```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF
 * PURE VIRTUAL CLASS, ABSTRACT CLASS AND LATE
 * BINDING IN THE POLYMORPHISM*/

 /* NAME: SAGAR GIRI, ROLL: 205 , SECTION : A */
#include <iostream>
using namespace std;
class Base          //ABSTRACT CLASS
{
      public:
            virtual void show() = 0;  //PURE VIRTUAL FUNCTION
};
class Derived1 : public Base
{
      public:
            void show()
            {
                  cout<<"This is Derived1..."<<endl;
            }
};
class Derived2 : public Base
{
      public:
            void show()
            {
                  cout<<"This is Derived2..."<<endl;
            }
};
int main()
{
      Base *ptr;
      Derived1 dv1;
      Derived2 dv2;
      cout<<"Enter the choice 1 or 2"<<endl;
      int c;
      cin>>c;

      //THE OUTPUT DEPENDS ON THE USER INPUT i.e. IN RUN-TIME
      if(c == 1)
      {
            ptr = &dv1;
      }
      else
      {
            ptr = &dv2;
      }
      ptr -> show();
return 0;
}
```

OUTPUT:

```
Enter the choice 1 or 2
1
This is Derived1...
```

```
Enter the choice 1 or 2
2
This is Derived2...
```

```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF VIRTUAL
 * FUNCTION AND EARLY BINDING OF STATIC POLYMORPHISM*/

 /* NAME: SAGAR GIRI, ROLL: 205, SECTION: A*/
#include <iostream>
using namespace std;
class Base   //ABSTRACT CLASS
{
      public:
            virtual void show()
            {
                  cout<<"This is base..."<<endl;
            }
};
class Derived1 : public Base
{
      public:
            void show()
            {
                  cout<<"This is derived one..."<<endl;
            }
};
class Derived2 : public Base
{
      public:
            void show()
            {
                  cout<<"This is derived two..."<<endl;
            }
};
int main()
{
      Base *ptr; //BASE CLASS POINTER

      /* BASE CLASS POINTER COULD HOLD THE
       * ADDRESS OF DERIVED CLAAS'S OBJECT
       * AND VICE-VERSA IS NOT POSSIBLE*/

      Derived1 dv1;
      Derived2 dv2;
      ptr = &dv1;
      ptr -> show();

      ptr = & dv2;
      ptr -> show();
return 0;
}
```

OUTPUT:

```
This is derived one...
This is derived two...
```

```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF STATIC FUNCTIONS
 * AND A DESTRUCTOR */

/* NAME: SAGAR GIRI, ROLL: 205, SECTION: A*/
#include <iostream>
using namespace std;
class gamma
{
	private:
		static int total;
		int id;
	public:
		gamma()                  //DEFAULT CONSTRUCTOR
		{
			total++;
			id = total;
		}
		~gamma()           //DESTRUCTOR
		{
			total--;
			cout<<"Destroying ID number "<<id<<endl;
		}
		static void showTotal() //STATIC FUNCTION
		{
			cout<<"Total is : "<<total<<endl;
		}
		void showID()             //NON-STATIC FUNCTION
		{
			cout<<"ID number is : "<<id<<endl;
		}
};
int gamma::total = 0;

int main()
{
	gamma g1;
	//STATIC FUNCTION ARE ASSOCIATED WITH THE CLASS ITSELF
	gamma::showTotal();

	gamma g2,g3;
	gamma::showTotal();

	//NON-STATIC FUNCTIONS ARE ASSOCIATED WITH THE OBJECTS ONLY
	g1.showID();
	g2.showID();
	g3.showID();

	cout<<"<-----End Of Program------>"<<endl;
//DESTRUCTORS ARE CALLED JUST BEFORE THE RETURN STATEMENT
return 0;
}
```

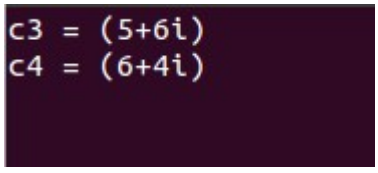
OUTPUT:

```
Total is : 1
Total is : 3
ID number is : 1
ID number is : 2
ID number is : 3
<-----End Of Program------>
Destroying ID number 3
Destroying ID number 2
Destroying ID number 1
```

```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF ASSIGNMENT AND
 * COPY INITILIZATION */

/* NAME: SAGAR GIRI, ROLL No. 205 , SECTION: A*/
#include <iostream>
using namespace std;
class Complex
{
      private:
            int real;int imag;
      public:
            Complex()                        //DEFAULT CONSTRUCTOR
            {     real=0; imag=0;     }
            Complex(int r,int i)      //2 ARGUMENT CONSTRUCTOR
            {
                  real = r;
                  imag = i;
            }
            Complex(Complex &x)       //COPY CONSTRUCTOR
            {
                  real = x.real;
                  imag = x.imag;
            }
            void display()
            {
                  cout<<"("<<real<<"+"<<imag<<"i)"<<endl;
            }
            void operator = (Complex &);     //OPERATOR OVERLOAD FOR ASSIGNMENT
};
void Complex::operator =(Complex &x)
{
      real = x.real;
      imag = x.imag;
}
int main()
{
      Complex c1(5,6),c2(6,4);
      Complex c3 = c1;                //INITILIZATION:CALLS COPY CONSTRUCTOR
      cout<<"c3 = ";c3.display();
      Complex c4;
      c4 =c2;                         //ASSIGNMENT: CALLS OPERATOR = OVERLOADING
      cout<<"c4 = ";c4.display();
return 0;
}
```

OUTPUT:

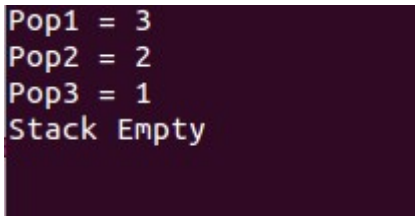


```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF EXCEPTION HANDLING*/

/* NAME: SAGAR GIRI, ROLL No. 205, SECTION : A*/
#include <iostream>
const int MAX = 3;
using namespace std;
class Stack
{
	private:
		int St[MAX];
		int top;
	public:
		Stack()
		{ top = -1; }
		class full   //CLASS TO THROW STACK FULL EXCEPTION
		{	};
		class empty  //CLASS TO THROW STACK EMPTY EXCEPTION
		{	};
		void push(int var)
		{
			if(top >= MAX-1)
			{
				//THROWS FULL OBJECT FOR STACK FULL EXCEPTION
				throw full();
			}
			St[++top] = var;
		}
		int pop()
		{
			if(top < 0)
			{
				//THROWS EMPTY OBJECT FOR STACK EMPTY EXCEPTION
				throw empty();
			}
			return (St[top--]);
		}
};
int main()
{
	Stack s1;
	try{		//TRY BLOCK
		s1.push(1);
		s1.push(2);
		s1.push(3);
		/*s1.push(4); */ //THROWS STACK FULL EXCEPTION
		cout<<"Pop1 = "<<s1.pop()<<endl;
		cout<<"Pop2 = "<<s1.pop()<<endl;
		cout<<"Pop3 = "<<s1.pop()<<endl;
		cout<<"Pop4 = "<<s1.pop()<<endl; //THROWS STACK EMPTY EXCEPTION

	}
	catch(Stack::full)	//CATCH BLOCK FOR STACK FULL EXCEPTION
	{
		cout<<"Stack Full"<<endl;
	}
	catch(Stack::empty)//CATCH BLOCK FOR STACK EMPTY EXCEPTION
	{
		cout<<"Stack Empty"<<endl;
	}
return 0;
}
```

OUTPUT:



```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF TEMPLATE FUNCTION
 * AND ITS USE */

 /* TEMPLATES IS USED TO HANDLE THE SINGLE
  * DEFINITION OF FUNCTION FOR MANY DATATYPES*/

/* NAME: SAGAR GIRI, ROLL No. 205, SECTION : A*/
#include <iostream>
using namespace std;
template <class T>        //FUNCTION TEMPLATE
T abs(T a)                      //RETURNS VALUE OF 'T' DATATYPES
{
      if(a < 0)
            return (-a);
      else
            return a;
}
int main()
{
      int a = 5;
      int b = -6;
      float c = -3.5;
      float d = 4.5;
      cout<<"|a| = "<<abs(a)<<endl;   //CALLS ABS() FUNTION WITH INT DATA-TYPE
      cout<<"|b| = "<<abs(b)<<endl;
      cout<<"|c| = "<<abs(c)<<endl;   //CALLS ABS() FUNTION WITH FLOAT DATA-TYPE
      cout<<"|d| = "<<abs(d)<<endl;
return 0;
}
```

OUTPUT:

```
|a| = 5
|b| = 6
|c| = 3.5
|d| = 4.5
```

```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF "this" POINTER*/
/* "this" POINTER IS USED TO REFER (THE ADDRESS) OF THAT OBJECT*/
/* "this" IS A KEYWORD IN C++ PROGRAMMING LANGUAGE */

/* NAME: SAGAR GIRI, ROLL No. 205, SECTION: A */
#include <iostream>
using namespace std;
class alpha
{
	private:
		int a;
	public:
		alpha()
		{ a = 0; }
		alpha(int x)
		{ a = x; }
		void displayAddress()
		{
			cout<<"Address is = "<<this<<endl;	//PRINTS ADDRESS
		}
		void display()
		{
			cout<<"\ta = "<<a<<endl;		//PRINTS VALUE
		}
};
int main()
{
	alpha a1;
	alpha a2(5);
	cout<<"Detials of a1 = "<<endl;
	a1.displayAddress();
	a1.display();
	cout<<"Detials of a2 = "<<endl;
	a2.displayAddress();
	a2.display();
return 0;
}
```

OUTPUT:

```
Detials of a1 =
Address is = 0x7fffa9699780
        a = 0
Detials of a2 =
Address is = 0x7fffa9699790
        a = 5
```