**Techno India NJR Institute of Technology**



**Lab Manual**

**Object Oriented Programming Lab (3CS4-22)**

**Gaurav Kumawat**

**Department of CSE**



**EXPERIMENT – 1**

**Aim:** Understand the basics of C++ Library, variables, data input-output.

**Solution:**

C++ comes with libraries that provide us with many ways for performing input and output. In C++ input and output are performed in the form of a sequence of bytes or more commonly known as **streams**.

* **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 (displayscreen) then this process is called output.



**Header files available in C++ for Input/Output operations are:**

1. **iostream**: iostream stands for standard input-output stream. This header file contains definitions of objects like cin, cout, cerr, etc.
2. **iomanip**: iomanip stands for input-output manipulators. The methods declared in these files are used for manipulating streams. This file contains definitions of setw, setprecision, etc.
3. **fstream**: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.

The two instances **cout in C++** and **cin in C++** of iostream class are used very often for printing outputs and taking inputs respectively. These two are the most basic methods of taking input and printing output in C++. To use cin and cout in C++ one must include the header file *iostream* in the program.

* **Standard output stream (cout)**: Usually the standard output device is the display screen. The C++ **cout** statement is the instance of the ostream class. It is used to produce output on the standard output device which is usually the display screen. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(**<<**).
* #include <iostream>
* **usingnamespace**std;
* **int**main()
* {
* **char**sample[] = "Techno India NJR";
*
* cout<< sample <<" - A computer science Lab Manual";
*
* **return**0;
* }

**Output:**

Techno India NJR **-** A computer science Lab Manual

In the above program, the insertion operator (**<<**) inserts the value of the string variable **sample** followed by the string “A computer science portal for geeks” in the standard output stream **cout** which is then displayed on the screen.

* **standard input stream (cin)**: Usually the input device in a computer is the keyboard. C++ cin statement is the instance of the class **istream** and is used to read input from the standard input device which is usually a keyboard.
The extraction operator (**>>**) is used along with the object **cin** for reading inputs. The extraction operator extracts the data from the object **cin** which is entered using the keyboard.
* #include <iostream>
* **usingnamespace**std;
* **int**main()
* {
* **int**age;
* cout<<"Enter your age:";
* cin>> age;
* cout<<"\nYour age is: "<< age;
* **return**0;
* }

**Input : 18**

**Output:**

Enter your age:

Your age is: 18

The above program asks the user to input the age. The object cin is connected to the input device. The age entered by the user is extracted from cin using the extraction operator(**>>**) and the extracted data is then stored in the variable **age** present on the right side of the extraction operator.

**EXPERIMENT – 2**

**Aim:**C++ program using with the concept of structures.

**Solution:**

#include<iostream.h>

#include<conio.h>

struct book

{

 char name[30];

 int pages;

 float price;

};

int main()

{

 struct book b[3];

for(int i=0;i<3;i++)

 {

cout<<"Enter Book name, pages & price"<<endl;

cin>>b[i].name>>b[i].pages>>b[i].price;

 }

cout<<"The Details are:"<<endl;

for(int i=0;i<3;i++)

 {

cout<<b[i].name<<"\t"<<b[i].pages<<"\t"<<b[i].price<<endl;

 }

 getch();

 return 0;

}

**EXPERIMENT – 3**

**Aim:**Implement class and object concepts and function overloading.

**Solution:**

#include<iostream.h>

#include<conio.h>

int volume(int); //cube

double volume(double, int); //cylinder

int volume(int,int,int); //Rectangular Box

int main()

{

cout<<"Volume of cube: "<<volume(10)<<endl;

cout<<"Volume of Cylinder: "<<volume(2.5,5)<<endl;

cout<<"Volume of rectunglar Box: "<<volume(4,6,8);

return 0;

}

int volume(int a)

{

 return (a\*a\*a);

}

double volume(double r, int h)

{

 return 3.14\*r\*r\*h;

}

int volume(int l, int b, int h)

{

 return(l\*b\*h);

}

**Class & Object:**

#include<iostream.h>

#include<conio.h>

class person

{

 char name[30];

 int age;

public:

 void getdata()

 {

cout<<"Enter Name & Age";

cin>>name>>age;

 }

 void putdata()

 {

cout<<name<<endl<<age;

 }

};

 int main()

{

 person p;

p.getdata();

p.putdata();

getch();

 return 0;

}

**EXPERIMENT – 4**

**Aim:**Write programs to understand dynamic memory allocation and array of objects.

**Solution:**

#include<iostream.h>

class employee

{

 char name [30];

 float age;

public:

 void getdata(void);

 void putdata(void);

};

void employee :: getdata(void)

{

 cout<<"Enter Name";

 cin>>name;

 cout<<"Enter age";

 cin>>age;

}

void employee :: putdata(void)

{

 cout<<"Name:"<<name<<"\n";

 cout<<"Age:"<<age<<"\n";

}

const int size=3;

int main()

{

employee manager[size];

 for(int i=0; i<size; i++)

 {

 cout<<"\n Details of manager"<<i+1<<"\n";

 manager[i].getdata();

 }

 cout<<"\n";

 for(i=0; i<size; i++)

 {

 cout<<"\n Manager"<<i+1<<"\n";

 manager[i].putdata();

 }

 return 0;

}

**EXPERIMENT – 5**

**Aim:**Program to understand different types of constructors and destructor.

**Solution:**

#include<iostream.h>

class complex

{

 float x, y;

public:

 complex(){}

 complex(float a)

 {

 x=y=a;

 }

 complex(float real, float img)

 {

 x = real;

 y = img;

 }

 friend complex sum(complex, complex);

 friend void show(complex);

};

complex sum(complex c1, complex c2)

{

 complex c3;

 c3.x = c1.x + c2.x;

 c3.y = c1.y + c2.y;

 return (c3);

}

void show(complex c)

{

 cout<<c.x<<"+j"<<c.y<<"\n";

}

int main()

{

 complex A(2.7, 3.5);

 complex B(1.5);

 complex C;

 C=sum(A,B);

 cout<<"A=";

 show(A);

 cout<<"B=";

 show(B);

 cout<<"C=";

 show(C);

 return 0;

}

**Copy Constructor**

#include<iostream.h>

#include<conio.h>

class code

{

 int id;

public:

 code(){}

 code(int a)

 {

 id = a;

 }

 code(code & x)

 {

 id = x.id;

 }

 void display(void)

 {

 cout<<id;

 }

};

int main()

{

 code A(100);

 code B(A);

code C = A;

 code D;

 D = A;

 cout<<"\n id of A:";

 A.display();

 cout<<"\n id of B:";

 B.display();

 cout<<"\n id of C:";

 C.display();

 cout<<"\n id of D:";

 D.display();

 return 0;

}

**Destructor**

#include<iostream.h>

#include<conio.h>

int count = 0;

class alpha

{

 public:

 alpha()

 {

 count++;

 cout<<"\n No. of object created"<<count;

 }

 ~alpha()

 {

 cout<<"\n No. of object destroyed"<<count;

 }

};

int main()

{

cout<<"\n \n Enter Main";

 alpha A1, A2, A3, A4;

 {

cout<<"\n Enter Block 1";

 alpha A5;

 }

{

cout<<"\n Enter Block 2";

 alpha A6;

 }

cout<<"\n Re-enter main";

 return 0;

}

**EXPERIMENT – 6**

**Aim:**Implement friend function to access private data of a class and usage of this pointer

**Solution:**

#include<iostream.h>

#include<conio.h>

class ABC;

class XYZ

{

 int x;

public:

 void setvalue(int i)

 {

 x=i;

 }

 friend void max(XYZ, ABC);

};

class ABC

{

 int a;

public:

 void setvalue(int i)

 {

 a=i;

 }

 friend void max(XYZ, ABC);

};

void max (XYZ m, ABC n)

{

 if(m.x>=n.a)

 cout<<m.x;

 else

 cout<<n.a;

}

int main()

{

 ABC abc;

 abc.setvalue(10);

 XYZ xyz;

 xyz.setvalue(20);

 max(xyz,abc);

 return 0;

}

**This Pointer**

#include<iostream.h>

#include<string.h>

class person

{

 char name[20];

 float age;

public:

 person(char \*s, float a)

 {

 strcpy(name, s);

 age = a;

 }

 person &person :: greater(person & x)

 {

 if(x.age>= age)

 return x;

 else

 return \*this;

 }

 void display(void)

 {

cout<<"Name:"<<name<<"\n"

 <<"Age: "<<age<<"\n";

 }

};

int main()

{

 person p1("John", 37.50),

 p2("Ahmed",29.0),

 p3("Hebber", 40.5);

 person p = p1.greater (p3);

 cout<<"Elder Person is:\n";

 p.display();

 p = p1.greater (p2);

 cout<<"Elder Person is:\n";

 p.display();

 return 0;

}

**EXPERIMENT – 7**

**Aim:**Write programs to understand the usage of constant data member and member function, static data member and member function in a class.

**Solution:**

#include<iostream.h>

#include<conio.h>

class test

{

 int code;

 static int count;

public:

 void setcode(void)

 {

 code = ++count;

 }

 void showcode(void)

 {

 cout<<"object number:"<<code<<"\n";

 }

 static void showcount(void)

 {

 cout<<"count:"<<count<<"\n";

 }

};

int test :: count;

int main()

{

test t1,t2;

 t1.setcode();

 t2.setcode();

 test :: showcount();

 test t3;

 t3.setcode();

 test :: showcount();

 t1.showcode();

 t2.showcode();

 t3.showcode();

 return 0;

}

**EXPERIMENT – 8**

**Aim:**Implement different types of inheritance, function overriding and virtual function

**Solution:**

#include<iostream.h>

class student

{

 protected:

 int roll\_number;

 public:

 void get\_number(int a)

 {

 roll\_number = a;

 }

 void put\_number(void)

 {

 cout<<"Roll No:"<<roll\_number<<"\n";

 }

};

class test : public student

{

 protected:

 float part1, part2;

public:

 void get\_marks(float x, float y)

 {

part1 = x;

 part2 = y;

 }

 void put\_marks(void)

 {

 cout<<"Marks obtained"<<"\n"

 <<"part1 ="<<part1<<"\n"

 <<"part2 ="<<part2<<"\n";

 }

};

class sports

{

 protected:

 float score;

 public:

 void get\_score(float s)

 {

 score = s;

 }

 void put\_score(void)

 {

 cout<<"Sports wt:"<<score<<"\n\n";

 }

};

class result : public test, public sports

{

 float total;

public:

 void display(void);

};

void result ::display(void)

{

 total = part1 + part2 + score;

 put\_number();

 put\_marks();

 put\_score();

 cout<<"Total Score:"<<total<<"\n";

}

int main()

{

 result student\_1;

 student\_1.get\_number (1234);

 student\_1.get\_marks (27.5, 33.0);

 student\_1.get\_score (6.0);

 student\_1.display ();

 return 0;

}

**Function Overriding and Virtual Function**

#include<iostream.h>

class Base

{

 public:

 void display()

 {

 cout<<"\n Display Base";

 }

 virtual void show()

 {

 cout<<"\n Show Base:";

 }

};

class Derived : public Base

{

 public:

 void display()

 {

 cout<<"\n Display Derived";

 }

 void show()

 {

 cout<<"\n Show Derived";

 }

};

int main()

{

 Base B;

 Derived D;

 Base \*bptr;

 cout<<"\n bptr points to Base\n";

 bptr = &B;

 bptr ->display ();

 bptr ->show ();

 cout<<"\n\n bptr points to derived\n";

 bptr = &D;

 bptr ->display ();

 bptr ->show ();

 return 0;

}

**EXPERIMENT – 9**

**Aim:**Implement Operator overloading concepts.

**Solution:**

#include <iostream.h>

class complex

{

 float real, imag;

public:

complex( ) { }

complex( float r, float i )

 {

 real = r;

imag = i;

 }

 void getdata( )

 {

 float r, i;

cout<<endl<< "Enter real and imaginary part ";

cin>> r >>i;

 real = r;

imag = i;

 }

 void setdata( )

 {

 real = r;

imag = i;

 }

void displaydata( )

 {

cout<<endl<< "real = " << real;

cout<<endl<<Imaginary = "<<imag;

 }

complex operator +( complex c )

 {

complex t;

t.real = real + c.real;

t.imag = imag + c.imag;

 }

complex operator \*( complex c )

 {

complex t;

t.real = real \* c.real - imag \* c.imag;

t.imag = real \* c.imag + c.real \* imag;

 return t;

 }

} ;

void main( )

{

complex c1, c2 ( 1.2, -2.5 ), c3, c4;

 c1.setdata( 2.0, 2.0 );

 c3 = c1 + c2;

 c3.displaydata( );

 c4.getdata( );

complex c5 ( 2.5, 3.0 ), c6;

 c6 = c4 \* c5;

c6.displaydata( );

complex c7;

 c7 = c1 + c2 \* c3;

 c7.displaydata( );

}

**EXPERIMENT – 10**

**Aim:**Write programs to understand function template and class template.

**Solution:**

#include<iostream.h>

template<class T>

void bubble(T a[],int n)

{

 for(int i=0;i<n-1;i++)

 for(int j=n-1;i<j;j--)

 if(a[j]<a[j-1])

 {

 swap(a[j],a[j-1]);//cals template function

 }

}

template<class X>

void swap(X &a, X &b)

{

 X temp =a;

 a=b;

 b=temp;

}

int main()

{

 int x[5]={10,50,30,40,20};

 int y[5]={1.1,5.5,3.3,4.4,2.2};

 bubble(x,5);///calls template function for int values

 bubble(y,5);//calls template function for float values

 cout<< "sorted x-array:";

for(int i=0;i<5;i++)

 cout<<x[i]<< "";

 cout<<endl;

 cout<< "sorted y-array:";

 for(int j=0;j<5;j++)

 cout<<y[j]<< "";

 cout<<endl;

 return 0;

}

**EXPERIMENT – 11**

**Aim:**Write programs to understand exception handling techniques.

**Solution:**

#include<iostream.h>

void test(int x)

{

 try

 {

 if(x==1)

 throw x;

 else

 if(x==0)

 throw 'x';

 else

 if(x==-1)

 throw 1.0;

 cout<<"End of try-black\n";

 }

 catch(char c)

 {

 cout<<"Caught a Character\n";

 }

 catch(int c)

 {

 cout<<"Caught an Integer\n";

 }

 catch(double c)

 {

 cout<<"Caught a Double\n";

 }

 cout<<"End of try-catch system\n";

}

int main()

{

 cout<<"Testing Multiple Catches\n";

 cout<<"x==1\n";

 test(1);

 cout<<"x==0\n";

 test(0);

 cout<<"x==2\n";

 test(2);

 return 0;

}

**EXPERIMENT – 12**

**Aim:**Write programs to understand file handling techniques.

**Solution:**

#include <fstream.h>

void main( )

{

 char cource[ 67 ],

target[ 67 ];

 char ch;

cout<<endl<< "Enter source filename";

cin>> source;

cout<<endl<< "Enter target filename";

cin>> target;

ifstreaminfile ( source );

ofstreamoutfile ( target );

while( infile )

 {

infile.get( ch );

outfile.put( ch );

 }

}