8.3 — Public vs private access specifiers

BY ALEX ON SEPTEMBER 4TH, 2007 | LAST MODIFIED BY ALEX ON DECEMBER 17TH, 2015

Public and private members

Consider the following struct:

```
1  struct DateStruct // members are public by default
2  {
3      int month; // public by default, can be accessed by anyone
4      int day; // public by default, can be accessed by anyone
5      int year; // public by default, can be accessed by anyone
6  };
7
8  int main()
9  {
10     DateStruct date;
11     date.month = 10;
12     date.day = 14;
13     date.year= 2020;
14
15     return 0;
16 }
```

In this program, we declare a DateStruct and then we directly access its members in order to initialize them. This works because all members of a struct are public members by default. **Public members** are members of a struct or class that can be accessed from outside of the struct or class. In this case, function main() is outside of the struct, but it can directly access members month, day, and year, because they are public.

On the other hand, consider the following almost-identical class:

```
1  class DateClass // members are private by default
2  {
3      int m_month; // private by default, can only be accessed by other members
4      int m_day; // private by default, can only be accessed by other members
5      int m_year; // private by default, can only be accessed by other members
6  };
7
8  int main()
9  {
10     DateClass date;
11     date.m_month = 10; // error
12     date.m_day = 14; // error
13     date.m_year = 2020; // error
14
15     return 0;
16 }
```

If you were to compile this program, you would receive errors. This is because by default, all members of a class are private. **Private members** are members of a class that can only be accessed by other members of the class. Because main() is not a member of DateClass, it does not have access to date's private members.

Access specifiers

Although class members are private by default, we can make them public by using the public keyword:

```
1  class DateClass
2  {
3  public: // note use of public keyword here, and the colon
4      int m_month; // public, can be accessed by anyone
5      int m_day; // public, can be accessed by anyone
6      int m_year; // public, can be accessed by anyone
```

```
};

int main()
{
    DateClass date;
    date.m_month = 10; // okay because m_month is public
    date.m_day = 14;  // okay because m_day is public
    date.m_year = 2020;  // okay because m_year is public

    return 0;
}
```

Because DateClass's members are now public, they can be accessed directly by main().

The public keyword, along with the following colon, is called an access specifier. **Access specifiers** determine who has access to the members that follow the specifier. Each of the members "acquires" the access level of the previous access specifier (or, if none is provided, the default access specifier).

C++ provides 3 different access specifier keywords: public, private, and protected. Public and private are used to make the members that follow them public members or private members respectively. The third access specifier, protected, works much like private does. We will discuss the difference between the private and protected access specifier when we cover inheritance.

Mixing access specifiers

Classes can (and almost always do) use multiple access specifiers to set the access levels of each of its members.

In general, member variables are usually made private, and member functions are usually made public. We'll take a closer look at why in the next lesson.

Rule: Make member variables private, and member functions public, unless you have a good reason not to.

Let's take a look at an example of a class that uses both private and public access:

```
#include <iostream>

class DateClass // members are private by default
{
    int m_month; // private by default, can only be accessed by other members
    int m_day; // private by default, can only be accessed by other members
    int m_year; // private by default, can only be accessed by other members

public:
    void setDate(int month, int day, int year) // public, can be accessed by anyone
    {
        // setDate() can access the private members of the class because it is a member of the
class itself
        m_month = month;
        m_day = day;
        m_year = year;
    }

    void print() // public, can be accessed by anyone
    {
        std::cout << m_month << "/" << m_day << "/" << m_year;
    }
};

int main()
{
    DateClass date;
    date.setDate(10, 14, 2020); // okay, because setDate() is public
    date.print(); // okay, because print() is public

    return 0;
```

```
}
```

This program prints:

```
10/14/2020
```

Note that although we can't access date's members variables m_month, m_day, and m_year directly from main (because they are private), we are able to access them indirectly through public member functions setDate() and print()!

The group of public members of a class are often referred to as a **public interface**. Because only public members can be accessed from outside of the class, the public interface defines how programs using the class will interface with the class. Note that main() is restricted to setting the date and printing the date. The class protects the member variables from being accessed or edited directly.

Some programmers prefer to list private members first, because the public members typically use the private ones, so it makes sense to define the private ones first. However, a good counterargument is that users of the class don't care about the private members, so the public ones should come first. Either way is fine.

Structs vs classes revisited

Now that we've talked about access specifiers, we can talk about the actual differences between a class and a struct in C++. A class defaults its members to private. A struct defaults its members to public. That's it!

Quiz time

1a) What is a public member?

Show Solution

1b) What is a private member?

Show Solution

1c) What is an access specifier?

Show Solution

1d) How many access specifiers are there, and what are they?

Show Solution

2) Write a simple class named Point3d. The class should contain:
* Three private member variables of type double named m_x, m_y, and m_z;
* A public member function named setValues() that allows you to set values for m_x, m_y, and m_z.
* A public member function named print() that prints the Point in the following format: <m_x, m_y, m_z>

Make sure the following program executes correctly:

```
int main()
{
    Point3d point;
    point.setValues(1.0, 2.0, 3.0);

    point.print();

    return 0;
}
```

This should print:

```
<1, 2, 3>
```

Show Solution

3) Now let's try something a little more complex. Let's write a class that implements a simple stack. Review lesson **7.9 -- The stack and the heap** if you need a refresher on a what a stack is.

The class should be named Stack, and should contain:
* A private fixed array of integers of length 10.
* A private integer to keep track of the length of the stack.
* A public member function named reset() that sets the length to 0 and all of the element values to 0.
* A public member function named push() that pushes a value on the stack. push() should return false if the array is already full, and true otherwise.
* A public member function named pop() that pops a value off the stack. If there are no values on the stack, it should return -1.
* A public member function named print() that prints all the values in the stack.

Make sure the following program executes correctly:

```
int main()
{
    Stack stack;
    stack.reset();

    stack.print();

    stack.push(5);
    stack.push(3);
    stack.push(8);
    stack.print();

    stack.pop();
    stack.print();

    stack.pop();
    stack.pop();

    stack.print();

    return 0;
}
```

This should print:

```
( )
( 5 3 8 )
( 5 3 )
( )
```

Show Solution

 **8.4 -- Access functions and encapsulation**

 **Index**

 **8.2 -- Classes and class members**