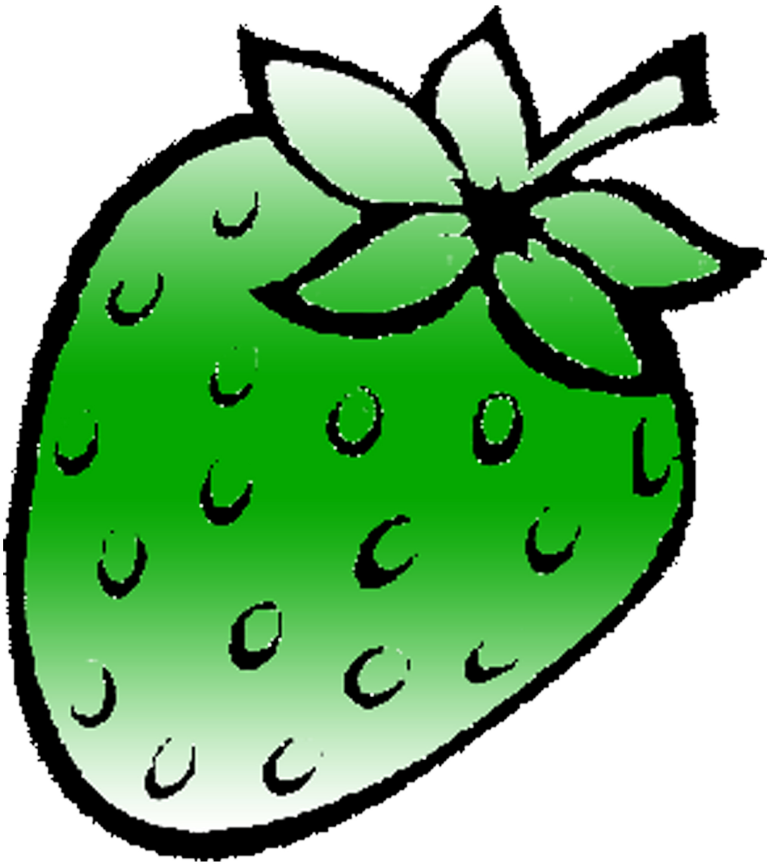
STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

Strawberrydevelopers.weebly.com

Experiment No. 10

PART A

(PART A: TO BE REFFERED BY STUDENTS)

A.1 Aim: To understand the below concept of File handling and templates in C++

1. **Opening a file**
2. **Closing a file**
3. **Storing object data in a file.**
4. **Reading data from the file.**
5. **Creating and using templates**

P1: A file contains a list of names and telephone numbers in the following form:
Name Telephone_number
Write a C++ program to read the file and output the list in tabular format. Use a class object to store each set of data.

P2: Write a function template for finding the maximum value contained in an array. Write main() function to find the maximum value of integer array and minimum value off floating point number in an array.

A.2 Prerequisite:

Knowledge of File Handling Concepts and templates.

A.3 Outcome:

After successful completion of this experiment students will be able to

1. Implement the concepts of file handling and generic programming using templates

A.4 Theory:

Data File Handling In C++

File. The information / data stored under a specific name on a storage device, is called a file.
Stream. It refers to a sequence of bytes.
Text file. It is a file that stores information in ASCII characters. In text files, each line of text is terminated with a special character known as EOL (End of Line) character or delimiter character. When this EOL character is read or written, certain internal translations take place.
Binary file. It is a file that contains information in the same format as it is held in memory. In binary files, no delimiters are used for a line and no translations occur here.
Classes for file stream operation

ofstream: Stream class to write on files
ifstream: Stream class to read from files
fstream: Stream class to both read and write from/to files.

Opening a file

OPENING FILE USING CONSTRUCTOR

ofstream *fout*(*"results"*); //output only

ifstream *fin*(*"data"*); //input only

OPENING FILE USING open()

Stream-object.open(*"filename"*, mode)

```
ofstream ofile;
ofile.open("data1");

ifstream ifile;
ifile.open("data2");
```

File mode parameter	Meaning
ios::app	Append to end of file
ios::ate	go to end of file on opening
ios::binary	file open in binary mode
ios::in	open file for reading only
ios::out	open file for writing only
ios::nocreate	open fails if the file does not exist
ios::noreplace	open fails if the file already exist
ios::trunc	delete the contents of the file if it exist

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open():

fstream *file*;

file.open (*"example.bin"*, ios::out | ios::app | ios::binary);

Closing File

fout.close();

fin.close();

INPUT AND OUTPUT OPERATION

put() and get() function

the function put() writes a single character to the associated stream. Similarly, the function get()

reads a single character form the associated stream.
example :
file.get(ch);
file.put(ch);
write() and read() function
write() and read() functions write and read blocks of binary data.
example:
file.read((char *)&*obj*, sizeof(*obj*));
file.write((char *)&*obj*, sizeof(*obj*));
ERROR HANDLING FUNCTION

FUNCTION	RETURN VALUE AND MEANING
eof()	returns true (non zero) if end of file is encountered while reading; otherwise return false(zero)
fail()	return true when an input or output operation has failed
bad()	returns true if an invalid operation is attempted or any unrecoverable error has occurred.
good()	returns true if no error has occurred.

File Pointers And Their Manipulation
All i/o streams objects have, at least, one internal stream pointer:
ifstream, like istream, has a pointer known as the get pointer that points to the element to be read in the next input operation.
ofstream, like ostream, has a pointer known as the put pointer that points to the location where the next element has to be written.
Finally, fstream, inherits both, the get and the put pointers, from iostream (which is itself derived from both istream and ostream).

These internal stream pointers that point to the reading or writing locations within a stream can be manipulated using the following member functions:

seekg()	moves get pointer(input) to a specified location
seekp()	moves put pointer (output) to a specified location
tellg()	gives the current position of the get pointer
tellp()	gives the current position of the put pointer

Basic Operation On Text File In C++

Program to write in a text file

```
#include<fstream.h>
int main()
{
    ofstream fout;
    fout.open("out.txt");
    char str[300]="Time is a great teacher but unfortunately it kills all its pupils. Berlioz";
    fout<<str;
    fout.close();
    return 0;
}
```

Program to read from text file and display it

```
#include<fstream.h>
#include<conio.h>
int main()
{
    ifstream fin;
    fin.open("out.txt");
    char ch;
    while(!fin.eof())
    {
        fin.get(ch);
        cout<<ch;
    }
    fin.close();
    getch();
    return 0;
}
```

Program to count number of characters.

```
#include<fstream.h>
#include<conio.h>
int main()
{
    ifstream fin;
    fin.open("out.txt");
    clrscr();
    char ch; int count=0;
    while(!fin.eof())
    {
```

```
        fin.get(ch);
        count++;
    }
    cout<<"Number of characters in file is "<<count;
    fin.close();
    getch();
    return 0;
}
```

Program to count number of words

```
#include<fstream.h>
#include<conio.h>
int main()
{
    ifstream fin;
    fin.open("out.txt");
    char word[30]; int count=0;
    while(!fin.eof())
    {
        fin>>word;
        count++;
    }
    cout<<"Number of words in file is "<<count;
    fin.close();
    getch();
    return 0;
}
```

Program to count number of lines

```
#include<fstream.h>
#include<conio.h>
int main()
{
    ifstream fin;
    fin.open("out.txt");
    char str[80]; int count=0;
    while(!fin.eof())
    {
        fin.getline(str,80);
        count++;
    }
    cout<<"Number of lines in file is "<<count;
```

```
    fin.close();
    getch();
    return 0;
}
```

Program to copy contents of file to another file.

```
#include<fstream.h>
int main()
{
    ifstream fin;
    fin.open("out.txt");
    ofstream fout;
    fout.open("sample.txt");
    char ch;
    while(!fin.eof())
    {
        fin.get(ch);
        fout<<ch;
    }
    fin.close();
    return 0;
}
```

Basic Operation On Binary File In C++

```
class student
{
         int admno;
         char name[20];
public:
        void getdata()
        {
                cout<<"\nEnter The admission no. ";
                cin>>admno;
                cout<<"\n\nEnter The Name of The Student ";
                gets(name);
        }
        void showdata()
        {
                cout<<"\nAdmission no. : "<<admno;
                cout<<"\nStudent Name : ";
                puts(name);
        }
```

```
        int retadmno()
        {
                return admno;
        }
};
```

function to write in a binary file

```
void write_data()
{
        student obj;
        ofstream fp2;
        fp2.open("student.dat",ios::binary|ios::app);
        obj.getdata();
        fp2.write((char*)&obj,sizeof(obj));
        fp2.close();
}
```

function to display records of file

```
void display()
{
        student obj;
        ifstream fp1;
        fp1.open("student.dat",ios::binary);
        while(fp1.read((char*)&obj,sizeof(obj)))
        {
                obj.showdata();
        }
        fp1.close();
}
```

Function to search and display from binary file

```
void search (int n)
{
        student obj;
        ifstream fp1;
        fp1.open("student.dat",ios::binary);
        while(fp1.read((char*)&obj,sizeof(obj)))
        {
                if(obj.retadmno()==n)
                        obj.showdata();
        }
```

```
        fp1.close();
}
```
Function to delete a record
```
void deleterecord(int n)
{
        student obj;
        ifstream fp1;
        fp1.open("student.dat",ios::binary);
        ofstream fp2;
        fp2.open("Temp.dat",ios::out|ios::binary);
        while(fp1.read((char*)&obj,sizeof(obj)))
        {
                if(obj.retadmno!=n)
                        fp2.write((char*)&obj,sizeof(obj));
        }
        fp1.close();
        fp2.close();
        remove("student.dat");
        rename("Temp.dat","student.dat");
}
```
Function to modify a record
```
void modifyrecord(int n)
{
        fstream fp;
        student obj;
        int found=0;
        fp.open("student.dat",ios::in|ios::out);
        while(fp.read((char*)&obj,sizeof(obj)) && found==0)
        {
                if(obj.retadmno()==n)
                {
                        obj.showdata();
                        cout<<"\nEnter The New Details of student";
                        obj.getdata();
                        int pos=-1*sizeof(obj);
                        fp.seekp(pos,ios::cur);
                        fp.write((char*)&obj,sizeof(obj));
                        found=1;
                }
        }
```

```
    fp.close();
}
```

Templates are a way of making your classes more abstract by letting you define the behavior of the class without actually knowing what datatype will be handled by the operations of the class. In essence, this is what is known as generic programming; this term is a useful way to think about templates because it helps remind the programmer that a templated class does not depend on the datatype (or types) it deals with. To a large degree, a templated class is more focused on the algorithmic thought rather than the specific nuances of a single datatype. Templates can be used in conjunction with abstract datatypes in order to allow them to handle any type of data. For example, you could make a templated stack class that can handle a stack of any datatype, rather than having to create a stack class for every different datatype for which you want the stack to function. The ability to have a single class that can handle several different datatypes means the code is easier to maintain, and it makes classes more reusable.
The basic syntax for declaring a templated class is as follows:
template <class a_type> class a_class {...};
The keyword 'class' above simply means that the identifier a_type will stand for a datatype. NB: a_type is not a keyword; it is an identifier that during the execution of the program will represent a single datatype. For example, you could, when defining variables in the class, use the following line:
a_type a_var;
and when the programmer defines which datatype 'a_type' is to be when the program instantiates a particular instance of a_class, a_var will be of that type.
When defining a function as a member of a templated class, it is necessary to define it as a templated function:
template<class a_type> void a_class<a_type>::a_function(){...}

When declaring an instance of a templated class, the syntax is as follows:
a_class<int> an_example_class;

An instantiated object of a templated class is called a specialization; the term specialization is useful to remember because it reminds us that the original class is a generic class, whereas a specific instantiation of a class is specialized for a single datatype (although it is possible to template multiple types).
Usually when writing code it is easiest to precede from concrete to abstract; therefore, it is easier to write a class for a specific datatype and then proceed to a templated - generic - class. For that brevity is the soul of wit, this example will be brief and therefore of little practical application.
We will define the first class to act only on integers.

```
class calc
{
  public:
    int multiply(int x, int y);
    int add(int x, int y);
 };
int calc::multiply(int x, int y)
{
  return x*y;
}
int calc::add(int x, int y)
{
  return x+y;
}
```

We now have a perfectly harmless little class that functions perfectly well for integers; but what if we decided we wanted a generic class that would work equally well for floating point numbers? We would use a template.

```
template <class A_Type> class calc
{
  public:
    A_Type multiply(A_Type x, A_Type y);
    A_Type add(A_Type x, A_Type y);
};
template <class A_Type> A_Type calc<A_Type>::multiply(A_Type x,A_Type y)
{
  return x*y;
}
template <class A_Type> A_Type calc<A_Type>::add(A_Type x, A_Type y)
{
  return x+y;
}
```

To understand the templated class, just think about replacing the identifier A_Type everywhere it appears, except as part of the template or class definition, with the keyword int. It would be the same as the above class; now when you instantiate an object of class calc you can choose which datatype the class will handle.

```
calc <double> a_calc_class;
```

Templates are handy for making your programs more generic and allowing your code to be reused later.

Template Functions :

The syntax for declaring a templated function is similar to that for a templated class:

template <class type> type func_name(type arg1, ...);
For instance, to declare a templated function to add two values together, you could use the following syntax:

```
template <class type> type add(type a, type b)
{
   return a + b;
}
```

Now, when you actually use the add function, you can simply treat it like any other function because the desired type is also the type given for the arguments. This means that upon compiling the code, the compiler will know what type is desired:

```
int x = add(1, 2);
```

will correctly deduce that "type" should be int. This would be the equivalent of saying:

```
int x = add<int>(1, 2);
```

where the template is explicitly instantiated by giving the type as a template parameter.

On the other hand, type inference of this sort isn't always possible because it's not always feasible to guess the desired types from the arguments to the function. For instance, if you wanted a function that performed some kind of cast on the arguments, you might have a template with multiple parameters:

```
template <class type1, class type2> type2 cast(type1 x)
{
   return (type2)x;
}
```

Using this function without specifying the correct type for type2 would be impossible. On the other hand, it is possible to take advantage of some type inference if the template parameters are correctly ordered. In particular, if the first argument must be specified and the second deduced, it is only necessary to specify the first, and the second parameter can be deduced.

For instance, given the following declaration

```
template <class rettype, class argtype> rettype cast(argtype x)
{
   return (rettype)x;
}
```

this function call specifies everything that is necessary to allow the compiler deduce the correct type:

```
cast<double>(10);
```

which will cast an int to a double. Note that arguments to be deduced must always follow arguments to be specified. (This is similar to the way that default arguments to functions work.)

You might wonder why you cannot use type inference for classes in C++. The problem is that it

would be a much more complex process with classes, especially as constructors may have multiple versions that take different numbers of parameters, and not all of the necessary template parameters may be used in any given constructor.

Templated Classes with Templated Functions

It is also possible to have a templated class that has a member function that is itself a template, separate from the class template. For instance,

```
template <class type> class TClass
{
   // constructors, etc

   template <class type2> type2 myFunc(type2 arg);
};
```

The function myFunc is a templated function inside of a templated class, and when you actually define the function, you must respect this by using the template keyword twice:

```
template <class type>  // For the class
   template <class type2>  // For the function
   type2 TClass<type>::myFunc(type2 arg)
   {
      // code
   }
```

The following attempt to combine the two is **wrong** and will not work:

```
// bad code!
template <class type, class type2> type2 TClass<type>::myFunc(type2 arg)
{
   // ...
}
```

because it suggests that the template is entirely the class template and not a function template at all.

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

Roll No. N008	Name: Akshay BANDA
Program: MBA TECh CS	Division: C
Semester: 2	Batch : C1
Date of Experiment: 8/4/2015	Date of Submission:
Grade :	

B.1 Software Code written by student:

(Paste your C++ code completed during the 2 hours of practical in the lab here)

1.

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

using namespace std;

class objects
{
   string n,t;
public:
   void getdata()
   {
      fstream f;
      f.open("telephone.txt",ios::app);
      cout<<"\nEnter name : ";
      cin>>n;
      f<<n;
      f<<"\t";
      cout<<"\Enter telephone number : ";
```

```
        cin>>t;
        f<<t;
        f<<"\n";
        f.close();
    }
    void display()
    {
        fstream f;
        f.open("telephone.txt");
        char ch;
        cout<<"\nName       Telephone";
        cout<<"\n";
        while(!f.eof())
        {
            f.get(ch);
            cout<<ch;
        }
        f.close();
        getch();
    }
};

int main()
{
    objects o;
    int c;
    do
    {
        cout<<"\nEnter 1 to write";
        cout<<"\nEnter 2 to read";
        cout<<"\nEnter your choice : ";
        cin>>c;
        if(c==1)
        {
            o.getdata();
        }
        else if(c==2)
        {
            o.display();
        }
```

```
    }while(c==1 || c==2);
    cout<<"\nThank you";
    return 0;
}
```

2.

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

template<class m>

m max(m a[100], int s)
{
   int i;
   m t=a[0];
   for(i=0;i<s;i++)
   {
      if(t<a[i])
      {
         t=a[i];
      }
   }
   return t;
}
template<class n>
n min(n a[100], int s)
{
   int i;
   n t=a[0];
   for(i=0;i<s;i++)
   {
      if(t>a[i])
      {
         t=a[i];
      }
   }
   return t;
}
```

```
int main()
{
   int a[100];
   float b[100],mi;
   int s,i,ma;
   cout<<"\nEnter size of array : ";
   cin>>s;
   cout<<"\nEnter integer values";
   for(i=0;i<s;i++)
   {
      cout<<"\nEnter number : ";
      cin>>a[i];
   }
   ma=max(a,s);
   cout<<"\nMaximum is "<<ma;
   cout<<"\nEnter floating point values";
   for(i=0;i<s;i++)
   {
      cout<<"\nEnter number : ";
      cin>>b[i];
   }
   mi=min(b,s);
   cout<<"\nMinimum is "<<mi;
   return 0;
}
```

B.2 Input and Output:

(Paste your program input and output in following format. If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)

1.

```
Enter 1 to write
Enter 2 to read
```

Enter your choice : 1

Enter name : akshay
←nter telephone number : 7045139285

Enter 1 to write
Enter 2 to read
Enter your choice : 2

Name Telephone
akshay 7045139285

2.

Enter size of array : 3

Enter integer values
Enter number : 7

Enter number : 18

Enter number : 27

Maximum is 27
Enter floating point values
Enter number : 7.7

Enter number : 1.8

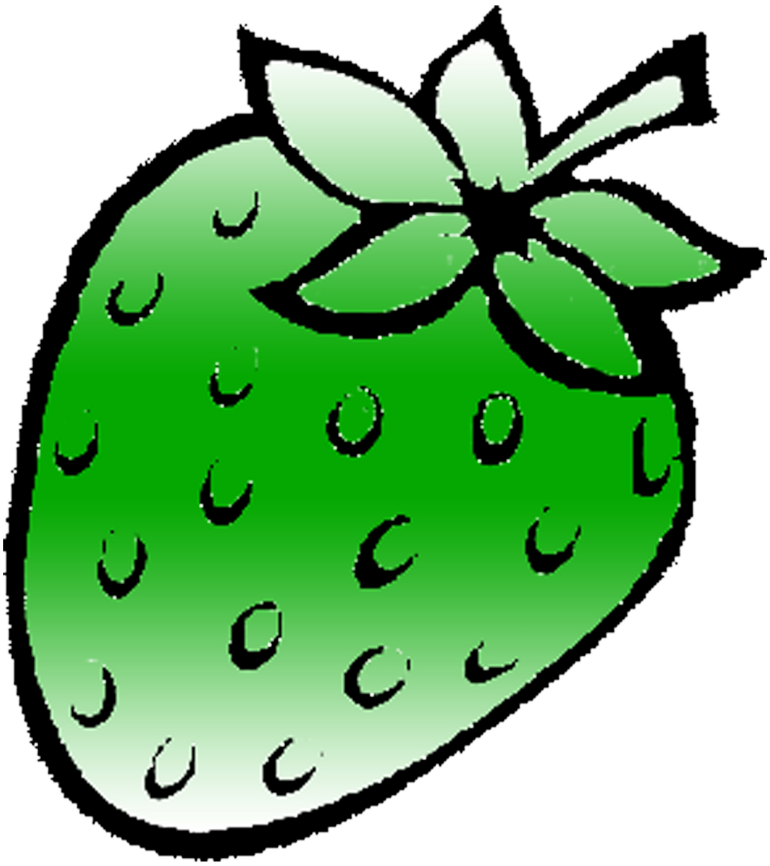
Enter number : 2.7

Minimum is 1.8

B.3 Conclusion:
(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.1)

I learned file handling,

STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

Strawberrydevelopers.weebly.com