CORE COURSE II

PROGRAMMING IN C++

Unit I

Basic Concepts of Object- Oriented Programming - Benefits of OOP - Object Oriented Languages - Applications of OOP – Structure of C++ Program - Tokens, Expressions and Control Structures – Functions in C++

Unit II

Classes and Objects – Constructors and Destructors –Operator Overloading and Type Conversions

Unit III

Inheritance : Extending Classes – Pointers - Virtual Functions and Polymorphism

Unit IV

Managing Console I/O Operations – Working with Files – Templates – Exception Handling

Unit V

Standard Template Library – Manipulating Strings – Object Oriented Systems Development

Part-A

1. What is Stream?

- A stream is an abstraction. It is a sequence of bytes.
- It represents a device on which input and output operations are performed.
- It can be represented as a source or destination of characters of indefinite length.
- It is generally associated to a physical source or destination of characters like a disk file, keyboard or console.
- C++ provides standard iostream library to operate with streams.

2. **Define input and output stream.**

- **Input Stream:** If the direction of flow of bytes is from the device (for example, Keyboard) to the main memory then this process is called input.
- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.

3. Define getline ( ) and write ( ) function.

- The getline( )function reads a whole line of the text and ends with newline character.
- This function can be invoked by,

cin.getline(line,size);

- The writeline( )function reads a whole line of the text and displays an entire line.
- This function can be invoked by ,

cout.write(line,size);

4. What are the Unformatted I/O Operations?

1. Cin
2. Cout
3. Get()
4. Put()

5. What are the ios format functions.?

1. Width()
2. Precison()
3. Fill()
4. Setf
5. Unsef

6. What is meant by manipulators?

The header file iomanip provides a set of functions called manipulators which can be used to manipulate the output formats. They provide same features as that of the ios member functions and flags.

7. What are manipulators in C++?

1. Setw(w)
2. Setprecison(d)
3. Setfill(c)
4. Setiosflags(f)
5. Resetiosflags(f)
6. Endl

8. What functions are used for manipulation of file pointers ?

- seekg ( ) – Moves get file pointer to a specific location.
- seekp ( ) - Moves put file pointer to a specific location.
- tellg ( ) – Returns the current position of the get pointer
- tellp ( ) - Returns the current position of the put pointer

9. What is a file ?

A file is a collection of related information defined by its creator. Commonly files represent programs and data. Files may be free-form, such as text files or may be rigidly formatted. In general, a file is a sequence of bits, bytes, lines, or records whose meaning is defined by its creator and user.

10. Differentiate file input stream and file output stream?

- I/P Stream extracts data from file
- O/P Stream inserts data to to file

11. What are operations on file?

1. Name the file on the disk
2. Open the file
3. Process the file(Read/Write)
4. Check for errors while processing
5. Close the file

12. What are the file stream class?

1. Filebuf
2. Fstreambase
3. Ifstream
4. Ofstream
5. Fstream

13. What is a function template?

Function templates are special functions that can operate with **generic types.** This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

In C++ this can be achieved using **template parameters.** A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function. These function templates can use these parameters as if they were any other regular type.

The format for declaring function templates with type parameters is:

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

14. What is Exception?

Exceptions provide a way to react to exceptional circumstances (like runtime errors) in programs by transferring control to special functions called **handlers.**

To catch exceptions, a portion of code is placed under exception inspection. This is done by enclosing that portion of code in a try-block. When an exceptional circumstance arises within that block, an exception is thrown that transfers the control to the exception handler. If no exception is thrown, the code continues normally and all handlers are ignored.

15. List out some of the unformatted I/O operators and formatted I/O operations.

Unformatted:

a)put()
b)get()
c)getline()
d)write()

Formatted:

a)ios stream class member functions and flags
b)Standard manipulators
c)User –defined manipulators

16. What are the advantages of using exception handling?

In C++, exception handling is useful because it makes it easy to separate the error handling code from the code written to handle the chores of the program. Doing so makes reading and writing the code easier.

C++ uses to handle exceptions means that it can easily handle those exceptions without any code in the intermediate functions.

Part-B

1.**Explain Console I/O operations in C++.**

Console input / output function take input from standard input devices and compute and give output to standard output device. Generally, keyboard is standard input device and monitor is standard output device.

In case of C++ it uses streams to perform input and output operations in standard input output devices (keyboard and monitor). A stream is an object which can either insert or extract the character from it.

The standard C++ library is iostream and standard input / output functions in C++ are:

- **cin**
- **cout**

There are mainly two types of consol I/O operations form:

- **Unformatted consol input output**
- **Formatted consol input output**

I - Unformatted consol input output operations

These input / output operations are in unformatted mode. The following are operations of unformatted consol input / output operations:

A) void get()

It is a method of cin object used to input a single character from keyboard. But its main property is that it allows wide spaces and newline character.

Syntax:

```
char c=cin.get();
```

Example:

```
#include<iostream>

using namespace std;

int main()

{

        char c=cin.get();

        cout<<c<<endl;

        return 0;

}
```

Output

```
I
I
```

B) void put()

It is a method of cout object and it is used to print the specified character on the screen or monitor.

Syntax:

```
cout.put(variable / character);
```

C) getline(char *buffer,int size)

This is a method of cin object and it is used to input a string with multiple spaces.

Syntax:

```
char x[30];
cin.getline(x,30);
```

Example:

```
#include<iostream>
using namespace std;
int main()
{
cout<<"Enter name :";
char c[10];
cin.getline(c,10); //It takes 10 charcters as input;
cout<<c<<endl;
return 0;
}
```

Output

```
Enter name :Divyanshu
Divyanshu
```

D) write(char * buffer, int n)

It is a method of cout object. This method is used to read n character from buffer variable.

Syntax:

```
cout.write(x,2);
```

E) cin

It is the method to take input any variable / character / string.

Syntax:

```
cin>>variable / character / String / ;
```

Example:

```
#include<iostream>
using namespace std;
int main()
{
int num;
char ch;
string str;
cout<<"Enter Number"<<endl;
cin>>num; //Inputs a variable;
cout<<"Enter Character"<<endl;
cin>>ch; //Inputs a character;
cout<<"Enter String"<<endl;
cin>>str; //Inputs a string;
return 0;
}
```

Output

```
Enter Number
07
Enter Character
```

```
h
Enter String
Deepak
```

F) cout

This method is used to print variable / string / character.

Syntax:

```
cout<< variable / charcter / string;
```

Example:

```
#include<iostream>
using namespace std;
int main()
{
 int num=100;
char ch='X';
string str="Deepak";
cout<<"Number is "<<num<<endl; //Prints value of variable;
cout<<"Character is "<<ch<<endl; //Prints character;
cout<<"String is "<<str<<endl; //Prints string;
return 0;
}
```

Output

```
Number is 100
Character is X
String is Deepak
```

II - Formatted console input output operations

In formatted console input output operations uses following functions to make output in perfect alignment. In industrial programming all the output should be perfectly formatted due to this reason C++ provides many function to convert any file into perfect aligned format. These functions are available in header file <iomanip>. iomanip refers input output manipulations.

A) width(n)

This function is used to set width of the output.

Syntax:

```
cout<<setw(int n);
```

Example:

```
#include<iostream>

#include<iomanip>

using namespace std;

int main()

{

int x=10;

cout<<setw(20)<<variable;

return 0;

}
```

Output

```
          10
```

B) fill(char)

This function is used to fill specified character at unused space.

Syntax:

```
cout<<setfill('character')<<variable;
```

Example:

```
#include<iostream>

#include<iomanip>
```

```
using namespace std;

int main()

{

int x=10;

cout<<setw(20);

cout<<setfill('#')<<x;

return 0;

}
```

Output

```
##################10
```

D) precison(n)

This method is used for setting floating point of the output.

Syntax:

```
cout<<setprecision('int n')<<variable;
```

Example:

```
#include<iostream>

#include<iomanip>

using namespace std;

int main()

{

float x=10.12345;

cout<<setprecision(5)<<x;

return 0;

}
```

Output

```
10.123
```

E) setflag(arg 1, arg,2)

This function is used for setting format flags for output.

Syntax:

setiosflags(argument 1, argument 2);

F) unsetflag(arg 2)

This function is used to reset set flags for output.

Syntax:

resetiosflags(argument 2);

G) setbase(arg)

This function is used to set basefield of the flag.

Syntax:

setbase(argument);

2. What is meant by Templates? Explain with example

Templates are one of the most powerful features in C++. Templates provide us the code that is independent of the data type.

In other words, using templates, write a generic code that works on any data type. In just need to pass the data type as a parameter. This parameter which passes the data type is also called a type name.

Templates can be implemented in two ways:

- **Function Template**
- **Class Template**

Function Template

Function Template is just like a normal function, but the only difference is while normal function can work only on one data type and a function template code can work on multiple data types.

Actually overload a normal function to work on various data types, function templates are always more useful as write the only program and it can work on all data types.

The general syntax of the function template is:

```
template<class T>
 T function_name(T args){
 ……
 //function body
 }
```

Here, T is the template argument that accepts different data types and class is a keyword. Instead of the keyword class, also write ‘typename’.

When a particular data type is passed to function_name, a copy of this function is made by the compiler with this data type as an argument and function are executed.

Example

```
#include <iostream>
using namespace std;
 template <typename T>
void func_swap(T &arg1, T &arg2)
{
  T temp;
  temp = arg1;
  arg1 = arg2;
  arg2 = temp;
}
 int main()
{
  int num1 = 10, num2 = 20;
  double d1 = 100.53, d2 = 435.54;
  char ch1 = 'A', ch2 = 'Z';
  cout << "Original data\n";
  cout << "num1 = " << num1 << "\tnum2 = " << num2<<endl;

  cout << "d1 = " << d1 << "\td2 = " << d2<<endl;

  cout << "ch1 = " << ch1 << "\t\tch2 = " << ch2<<endl;

   func_swap(num1, num2);

  func_swap(d1, d2);

  func_swap(ch1, ch2);

  cout << "\n\nData after swapping\n";

  cout << "num1 = " << num1 << "\tnum2 = " << num2<<endl;

  cout << "d1 = " << d1 << "\td2 = " << d2<<endl;

  cout << "ch1 = " << ch1 << "\t\tch2 = " << ch2<<endl;

  return 0;

}
```

Output:

```
Original data
num1 = 10 num2 = 20
d1 = 100.53 d2 = 435.54
ch1 = A ch2 = Z

Data after swapping
num1 = 20 num2 = 10
d1 = 435.54 d2 = 100.53
ch1 = Z ch2 = A
```

In the above program, defined a function template "func_swap" that swaps two values. The function takes two reference arguments of type T. It then swaps the values. As the arguments are references, whatever changes to arguments in the function will be reflected in the caller function.

In the main function, data of type int, double and char. call function func_swap with each type of data. Then display the swapped data for each data type.

Thus this shows that need not write three functions for three data types. It suffices to write only one function and makes it a template function so that it is independent of the data type.

Class Templates

Like in function templates, a requirement to have a class that is similar to all other aspects but only different data types.

In this situation, different classes for different data types or different implementation for different data types in the same class. But doing this will make our code bulky.

The best solution for this is to use a template class. Template class also behaves similar to function templates. To pass data type as a parameter to the class while creating objects or calling member functions.

The general syntax for the class template is:

```
template <class T>

class className{

…..

public:

            T memVar;

            T memFunction(T args);

};
```

In the above definition, T acts as a placeholder for the data type. The public members' memVar and memFunction also use T as a placeholder for data types.

Once a template class is defined as above, create class objects as follows:

```
className<int> classObejct1;

className<float> classObject2;

className<char> classObject3;
```

Example :

```
#include <iostream>

 using namespace std;

template <class T>

class myclass {

  T a, b;
 public:
  myclass (T first, T second)
     {a=first; b=second;}
  T getMaxval ();
};
template <class T>
T myclass<T>::getMaxval ()
{
  return (a>b? a : b);
}
int main () {
myclass <int> myobject (100, 75);
cout<<"Maximum of 100 and 75 = "<<myobject.getMaxval()<<endl;
 myclass<char> mychobject('A','a');
 cout<<"Maximum of 'A' and 'a' = "<<mychobject.getMaxval()<<endl;
return 0;
}
```

Output:

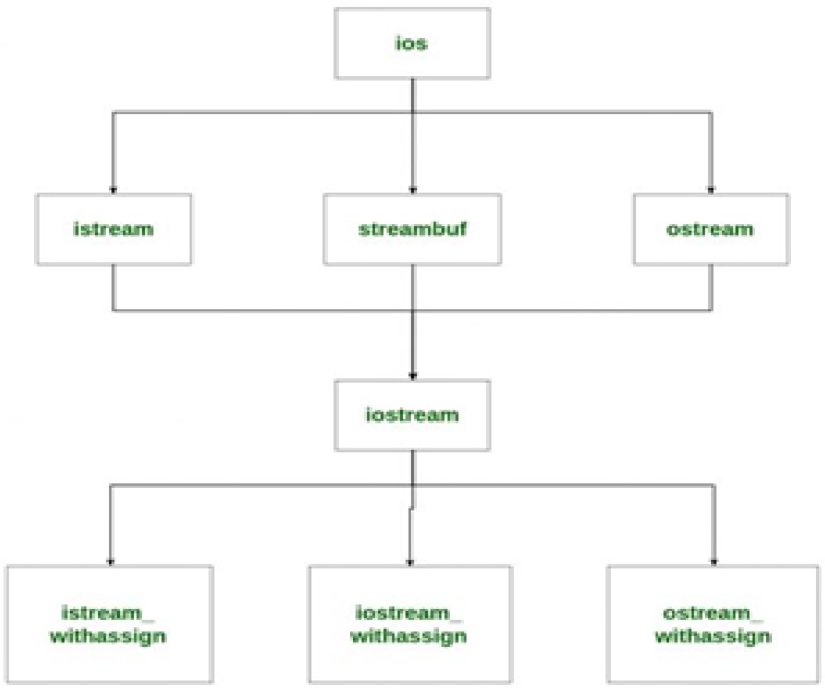
```
Maximum of 100 and 75 = 100
Maximum of ‘A' and ‘a' = a
```

Part-C

1.Describe briefly on C++ stream classes hierarchy with it's diagram

C++ Stream Classes Structure

In C++ there are number of stream classes for defining various streams related with files and for doing input-output operations. All these classes are defined in the file iostream.h**. Figure given below shows the hierarchy of these classes.**



1. ios class is topmost class in the stream classes hierarchy. It is the base class for istream, ostream, and streambuf class.
2. istream and ostream serves the base classes for iostream class. The class istream is used for input and ostream for the output.
3. Class ios is indirectly inherited to iostream class using istream and ostream. To avoid the duplicity of data and member functions of ios class, it is declared as virtual base class when inheriting in istream and ostream as

```
class istream: virtual public ios
{
};
class ostream: virtual public ios
{
};
```

4. The _withassign classes are provided with extra functionality for the assignment operations that's why _withassign classes.

Facilities provided by these stream classes.

1. **The ios class:** The ios class is responsible for providing all input and output facilities to all other stream classes.
2. **The istream class:** This class is responsible for handling input stream. It provides number of function for handling chars, strings and objects such as **get, getline, read, ignore, putback** etc..

   Example:
   ```
   #include <iostream>
   using namespace std;
   int main()
   {
           char x;
           // used to scan a single char
           cin.get(x);
           cout << x;
   }
   ```
   Input:
   ```
     g
   ```
   Output:
   ```
     G
   ```
3. **The ostream class:** This class is responsible for handling output stream. It provides number of function for handling chars, strings and objects such as write, put etc..

   Example:
   ```
   #include <iostream>
   using namespace std;
   int main()
   {
     char x;
     // used to scan a single char
     cin.get(x);
     // used to put a single char onto the screen.
     cout.put(x);
   }
   ```
   Input:
   ```
     g
   ```
   Output:
   ```
     g
   ```
4. **The iostream:** This class is responsible for handling both input and output stream as both **istream class** and **istream class** is inherited into it. It provides function of both **istream class** and **istream class** for handling chars, strings and objects such as **get,**

getline, read, ignore, putback, put, write etc..

Example:

```
#include <iostream>
using namespace std;
int main()
{
// this function display
// ncount character from array
cout.write("geeksforgeeks", 5);
}
```

Output:

```
geeks
```

5. **istream_withassign class:** This class is variant of **istream** that allows object assigment. The predefined object **cin** is an object of this class and thus may be reassigned at run time to a different **istream** object.
   Example:To show that **cin** is object of **istream** class.

```
#include <iostream>
using namespace std;
class demo
{
public:
int dx, dy;
// operator overloading using friend function
friend void operator>>(demo& d, istream& mycin)
{
// cin assigned to another object mycin
mycin >> d.dx >> d.dy;
}
};
int main()
{
demo d;
cout << "Enter two numbers dx and dy\n";
// calls operator >> function and
// pass d and cin as reference
d >> cin; // can also be written as operator >> (d, cin) ;
cout << "dx = " << d.dx << "\tdy = " << d.dy;
}
```

Input:

```
4 5
```

Output:

```
Enter two numbers dx and dy
4 5
dx = 4  dy = 5
```

6. **ostream_withassign class:** This class is variant of **ostream** that allows object assigment. The predefined objects **cout, cerr, clog** are objects of this class and thus may be reassigned at run time to a different **ostream** object.
Example:To show that cout is object of **ostream** class.

```
#include <iostream>
using namespace std;
class demo
{
public:
int dx, dy;
demo()
{
dx = 4;
dy = 5;
}
// operator overloading using friend function
friend void operator<<(demo& d, ostream& mycout)
{
// cout assigned to another object mycout
mycout << "Value of dx and dy are \n";
mycout << d.dx << " " << d.dy;
}
};
int main()
{
demo d; // default constructor is called
// calls operator << function and
// pass d and cout as reference
d << cout;  // can also be written as operator << (d, cin) ;
}
```

Output:

```
Value of dx and dy are
4 5
```

2 .Discuss on File Handling in C++.

File handling is used to store a data permanently in computer. The data can be stored in secondary memory (hard disk) using file handling.

The I/O data can easily transferred from one computer to another by using files.

The C++ standard library provides fstream class for performing Read and Write operations.

The fstream defines three new datatypes are as:

Datatype	Description
Ofstream	Used for creating a file and write data on files.
Ifstream	Used for reading the data from files.
Fstream	Used for both read and write data from files.

Following are the functions used in File Handling.

Function	Description
open()	It is used to create a file.
close()	It is used to close an existing file.
get()	It is used to read a single unit character from a file.
put()	It is used to write a single unit character in file.
read()	It is used to read data from file.
write()	It is used to write data into file.

Following are the operations of File Handling.

1. Naming a file
2. Opening a file
3. Reading data from file
4. Writing data into file
5. Closing a file

Opening a File

The open() function is used to open multiple files which uses the same stream object.

The fstream or ofstream object is used to open a file for writing and ifstream object is used to open a file for reading.

Syntax:

```
void open(const char *filename, ios::openmode mode);
```

where, First argument *filename specifies the name of file and location.
Second argument open() member function defines the mode in which the file should be opened.

Following are the file opening modes

File mode	Description
ios::out	This mode is opened for writing a file.
ios::in	This mode is opened for reading a file.
ios::app	This mode is opened for appending data to end-of-file.
ios::binary	The file is opened in binary mode.
ios::ate	This file moves the pointer or cursor to the end of the file when it is opened.
ios::trunc	If the file is already exists, using this mode the file contents will be truncated before opening the file.
ios::nocreate	This mode will cause to fail the open() function if the file does not exists.
ios::noreplace	This mode will cause to fail the open function if the file is already exists.

All these above modes can be combined using the bitwise operator (|). for example, ios::out | ios::app | ios::binary

Closing a File

The close() function is used for closing a file.

When a program terminates, it automatically closes or flushes all the streams, releases all the allocated memory and closes all the opened files. But a good practice for programmer to close all the opened files before the program termination.

Syntax:

void close();

A file must be closed after completion of all operation related to a file.

Writing to a File

The insertion operator (<<) is used to write information in a file.

The ofstream or fstream object is used instead of cout object.

Reading from a File

The extraction operator (>>) is used to read information from a file into your program.

The ifstream or fstream object is used instead of cin object.

Input and Output Operation

Following functions are used for I/O Operation:

Function	Description
put()	This function writes a single character to the associated stream.
get()	This function reads a single character to the associated stream.
write()	This function is used to write the binary data.
read()	This function is used to read the binary data.

File Pointers

File pointer is associated with two pointers,
1. Input pointer reads the content of a given file location.
2. Output pointer writes the content to a given file location.

- All Input/Output stream objects have at least one internal stream pointer.
- The ifstream has a get pointer which points to the element to read in the next input operation.
- The ofstream has a put pointer which points to the location where the next element has to be written.
- The fstream inherits both the get and put pointers from iostream.

Following are the member function for manipulation of file pointer:

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

Example: Program demonstrating Read and Write mode of a file

Following program reads the information from the file and displays it onto the screen.

```
#include <fstream>
#include <iostream>
using namespace std;
int main ()
{
  char name[50];
  ofstream ofile;                    // open a file in write mode.
  ofile.open("abc.dat");
  cout << "Writing to the file" << endl;
  cout << "Enter your name: "<<endl;
  cin.getline(name, 50);
  cout<<endl;
  ofile << name << endl;         // write inputted data into the file.
  ofile.close();                  // close the opened file.
  ifstream ifile;                 // open a file in read mode.
  ifile.open("abc.dat");
  cout << "Reading from the file" << endl;
  ifile >> name;
  cout << name << endl;      // write the data at the screen.
  ifile.close();              // close the opened file.
```

```
   return 0;
}
```

Output:

```
Writing to the file
Enter your name:
Prajakta Pandit

Reading from the file
Prajakta Pandit
```

3. Explain Exception Handling used in c++ with example.

An exception transfers the control to special functions called Handlers. Exception handling makes it easy to separate the error handling code. It provides a way to transfer control from one part of a program to another. Exception handling makes code readable and maintainable. It separates the code to the normal flow. Function can handle or specify any exceptions they choose.

Following are the three specialized keywords where exception handling is built.

1. throw
2. try
3. catch

1. throw

Using throw statement, an exception can be thrown anywhere within a code block.

It is used to list the exceptions that a function throws, but doesn't handle itself.

Following example shows throwing an exception when dividing by zero condition occurs.

```
float div(int x, int y)
{
   if(y==0)
   {
       throw ”Divide by zero condition!!!”;
   }
   return (x/y);
}
```

2. try

Try represents a block of code that can throw an exception.

It identifies a block of code which particular exceptions will be activated.

Try block followed by one or more catch blocks.

Syntax:

```
try
{
   //Code
}
catch(ExceptionName e1)
{
   //Catch block
}
catch(ExceptionName e2)
{
   //Catch block
}
catch (ExceptionName e3)
{
   //Catch block
}
```

3. Catch

Catch represents a block of code executed when a particular exception is thrown.

It follows the try block which catches any exception.

Syntax:

```
try
{
   //code
}
catch(ExceptionName e)
{
   //code to handle exception
}
```

In the above syntax, code will catch an exception of ExceptionName type.

Example: Program demonstrating flow of execution of Exception

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

```
int main()
{
  int a = -2;
  cout << "Before Try Block"<<endl;
  try
  {
    cout << "Inside Try Block"<<endl;
    if (a < 0)
    {
      throw a;
      cout << "After Throw Exception (Never executed)"<<endl;
    }
  }
  catch (int a )
  {
    cout << "Exception Caught \n";
  }
  cout << "Executed After Catch Exception"<<endl;
  return 0;
}
```

Output:

```
Before Try Block
Inside Try Block
Exception Caught
Executed After Catch Exception
```