IT206 Data Structures Lab with OOP[1]

Rachit Chhaya

DA-IICT

May 24, 2022

[1]Content taken from balgurusamy book and geeksforgeeks.org,programiz.com

Function Overloading

- Using a function with same name to perform different tasks.

Function Overloading

- Using a function with same name to perform different tasks.
- We can design a family of functions with one function name but different argument lists.

Function Overloading

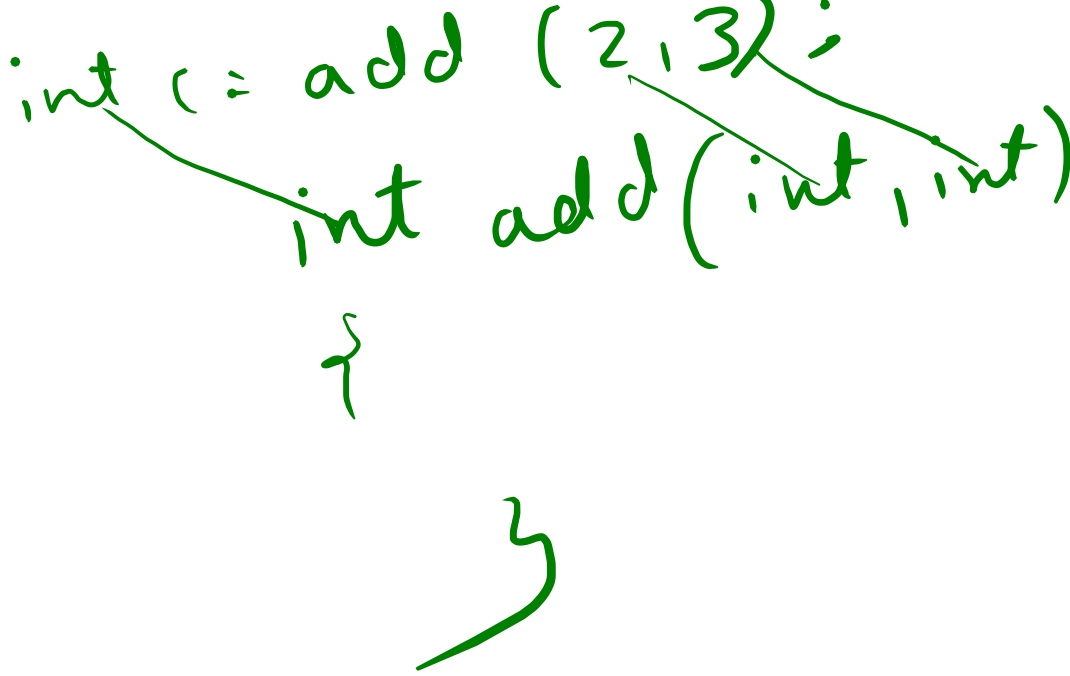
- Using a function with same name to perform different tasks.
- We can design a family of functions with one function name but different argument lists.

```
// Declarations
int add(int a, int b);
int add(int a, int b, int c);
double add(double x, double y);
double add(int p, double q);
double add(double p, int q);

// Function calls
cout << add(5, 10);
cout << add(15, 10.0);
cout << add(12.5, 7.5);
cout << add(5, 10, 15);
cout << add(0.75, 5);
```

Steps for Function Selection

1. Try to find exact match



Steps for Function Selection

char, int, long int

long

1. Try to find exact match
2. Else Use Integral promotions. e.g. : char to int, float to double

Steps for Function Selection

float to int
int to float

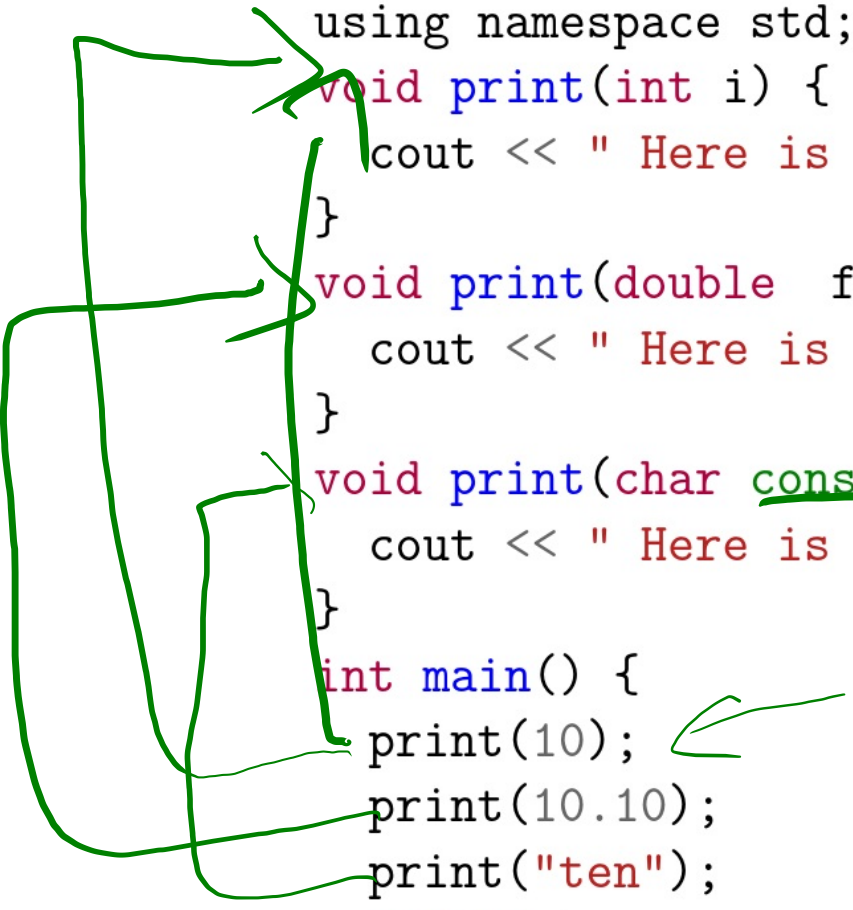
1. Try to find exact match
2. Else Use Integral promotions. e.g. : char to int, float to double
3. Use implicit conversions. If match is not unique , error!

Steps for Function Selection

1. Try to find exact match
2. Else Use Integral promotions. e.g. : char to int, float to double
3. Use implicit conversions. If match is not unique , error!
4. Use user defined conversions if all else fails.

Example Function Overloading

```
#include <iostream>
using namespace std;
void print(int i) {
  cout << " Here is int " << i << endl;
}
void print(double  f) {
  cout << " Here is float " << f << endl;
}
void print(char const *c) {
  cout << " Here is char* " << c << endl;
}
int main() {
  print(10);
  print(10.10);
  print("ten");
  return 0;
}
```



constant

Runtime Polymorphism

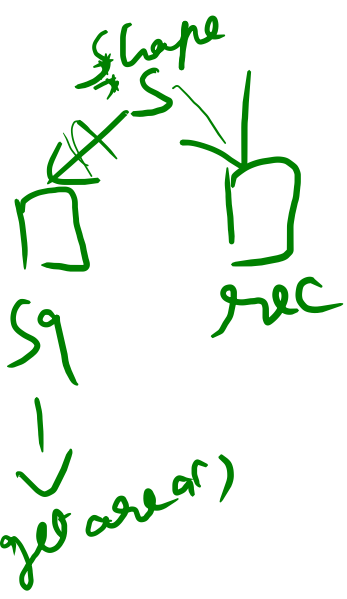
```
#include <iostream>
using namespace std;
// Base class
class Shape {
public:
    Shape(int l, int w)
    {
        length = l;
        width = w;
    }
    int get_Area()
    {
        cout << "This is call to parent class area\n";
        return 1;
    }
protected:
    int length, width;
};
```

Compile time
↓
Static binding

Runtime
↓
Dynamic

```
// Derived class
class Square : public Shape {
public:
    Square(int l = 0, int w = 0)
        : Shape(l, w)
    {
    } // declaring and initializing derived class
    // constructor
    int get_Area()
    {
        cout << "Square area: " << length * width << '\n';
        return (length * width);
    }
};
```

```
    // Derived class
class Rectangle : public Shape {
public:
    Rectangle(int l = 0, int w = 0)
        : Shape(l, w)
    {
    } // declaring and initializing derived class
    // constructor
    int get_Area()
    {
        cout << "Rectangle area: " << length * width
             << '\n';
        return (length * width);
    }
};
```




```
int main()
{
    Shape* s;

    // Making object of child class Square
    Square sq(5, 5);

    // Making object of child class Rectangle
    Rectangle rec(4, 5);
    s = &sq; // reference variable
    s->get_Area();
    s = &rec; // reference variable
    s->get_Area();

    return 0; // too tell the prrogram executed
    // successfully
}
```

Output??

pointer to base class

sq.getarea();

- The derived class's function is called using a base class pointer.

- The derived class's function is called using a base class pointer.
- Solution : Virtual Function

(Dynamic binding
Run time polym)

- The derived class's function is called using a base class pointer.
- Solution : Virtual Function
- A virtual function is a member function that is declared in the base class using the keyword virtual and is re-defined (Overriden) in the derived class

- The derived class's function is called using a base class pointer.
- Solution : Virtual Function
- A virtual function is a member function that is declared in the base class using the keyword virtual and is re-defined (Overriden) in the derived class
- Tells the compiler to perform late binding where the compiler matches the object with the right called function and executes it during the runtime.

Virtual Function Example

```
#include <iostream>
using namespace std;

// Declaration of Base class
class Shape {
public:
    // Usage of virtual constructor
    virtual void calculate()
    {
        cout << "Area of your Shape ";
    }
};
```

```
class Rectangle : public Shape {
public:
    int width, height, area;

    void calculate()
    {
        cout << "Enter Width of Rectangle: ";
        cin >> width;

        cout << "Enter Height of Rectangle: ";
        cin >> height;

        area = height * width;
        cout << "Area of Rectangle: " << area << "\n";
    }
};
```

calculate (١ ، ٣)

```
class Square : public Shape {
public:
    int side, area;

    void calculate()
    {
        cout << "Enter one side your of Square: ";
        cin >> side;

        area = side * side;
        cout << "Area of Square: " << area;
    }
};
```

```
int main()
{

    Shape* S;
    Rectangle r;
    S = &r;

    S->calculate();
    Square sq;
    S = &sq;
    S->calculate();
    return 0;
}
```

Rules for Virtual Function

1. Must be members of some class
2. Cannot be static members
3. Must be accessed using object pointers
4. Can be friend of another class

5. Must be defined in base class
6. Must have identical prototype in all classes.
7. No virtual constructors
8. Base pointer can point to any class object, but reverse is not true
9. If defined in base class not necessarily needs to be defined in derived class.

Pure Virtual Function

```
// An abstract class
class Test
{
    // Data members of class
public:
    // Pure Virtual Function
    virtual void show() = 0;

   /* Other members */
};
```

Pure virtual functions are used

- if a function doesn't have any use in the base class
- but the function must be implemented by all its derived classes

A class is abstract if it has at least one pure virtual function. We cannot make objects of abstract class but it can have pointers, constructors.