Polymorphism in C++

Professor Hossein Saiedian

EECS 348: Software Engineering

November 7, 2023

Polymorphism

- Polymorphism is a concept in programming that allows objects of different types to be treated as if they are objects of a common type
- It's like having a universal remote control that works on different brands of TVs
- In programming, it means that you can write code that works with different objects in a similar way, even if those objects belong to different classes or types

Polymorphism in C++

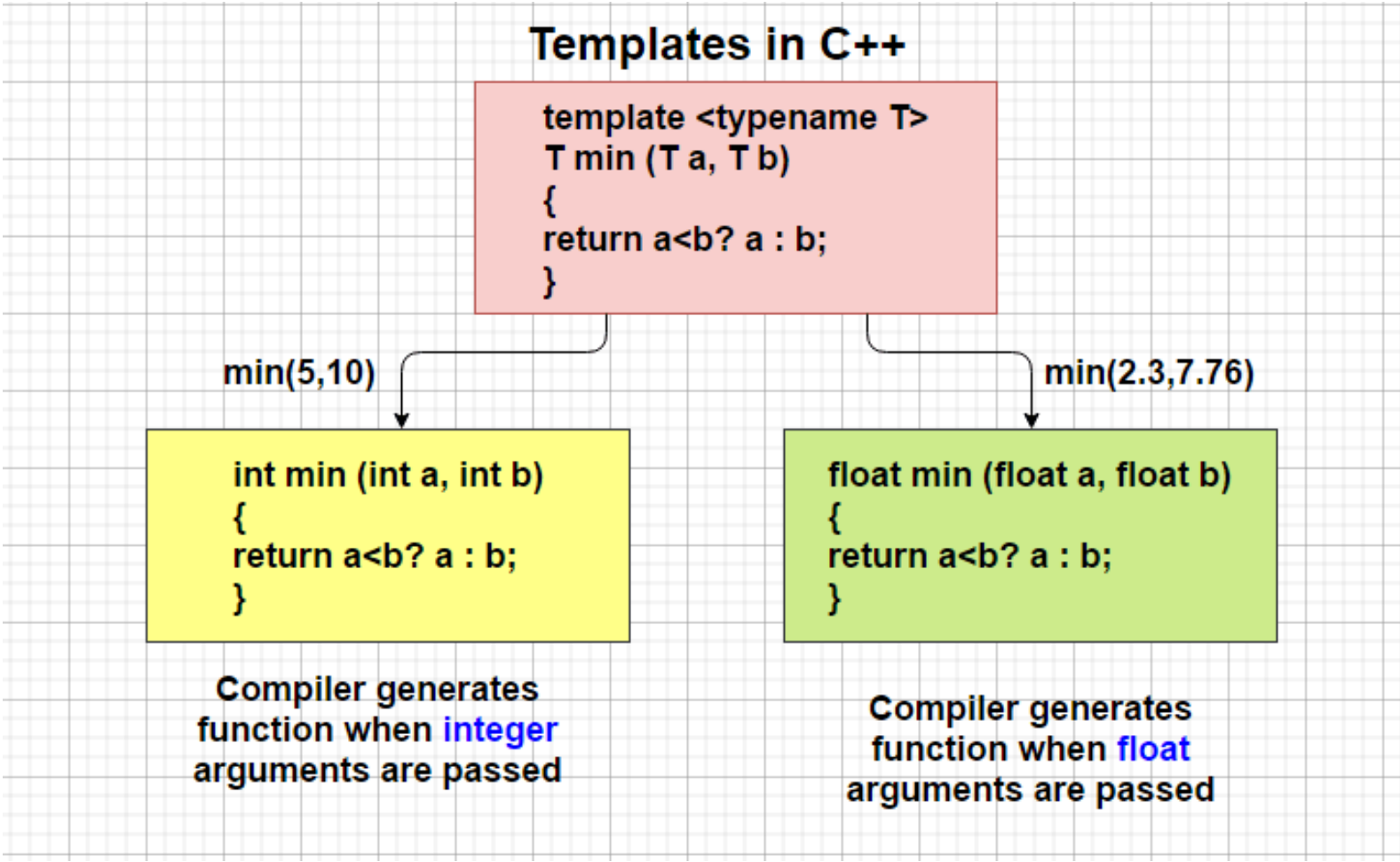
- Overloaded functions

```
// Function to add two double numbers
double add(double a, double b) {
    return a + b;
}
```

```
// Function to concatenate two strings using array notation
const char* add(const char* str1, const char* str2) {
    char result[100]; // Assuming a fixed buffer size for simplicity
    snprintf(result, sizeof(result), "%s%s", str1, str2);
    return result;
}
```

Polymorphism in C++

- Template classes



Polymorphism in C++

- Template functions

```
template <typename T>
T myMax (T x, T y)
{
  return (x > y) ? x: y;
}

int main ()
{
  count << myMax<int> (3, 7) << endl;
  count << myMax<char> ('g' , 'e') << endl;
  return 0;
}
```

Compiler internally generates and adds below code

```
int myMax (int x, int y)
{
  return (x > y) ? x: y;
}
```

Compiler internally generates and adds below code

```
char myMax (char x , char y)
{
  return (x > y) ? x: y;
}
```

Polymorphism in C++

- Template classes

```
template <typename T>
class Measurement
{
public:
  Measurement (T val): value(val){}
  T getValue(){ return value; }
  void setValue(T val){ value = val; }

private:
  T value;
};


int main()
{
  Measurement<int> m1(100);
  Measurement<double> m2(10.967);
}
```

```cpp
template <typename T>
class Stack {
public:
    Stack(int in_capacity) : capacity(in_capacity), size(0) {
        data = new T[capacity];
    }

    ~Stack() {
        delete[] data;
    }

    void Push(const T& value) {
        if (size < capacity) {
            data[size++] = value;
        } else {
            std::cerr << "Stack is full. Cannot push more elements." << std:
        }
    }

    // Additional method to set the size
    void SetSize(int new_size) {
        assert(new_size <= capacity);
        size = new_size;
    }

    int GetSize() const {
        return size;
    }

private:
    T* data;
    int capacity;
    int size;
};
```

Polymorphism in C++: Template classes

```cpp
int main() {
    Stack<double> doubleStack(5);
    Stack<char> charStack(3);

    return 0;
}
```

Polymorphism in C++

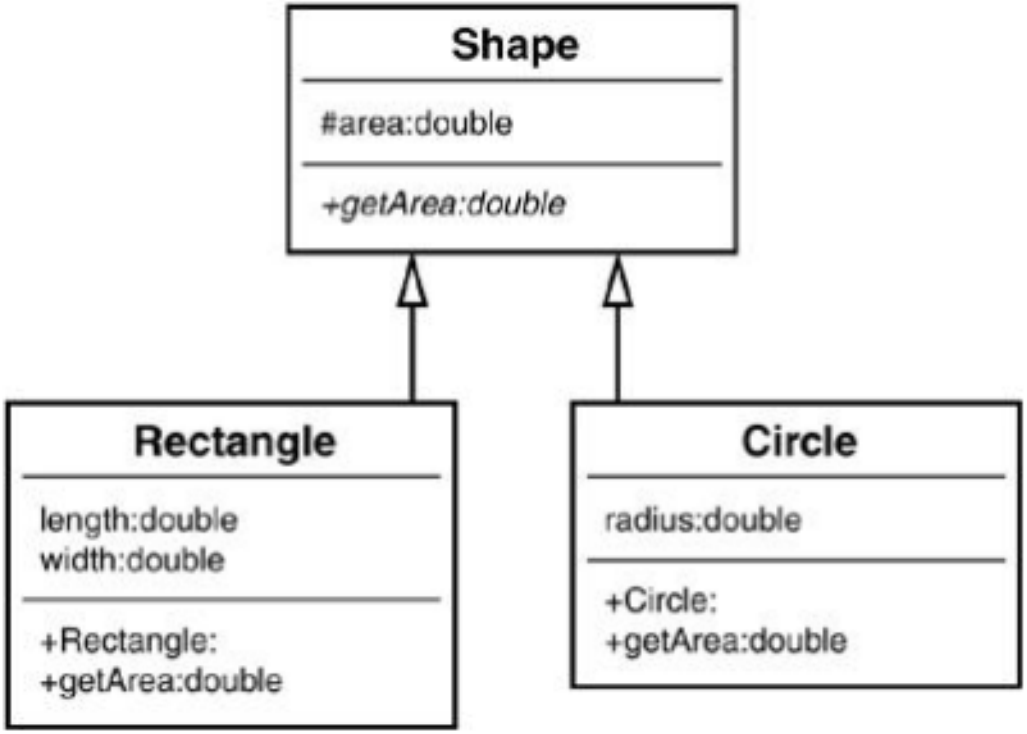
- Pure polymorphism
  - Very powerful
  - Achieved via inheritance and subtyping (derived classes)
  - **Subtyping allows derived classes to be treated as instances of the base class**
  - **Pure polymorphism the ability to treat objects of different derived classes as objects of their base class**
  - **Dynamic binding: during runtime, the correct method is invoked based on the actual type of the object**

Polymorphism in C++

- Supertype in place of a subtype
  - You can substitute a base class object, reference, or pointer in place of a derived class object when calling functions or methods
  - When you do this, the code that operates on the base class reference or pointer works with objects of derived classes as well
  - The flexibility allows you to write code that is more generic and can work with a range of object types
  - It promotes code reusability because you can write functions that operate on base class types and use them with various derived classes

Polymorphism in C++

- The Shape class (again)



```cpp
class Shape {
public:
    virtual void Draw() {
        std::cout << "Drawing a generic shape" << std::endl;
    }
};

class Circle : public Shape {
public:
    void Draw() override {
        std::cout << "Drawing a circle" << std::endl;
    }
};

class Square : public Shape {
public:
    void Draw() override {
        std::cout << "Drawing a square" << std::endl;
    }
};

// Function that takes a Shape object as an argument
void DrawShape(Shape* shape) {
    shape->Draw();
}
```

Polymorphism in C++

- The Shape class (again)

```
int main() {
    Shape* shape1 = new Circle();
    Shape* shape2 = new Square();

    // Pass Shape objects to the DrawShape function
    DrawShape(shape1); // Calls the Draw method for Circle
    DrawShape(shape2); // Calls the Draw method for Square

    delete shape1;
    delete shape2;

    return 0;
}
```

Polymorphism in C++

```cpp
int main() {
    Shape* shape1 = new Circle();
    Shape* shape2 = new Square();

    // Pass Shape objects to the DrawShape function
    DrawShape(shape1); // Calls the Draw method for Circle
    DrawShape(shape2); // Calls the Draw method for Square

    delete shape1;
    delete shape2;

    return 0;
}
```

```cpp
// Function that takes a Shape object as an argument
void DrawShape(Shape* shape) {
    shape->Draw();
}
```