z);
        int r=MyType1<int>(m,n,p);
        cout <<r <<"\t" <<k <<"\t" <<f;
}
```

In the above program, MyType is declared to be a template. The template function MyType1 is then defined to be of the type MyType and takes three parameters when called. The function of the template function is to return the sum of two of the values passed depending on the condition that the three numbers satisfy. In the definition of the main function, the template function is called three times. The first call passes to the template function three double values but then the <int> in the call tells the compiler to treat the parameters as integers and return the result to the integer variable k. Similarly, the second call passes three integer variables and instruct the compiler to treat the variables as doubles and return a double result to the variable f. When the above program is run, the following will be displayed to the screen.

```
11    3    12
Press any key to continue
```

Let us explain the results. For the 11, the float values 4.0, 11.3 and 0.08 are passed to the MyType1. The call suggests that the parameters should be treated as integers hence 4, 11 and 0 are used respectively. From the definition of the function, since 4 is not greater than 14, the sum of 11 and 0 are to be returned and hence why 11 is returned. For the 3, the double values 1.2, 2.5 and 1.9 by the function call are to be treated as integers hence they are converted to integers as 1, 2 and 1 respectively. Again, since the first 1 is not greater than 2, the result returned is the sum of 2 and the 1 following it and that gives 3 hence the 3.

Note: If you should use Visual C++ to compile the above program it will give zero error and nine warnings. However, if you build after that and recompile, the warnings disappear which suggest that the above is completely right. Compiling the same program using the bloodshed C++ will give no error and no warning messages.

Please note that once a template is declared, it can be used in structures and classes. It can also be used to define template functions that can allow any two or more different parameter types to be passed. For example, to define a template that can allow two different data types to be passed, we can have say

```
template <class Type1, class Type2>
```

where *Type1* and *Type2* represent different data types. Let us consider the following program.

```
#include <iostream>
#include <cstdlib>
using namespace std;
template <class MyType, class YourType>
MyType MyType1(MyType a, MyType b, YourType c)
{
        if (a>b&&b>c)
                return (a+b);
        else if (a>b&&b<c)
          return (a+c);
        else
        return (b+c);
}
int main()
{
        int x=2,y=4,z=8;
        double t=1.2,d=2.5,e=1.8;
        float m=4.0,n=11.3,p=0.08;
        int k=MyType1<int> (t,d,m);
        double f =MyType1<double>(x,y,t);
        int r=MyType1<int>(m,n,z);
        cout <<r <<"\t" <<k <<"\t" <<f <<endl;

    system("PAUSE");
    return 0;
}
```

According to the template definition, the template has two different data types, however the data type of the first two parameters are the same hence when calling it we have to ensure that the data types of the first two parameters are different. A run of the above program gives the following result. See if you can explain the result. The bolded variables are the variables of data types different from the others.

```
19      6       5.2
Press any key to continue . . .
```

Exercise: C++ uses the functions abs, labs and fabs to return the absolute value of an integer, double or float value respectively. Write a function template that can return the absolute value of any number value passed to it, be it integer, double or float.

EXERCISE : HARD WORKS BREAKS NO BONE COMPANY wants a computerized system for calculating taxes on imported items that pass through their hands. The following information are first to be recorded on an invoice and then handed over to a DATA Entry Clerk to be fed into the Computer for the tax to be calculated.

Name of importer
Address of importer
Classification of importer (G for Ghanaian)
and Total value of items imported.

The processes involved are as follows:

- If an importer is a Ghanaian, the tax payable is one-eighth of the total value of the items imported.
- If an importer is a Foreigner, then the tax is calculated as follows:
  25% of the first ¢2000 of the total value of the items
  20% of the next ¢2500 of the total value of the items
  30% of the next ¢7543 of the total value of the items
  10% of the excess over ¢12043 of the total value of the items.

Assuming you have been employed by the above institution or company and you are required to computerize this tax calculation, write a C++ program for this Computerization if they expect to get the report given below as output for each importer on paper:

**** HARD WORKS BREAKS NO BONE COMPANY LIMITED ****
NAME OF IMPORTER **********
ADDRESS OF IMPORTER **********
TOTAL VALUE OF IMPORTED ITEM(S) **********
TAX ON IMPORTED ITEM(S) **********
DATE day Month, Year

(CASHIER'S SIGNATURE)

QUESTION: Given that the first of January 1800 is Wednesday and that the years 1800 and 1900 are not leap years (although they have all the properties of a leap year) write a program that will accept as input a year (such as 1993) and output the calendar for that year.Your program should be written such that it works for all years between 1800 and the year 3000. You should actually write two program, one without using user defined functions and the other using user defined functions.

Program listing: We will give only the answer for without using any user defined functions

```
#include <iostream>
#include <iomanip>
using namespace std;
#define sline "--------------------------------------------------\n";

int main()
{
 /* We shall store the number of days in each month in an array days and the first
    first three letters of each month's name in an array mths. The variable start will
    be used to track the day of the week on which the first of January of the required
    year falls.
 */
 int days[]={31,28,31,30,31,30,31,31,30,31,30,31};
 char
mths[12][4]={"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"};
 int year, start=4;
 cout <<"Please enter the year required for calender: ";
 cin>> year;
 for(int yr=1800;yr<year;yr++)
 {
    if(yr%4==0 && yr!=1800 && yr!=1900) //check if yr is a leap year
     start+=2;
    else
      start++;
 }
  start=start%7; if(start==0) start=7;
  if(year%4==0 && year!=1800 && year!=1900)
    days[1]++; //increase number of days in February by 1 if year required isa leap year.
  for(int i=1;i<13;i++)
  {
   //Print the name of the month, year and the names of the week days.
```

```
   cout<<"\n\n\n\t\t\t" <<mths[i-1] <<"  " <<year <<"\n";
   cout <<"Sun\tMon\tTue\tWed\tThu\tFri\tSat\n";
   cout<<sline; // underline the names of the week days
   for(int k=1;k<start;k++)
      cout <<"\t";
    for(int j=1;j<days[i-1]+1;j++)
    {
     cout <<setw(3) <<j <<"\t"; start++;
     if(start>7)
       {start=1;cout <<endl;}
    }
  }
  cout <<endl;
  system("pause");
  return 0;
}
```

Exercise

Rewrite or modify the above program so that the output will appear exactly in the format above, that is having two months juxtapose.

Exercise

Write a program that will accept as input a nine digit positive integer. The first four digits indicate a year such as 1992. The next two digits indicate a month, with 01= January, 02 = February, etc. The next digit has a value of 1 through 7 and indicates the day on which the first of the month in question falls, with 1= Sunday, 2 = Monday, etc. The last two digits indicate the number of days in the month.

You are required to print a calendar for the required month. If for example the input is 199204430, then your output should be as shown below:

APRIL 1992

SUN	MON	TUE	WED	THU	FRI	SAT
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

QUESTION

A pair of integers are said to be AMICABLE if the sum of all the divisors of each of the numbers (excluding the number itself) equals the other. Thus, the integers 1184 and 1210 are amicable. The divisors of 1184 are 1, 2, 4, 8, 16, 32, 37, 74, 148, 296 and 592 which sum to 1210 whiles the divisors of 1210; 1, 2, 5, 10, 11, 22, 55, 110, 121, 242 and 605 also sum to 1184.

You are required to write a program that accepts pairs of integers as input and output AMICABLE if a pair is amicable and NOT AMICABLE if otherwise.

Program listing

```
#include<iostream>

using namespace std;
int main()
{
 int n1,n2,sumn1=0,sumn2=0,i;
 do
 {
  system("cls");
  cout<<"Please enter two positive integers: ";
  cin >>n1 >>n2;
 }while (n1<0||n2<0);
 system("cls");
 for(i=1;i<=n1/2;i++)
   if(n1%i==0)
     sumn1=sumn1+i;
 if(sumn1!=n2)
  cout <<"The numbers " <<n1 <<" and " <<n2 <<" are NOT Amicable";
 else
 {
   for(i=1;i<=n2/2;i++)
     if(n2%i==0)
       sumn2=sumn2+i;
 if(sumn2!=n1)
    cout <<"The numbers " <<n1 <<" and " <<n2 <<" are NOT Amicable";
 else
    cout <<"The numbers " <<n1 <<" and " <<n2 <<" ARE Amicable";
 }
 cout <<endl;
 system("pause");
 return 0;
```

}

QUESTION

The Julien Calendar is a method of assigning an integer to every day of the year e.g. January 30 will be 30, February 1 will be 32, December 31 will be 365 or 366 for leap years.Write a program that will accept as input a date (an 10-digit string) in the form DD MM YYYY where DD is the day, MM the month and YYYY the year, thus 22 12 1988 will mean 22nd December 1988, and output the date in Julien Calendar form. Your program should ensure that the input string is a valid date.

Program listing

```
#include<iostream>
using namespace std;
int main()
{
 int mths[12]={31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
 int days=0,d,m,y;
 char dd[3],mm[3],yy[5];
 do
 {
   cout <<"Please enter a valid 10-digit string in the form dd mm yyyy: ";
   cin>>dd >>mm >>yy;
   d=atoi(dd);m=atoi(mm);
   y=atoi(yy);
   if(y%4==0&&y!=1800&&y!=1900) mths[1]++;
 }while((m<1||m>12)||(y<1900||y>9999)||(d<1||d>mths[m-1]));

 for(int i=0;i<m-1;i++)
  days=days+mths[i];
 days=days+d;
 cout <<"The Julien number for the date " <<d <<"/" <<m <<"/" <<y
      <<" is " <<days;
 cout <<endl;
 system("pause");
 return 0;
}
```

Program run

Please enter a valid 10-digit string in the form dd mm yyyy: 01 03 2008

The Julien number for the date 1/3/2008 is 61
Press any key to continue . . .

QUESTION

An ICE CREAM producer, JIMMY has a number of sales agents who sells his ice cream. A salesman is expected to return to Jimmy the total amount of sales that he makes at the end of the day. Jimmy has a computerised database package that allows him to enter a sale made by a salesman directly into his Computer. Salesmen are given their commissions at the end of the week. Jimmy wants a program that can make use of his weekly database file to produce the following output.

SALESMEN'S SUMMARY FOR THE WEEK

SALESMAN	TOTAL	COMM	BALAN	NO. OF DAYS
:	:	:	:	:
:	:	:	:	:
:	:	:	:	:
:	:	:	:	:

Note that the order in which salesmen send their sales varies from one day to another and it is not a must that a salesman makes a sale on every blessed day. The commission is calculated as 15% of the total sales that one makes.

Program listing

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include<cstring>
#include <iomanip>
using namespace std;
ifstream fcin;
ofstream fcout;
void getInput(void);
void sortInput(void);
void computeCommDisp(void);
void displayOutput(void);
int code[1000];
char name[1000][15];
double sales[1000];
```

```
int NOS=0,totdays=0;//NOS is number of sales

int main()
{
  fcin.open("c:/myc++book/input.dat");
 if(fcin.fail())
  {
    cout <<"\nError opening input file, please check file name\n";
    exit(1);
  }
  fcout.open("output.dat");
  getInput();
  sortInput();
  computeCommDisp();
  cout<<"\n_____________________________________________________________"
        <<"_____________\n";
  cout <<"\n\t\t\t\t\tThe number of sales =" <<setw(10) <<totdays <<endl;
  cout<<endl;
  system("PAUSE");
  return 0;
}

void getInput()
{
 while (fcin >>code[NOS] >>sales[NOS] >>name[NOS])
    NOS++;

}

void sortInput(void)
{

  for(int i=0;i<NOS-1;i++)
   for(int j=0;j<NOS-i-1;j++)
      if(code[j]>code[j+1])
       {
         int temp=code[j+1];
         code[j+1]=code[j];
         code[j]=temp;
         double stemp=sales[j+1];
         sales[j+1]=sales[j];
         sales[j]=stemp;
         char st[15];
         strcpy(st,name[j+1]);
```

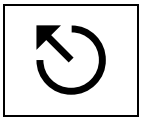
```
          strcpy(name[j+1],name[j]);
          strcpy(name[j],st);
         }
}

void computeCommDisp(void)
{
 int i=0,hold=code[0], days=0;
 char nme[15];
 strcpy(nme,name[0]);
 float sum=0;
 cout <<"\t\t SALESMAN'S SUMMARY FOR THE WEEK\n";
  cout <<"NUMBER   NAME\t\tTOTAL\t\tCOMM \t   BALANCE\tNO. OF DAYS\n";
 cout<<"_____________________________________________________________"
     <<_____________________\n";
 while (i<NOS)
 {
  while (code[i]==hold && i<NOS)
  {
   sum=sum+sales[i];
   i++;
   days++;
  }

   cout <<hold <<"\t" <<nme << "\t" <<setw(10) <<ceil(sum) <<".00"
        <<"\t" <<setw(10) <<ceil(sum*0.15) <<".00" << "\t"
        <<setw(10) <<ceil(sum-sum*0.15) <<".00"
        <<"\t" << setw(7) <<days <<endl;
   sum=0;totdays+=days;
   days=0;
   hold=code[i];
   strcpy(nme,name[i]);
 }
   if(i!=NOS)
   {
     sum=0.15*sum;
     cout <<hold <<"\t" <<name[NOS] << "\t" <<ceil(sum) <<".00" <<endl;
   }
}
```

Self Assessment 2-6

1. How are pointers used in a program?

2. What are the difference(s) between an array and a pointer?

3. Why is it important to use dynamic arrays?



Unit Summary

1. In this unit we have seen the importance of files. The two main types of files discussed are the sequential and the random files. The sequesntial is one in which the records are accessed in the order as they were stored based on a unique key that uniquely identifies a record. In looking for a particular record, the searching normally starts with the first record and through to the required record. By so doing, a sequential file normally leads to high seek time. Seek time is defined as the total time required to locate a desired record. Random files are those for which it is possible to position the file pointer directly at the required record thus making record retrieval far faster than sequential file irrespective of where the record appears in the file.

2. We have also learnt that rather than including specific user defined functions in a program, it is always a good idean to place all user defined function in a common hearder file so that the header file can be included in any program that requires one or more of the functions in the user defined file. By this way a lot of time is saved from having to re-write or copy and paste user defined functions from one program into another.

3. We have also learnt in this unit about pointers that they are constructs that give programmers more control of the Computer's memory. A pointer points to the memory address of a variable.

4. Before this unit, whenever we wanted to use an array, we will have to assume that size of the array and try to work within the size. This has the tendency of either wasting a lot of memory locations because only a small section of the assumed size iend up being used or forcing program execution to be terminated because an attempt is made at referencing elements with a subscript being out of range. Dynamic arrays solves these problems as the array and the size is created during runtime.

Course Summary

We started with this course by first looking at the five generations of programming languages. We also saw that as we move from one generation to the other the main rationale was to make programming less time consuming and very easy to understand. This was followed the different program translators, that is assembler, interpreter and compiler. We also that learnt that no matter good we may be at programming we are likely to make errors in our programs. These errors can be either syntx, logic or runtime rrors.

We have also learnt a number of C++ commands, expressions and other object oriented concepts

Selected Answers to Self Assessments and Unit Assignments

Course Quiz/Exams

Multiple Choice Questions (MCQs)

1. Which of the following can best be used to alter the execution of statement order in a program with a conditional statement?
   - A A Boolean expression
   - B A question type
   - C A selection statement
   - D A sequence

2. Boolean expressions can only be used in an IF statement. This is
   - A True
   - B False

3. An input data can be read into a Boolean variable using a *cin* statement. This is
   - A True
   - B False
   - C Difficult to tell

4. Given that A = true, B = false and C = false, what is the result of the following expression?
   A || B && C

   - A True
   - B False

5. Which of the following statements correctly declares the variable *MyBalance* to be used in storing account balances in a program
   - A MyBalance float;
   - B real MyBalance;
   - C int eger MyBalance;
   - D double : MyBalance;
   - E None of the above

6. Which of the following statements correctly display the data held in variable *total*?
   - A cout << "total" ;
   - B cout >> total ;
   - C cout >> 'total';
   - D write total;
   - E None of the above

7. Which of the following statements correctly assigns the sum of the variables *midsem* and *exams* and store the result in a variable *total*?
   - A total = midsem + exams
   - B total ==midsem + exams;
   - C total = midsem + exams;
   - D total += midsem + exams

8. Which of the following statements best describes the expression "What is your name ?" in the following statements?

```
cout <<"What is your name ? "
cin>> name;
```

A A question
B Output statement
C A prompt
D cout statement
E All of the above

9. Which of the following statements best instructs the Computer to display to the screen the data held in the variable *Sum* such that it will have at most three print positions.

A cout >> "small_value",3;
B cout << small_value:3 ;
C write 'small_value:3';
D cout << small_value:3;
E None of the above

10. Which of the following correctly adds 10 and 20 when given the input '2 10 20' and correctly adds 30 20 and 10 given the input to be '3 30 20 10'?

(i)

```
cin>>n;
 If (n==2)
 {
        cin>> a1 >>a2
        sum= a1+a2;
 }
else
    if (n==3)
    {
        cin>>a1 >>a2 >>a3;
        sum= a1+a2+a3;
    }
   else
        cout <<"Please, wrong input!!!!!";
```

(ii)

```
cin>>n;
 if (n>1 &&  n < 4)
 {
        cin>> a1 >> a2
        sum= a1+a2;
        if (n==3)
        {
          cin >> a3;
          sum=sum+a3;
        }
 }
else
    cout <<"Please, wrong input!!! ";
```

A I only
B II only
C I and II
D None of them

11. In a complete program, under no circumstance should one have program statements after the *return 0*; of the main program. This is

A True
B false

12-14. The following program is logically incorrect as its function is to compare input data against 666 when the first data is strictly smaller than the second. Use this to answer questions 12-14.

```
1   #include<iostream>
2   using namespace std;
3   int main()
4   {
5   const int large(666);
6   int a,b;
7   cout <<"Please enter values of A and B ";
8   cin>>a >>b;
9   If (a > b)
10      if (a <= large)
11          cout <<(a+b);
12      else
13          cout <<a <<" is larger than " <<b <<" and maximum";
14  else
15          cout <<(a+b);
16  system("pause");
17  return 0;
18  }
```

12. Which of the following correctly modifies the program to do exactly what is expected?

i. interchange line 15 with 10 - 13
ii. interchange a and b on line 9
iii. replace "a, b" with "b, a" on line 8

A I only
B II only
C III only
D II and III only
E I, II and III

13. Without the modification to the above program, what will be displayed to the screen when the inputs are entered as 800 and 250 in that order.

A 1050
B 800 is larger than 250 and the maximum
C None of the above

14. Which of the following statements correctly declares the integer variables *Sum* and *Counter*, and initialises both variables to zero?

A int sum=counter=0;
B integer sum=0 and counter=0;
C int sum=0 && counter = 0;
D int = 0 sum, counter;
E At least two of the above

15. Which of the following is an invalid C++ relational operator

A =
B <>
C <
D >

16. Write a C++ statement which compares the integer variable *sum* to the constant value 99, and if it is the same prints the string 'XCIX'

A if sum = 99 then cout <<"XCIX";
B if (sum == 99) cout << 'XCIX';
C if (sum = 99) cout<<'XCIX';
D if sum = 99 cout << ('XCIX');

17. Which of the following statements correctly defines *MyAge* as integer?

A MyAge :integer;
B MyAge : year;
C MyAge year;
D int MyAge;
E None of the above

18. Which of the following statements can best be used to accept data into the variable *alphabet*?

A cin >> 'letter';
B cin >> "letter";
C read letter ;
D cin << letter ;
E None of the above

19. Write a C++ statement which compares the string variable *letter* to the character variable *YourLetter*, and if they are not the same, prints the value of *YourLetter*

A if (letter <> YourLetter) cout << 'YourLetter';
B if strcmp(letter, YourLetter) cout <<YourLetter;
C if (letter != YourLetter) cout <<YourLetter;
D if (strcomp(letter,YourLetter) !=0) cout <<YourLetter;
E At least two of the above are correct

20. Which of the following can you use to specify the number of characters that should be displayed for a string variable?

A setw
B setwidth
C length
D charlength

21. If the characters "A" and "C" are compared, the Computer will indicate that "A" is smaller than "C". Which of the following best describes why 'A' is smaller than 'B'

A Alphabetically, A is before C
B If A is the first alphabet and C is the third then A is smaller than C
C The ASCII value of A is smaller than that of C then so A is smaller than C
D All the above

22. Which of the following C++ user defined function headers indicates that the elements of the array *Wages* cannot be modified by the function, MyFunction.

A const int MyFunction(int Wages);
B int const MyFunction(int Wages);
C const MyFunction(int Wages);
D int MyFunction(const wages);
E All the above

23. Write a C++ statement which declares a constant called MAXSIZE with a value of 80

A constant MAXSIZE = 80;
B const MAXSIZE == 80;
C const MAXSIZE 80;
D constant MAXSIZE = 80
E none of the above

24. Which of the following *for* loops can correctly display the following output?

```
1
22
333
4444
55555
```

A
```
for (i=1;i<6;i++)
{
        for (j=1;j<=i;j++)
           cout << i;
            cout <<endl;
}
```

B
```
for (i=1;i<=5;i++)
{
        for (j=1;j<=i;j++)
```

```
            cout << j;
            cout <<endl;
        }
```

C

```
        for (i=1;i<=5;i++)
        {
            for (j=1;j<i;j++)
               cout << j;
            cout <<endl;
        }
```

25. A variable which is not defined before or at the start of the main function cannot NEVER be used in the body of the main function. This is

    A True
    B False
    C Not easy to tell
    D It depends on the compiler

26. Which of the following will you say correctly declares the variable *MyInitials* as character variable?

    A character : MyInitials;
    B char : MyInitials;
    C char := MyInitials
    D MyInitials character;
    E char MyInitials ;

27. Which of the following statements can correctly display the alphabets A to Z inclusive?

A

```
        loop = 'A';
        while (loop <= 'Z') do
        {
            cout << loop << ' ';
            loop = loop + 1;
        }
```

B

```
        loop = 'Z';
        while (loop > 'A') do
        {
            cout << loop << ' ');
            loop = loop – 1;
        }
```

C

```
        loop:= A;
        while (loop < =Z) do
        {
```

```
        cout <<  loop << ' ' ;
        loop = loop ++;
    }
```

28. Which of the following can be used to add comments in a C++ program?
    - A { }
    - B /* */
    - C ; ;
    - D comment
    - E At least two of the above

29. Which of the following correctly moves the cursor to the start of the next line?
    - A cout <<;
    - B cout <<nextLine;
    - C cout <<endOfLine;
    - D cout << 'endl';
    - E None of the above

30. The statement that declares a variable called *WholeNumber*, capable of storing any integer value, is
    - A WholeNumber : integer;
    - B integer WholeNumber;
    - C WholeNumber as integer;
    - D int WholeNumber;
    - E WholeNumber of integers;

31. To print a single quote as part of an output string which of the following will you use with the *cout* statement?
    - A use a double quote in place of the single quote
    - B use two double quotes in place of the single quote
    - C preceed the single quote with a backslash
    - D preceed the single quote with a single quote

32. In C++ all valid program statements should be terminated by semi-colon, this is
    - A True
    - B False

33. Which of the following statements declares an array called numbers to store integer 20 integer values?
    - A int numbers = ARRAY[1..20];
    - B int numbers = ARRAY[0..19];
    - C int numbers[20];
    - D int numbers = ARRAY[20];

34. Which of the following is syntactically correct?

A WelcomeMessage = 'Hello there! ';
B WelcomeMessage == 'Hello there! ';
C WelcomeMessage = "Hello there! ";
D WelcomeMessage = Hello there!;

35. Which of the following statements will correctly display ASCII value of the letter "A" to the screen?

A cout << chr('A');
B cout << str('A');
C cout <<ASCII('A') ;
D cout << char('A');
E None of the above

36. Which of the following statements correctly declares the variable *counter* and initialise it to 100?

A counter = 100;
B int counter[100];
C integer counter =100;
D int counter(100)

37. Which of the following operators has the least priority when used in the same expression?

A =
B +
C /
D NOT

38. Which of the following statements correctly stores the number 20 at the 10^{th} location of the array, MyNumbers?

A MyNumbers[20] = 10;
B MyNumbers [10] = 20;
C MyNumbers [9] = 20;
D MyNumbers [20] = 9;
E B and C are both correct

39. What is the only programming language that a computer understands directly?

A English, as spoken in countries like Ghana.
B BASIC, the Beginners' All-purpose Symbolic Instruction Code
C machine language, different for every type of CPU
D All the above

40. What are the three main *types* of computer programming languages?

A machine language, assembly language, high level language
B imperative language, functional language, declarative language

C Pascal, Java, C++
D Interpreter, Compiler, Assembler

41. Which of the following is the main reason why computer programmers prefer to a high-level language to the internal machine code or assembler language?
    A Program portability from one machine to another
    B ease and speed of writing program
    C Efficiency
    D All

42. A compiler produces an error in compiling a program because a closing curly bracket has been missed out in the source code. What type of error is this?
    A Syntax Error
    B Logical Error
    C Linker Error
    D Compiler Error
    E Missing Error

43. Which of the following is essential statement type for describing algorithms?
    A sequence
    B selection
    C repetition
    D All the above

44. Which of the following best describes a condition?
    A A value which is **true** or **false**
    B A numerical value
    C A value which is yes or no
    D A and B are correct
    E is a question type

45. Which of the following statements best describes a compiler?
    A "A compiler translates your java program into machine code and then, if possible, runs the program"
    B "A compiler produces a list of syntax and logical errors"
    C "A compiler translates a correct java program into machine code. Anything that cannot be translated causes a syntax error."
    D "A compiler simply joins the different classes in your program together."
    E None of these

46. C++ compilers allow for nested comments. This is
    A True
    B False

46-53. For each of the following indicate whether the statement is true or false

46. A user defined function cannot be defined before the main function

47. A user defined function can be defined within the body of the main function

48. After the return 0 of the main function, a user defined function can follow

49. If a user defined function is not called by the main function or other user defined function, the compiler will display an error message during program compilation.

50. The use of curly brackets to enclose the body of a user defined function is optional

51. The C++ compiler is not likely to indicate there is an error if the following are used as user defined function names: *double*, *for* and *repeat*.

52. All elements of an array are placed randomly in a computer memory for use

53. It is not possible to write an arithmetic expression such that * is to the left of the = sign

54. The smallest amount of information possible is known as what?
    - A Word
    - B byte
    - C Bit
    - D Data

55. Which of the following defines the correct order of steps to get a program to work on a personal computer?
    - A type, edit, link, run, compile
    - B compile, link, edit, run
    - C edit, compile, link, run
    - D edit, link, compile, run

56. If at the end of a compilation process, a compiler displays the number of errors to be 1, then which of the following may best give the total numbers of errors in the program?
    - A 1
    - B 8
    - C 99
    - D any of the above
    - E Cannot tell exact number

57. Which of the following best describes the program that allows you to type your source program so that it can later be compiled without having to exit the program?
    - A Compiler
    - B Editor
    - C Linker
    - D Word processor such as Microsoft Word

E Algorithm

58. Which of the following is best reason why one will have to document a program?
    A to assist the program user
    B to assist the programmer
    C to give to the software owner
    D All the above
    E both A and B above

59. The main function of a program should have at least one return statement. This is
    A True
    B False

60. If two or more independent and very good programmers are asked to write an algorithm or program for finding the area under a curve using the Simpson's rule or formula, then one would expect that all things being equal (that is if the solutions are correct) the programmers will come out with algorithms and programs of the same length or same number of lines in the source code. This is most of the time
    A True
    B False

detach and return to IDL, KNUST

Learner Feedback Form/[insert course code]

Dear Learner,
While studying the units in the course, you may have found certain portions of the text difficult to comprehend. We wish to know your difficulties and suggestions, in order to improve the course. Therefore, we request you to fill out and send the following questionnaire, which pertains to this course. If you find the space provided insufficient, kindly use a separate sheet.

1. How many hours did you need for studying the units

Unit no.	1	2	3	4	5	6
No. of hours						

2. Please give your reactions to the following items based on your reading of the course

Items	**Excellent**	**Very good**	**Good**	**Poor**	*Give specific examples, if poor*
Presentation quality	☐	☐	☐	☐	
Language and style	☐	☐	☐	☐	
Illustrations used (diagrams, tables, etc.)	☐	☐	☐	☐	
Conceptual clarity	☐	☐	☐	☐	
Self assessment	☐	☐	☐	☐	
Feedback to SA	☐	☐	☐	☐	

3. Any other comments