CS-1201 Object Oriented Programming

Const Member Functions and Objects, Friend Function

Arbish Akram

Department of Computer Science
Government College University

Constant Member Functions

- Constant member functions are functions that cannot modify the values of the class's data members.
- To declare a member function as constant, append the keyword `const` to the function prototype and definition header.

Syntax of Constant Member Functions

1. Declaration within a Class:

```
return_type function_name() const;
int get_data() const;
```

2. Definition within Class Declaration:

```
return_type function_name() const {
    // function body
}
int get_data() const {
    // function body
}
```

3. Definition Outside the Class:

```
return_type ClassName::function_name() const {
    // function body
}
int Demo::get_data() const {
    // function body
```

Const Member Functions

- Non-const functions can modify member data.
- Const functions cannot modify member data — compiler will generate an error if attempted.
- Const function can be called on both `const` and non-`const` objects.
- Ideal for functions that only read data (e.g., getters).
- Helps prevent accidental modifications and communicates intent.

Const Objects

- Objects of classes can be declared as const.
- A const object cannot be modified after it's created.
- Only const member functions can be called on const objects.
- Non-const objects can call both const and non-const member functions.

Example 1: Const Objects

```
1   #include<iostream>
2   using namespace std;
3   class Rectangle {
4       private:
5           int width, height;
6       public:
7           Rectangle(int w, int h) : width(w), height(h) {}
8           int getWidth() const {
9               return width;
10          }
11          int getHeight() const {
12              return height;
13          }
14  };
15  int main() {
16      const Rectangle rect(10, 5); // Creating a const object
17      // Accessing const member functions
18      cout<<"width: "<<rect.getWidth()<<endl;  // Valid
19      cout<<"height: "<<rect.getHeight()<<endl; // Valid
20      // rect.width = 20; // Error: cannot modify a const object
21      return 0;
22  }
```

Example 2: Const Objects

```
1  class Distance
2  {
3      private:
4          int feet;
5          float inches;
6      public:
7          Distance(int ft, float in) : feet(ft), inches(in)
8          { }
9      void getdist()
10     {
11         cout << "\nEnter feet: "; cin >> feet;
12         cout << "Enter inches: "; cin >> inches;
13     }
14     void showdist() const
15         { cout << feet << "\'-" << inches << '\"'; }
16 };
17 int main()
18 {
19     const Distance football(300, 0);
20     cout << "football = ";
21     football.showdist();
22     cout << endl;
23     return 0;
24 }
```

What is the `this` Pointer?

- Each object has its own copy of data members.
- All objects share the same function definitions.
- The compiler uses the implicit ‘this‘ pointer to access the correct data members for each object.

Example 1

When local variable names are the same as member variable names.

```
1  class Test {
2      private:
3          int x;
4      public:
5          void setX(int x) {
6              this->x = x; // Using 'this' to refer to the member variable
7          }
8          void print() {
9              cout << "x = " << x << endl;
10         }
11 };
12 int main() {
13     Test obj;
14     int x = 20;
15     obj.setX(x);
16     obj.print(); // Output: x = 20
17     return 0;
18 }
```

Example 2

```
1  class Test {
2      public:
3          Test(int = 0);
4          void print() const;
5      private:
6          int x;
7  };
8  // constructor
9  Test::Test(int value) : x(value) {}
10 void Test::print() const {
11     // implicitly use the this pointer to access the member x
12     cout << "x = " << x;
13     // explicitly use the this pointer and the arrow operator
14     cout << "\n  this->x = " << this->x;
15     // explicitly use the dereferences this pointer and the dot operator
16     cout << "\n(*this).x = " << (*this).x << endl;
17 }
18
19 int main( ) {
20     Test testObject(12);
21     testObject.print();
22     return 0;
23 }
```

Friend Functions and Classes

- A friend is a function or class that is not a member of a class but has access to its private members.
- Private members are hidden from all parts of the program outside the class, requiring public member functions for access. However, a friend function can bypass this restriction.
- A friend function can be a stand-alone function or a member of another class.
- An entire class can be declared as a friend of another class.

Example 1: Friend Function

```
1  // C++ program to demonstrate the working of the friend function
2   class Distance {
3       private:
4           int meter;
5           // friend function
6           friend int addFive(Distance);
7       public:
8           Distance() : meter(0) {}
9   };
10  // friend function definition
11  int addFive(Distance d) {
12      //accessing private members from the friend function
13      d.meter += 5;
14      return d.meter;
15  }
16  int main() {
17      Distance D;
18      cout << "Distance: " << addFive(D);
19      return 0;
20  }
```

Example 2: Friend Function

```
1  // Add members of two different classes using friend functions
2   #include <iostream>
3   using namespace std;
4   // forward declaration
5   class ClassB;
6   class ClassA {
7       public:
8           // constructor to initialize numA to 12
9           ClassA() : numA(12) {}
10      private:
11          int numA;
12           // friend function declaration
13           friend int add(ClassA, ClassB);
14  };
15  class ClassB {
16      public:
17          // constructor to initialize numB to 1
18          ClassB() : numB(1) {}
19      private:
20          int numB;
21          // friend function declaration
22          friend int add(ClassA, ClassB);
23  };
```

Example 2: Friend Function

```
1  // access members of both classes
2  int add(ClassA objectA, ClassB objectB) {
3      return (objectA.numA + objectB.numB);
4  }
5
6  int main() {
7      ClassA objectA;
8      ClassB objectB;
9      cout << "Sum: " << add(objectA, objectB);
10     return 0;
11 }
```

- `ClassA` and `ClassB` have declared `add()` as a friend function, it can access the private data of both classes.
- The friend function inside `ClassA` may utilize `ClassB`, even if `ClassB` is not defined at that point.
- To resolve this, a forward declaration of `ClassB` is required in the program.