Software engineering

v2024

Lab 2:
Programming with Modern C++ – Part I

Contents

FESB in cooperation with ZORAJA Consulting d.o.o.

Exercise 1: Objects

In this exercise you will create a C++ class and define its members: *fields* and *methods*. From that class you will instantiate *objects* and test their functionality

You will create three types of objects:

- **Global objects**
  These objects are declared at the global scope. They are created before method main is called and destroyed after the main method has finished its execution.

- **Local objects**
  The objects are declared inside a function (method). They are created at the place of the declaration and destroyed before the function exits. You can extend the life of a local object using keyword static. In that case local objects are destroyed as global ones.

- **Dynamic Objects**
  The pointers to those objects are declared in any scope. The actual objects are created using the new operator which allocates the memory on the **heap**. Dynamic objects must be explicitly destroyed using the delete operator.

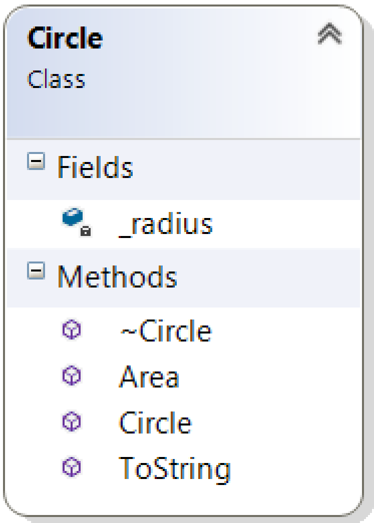
The most common errors in C++ are:

1. The programmer forgets to use the delete operator.
2. The programmer uses a pointer that points to a wrong memory location.

The main differences between other modern OO languages such as Java and C# and modern C++ are:

1. There is no delete operator and the runtime, using garbage collection mechanisms, deletes objects when there is no (reachable) references to them.
2. There are no pointers available for programmers. In C# you can use pointers in unsafe code.

The following UML class diagram shows the class to be created.



Create a Project

1. Create an Empty Project named Objects.
2. Add file Program.cpp to the project.

Add a Header File to the Project

1. Add a file named Circle.h to the project.
2. Protect the file from multiple inclusions.
3. Include the string header file.
4. Add a namespace abc.
5. Inside the abc namespace add a class named Circle.
6. Add a member variable _radius of type double.
7. Add an ordinary constructor with a parameter of type double. Set the default value of the parameter to 1.0.
8. Declare a constant, virtual member function named Area which returns a double.
9. Declare a constant, virtual member function ToString which returns a string.
10. Declare a virtual destructor.
11. Declare the following free functions (outside the Circle class, but inside the abc namespace):
    - Function f, which both takes in and returns an object of type Circle,
    - Function g, which takes in a pointer to an object of type Circle, and has a return type void,
    - Function h, which has no parameters and returns a pointer to an object of type Circle.

Add an Implementation File to the Project

1. Add a file named Circle.cpp to the project.
2. Include the iostream and string header files.
3. Include the previously created Circle.h header file.
4. To avoid using fully qualified names, use the using directive for namespaces std and abc.
5. Add a global constant expression PI of type double and initialize it to 3.1415926535897.
6. Implement the constructor of the Circle class. Initialize the _radius variable and have the constructor output the return value of the ToString function.
7. Implement the Area function so that it calculates the area of the circle.
8. Implement the ToString function so that it returns a string containing the key information about the type.
9. Implement the destructor.
10. Implement the function f to return a Circle object. The object's radius value should be larger than the radius of the argument by 10.
11. Implement the function g so that it increases the radius value of the argument by 100 and outputs the new value.
12. Implement the function h so that it creates a dynamic object of type Circle and returns a pointer to it.
    - What would happen if you were to create a non-dynamic object inside the function and return its address?

Create Objects

1. In the Program.cpp file include the iostream and Circle.h header files.
2. Include the file Helpers.h, which can be found in the course materials section.
3. Use the using directive for namespaces std and abc.
4. Add a global object a of type Circle.
5. In the main function enable the use of UTF8 encoding on the wcout stream by using the SetUTF8 function from the Helpers.h file.
6. Output a string to test the previously mentioned function.
7. Calculate the area of object a and output it to the console window.
8. Declare a variable radius of type double. Inside a try-catch block prompt the user to enter a value and use it to initialize the radius variable.
9. Add a local object b of type Circle and use radius to initialize it.
10. Output the result of b's Area function to the console window.

11. Pass a as an argument to function f and assign the result to a local variable c. Use type inference (keyword auto) when declaring c.
12. Output the area of c to the console.
13. Create a dynamic Circle object with radius 3.0 and assign it to constant pointer p.
14. Call the function g with p as its argument.
15. Use the delete operator on p to delete the object it points to.
16. Call the function h and assign its result to a pointer q.
17. Call function Area on q and output the result to the console.
18. Delete the object q points to.

Run the App

```
Constructing Circle (1.000000)
Zuji , zveči , zvoni , zvuči
Šumi, grmi, tutnji, huči
...

Area: 3.14159
Enter the radius: 3
Constructing Circle (3.000000)
Area: 28.2743
Destroying Circle (3.000000)

Destroying Circle (1.000000)
Area: 380.133

Constructing Circle (3.000000)
Area: 33329.2
Destroying Circle (103.000000)

Constructing Circle (4.000000)
Area: 50.2655
Destroying Circle (4.000000)

Destroying Circle (11.000000)
Destroying Circle (1.000000)

C:\Users\Perica\source\repos\Objects\x64\Debug\Objects.exe (process 29396) exited with code 0 (0x0).
Press any key to close this window . . .
```

Exercise 2: Pointers and References

In this exercise you will implement references as (1) aliases for variables and (2) for passing variables (objects) to functions.

An expression (variable) can be an **l-value** or **r-value**. An l-value has a memory address visible to the programmer and **r-values** are temporary variables that cannot be directly accessed by the programmer.

A reference is an alias for a variable.

Rules for refereeing:

- An **l-value** reference can directly refer to an **l-value**.
- A constant **l-value** reference can refer to an **r-value**, but it cannot modify that **l-value**.
- An **r-value** reference can directly refer to an **r-value** and can also modify it.
- An **r-value** reference cannot directly point to an **l-value** but since that **r-value** is known to the compiler we can kindly ask her to make that value accessible to the programmer via casting to Type&&.
- An r-value reference to an auto type (auto&&) or to a template parameter (T&&) can be deduced to both **l-value** and **r-value** references via deduction rules (Hm!)

Create a Project

1. Create an Empty Project named References.
2. Add file Program.cpp to the project.

Add a File to the Project

1. Add file Something.h to the project.
2. Protect the file from multiple inclusions.
3. Include the iostream header file.
4. Use the using directive for namespace abc.
5. Inside the namespace add a class Something with a member variable value of type int.
6. Implement an ordinary constructor with a parameter of type int and a default value of -1.
7. Implement a virtual destructor which outputs to the console.
8. Implement a free function f which takes in a constant l-value reference to type Something, outputs its value to the console and returns a void.
   - What would happen if you were to increment the reference's value?
9. Implement a free function g which takes in an r-value reference to type Something, increments its value and outputs the new value to the console.

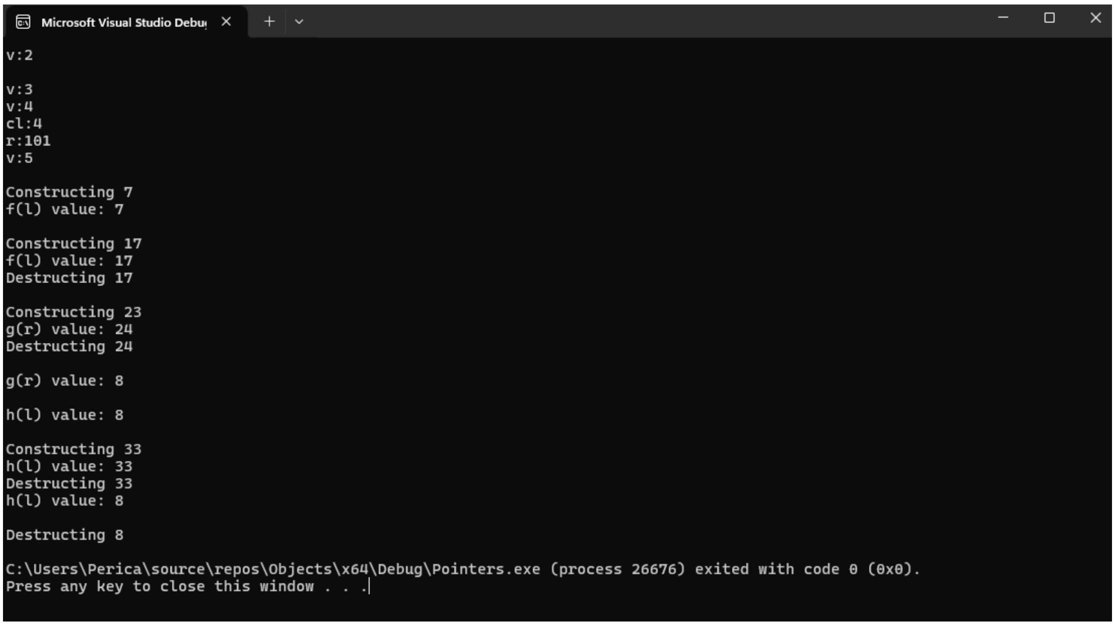
10. Implement a free function h which takes in a constant l-value reference to type Something and outputs its value to the console.
11. Overload function h to take in an r-value reference to Something, increment it and output the new value. Comment the function out for now.

Test Pointers and References

1. Include files iostream.h and Something.h into Program.cpp.
2. Use the using directive for namespaces abc and std.
3. Inside the main function declare variables u and v of type int and initialize them to values 1 and 2, respectively.
4. Output the value of v to the console.
5. Add a pointer to int p have it point to v.
6. Increment v's value by using pointer p.
7. Output the new value of v.
    - What would happen if you were to add another pointer to int and assign to it the address of expression 100?
        - i. And if the pointer was a read-only pointer?
8. Add an l-value reference l to type int and assign v to it.
    - What would happen if you tried to assign something else to l?
    - What would happen if you tried assigning v to a constant l-value reference to int?
9. Increment l and output v's value to the console.
    - Can you assign 100 to an l-value reference to int?
10. Add a constant l-value reference cl to 100.
    - Can you increment cl?
11. Output the value of cl.
12. Add an r-value reference r to int and assign expression 100 to it.
13. Increment r and output its new value to the console.
14. Add an r-value reference to int and initialize it by using static_cast<int&&> on v.
15. Add another r-value reference to int and initialize it with v by explicitly casting it to an r-value using std::move.
16. Increment r and output the value of v to the console.
17. Add a local object a of type Something. Call the function f and pass a as its argument.
18. Call function f and pass it a temporary object of type Something.
19. Call function g and pass it a temporary object of type Something.

20. Call function g once again. Pass a as its argument by casting it to an r-value using std::move.
21. Call function h and pass a as its argument.
22. Call function h and pass it a temporary object of type Something.
23. Call function h once again, but this time use std::move on a and pass that as an argument.
    - What would have happened had you uncommented the overloaded function prior to making calls to h?

Run the App



```
Microsoft Visual Studio Debu
v:2

v:3
v:4
cl:4
r:101
v:5

Constructing 7
f(l) value: 7

Constructing 17
f(l) value: 17
Destructing 17

Constructing 23
g(r) value: 24
Destructing 24

g(r) value: 8

h(l) value: 8

Constructing 33
h(l) value: 33
Destructing 33
h(l) value: 8

Destructing 8

C:\Users\Perica\source\repos\Objects\x64\Debug\Pointers.exe (process 26676) exited with code 0 (0x0).
Press any key to close this window . . .
```