Intermediate programming(C++)-

Lab 10: OOP – Part 2

CSCI112
Nile University

Content

- ✓ Passing objects to functions.
- ✓ Returning objects from functions.
- ✓ Static class members
- ✓ Static class methods
- ✓ Operator overloading

Passing objects to function

1. Passing objects by value

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

class MyClass {
public:
    int data;
    MyClass(int val) : data(val) {}
};

void modify_object(MyClass obj) {
    obj.data = 28;
}

void display_object(MyClass obj) {
        cout << "Object data: " << obj.data << endl;
}

int main() {
    MyClass obj(8);
    cout << "Object data before modification: ";
    display_object(obj);

    cout << "Object data after modification: ";
    modify_object(obj);
    display_object(obj);
}
```

Output

```
Object data before modification:
Object data: 8
Object data after modification:
Object data: 8
```

Passing objects to function

2. Passing objects by Reference

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

class MyClass {
public:
    int data;
    MyClass(int val) : data(val) {}
};

void modify_object(MyClass &obj) {
    obj.data = 28;
}

void display_object(MyClass obj) {
        cout << "Object data: " << obj.data << endl;
}

int main() {
    MyClass obj(8);
    cout << "Object data before modification: ";
    display_object(obj);

    cout << "Object data after modification: ";
    modify_object(obj);
    display_object(obj);
}
```

Output

```
Object data before modification: Object
data: 8
Object data after modification: Object
data: 28
```

Passing objects to function

3. Passing objects by pointer

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

class MyClass {
public:
    int data;
    MyClass(int val) : data(val) {}
};

void modify_object(MyClass* obj) {
    obj ->data = 28;
}

void display_object(MyClass obj) {
        cout << "Object data: " << obj.data << endl;
}

int main() {
    MyClass obj(8);
    cout << "Object data before modification: ";
    display_object(obj);

    cout << "Object data after modification: ";
    modify_object(&obj);
    display_object(obj);
}
```

Output

```
Object data before modification: Object
data: 8
Object data after modification: Object
data: 28
```

Returning objects from functions

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

class MyClass {
public:
    int data;
    MyClass(int val) : data(val) {}
};

MyClass return_obj() {
    MyClass obj(120);
    return obj;
}

int main() {
    MyClass obj = return_obj();
    cout << "Returned data: " << obj.data;  // 120
}
```

Output

```
Returned data: 120
```

Returning objects from functions – as a pointer

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

class MyClass {
public:
    int data;
    MyClass(int val) : data(val) {}
};

MyClass* return_obj() {
    return new MyClass(40);
}

int main() {
    MyClass* obj = return_obj();
    cout << "Returned data: " << obj -> data;  // 40
}
```

Output

```
Returned data: 40
```

Static class members

- Shared across all created objects.
- Must be initialized outside the class; before its sharing property.
- Declared using `static` keyword`, and accessed using the class name `{ClassName}::{StaticVariable}`

```
class MyClass {
public:
    static int cnt;

    MyClass() {
        cnt++;
    }
};

int MyClass::cnt = 0;

int main() {
    cout << "Creating object (1)… \n";
    MyClass obj1;
    cout << "Cnt for object (1): " << obj1.cnt << endl;

    cout << "Creating object (2)… \n";
    MyClass obj2;
    cout << "Cnt for object (2): " << obj2.cnt << endl;

    cout << "Creating object (3)… \n";
    MyClass obj3;
    cout << "Cnt for object (3): " << obj3.cnt << endl;
}
```

Output

```
Creating object (1)…
Cnt for object (1): 1
Creating object (2)…
Cnt for object (2): 2
Creating object (3)…
Cnt for object (3): 3
```

Static class methods

- Functions associated with the class, not a specific objects.
- Can only access the static data members or other static functions
- Declared using `static` keyword, and called by the class name `{ClassName}::{StaticFunction}`

```
class MyClass {
public:
    static int cnt;
    MyClass() {
        cnt++;
    }
    static int return_cnt() {
        return cnt;
    }
};

int MyClass::cnt = 0;
int main() {
    cout << "Creating object (1)... \n";
    MyClass obj1;
    cout << "Cnt for object (1): " << obj1.return_cnt() << endl;

    cout << "Creating object (2)... \n";
    MyClass obj2;
    cout << "Cnt for object (2): " << obj2.return_cnt() << endl;

    cout << "Creating object (3)... \n";
    MyClass obj3;
    cout << "Cnt for object (3): " << obj3.return_cnt() << endl;
}
```

Output

```
Creating object (1)...
Cnt for object (1): 1
Creating object (2)...
Cnt for object (2): 2
Creating object (3)...
Cnt for object (3): 3
```

Operator overloading

- Allowing to re-define how operators (e.g., +, -, *, ==, etc.) work for user-defined data types such as objects of classes.
- It allows the object to behave as the primitive data types.

- For example, if we want to sum two objects:

```
MyClass obj1, obj2;
Result = obj1 + obj2;
```

This operation cannot be implemented without + operator overloading; as each object has another data inside it.

Types of operators that can be overloaded:

- Binary operators: +, -, *, /, %
- Relational operators: ==, !=, <, >, <=, >=
- Unary operators:++, --
- Assignment operators: =, +=, -=
- Input/ output operators: <<, >>
- Other advanced operator (e.g., Array subscript [], function call operator (), Dereference operator * (to get values of a pointer)).

Operators that can not be overloaded:

- ::
- sizeof
- ?: → ternary operator

Binary operator(+) overloading example:

```
class MyClass {
public:
    int val;
    MyClass (int v = 0) : val(v) {}

    MyClass operator+ (MyClass& obj) {
        MyClass temp;
        temp.val = this -> val + obj.val;
        return temp;
    }
};

int main() {
    MyClass obj1 (10), obj2 (50);
    MyClass result = obj1+ obj2;
    cout << "Result: " << result.val << endl;
}
```

Output

```
Result: 60
```

Relational operator(<=) overloading example:

```
class MyClass {
public:
    int val;
    MyClass (int v = 0) : val(v) {}

    bool operator <= (MyClass& obj) {
        if (this ->val <= obj.val)
            return true;
        else
            return false;

        // OR
        // return (this -> val <= obj.val)
    }
};

int main() {
    MyClass obj1 (10), obj2 (50);
    if(obj1 <= obj2) {
        cout << "Obejct 2 is less than object 1\n";
    }
    else {
        cout << "Object 1 is less than object 2\n";
    }
}
```

Output

```
Object 2 is less than
object 1
```

Unary operator(++) overloading example:

```cpp
class MyClass {
public:
    int val;
    MyClass (int v = 0) : val(v) {}

    // prefix increment (++obj)
    MyClass operator++ () {
        val++;
        return *this;
    }
};

int main() {
    MyClass obj1 (10), obj2 (50);
    MyClass incremented_obj1 = ++obj1;
    MyClass incremented_obj2 = ++obj2;

    cout << "Obj1++: " << incremented_obj1.val
         << "\nObj2++: " << incremented_obj2.val;
}
```

Output

```
Obj1++: 11
Obj2++: 51
```

Assignment operator(==) overloading example:

```
class MyClass {
public:
    int val;
    MyClass (int v = 0) : val(v) {}

    bool operator== (MyClass& obj) {
        return (this ->val == obj.val);
    }
};

int main() {
    MyClass obj1 (10), obj2 (50);
    if (obj1 == obj2) {
        cout << "Object 1 is equal to object 2\n";
    }
    else {
        cout << "Obj1 and obj2 are different\n";
    }
}
```

Output

```
Obj1 is equal to obj2
```

Task

- Overload (-) operator.
- Overload (<) operator.