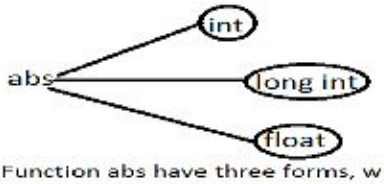
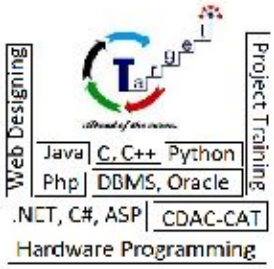
Polymorphism:

- The C++ technology is based on concepts OOP (Object Oriented Programming) approach.
- Polymorphism is one of the **main features** of OOP.
- **Poly means many** whereas **morphism stands for forms** i.e. many forms of an element is polymorphism such as you and you have this feature. You play many roles such in life as son, father, teacher, uncle etc.
- **Multi-behaving ability of the element is called polymorphism.**

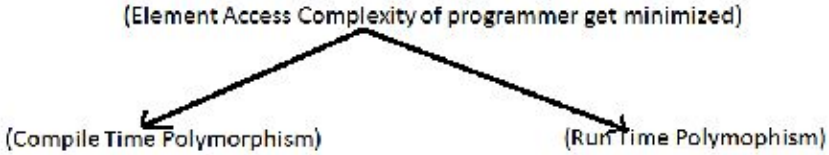


Here Function abs have three forms, when we call it we only remember one command because all forms name is abs.



Polymorphism

Def : One Element (object) having many behaviors.
(Element Access Complexity of programmer get minimized)



- **Compile Time Polymorphism**
  - Overloading is compile time polymorphism where more than one methods share the **same name with different parameters**.
  - It is also known as **static polymorphism, false polymorphism**
  - This can be achieving by **absolute address binding**, popularly known as compile time binding or static binding.
  - e.g.
    - **Function overloading**
    - **Constructor overloading**
    - **Operator overloading**
- **Run Time Polymorphism**
  - Dynamic Polymorphism is where the decision to choose which methods to execute, is set during the run-time.

e.g.

```
#include<iostream>
#include<process.h>
using namespace std;
class Base
{
    public :
            void display ()
            {
                    cout << "\n I am the display function of Base class";
            }
```

```
};
class Derive : public Base
{
    public :
            void display ()
            {
                    cout << "\n I am the display function of Derive CLASS";

            }
            void show()
            {
                    cout <<"\ni am in the show of derive class ";
            }
};


main ()
{
    system("cls");
    Base b ;
    Derive d ;

    Base *bptr  = &b ;
    bptr -> display();

    Derive *dptr = &d;
    dptr -> display();
    dptr -> show();

    Base &bref = b ;
    bref.display();

    Derive &dref = d ;
    dref.display();

    bptr = &d ;  // Yes
    bptr->display();  // Base class version of display invoked

}
```


- Why Base class pointer store the address of derive class object but opposite is not true?
    - Base class pointer can store the address of the derive class because derive class have sub object of base class.
- **Note**
    - **Base class pointer, not only store address of same class object as well as store address of any object whose class is directly or indirectly inherit from the base class.**
    - **Base class pointer only accesses that member they are direct member of the base class.**

- If we try to access the member of derive class from the Base class pointer then technology generate compile time error.

Virtual function

- A virtual function a member function which is declared within base class and is re-defined (Overridden) by derived class.
- When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.
- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve **Runtime polymorphism**
- Functions are declared with a virtual keyword in base class.
- The resolving of function call is done at Run-time.
- Rules for Virtual Functions
  - They must be declared in **public section of class.**
  - Virtual functions cannot be static and also cannot be a friend function of another class.
  - **Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.**
  - The prototype of virtual functions should be same in base as well as derived class.
  - They are always defined in base class and overridden in derived class.
  - It is not mandatory for derived class to override (or re-define the virtual function), in that case base class version of function is used.
  - A class may have virtual destructor but **it cannot have a virtual constructor.**


```
#include<iostream>
using namespace std;
class base
{
    public :
            virtual void display()
            {
                    cout <<"\ndisplay of base";
            }
};

class derive1 : public base
{
    public :
            void display()
            {
                    cout <<"\ndisplay of derive1";
            }
            void show()
            {
                    cout << "show of derive1";
            }
};
```

```
class derive2 : public base
{
    public :
            void display()
            {
                    cout <<"\ndisplay of derive2";
            }

} ;

main ()
{
    base *bptr;
    derive1 d1;
    derive2 d2;
    bptr = &d1 ;// Yes
    bptr -> display();
    //bptr -> show();
    bptr = &d2 ;// Yes
    bptr -> display();
}
```


e.g.

```
#include<iostream>
using namespace std;
class Dadaji
{
        public :
                virtual void f1 ()
                {
                        cout <<"\n I am in f1 of base";
                }
                void f2()
                {
                        cout << "\n I am in f2 of base ";
                }
} ;
class Pitaji : public Dadaji
{
        int a ;
        public :
                void f1()
                {
                        cout <<"\n I am in f1 of derive";
                }
                void f3()
                {
                        cout << "\n I am in f3 of derive";
                }
} ;
class Betaji : public Pitaji
```

```
{
        int k,l ;
        public :
                void f1()
                {
                        cout <<"\n I am in f1 of derive";
                }
                void f3()
                {
                        cout << "\n I am in f3 of derive";
                }
} ;

main ()
{

        Dadaji d ;
        Pitaji p ;
        Betaji b ;
        cout <<"\nsizeof d" << sizeof d ;
        cout <<"\nsizeof p" << sizeof p ;
        cout <<"\nsizeof b" << sizeof b ;
}
```


Output :

```
sizeof d  - 2 byte
sizeof p  - 4 byte
sizeof b  - 8 byte
```

- **How this is possible?**
  - When we apply the virtual function in class, then technology silently place a virtual pointer (vptr) and initialized with the help of constructor.
- **Why The Technology putting such virtual pointer.**
  - virtual pointer stores the address of VTABLE(virtual table)
- **What kind of information stores in the Virtual Table.**
  - Address of all virtual members of that class.

Example for Dynamic Polymorphism

e.g.

```
#include <iostream>
#include <stdlib.h>
using namespace std;
//MicroSoft set Partial Standard
class printer
{
        public:
        virtual void print(char *matter)
        {

        }
};
```

```
//Company Hewlett Packard
class Hp1018 : public printer
{
        public:
        void print(char *matter)
        {
                cout << "\nPrinting on 1018 Laser Printer";
                cout << matter;
        }
};
//Company Hewlett Packard
class HpF012 : public printer
{
        public:
        void print(char *matter)
        {
                cout << "\nPrinting on F012 Laser Printer";
                cout << matter;
        }
};
//Company Intex
class Intex : public printer
{
        public :
                void print(char *matter)
                {
                        cout << "\nPrinting on intex Laser Printer";

                        cout << matter;
                }
};


//Microsoft
main()
{
        int choice ;
        char *mtr ;
        printer *ptr ;
        do
        {
                system("cls");
                cout <<"\n1. HP 1018 ";
                cout <<"\n2. Hp F012 ";
                cout <<"\n3. Intex ";
                cout <<"\n4. exit ";
                cout <<"\nEnter your choice";
                cin >> choice;
                switch(choice )
                {
                        case 1 :
```

```
                ptr = new Hp1018();
                mtr = "doc1";
        break ;
        case 2:
                ptr = new HpF012();
                mtr = "doc2";
        break ;
        case 3:
                ptr = new Intex();
                mtr ="doc3";
        break ;
        case 4:
                exit(1) ;
    }

    ptr->print(mtr);   //dynamic polymorphism


    system("pause");


  }while(1) ;
}
```


- When a class contains at least one pure virtual function then the class is known as abstract class.

  e.g.

```
class printer
{
        public:
                virtual void print(char *matter) = 0; // pure virtual function
};
```

- In this case printer is an **abstract class.**
- Note

  virtual void print(char *matter) = 0 ;

  Here 0 does not assign to a function print, it simply means print is a **pure virtual function**

- Once a class become abstract, and then any developer can use it only by **inheritance and must be override pure virtual function.**
- However abstract class not only contains normal members as well contains some standards (pure virtual functions) that must overridden by sub classes if subclasses want the service of the base class.
- Abstract class made force to the developer who inherit the abstract class to override the pure virtual function, if the developer of derive class not to override the pure virtual method in derive class then derive class become abstract class.
- Nobody can create the object of abstract class because they partially implemented class.

- Note
  - **Constructor cannot be virtual but Destructor can be virtual**

e.g.

```
#include<iostream.h>
#include<conio.h>
class base
{
        public:
                base()
                {
                        cout <<"\nbase class constructor" ;
                }
                virtual ~base()
                {
                        cout << "\nbase class destructor";
                }
} ;
class derive: public base
{
        public:
                derive()
                {
                        cout <<"\nderive class constructor" ;
                }
                ~derive()
                {
                        cout << "\nderive class destructor";
                }
} ;
main ()
{
        base *ptr ;
        ptr = new derive ;
        delete ptr ;
}
```


- Note
  - **The mechanism of the virtual is start after success full creation of the object** which only possible after the completion of the constructor.

- **virtual function in a derive class :**

e.g.

```
#include<iostream.h>
#include<conio.h>
class base
{
        public:
                base()
                {
                        cout <<"\nbase class constructor" ;
                }
```

```
        virtual ~base()
        {
                cout << "\nbase class destructor";
        }
        virtual void f1()
        {
                cout << "\nf1 of base class ";
        }
} ;
```



```
class derive: public base
{
        public:
                derive()
                {
                        cout <<"\nderive class constructor" ;
                }
                ~derive()
                {
                        cout << "\nderive class destructor";
                }
                void f1()
                {
                        cout <<"\nf1 of derive class";
                }
                virtual f2()
                {
                        cout <<"\nf2 in derive class";
                }

} ;
main ()
{
        clrscr();
        base *ptr ;
        ptr = new base;
        /*other code */
        delete ptr ;
        ptr = new derive ;
        ptr -> f1() ;
        ptr->f2(); // Error- f2 is not accessible by base class pointer
        delete ptr ;
}
```


Note : virtual keyword should be place in the base class

Interface: It is a contract

```
class InterfaceForXYZ
```

```
{       virtual void a()= 0 ;
        virtual void b()= 0 ;
        virtual void c()= 0 ;
        virtual void d()= 0 ;
        virtual void e()= 0 ;
} ;
class Test : InerfaceForXYZ
{
        run(InterfaceForXYZ *ptr) ;
        {
                ptr->a();
        }
}
```