Lecture 4
Introduction to OOP

Khola Naseem
khola.naseem@uet.edu.pk

Constructor Initializer List

- you've set values for member variables in the body of a constructor using explicit assignment. You can use an alternative technique that uses a member initializer list.
- The initialization list is separated from the parameter list by a colon (:), and each initializer is separated from the next by a comma (,).
- Syntax:

```
Box(double height, double breadth) : m_height(height), m_breadth{breadth}
        {
                cout << "Box constructor called." << endl;
        }
```

- The order in which the member variables are initialized is always determined by the order in which they are declared in the class definition

Credit: Khola Naseem

Constructor Initializer List

- Example:

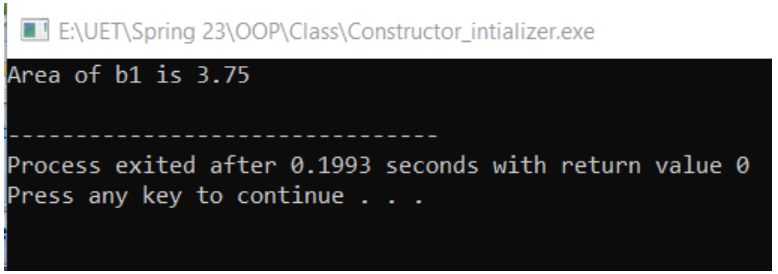
```
class Box{
    private:
        float length;
        float breadth;
    public:
    Box(float a,float b ) : length(a),breadth(b)
    {
        //length=a;
        //breadth=b;
    }

    double Area()
    {
        return length*breadth;
    }
};
int main(int argc, char** argv) {
    Box b1(2.5,1.5);

    cout<<"Area of b1 is "<<b1.Area()<<endl;

}
```

Output:



Constructor Initializer List

- Example: Usage

```
51  #include<iostream>
52  using namespace std;
53
54  class Test {
55      const int t;
56  public:
57    //Initializer list must be used
58      Test(int t1) {
59      t=t1;}
60      int getT() { return t; }
61  };
62
63  int main() {
64      Test t1(10);
65      cout<<t1.getT();
66      return 0;
67  }
```

Compiler (5) | Resources | Compile Log | Debug | Find Results | Close

Line	Col	File	Message
58	5	E:\data_UET\Spring 2024\OOP A\Class\main.cpp	[Error] uninitialized const member in 'const int' [-fpermissive]
55	15	E:\data_UET\Spring 2024\OOP A\Class\main.cpp	[Note] 'const int Test::t' should be initialized
59	3	E:\data_UET\Spring 2024\OOP A\Class\main.cpp	[Error] assignment of read-only member 'Test::t'
28		E:\data_UET\Spring 2024\OOP A\Class\Makefile.win	recipe for target 'main.o' failed

Credit: Khola Naseem

Constructor Initializer List

- Example:

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

class Test {
    const int t;
public:
  //Initializer list must be used
    Test(int t1):t(t1) {
    }
    int getT() { return t; }
};

int main() {
    Test t1(10);
    cout<<t1.getT();
    return 0;
}
```

```
E:\data_UET\Spring 2024\OOP A\
10
--------------------------------
Process exited after 0.219 se
Press any key to continue . .
```

Constructor Initializer List

- Use:

```
#include <iostream>
using namespace std;
class Sample3 {
    int i;          /* Member variable name : i */
    public:
   Sample3 (int i)    /* Local variable name : i */
   {
        i = i;
        cout<<i<<endl;   /* Local variable: Prints the correct value which we passed in constructor */
    }

    int getI()
    {
        cout<<i<<endl;    /*global variable: Garbage value is assigned to i. the expected value should be which we passed in constructor*/
        return i;
    }
};
int main()
{
   Sample3 s3(1);
   cout<<s3.getI();

}
```

Output:

```
1
0
0
```

Constructor Initializer List

- Usage 2:

```
68  #include<iostream>
69  using namespace std;
70
71  class Test {
72      int &t;
73  public:
74      Test(int &t) {
75      t=t;
76      }  //Initializer list must be used
77      int getT() { return t; }
78  };
79  int main() {
80      int x = 20;
81      Test t1(x);
82      cout<<t1.getT()<<endl;
83      x = 30;
84      cout<<t1.getT()<<endl;
85      return 0;
86  }
```

Compiler (4) | Resources | Compile Log | Debug | Find Results | Close

Line	Col	File	Message
		E:\data_UET\Spring 2024\OOP A\Class\main.cpp	**In constructor 'Test::Test(int&)':**
74	5	E:\data_UET\Spring 2024\OOP A\Class\main.cpp	[Error] uninitialized reference member in 'int&' [-fpermissive]
72	10	E:\data_UET\Spring 2024\OOP A\Class\main.cpp	[Note] 'int& Test::t' should be initialized
28		E:\data_UET\Spring 2024\OOP A\Class\Makefile.win	recipe for target 'main.o' failed

Credit: Khola Naseem

Constructor Initializer List

- Usage 2:

```
class Test {
    int &t;
public:
    Test(int &t):t(t) {
//  t=t;
    }  //Initializer list must be used
    int getT() { return t; }
};
int main() {
    int x = 20;
    Test t1(x);
    cout<<t1.getT()<<endl;
    x = 30;
    cout<<t1.getT()<<endl;
    return 0;
}
```

```
20
30

--------------------------------
Process exited after 0.2615 seconds with return value 0
Press any key to continue . . .
```

Why: It's not just a matter of syntax; it's a requirement imposed by the nature of C++ references.

Credit: Khola Naseem

Constructor Initializer List

- Usage:

```
#include <iostream>
using namespace std;
class Sample3 {
    int i;          /* Mem
    public:
   /*Sample3 (int i)     /
    {
        i = i;
        cout<<i<<endl;
    }
    */
    Sample3 (int i):i(i){
    }

    int getI()
    {
        cout<<i<<endl;
        return i;
    }
};
int main()
{
    Sample3 s3(1);
    cout<<s3.getI();

}
```

```
1
1
--------------------
```

Constructor Initializer List

- Usage 3:

```
87  class Sample1
88  {
89      int i;
90      public:
91      Sample1 (int temp)
92      {
93         i = temp;
94      }
95  };
96   // Class Sample2 contains object of Sample1
97   class Sample2
98   {
99    Sample1  a;
100   public:
101   Sample2 (int x)      /* Initializer list must be used */
102   {
103     //a=sample1(x);
104     a(x);
105   }
106  };
107   int main()
108  {
109     Sample2 s3(1);
110  }
```

Compiler (9) | Resources | Compile Log | Debug | Find Results | Close

Line	Col	File	Message
		E:\data_UET\Spring 2024\OOP A\Class\main.cpp	**In constructor 'Sample2::Sample2(int)':**
102	3	E:\data_UET\Spring 2024\OOP A\Class\main.cpp	[Error] no matching function for call to 'Sample1::Sample1()'
102	3	E:\data_UET\Spring 2024\OOP A\Class\main.cpp	[Note] candidates are:
91	6	E:\data_UET\Spring 2024\OOP A\Class\main.cpp	[Note] Sample1::Sample1(int)
91	6	E:\data_UET\Spring 2024\OOP A\Class\main.cpp	[Note] candidate expects 1 argument, 0 provided

Credit: Khola Naseem

Constructor Initializer List

- Usage 3:

```
class Sample1
 {
     int i;
     public:
     Sample1 (int temp)
     {
        i = temp;
     }
 };
  // Class Sample2 contains
 class Sample2
 {
  Sample1  a;
  public:
  Sample2 (int x):a(x)
  {
    //a=sample1(x);
    //a(x);
  }
 };
  int main()
 {
    Sample2 s3(1);
 }
```

```
--------------------------------
Process exited after 0.2233 seconds with return value 0
Press any key to continue . . .
```

Credit: Khola Naseem

Getter and setter functions

- **Getter Function:**
- Inhibiting all external access to the values of private member variables of a class is rather extreme
- Find out the value of the member variable
- But expose the member variables by using the public keyword either.
- You can provide access to the values of private member variables by adding member functions to return their values.
- The values of the member variables are fully accessible, but they can't be changed from outside the class
- Functions that retrieve the values of member variables are referred to as **accessor functions**.

Getter and setter functions

- **Getter Function:**
- Example

```
#include <iostream>
using namespace std;
/* run this program using the console pauser or add your own
class Box{
    private:
        float length;
        float breadth;
    public:
    Box()
    {
        length=2.2;
        breadth=2.4;
    }
    float getlength()
    {
        return length;
    }
    float getbreadth()
    {
        return breadth;
    }
    double Area()
    {
        return length*breadth;
    }
};
int main(int argc, char** argv) {
    Box b1;

    cout<<"Area of b1 is "<<b1.Area()<<endl;
    cout<<"Get value of the length  "<<b1.getlength()<<endl;
    cout<<"Get value of the length  "<<b1.getbreadth()<<endl;
}
```

Output:

```
Area of b1 is 5.28
Get length value  2.2
Get breadth value  2.4
```

Credit: Khola Naseem

Getter and setter functions

- **Setter Function:**
- Modify value of private data member using the member function
- Member functions that allow member variables to be modified are sometimes called mutators.
- allow member variables to be changed from outside the class
- Has opportunity to apply checks on the values.

Getter and setter functions

- **Setter Function:**
- Modify value of private data member using the member function

```
class Box{
    private:
        float length;
        float breadth;
    public:
    Box()
    {
        length=2.2;
        breadth=2.4;
    }
    float getlength()
    {
        return length;
    }
    float getbreadth()
    {
        return breadth;
    }
    void setlength(float a)
    {
        if (a>2.3)
        {
            length=a;
        }
    }
    void setbreadth(float b)
    {
        if (b>0)
        {
            breadth=b;
        }
    }
    double Area()
    {
        return length*breadth;
    }
};
int main(int argc, char** argv) {
    Box b1;
    float new_len=3.0, new_breadth=-9.0;
    cout<<"Area of b1 is "<<b1.Area()<<endl;
    cout<<"Get value of the length  "<<b1.getlength()<<endl;
    cout<<"Get value of the length  "<<b1.getbreadth()<<endl;
    b1.setlength(new_len);
    b1.setbreadth(new_breadth);
    cout<<"return the value of the length after setter function call "<<b1.getlength()<<endl;
    cout<<"return the value of the breadth after setter function call "<<b1.getbreadth()<<endl;
}
```

Output:

```
Area of b1 is 5.28
Get value of the length  2.2
Get value of the length  2.4
return the value of the length after setter function call 3
return the value of the breadth after setter function call 2.4
```

Credit: Khola Naseem