
Week 3: Accessor Modifiers

— By, Mahrukh Khan —

Access Modifiers

- private
- protected
- public

```
class Class_Name
{
    public:
        Member_Specification_1
        Member_Specification_2
        .
        .
        .
        Member_Specification_n
    private:
        Member_Specification_n+1
        Member_Specification_n+2
        .
        .
        .
};
```

public access:

```
1  #include <iostream>
2  using namespace std;
3
4  class A {
5  public :
6      int a;
7  protected:
8      int b;
9  private :
10     int c;
11 };
12
13 int main ()
14 {
15     A obj1;
16     obj1.a=1;
17     A obj2;
18     obj2.b=1;
19     return 0;
20     A obj3;
21     obj3.c=1;
22 }
23
24
25
26
```

mpiler (5) Resources Compile Log Debug Find Results Close

Col	File	Message
	C:\Users\Administrator\Desktop\CP-Spring2019\Less...	In function 'int main()':
6	C:\Users\Administrator\Desktop\CP-Spring2019\Lessons...	[Error] 'int A::b' is protected
8	C:\Users\Administrator\Desktop\CP-Spring2019\Lessons...	[Error] within this context
7	C:\Users\Administrator\Desktop\CP-Spring2019\Lessons...	[Error] 'int A::c' is private
8	C:\Users\Administrator\Desktop\CP-Spring2019\Lessons...	[Error] within this context

private access: Set and get functions

```
[*] ex.cpp | Untitled2.cpp | Untitled3.cpp
2  using namespace std;
3  class A {
4      public :
5          int a;
6          protected:
7          int b;
8          private :
9              int c;
10
11      public:
12      int getC()
13      {
14          return c;
15      }
16
17      void setC(int cc)
18      {
19          c=cc;
20      }
21  };
22
23  int main ()
24  {
25      A obj1;
26      obj1.a=1;
27      A obj3;
28      obj3.setC(3);
29      int c= obj3.getC();
30      cout<<"This is "<<c;
31      return 0;
32  }
```

```
C:\Users\Administrator\Desktop\CP-Spring2019\Lessons\Week 1\Untitled3.exe
This is 3
-----
Process exited after 0.1219 seconds with return value 0
Press any key to continue . . .
```

Accessor and Mutator Functions

Member functions that allow you to find out the values of the private member variables of a class are called **accessor functions**. The accessor functions need not literally return the values of each member variable, but they must return something equivalent to those values. Although this is not required by the C++ language, the names of accessor functions normally include the word `get`.

Member functions that allow you to change the values of the private member variables of a class are called **mutator functions**. Although this is not required by the C++ language, the names of mutator functions normally include the word `set`.

It is important to always include accessor and mutator functions with each class definition so that you can change the data stored in an object.

Example

```
class Automobile
{
public:
    void set_price(double new_price);
    void set_profit(double new_profit);
    double get_price();
private:
    double price;
    double profit;
    double get_profit();
};

int main()
{
    Automobile hyundai, jaguar;
}
```

```
hyundai.price = 4999.99;
jaguar.set_price(30000.97);
double a_price, a_profit;
a_price = jaguar.get_price();
a_profit = jaguar.get_profit();
a_profit = hyundai.get_profit();
```

Which of the statements are then allowed in the main part of your program?

protected

A protected member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

```
class Box {  
    protected:  
        double width;  
};  
class SmallBox:Box {  
    public:  
        void setSmallWidth( double wid );  
        double getSmallWidth( void );  
};  
double SmallBox::getSmallWidth(void) {  
    return width ;  
}  
void SmallBox::setSmallWidth( double wid ) {  
    width = wid;  
}
```

```
int main() {  
    SmallBox box;  
    box.setSmallWidth(5.0);  
    cout << "Width of box : " << box.getSmallWidth() << endl;  
    return 0;  
}
```

noile Loa | Debug | Find Re

C:\Users\Administrator\Desktop\CP-Spring2019\Lessons\Week 3\protected.exe

Width of box : 5

Process exited after 0.1043 seconds with return value 0

protected

A protected member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

Specifiers	Within Same Class	In Derived Class	Outside the Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

Inline Functions

- C++ provides inline functions to help reduce function call overhead especially for small functions.
- The compiler places a copy of the code of that function at each point where the function is called at compile time.

```
inline return-type function-name(parameters)
{
    // function code
}
```

Advantages

- Function call overhead doesn't occur.
- It also saves the overhead of push/pop variables on the stack when function is called.
- It also saves overhead of a return call from a function.

Disadvantages

- Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.
- The added variables from the inlined function consumes additional registers, After in-lining function if variables number which are going to use register increases than they may create overhead on register variable resource utilization.
- If you use too many inline functions then the size of the binary executable file will be large, because of the duplication of same code.

Example

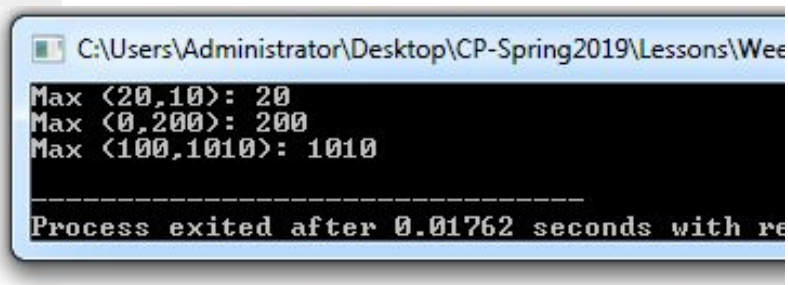
```
#include <iostream>

using namespace std;

inline int Max(int x, int y) {
    return (x > y)? x : y;
}

// Main function for the program
int main() {
    cout << "Max (20,10): " << Max(20,10) << endl;
    cout << "Max (0,200): " << Max(0,200) << endl;
    cout << "Max (100,1010): " << Max(100,1010) << endl;

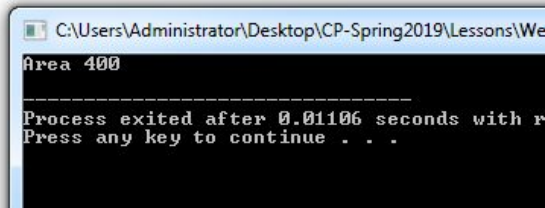
    return 0;
}
```



```
C:\Users\Administrator\Desktop\CP-Spring2019\Lessons\Week 1>
Max (20,10): 20
Max (0,200): 200
Max (100,1010): 1010
-----
Process exited after 0.01762 seconds with return code 0
```

Example

```
1  #include <iostream>
2
3  using namespace std;
4
5  inline int area(int len, int hi) {
6      return len*hi;
7  }
8
9  // Main function for the program
10 int main() {
11     cout << "Area " << area(20,20) << endl;
12     return 0;
13 }
```



```
C:\Users\Administrator\Desktop\CP-Spring2019\Lessons\We
Area 400
-----
Process exited after 0.01106 seconds with r
Press any key to continue . . .
```

```
int main()
{
    cout << len * hi;
    OR
    cout << 10 * 20;
}
```

Inline Function in class

- all the functions defined inside the class are implicitly inline.

```
class S
{
public:
    int square(int s); // declare the function
};

inline int S::square(int s) // use inline prefix
{
}
```

Void Functions

- A function that returns no value.
- In the case of a function that returns a value, the return statement specifies the value returned. In the case of a void function, the return statement simply ends the function call.

```
void initialize_screen( )  
{  
    using namespace std;  
    cout << endl;  
    return;  
}
```

← This return is optional.

Use of return in void function

Function Declaration

```
1 void ice_cream_division(int number, double total_weight);
2 //Outputs instructions for dividing total_weight ounces of
3 //ice cream among number customers.
4 //If number is 0, nothing is done.
```

Function Definition

```
1 //Definition uses iostream:
2 void ice_cream_division(int number, double total_weight)
3 {
4     using namespace std;
5     double portion;
6
7     if (number == 0)
8         return;
9     portion = total_weight/number;
10    cout.setf(ios::fixed);
11    cout.setf(ios::showpoint);
12    cout.precision(2);
13    cout << "Each one receives "
14          << portion << " ounces of ice cream." << endl;
15 }
```

If number is 0, then the function execution ends here.

Abstraction

- Data Abstraction is a process of providing only the essential details to the outside world and hiding the internal details, i.e., representing only the essential details in the program.
- It is process of removing characteristics from something in order to reduce it to a set of essential characteristics.
- Data Abstraction is a programming technique that depends on the separation of the interface and implementation details of the program.
- Through Abstraction, a programmer hides all but the relevant data about a class in order to reduce complexity and increase reusability.
- Examples,
 - You know only how to drive a car but don't know internal details how car built.
 - Using mobile phone we make calls and chat but don't know how it work.
 - We know how to operate ATM machine but don't have any knowledge about its internal working process.

Polymorphism

- In OOP you implement polymorphism through a process called overloading.
- You can implement different methods of an object that have the same name.
- The object can then tell which method to implement depending on the context (in other words, the number and type of arguments passed) of the message.
- Example:

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person have different-different behaviors.