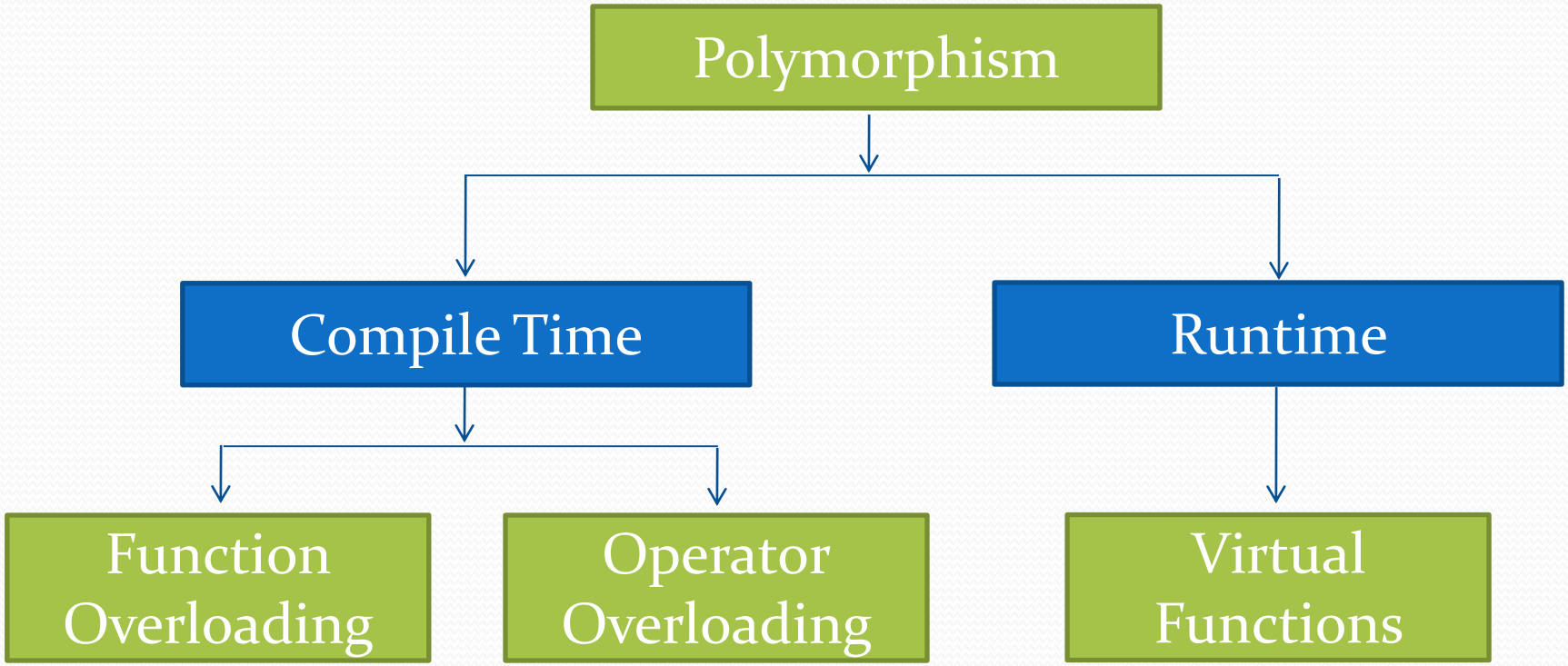
Operator Overloding

Haresh Jaiswal

Rising Technologies, Jalna.

Introduction

- Operator overloading is an important branch of Polymorphism in C++.



Introduction

- In simple words, overloading an operator means assigning additional operation or job to it; relative to a specific class (user defined type).

Introduction

- One of C++'s great features is its extensibility, Operator Overloading is major functionality related to extensibility.
- In C++, most of operators can be overloaded so that they can perform special operations relative to the classes you create.
- For example, + operator can be overloaded to perform an operation of string concatenation along with its pre-defined job of adding two numeric values.
- When an operator is overloaded, none of its original meaning will be lost.

Introduction

- After overloading the appropriate operators, you can use objects in expressions in just the same way you use C++'s built-in data types.

Introduction

- Consider following piece of code. Considering the + operator is overloaded in such a way with class String that it will concatenate two string objects.

```
int main()
{
  int n1 = 10, n2 = 20, n2;
  String s1 = "Rising", s2="Tech", s3;

  n3 = n1 + n2; // addition
  s3 = s1 + s2; // concatenation
}
```

Overloading Operators

- Operators come in three flavours:
  - Unary (++, --)
  - Binary (+, -)
  - Ternary (?:)
- You are free to overload most of the built-in operators, but you cannot create new operators of your own.
- Therefore, although you can give your class an increment operator (++), you cannot create a squared operator.

Overloading Operators

- You can overload these operators to do anything you want, but it is a good programming practice for them to make sense.
- That means you can have the ++ operator to decrement, but it would make no sense to do so.

Operators Function

- Operators are overloaded by creating operator functions
- An *operator function defines the* operations that the overloaded operator will perform relative to the class upon which it will work.
- An operator function is created using the keyword operator
- Operator functions can be either
  - Member function of a class or
  - Non-member function *(Mostly Friend Functions)*
- The way operator functions are written differs between member and non-member functions.

Member Operators Function

- A member operator function for binary operators

```
RetType ClassName::operator#(arg-list)
{
    // operations
}
```

- Often, operator functions returns an object of the class they operate on, but ret-type can be any valid type.
- The # is a placeholder. When you create an operator function, substitute the operator for the #.

Member Operators Function

- A member operator function for binary operators

```
RetType ClassName::operator#(arg-list)
{
    // operations
}
```

- For example, if you are overloading the + operator, use operator+
- While overloading binary operators using member function, the arg-list will contain one parameter.

Member Operators Function

- A member operator function for Unary operators

```
RetType ClassName::operator#()
{
    // operations
}
```

- While overloading an unary operator, arg-list will be empty.

Friend Operators Function

- Instead of Member function, you can define operator functions that will be friend to the class you have created.

Friend Operators Function

- A friend operator function for binary operators

```
RetType operator#(arg-list)
{
    // operations
}
```

- While overloading a binary operator using friend function; argument list must take 2 arguments, one of them must be of user defined type.

Friend Operators Function

- A friend operator function for Unary operators

```
RetType operator#(arg-list)
{
    // operations
}
```

- While overloading an unary operator using friend function; argument list must have 1 argument as reference to the object.

Overloading Operators : Rules

- You can overload virtually any c++ operator, but you cannot create your own set of operators.
- You cannot change any operators precedence.
- You cannot change syntax to use an operator.
- You cannot alter the pre-defined job performed by any operator, that means you cannot overload + operator in such a way to perform subtraction of two integers.

Overloading Operators : Rules

- You can overload most of Unary and Binary Operators, but you cannot overload the ternary operator (?:)
- Binary as well as Unary operators can be overloaded by both Approaches
  - Member Functions Approach
  - Friend Function Approach

Overloading Binary Operators

- While overloading binary operators; at least one of its operand should be of user defined types.
- The operator function must return an object.