

1 Week 8

1.1 Classes

Classes are the most common form of object-oriented programming across programming languages. Classes are used similarly to structs, but they have more features. One of the first differences we encounter is that in a struct, all member variables and functions are public by default. This means that we are able to access member variables or call member functions using the member access operator (“.”) from outside of the struct definition (i.e., call a member function or access a member variable from `main`). In a class, all member variables and functions are private by default.

One of the benefits of having member functions and variables private is that it allows us as coders to have some things accessible to the users and some things hidden. For example, you might create a Graphic User Interface (GUI) which has a colored button. You want the user to be able to control the button’s color, but you may not want to give the user access to other things such as how it responds to being clicked. In PIC 10A, you won’t have to worry about such complex applications, but you will be able to set member variables and member functions in a class or struct as private or public, as the example below shows.

Keep in mind that you can access private variables or call private functions inside the class definition but not outside. This might seem annoying at first. You are limiting what an object of the class is capable of being used for by letting a user only have access to what’s public. As the programs you write become more complex, they would need variables and functions that support the object’s main functionalities. Having certain things hidden (or private) will prevent the user from making mistakes when using something you created.

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

class Point {
private:
    int x;
    int y;

public:
    // Constructors are best made public so you can make
    // new Point objects outside the class.
    Point() : x(0), y(0) {}
    Point(int _x, int _y) : x(_x), y(_y) {}

    // Member function
    // We are using the private member variables: x & y.
    // This is fine because it's inside the class definition.
    void print() const {
        cout << "(" << x << ", " << y << ")\n";
    }
};
```

```

int main() {
    Point pt00;

    Point pt11(1,1);

    pt00.print();
    pt11.print();

    // Trying to access the private member variables
    // outside the class definition gives us an error.
    cout << "(" << pt00.x << ", " << pt00.y << ")\n";

    return 0;
}

```

The code above demonstrates a typical usage of private and public keywords. Member variables `x` and `y` are made private. A user can use the public member function (`print`) to show their status. The constructors should always be made public. Once constructed, then there is no way in our current code to change the member variables. This is what the coder might intend to happen. Suppose in the constructor, the member variables are initialized to make sure the `Point` object we create has positive `x` and `y`. Then we don't want a user to be able to later change the member variable to something negative. Instead, if we want to let the user update the member variables, we could add functions under `public` that do specific tasks that are inline with what we want our `Point` to be able to do. For example, a public function could measure the distance from one `Point` to another `Point` or another public function could translate the `Point`.

1.2 Exercises

1.2.1 Practice coding exercises

1. (a) Create the `CompletedCourse` struct with a constructor that takes a course name (`string`) and a grade (`double`). Include a member function `printer()` to print the course name and grade.
- (b) Test `CompletedCourse` struct by creating an instance, adding a course, and printing the details.
- (c) Create the `Student` struct with member variables for the student's name (`string`), student ID (`int`), a vector of completed courses (`prevCourses`), and a vector of current courses (`currentCourses`). Include a constructor that initializes the name and student ID.
- (d) Implement the following member functions in the `Student` struct:
 - `printer()`: to print the student's name and ID.
 - `addCompletedCourse()`: takes a `CompletedCourse` object and adds it to the vector of completed courses.
 - `addCompletedCourse()`: takes a course name (`string`) and a grade (`int`) and adds it as a completed course.

- `addCurrentCourse()`: takes a course name (`string`) and adds it to the vector of current courses.
 - `displayCourses()`: to print all completed courses using the `printer()` function.
- (e) Demonstrate the usage of the `Student` struct by creating an instance, adding completed and current courses, and printing the student details along with
- (f) Structs and classes are pretty similar. Show this by taking the exercise (or a previous exercise where you created a struct) and turn the struct into a class.
2. (a) Define a class called `Rectangle` with private member variables `length` and `width`. Provide a constructor that initializes these values.
- (b) Create public member functions for the `Rectangle` class:
- `getArea` returns the area of the rectangle.
 - `getPerimeter` returns the perimeter of the rectangle.
 - `print()` prints the details of the rectangle.
 - `compare` takes in a `Rectangle` as an argument and compares it with the one calling the member function. It returns `true` if they have the same dimensions, even if measured differently.
- (c) In the `main()` function:
- Create two `Rectangle` objects, `r1` and `r2`, with dimensions (1,4) and (2,2) respectively.
 - Call the `print()` method on both `r1` and `r2` to display their details.
 - Use the `compare()` method to compare `r1` with `r2` and display the result.
 - Use the `compare()` method to compare `r2` with `r1` and display the result.
3. • Create a `Date` class that stores the day, month, and year as private member variables. Include appropriate getters and setters for these variables. Implement this in `date.hpp` and `date.cpp` files.
- Create a constructor that initializes the date with provided day, month, and year values.
 - Implement functions for displaying the date in different formats: one to display as `day/month/year` and another to display as `Month day, year` (e.g., May 25, 2023).
 - Ensure that the date is valid (e.g., day between 1-31, month between 1-12) when setting these values.
 - Create instances of the `Date` class, set different dates, and display them using the implemented functions.

1.2.2 Predict the output exercises

Write the console outputs (not including the `Program ended with exit code: 0` line). Assume the code runs error-free, includes the appropriate headers, and is using `std` namespace.

```

4. #include <iostream>
   using namespace std;

   struct PrivateMember {
   private:
       int x;

   public:
       PrivateMember(int val) : x(val) {}
       void printValue() { cout << "Value: " << x << endl; }
   };

   int main() {
       PrivateMember pm(7);
       pm.printValue();
       return 0;
   }

```

```

5. #include <iostream>
   using namespace std;

   class PrivateFunction {
   private:
       void printMessage() {
           cout << "This is a private function." << endl;
       }

   public:
       void publicCall() {
           printMessage();
       }
   };

   int main() {
       PrivateFunction pf;
       pf.publicCall();
       return 0;
   }

```

```

6. #include <iostream>
   using namespace std;

   class PrivateVariable {

```

```

    int x;

public:
    PrivateVariable() : x(8) {}
    void printValue() { cout << "Value: " << x << endl; }
};

int main() {
    PrivateVariable pv;
    pv.x = 88;
    pv.printValue();
    return 0;
}

```

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

```

class PublicMember {
public:
    int x;

    PublicMember(int val) : x(val) {}
};

int main() {
    PublicMember pm(7);
    pm.x = 77;
    cout << "Value: " << pm.x << endl;
    return 0;
}

```

8. Consider the structs below.

```

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

struct A {
    A() { cout << "A0 "; }
    A(int i1) { cout << "A1 "; }
    A(int i1, int i2) { cout << "A2 "; }
    A(const A& a) { cout << "A3 "; }
}

```

```

};

struct B {
    A x;
    A y;
    A z;

    B() : x(), y(0), z(0, 0) { cout << "B0 "; }
    B(int i1) : x(), z(0, 0), y(0) { cout << "B1 "; }
    B(int i1, int i2) : y(0), x(), z(0, 0) { cout << "B2 "; }
    B(int i1, int i2, int i3) : y(0), z(0, 0), x() { cout << "B3 "; }
    B(int i1, int i2, int i3, int i4) : z(0, 0), x(), y(0) { cout << "B4 "; }
    B(int i1, int i2, int i3, int i4, int i5) : z(0, 0), y(0), x() { cout << "B5 "; }

    B(double d) {cout << "B6 "; z = A(0, 0); y = A(0); x = A();}
    B(A& a) : x(), y(a), z() {cout << "B7 ";}
    B(A a, int i1) : x(), y(a), z() {cout << "B8 ";}
};

```

What is the output when the following snippets of code are run in `main`?

- (a) `A a0;`
- (b) `A a1(2);`
- (c) `A a2(2, 3);`
- (d) `A a3(a0);`
- (e) `B b0;`
- (f) `B b1(1);`
- (g) `B b2(1, 2);`
- (h) `B b3(1, 2, 3);`
- (i) `B b4(1, 2, 3, 4);`
- (j) `B b5(1, 2, 3, 4, 5);`
- (k) `B b6(0.0);`
- (l) `B b7(a0);`

(m) B b8(a1, 1);

1.2.3 Error spotting exercises

Error spotting will be on the exam. Look at the follow code below and state:

- What errors occurred (build, run time, or logic) or if there are none.
- In the case of a logic error, what does the code currently accomplish?
- How the code can be fixed (use the line numbers to help with the description).

A description of how the code should perform will be put in a block comment. After you make your initial guess, type the code in your IDE and run it. Copying very carefully may help you find some of the errors. Play around with the code to fix it.

```
9.  1  #include <iostream>
    2  #include <string>
    3  using namespace std;
    4
    5  class Dog {
    6      string name;
    7      int age;
    8  };
    9
   10  /* Creates a instance of a Dog.
   11     Assigns it a name and an age. */
   12  int main() {
   13      Dog d;
   14      d.name = "Fido";
   15      d.age = "3";
   16
   17      return 0;
   18  }
```

10. Consider the following class definition error-free.

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

class Character{
public:
    Character() : magic_power("nothing"){}
```

```

    Character(const string& _magic_power) : magic_power(_magic_power){}
    void query(){cout << magic_power << endl;}

private:
    string magic_power;
};

```

Select all the error-free ways to create an instance of character labeled "Mirabel" and print out her magic power: "nothing".

- (a) `Character Mirabel;`
`Mirabel.query();`
- (b) `Character Mirabel();`
`Mirabel.query();`
- (c) `Character Mirabel("nothing");`
`cout << Mirabel.magic_power << endl;`
- (d) `Character Mirabel();`
`cout << Mirabel.magic_power << endl;`
- (e) `Character Mirabel;`
`cout << Mirabel.magic_power << endl;`
- (f) `Character Mirabel("nothing");`
`Mirabel.query();`

11.

```

1  #include <iostream>
2  using namespace std;
3
4  class Circle {
5  private:
6      double radius;
7
8      Circle() : radius(1.0) {}
9      Circle(double _radius) : radius(_radius) {assert(_radius >= 0);}
10
11 public:
12     double getRadius() const {
13         return radius;
14     }

```

```

15
16     void setRadius(double _radius) {
17         assert(_radius >= 0);
18         radius = _radius;
19     }
20
21     double getArea() const {
22         return 3.14159265359 * radius * radius;
23     }
24
25     double getCircumference() const {
26         return 2 * 3.14159265359 * radius;
27     }
28 };
29
30 /* This code is meant to create two circles and
31    print out their radius, area, and circumferences. */
32 */
33 int main() {
34     Circle circle1;
35     Circle circle2(2.5);
36
37     cout << "Circle 1 - Radius: " << circle1.getRadius() <<
38           ", Area: " << circle1.getArea() <<
39           ", Circumference: " << circle1.getCircumference() << endl;
40     cout << "Circle 2 - Radius: " << circle2.getRadius() <<
41           ", Area: " << circle2.getArea() <<
42           ", Circumference: " << circle2.getCircumference() << endl;
43
44     return 0;
45 }

```

```

12. 1  #include <iostream>
2     using namespace std;
3
4
5     /* This code is mean to print the value of the
6        member variable of obj. */
7     class Exercise {
8         int x;
9
10    public:
11        void printValue() { cout << "Value: " << x << endl; }
12    };

```

```
13
14  int main() {
15      Exercise obj;
16      obj.printValue();
17      return 0;
18  }
19
```