POLYMORPHISM AND POINTERS IN C++

By J.V.Patil

Edit with WPS Office

INTRODUCTION

- A pointer is a variable which holds the memory address of another variable.

Pointer Declaration:

- When a pointer variable is declared, the variable name must be preceded by an asterisk(*).
- The syntax of declaration of pointer is:

```
data_type *pointer_variable;
```

- Example:

```
int *p;
```

It declares the variable p as a pointer variable that points to an integer data type.

Edit with WPS Office

- Accessing the Address of a Variable
  - We can obtain, address of a variable with the help of the operator &.
  - The operator & preceding a variable returns the address of the variable associated with it.
  - Example,

```
p=&ptr;
```

The above, statement will assign the address 5000 to variable p.

- Initialization of Pointer Variable
  - The process of assigning the address of a variable to a pointer variable is known as initialization.
  - Example-

```
int quantity;
int *p;          //declaration
p=&quantity;  //initialization
```

Edit with WPS Office

- Accessing a Variable through its Pointer
  - Once pointer has been assigned the address of a variable, the remains as to how to access the value of the variable using the pointer?
  - This is done by using another unary operator *(asterisk), usually known as indirection operator or de-referencing operator.
  - Example,

```
int quantity,*p, n;
quantity=179;
p=&quantity;     //assigns the address of quantity to the pointer variable p
n=*p;            //the pointer returns the value of the variable of which pointer value is address
```

In this case, *p returns the value of the variable quantity, because p is the address of quantity.

  - The * can be remembered as 'value at address'
  - E.g.

```
int x, y, *p1, *p2;
p1=&x;
p2=p1;//The address of p1 is assigned to the pointer variable p2. The contents of p1 and p2 will be same as these two pointer variables hold the same address.
```

Edit with WPS Office

ACCEPT A NUMBER FROM USER AND PRINT ITS VALUE AND ADDRESS USING POINTER.

```
#include<iostream>
using namespace std;
void main(){
    int n, *p,x;
    p=&n;
    cout<<"Enter number: ";
    cin>>n;
    cout<<"The number is:  "<<*p<<endl;
    x=*p;
    cout<<"The number is: "<<x<<endl;
    cout<<"It's address is: "<<p<<endl;
}
```

OUTPUT

```
Enter number: 3
The number is: 3
The number is:3
It's address is:0x8fa6ffff4
```

Edit with WPS Office

POINTER TO OBJECTS

- A pointer can point to an object created by a class. We can also make a pointer to point to an object created by a class similar to that of normal variables. A pointer is a variable that holds the address of other variables or objects.
- In pointer to objects the starting address of the object is assigned to the pointer. After that, we can access the members of a class with the help of the arrow operator (->) rather than the dot operator (.).
- For example: s.get();

  ptr->get()
- The Syntax for creating pointer to object is as follows:

  ```
  class_name *pointer_name, object_name;
  pointer_name=&object_name;
  ```
- In the above syntax, the & operator returns the starting address of the object specified by the object_name. Next, the resultant address is assigned to the pointer variable specified by the pointer_var.

Edit with WPS Office

EXAMPLES

- Write a program to declare a class book containing data members book_title, author name and price. Accept and display this information for one object of the class using pointer to that object.
- WAP to declare a class ‘Product” having data members as product_name and product_price. Accept and display this data for one object using pointer to the object.
- WAP to declare a class employee having data members emp_id, emp_name and department. Accept & display this information for 5 objects.

Edit with WPS Office

POINTER TO DERIVED CLASSES

- We can use pointers not only to base objects but also to the objects of derived classes.
- Pointers to objects of a base class are type compatible with pointers to objects of a derived class, thus allowing a single pointer variable to be used as a pointer to objects of a base class and its derived classes.
- For example, if B is a base class and D is derived class from B, then a pointer declared as pointer to B can also be a pointer to D.
- Consider the following declarations:

```
B *ptr;     //Pointer to class B
   B b;     //base object
   D d;     //derived object
   ptr=&b;     //ptr points to object b
   ptr=&d;     //ptr points to object d
```

  This is valid because **d** is an object derived from the class **B**.
- However, there is a problem in using ptr to access the public members of the derived class D.

Edit with WPS Office

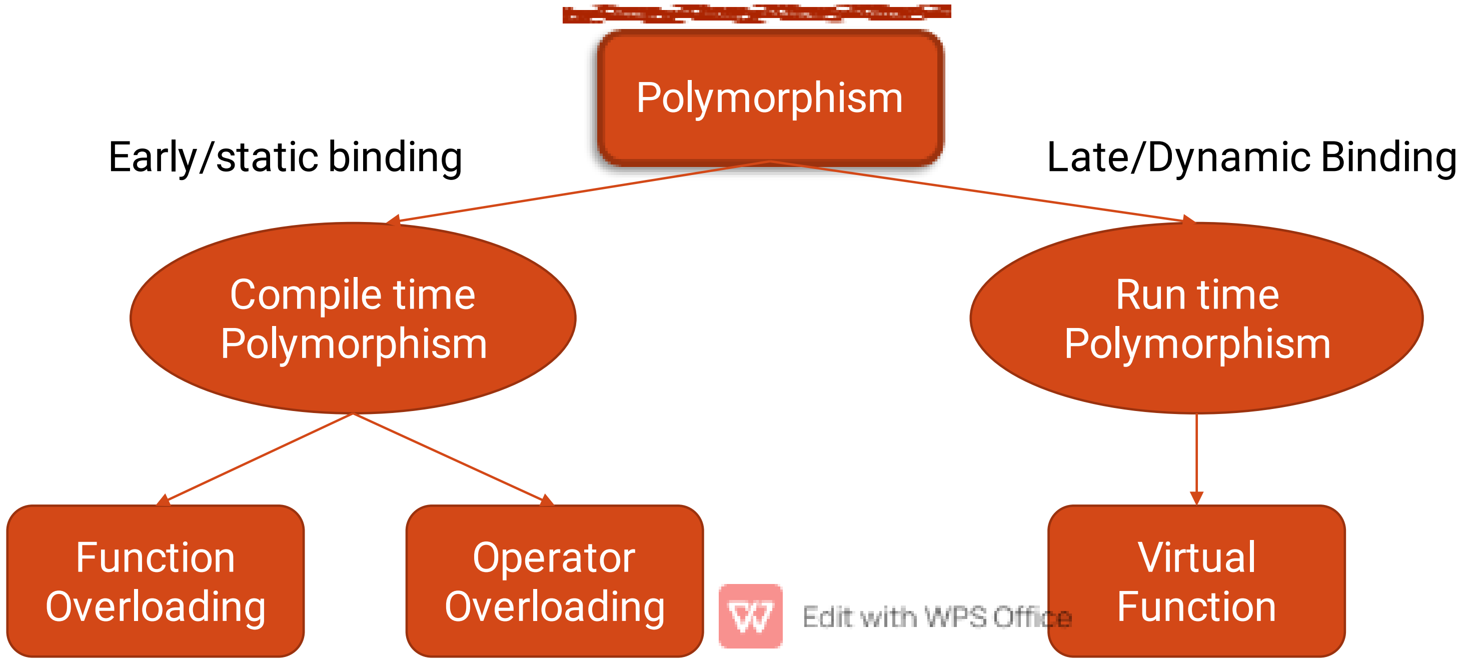
POINTER TO DERIVED CLASSES

- Using ptr, we can access only those members which are inherited from B and not the members that originally belongs to D.
- In case a member of D has the same name as one of the member of B, then any reference to that member by ptr will always access the base class member.
- Although C++ permits a base pointer to point any object derived from that base, the pointer cannot be directly used to access all the members of derived class.
- We may have to use another pointer declared as pointer to the derived type.

Edit with WPS Office

INTRODUCTION TO POLYMORPHISM

- Polymorphism is one of the crucial feature of OOP. Poly means many and Morphism means forms.
- It simply means that 'one name, multiple forms'.
- Polymorphism is implemented by using the overloaded functions and operators.



Edit with WPS Office

TYPES OF POLYMORPHISM

- 1. Compile Time Polymorphism: Information is known to the compiler at compiler time and therefore compiler is able to select the appropriate function for a particular call at compile time itself. This is called **early binding** or **static binding** or **static linking**. Also known as compile time polymorphism.
- Early binding is implemented using operator overloading and function overloading.

- 2. Run-time Polymorphism: When the required information to call a function is known to the compiler at the execution time or run-time, it is known as run-time polymorphism, **late binding, or dynamic binding.**
- Dynamic binding requires the use of pointer to objects.
- The dynamic binding is implemented using virtual function.

Edit with WPS Office

DIFFERENCE BETWEEN RUN TIME AND COMPILE TIME POLYMORPHISM

Compile Time	Run Time
When the required information to call a function is known to the compiler at the compile time, it is known as compile time polymorphism	Calling a function or assigning a value to a variable, at run time is called run time polymorphism.
Function to be called is unknown until appropriate selection is made.	Functions to be called are known well before.
This does not require use of pointers to object.	This requires use of pointers to object
Function calls execution are faster	Function calls execution are slower
It is also known as static binding or early binding	It is also known as dynamic binding or late binding
It is implemented with operator overloading or function overloading	It is implemented with virtual function.

Edit with WPS Office

FUNCTION OVERLOADING

- Function overloading: The term overloading refers to the use of same thing for different purpose.
- Function overloading means we can use same function name to create functions that perform different tasks depending upon the context.
- When a function is overloaded, the same function has more than one definition where difference is either in number of parameters or in data type of parameter or both.
- Therefore depending on the way the function called that corresponding function definition is used to call the function.
- Syntax: return_type function_name(actual arguments);

Edit with WPS Office

FUNCTION OVERLOADING:

- Example:

```
int add(int x, int y);
void add(float, float, float);
```

- Above function add is with two integer parameters and with three float parameters therefore when function is called as add(2,4) first function definition will be used and if function is called as add(1.2, 2.2,3.2) the second definition will be used.
- Function Overloading can be written as :
  - Different types of parameters

    In this type, we define two or more functions with same name and same number of parameters, but
    the type of parameters is different.
  - Different Numbers of parameters

    In this type of function overloading, we define two functions with the same name but different numbers of parameters of the same type

Edit with WPS Office

ADVANTAGES OF FUNCTION OVERLOADING

- The main advantage of function overloading
  - Improve the **code readability** and
  - Allows **code reusability**.

Edit with WPS Office

OPERATOR OVERLOADING

- Operator overloading is an idea of giving special meaning to an existing operator in C++ without changing their original meaning.
- In overloading, the operators are implemented as functions using the operator keyword.
- We can redefine or overload some of the built-in operators by giving user defined meaning to it.
- For example, we can overload an operator '+' in a class for String so that we can concatenate two strings by just using '+'.
- The general form of an operator function is:

```
return_type  operator op(arglist);        // function declaration
return_type class_name:: operator op(arglist) // function definition
{
    ……
}
```

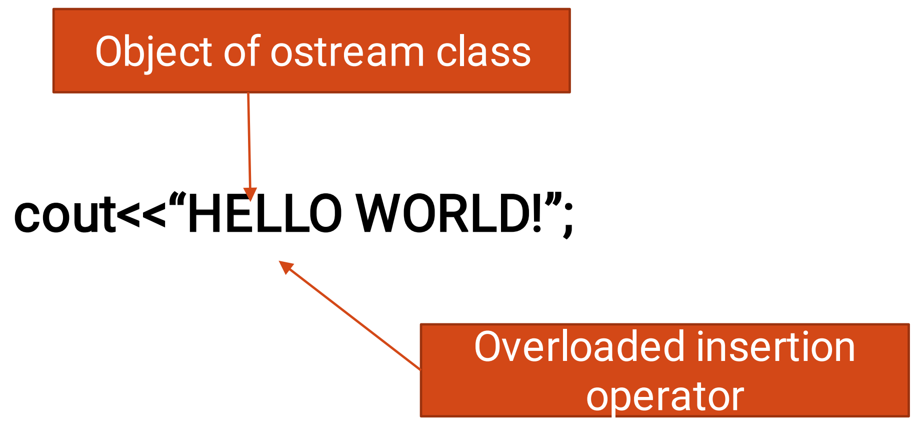
- Where return type is the type of value returned by the specified operation and op is the operator being overloaded. Operator op is the function name.

Edit with WPS Office

OPERATOR OVERLOADING

Example

The input/output operators << and >> are good examples of operator overloading



```
cout<<“HELLO WORLD!”;
```

The built-in definition of the **<<** operator is for **shifting of bits**.
It is also used for **displaying the values** of various data types.

Edit with WPS Office

OPERATOR OVERLOADING

- Operator functions must be either by member function or friend functions.
- The basic differences between them is that a friend function will have only one argument for unary operator and two for binary operators.
- While a member function has no arguments for unary operators and only one for binary operators.
- This is because the object used to invoke the member function is passed implicitly and therefore is available for the member function.
- This is not the case with friend functions. Arguments may be passed either by value or by reference.

Edit with WPS Office

OPERATOR OVERLOADING

- Almost all operators in C++ can be overloaded except the given below:

1. Class member operators (.)
2. Scope resolution operator(::)
3. Sizeof operator(sizeof)
4. Conditional operator(?:)

Edit with WPS Office

OVERLOADING UNARY AND BINARY OPERATOR

- There are two types of operators:

1. Unary Operator:

It requires only one operand to perform operation.

++ or –

Example : i++ , –i

2. Binary Operators:

It requires two operands to perform operation.

+, -, *

Example : a+b , a*b

Edit with WPS Office

OVERLOADING UNARY OPERATOR

- Program:

Overload the unary – operator so that when it is used with an object, the value of numeric data members of the class will be negated.

- By using member function
- By using friend function

Edit with WPS Office

OVERLOADING BINARY OPERATOR

- Program

1. Overload the unary + operator so that when it is used with two objects, the value of numeric data members of a class for both objects will be added.
2. Overload unary '>' operator to compare length of two strings.

Edit with WPS Office

RULES OF OPERATOR OVERLOADING:

1. Only existing operators can be overloaded. New operators cannot be created.
2. The overloaded operator must have at least one operand that is of user-defined type.
3. We cannot change the basic meaning of an operator. That is, we cannot redefine the plus (+) operator to subtract one value from the other.
4. Overloaded operators follow the syntax rules of the original operators.
5. We can not use friend function to overload certain operators (Assignment operator (=), function call (()),subscript operator ([]),class member operator ((->))
6. Unary operators, overloaded by means of a member function, take no explicit arguments and return no explicit values. But those overloaded by means of a friend function take one reference argument.
7. Binary operators overloaded through a member function take one explicit argument and those which one overloaded through a friend function take two.
8. When using binary operators overloaded through a member function, the left-hand operand must be an object of the relevant class.
9. Binary arithmetic operators such as +, - *, and / must explicitly return a value. They must not attempt to change their own arguments.

Edit with WPS Office

VIRTUAL FUNCTION

- Virtual function belongs to runtime polymorphism.
- When we use the same function name in both the base and derived classes, the function in base class is declared as virtual using keyword virtual.
- When a function is made virtual, C++ determines which function to use at run time based on the type of object pointed to by the base pointer, rather than the type of pointer.
- Thus by making the base pointer to point to different objects, we can execute different versions of the virtual function.
- For example,

```
virtual void draw();
```

Edit with WPS Office

RULES FOR VIRTUAL FUNCTION

- The virtual function must be member of the some class.
- They cannot be static members
- A virtual function can be a friend function for another class.
- A virtual function in a base class must be defined, even though it may not be used.
- They are accessed using objects pointers

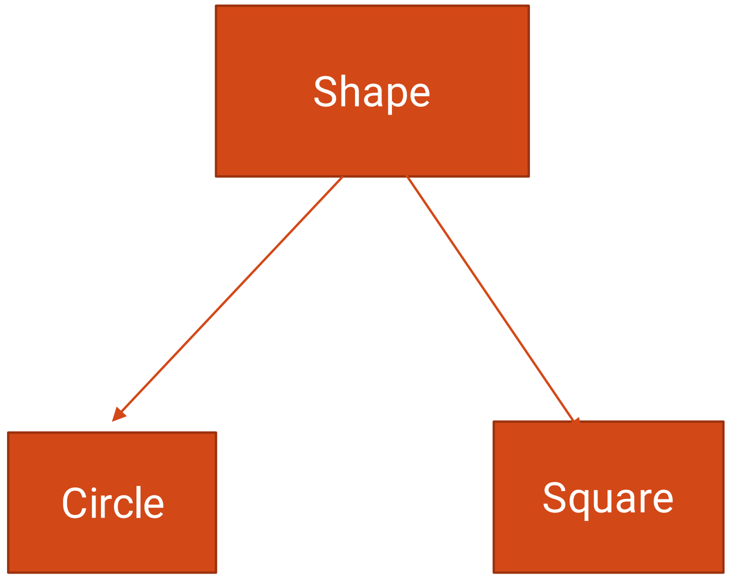
Edit with WPS Office

PURE VIRTUAL FUNCTION

- A pure virtual function is a function declared in a base class that has no definition.
- The compiler requires each derived class to either define the function or re-declare it as a pure virtual function.
- Such functions are also called as ’do-nothing’ functions.
- For example,

```
class ABC
{
public:
virtual void display( )=0;
};
```

Edit with WPS Office



Edit with WPS Office

THANK YOU!!!

Edit with WPS Office