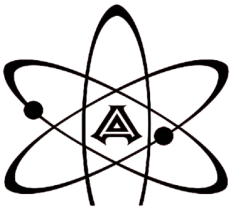

C++ Programming Exercises

By/Abdelwahab Ayman AbuGhaleb

Summary

Try to work through the exercises on each lesson before moving onto the next. You may find the following web resource helpful:

Provides tutorials, a user's forum and documentation on the most widely used libraries including strings, I/O and the STL.

Lesson 1 - Hello World

1. Find out what is the difference between a "signed" and an "unsigned" integer. When might you use an unsigned integer instead of a signed integer?
2. You have seen the "fundamental data type" called `int` but you will eventually need to use others. Amend the program to create a `char` data type, a `bool` data type and a `double` data type. Give them appropriate names and values and print their values to screen using `cout`.
3. Amend the code so that it asks for two numbers from the user. Store these numbers in separate integer variables and then print to screen their sum and product on separate lines. Use `cin` and `cout`—as well as the new line charactor—to accomplish this task.
4. Amend the code from Lesson 1 so that the program asks for your first name and then prints "Hello (your name)!" to the screen. Use the `cin` and `cout` classes to accomplish this. In addition, use the `string` variable type in the `std` namespace to store your name.

Lesson 2 - Functions

1. Not every function we may write has to return a value. Write a function called `product` which accepts two integers as arguments (like `add` and `minus`) but returns no value. In the function body the `product` function should use `cout` to print the product directly to screen.
2. Write a function called `quotient` which should take a double argument and an integer argument. This function should also return a double data type. Now if you provide the `quotient` function with the values 5 and 3 say, it should return the value 1.66667 or there-abouts. Test this by calling the function within a `cout` statement. Once this works change the double argument to an integer and notice the result; why does `quotient` no longer return a double?
3. Write a "calculator program" which asks for two numbers and a mathematical operator (represented as a `char` type. Depending on the operator (+, -, * or /) call the appropriate function and display the result. Use a `switch` statement to filter the choices.

Lesson 3 - Program Flow

1. Write a function that accepts an integer argument and returns a boolean value. The function should return `true` if the integer argument is greater than or equal to zero, and `false` otherwise. Call this function with a positive and negative value to test that it works, using `cout` to display the result.

2. Write another function that continuously asks for numbers (use `cin`), until the user provides a negative number. You'll need some kind of loop and you should make use of the function you created in Exercise 1.

3. Recursion is an alternative to writing loops where a function "calls itself". Write another version of the factorial function that uses Recursion instead of loops. What is one advantage and one disadvantage of using Recursion as an alternative to loops?

Lesson 4 - Pointers

1. Given that a pointer holds the value of a memory address, why is it permitted to add an integer data type value to a pointer variable but not a double data type?

2. Suppose we have a pointer to `float` data type which contains the memory address value 100. If we add the integer value 3 to this pointer, what will be the value of the pointer if `float` data types are 4 bytes in length?

3. Write a function called `swap` that takes two pointers to integer arguments and returns void. Inside the function, swap the values of the integers. Write a main function that calls this function and verify that the value of the integers has indeed been swapped after this function call.

4. Pointer arithmetic can be used to access specific values when a sequence of such values exist. Create an array of 10 integers numbered 1 to 10 with the syntax:
`int arr[10] = {1,2,3,4,5,6,7,8,9,10};`
Now create a function which accepts a integer pointer as its argument and returns void. Implement the function to print out the even numbers of the array using pointer arithmetic.

Lesson 5 - Memory

1. What is "Stack Overflow" and why might a Recursive method be susceptible to this kind of problem?

2. Memory Leaks are a problem associated with which type of memory? What must you do to try and avoid causing Memory Leaks in your programs?

3. Amend the code to create a `char` variable on the stack and a `string` variable on the heap.

4. Examine some of the other options provided when calling `new` and `delete`. Amend the code to create an array of 10 integers on the heap. Ensure that you release the memory you created using the correct form of `delete`.

5. Tree structures are a widely used data construct in programming (including games programming). One such tree structure is a Binary Search Tree which stores data in a manner that allows fast searching. Familiarise yourself with this technique and write your own Binary Search Tree using the code below to help you:

```
1  struct node {
2      int value;
3      struct node* left;
4      struct node* right;
5  };
6
7  struct node* root = NULL;
8
9  // implement the functions described by these headers
10 void insert_integer(struct node** tree, int value);
11 void print_tree(struct node* tree);
12 void terminate_tree(struct node* tree);
13
```

```
14 /**
15 * Main function
16 */
17 int main() {
18    // call your implemented functions here to test
19    // the binary search tree
20    return 0;
21 }
```

memory.cpp

A Tree can be considered as a collection of nodes, linked to one another via pointers. You have been given the definition of a node in the Binary Search Tree. This is in the form of a `struct`; a record for holding together related variables.

You "build" the tree recursively using the `insert_integer` function. You will need to find the correct place to insert a node to hold the new value, then create a node on the heap.

The `print_tree` function should print out the values in the tree in ascending order, and the `terminate_tree` function should reclaim all the memory used in the tree.

Lesson 6 - Scope and Extent

1. When are global variables created and destroyed? How does this differ to variables created in a function?
2. What is the scope of `x` in the following lines of code...

```
1 int y = 2;
2 for(int x = 0; x < y; ++x) {
3    cout << arr[x] << ", ";
4 }
```

scope_extent.cpp

3. Amend the code to create two integer variables in the main function with the same name, use what you've learned about scope and extent to prevent a naming conflict.
4. Alter the program such that the `some_function` function belongs to a newly created namespace called `some_namespace`. Now create another `some_function` function as a duplicate of the original, within another namespace called `some_other_namespace`. Finally, call both functions within the main function using the namespace syntax (i.e. namespace::function).

Lesson 7 - References

1. What is wrong with these lines of code...

```
1 int& empty_reference;
2 int& unnamed_reference = 5;
```

references.cpp

2. what is the value of `x_ref` after these lines of code execute...

```
3 int x = 3, y = 4;
4 int& x_ref = x;
5 x_ref = y;
```

references.cpp

3. Why should you never return a reference type from a function, if the return value is "local variable" of that function?

4. Create another function called `swap` which swaps two characters and returns `void`. Ensure that your function swaps the original variables and does not create copies. Call this function from the main function to make sure it works.

5. Create a global variable in the form of an array of 10 integers, i.e:
   `int nums[10] = {7,3,5,2,1,4,6,9,10,8};`
   and create a new function which sorts the numbers of the array into ascending order; use the `swap_ref` function to help you.

Lesson 8 - Arrays

1. Amend the code to create an array of characters on the stack, which contains your full name (including room for a space between your first and last names). Now create a function which prints to screen your name using the array you just created.

2. Create two arrays on the heap called "first" and "last". Copy your first name into the "first" array and your last name into the "last" array.

3. Create a "battleship" game using a 2-dimension array of booleans to represent the battle-ground. Use appropriate dimensions (say 5 by 5) and denote the presence of your "battleship" by setting the boolean values to `true` in array indices where your battleship is situated. Provide the user with a certain number of guesses to pick coordinates within the 2-D array. Your program should keep track of the number of guesses made and inform the user whether a guess is successful or not.

Lesson 9 - Classes

1. Modify the "Enemy" constructor so that the `score` field is set to an inital supplied value like `hit_points`.

2. Create a new class called "player" which possesses the same fields and methods as enemy. In addtion, give the player a string field called "name", use the `std::string` class. Create "name" on the heap and provide methods to "get" and "set" the name of player.

3. Re-write the Binary Search Tree code from Lesson 5 to create a Binary Search Tree class. Implement a Constructor, Destructor in addition to methods which allow an integer to be inserted and the tree to be printed to screen. Implement a search method which returns `true` if a supplied value is present in the tree, `false` otherwise.

4. Modify the "battleship" game in the previous exercise to use classes. Create a battleship class which possess a "hit-points" and "score" field. When the user lands a shot, the score field should be increased until hit-points is reached. This should denote that your battleship has been sunk. How you represent the battleground is your choice.

Lesson 10 - Inheritance

1. If we did not make the "Enemy" Class Destructor `virtual`, how might a memory leak be introduced into our program when the "ArmedEnemy" class Destructor is called?

2. How does giving a class one or more virtual methods, affect the memory requirements for the C++ compiler compared to a class which possesses no virtual methods?

3. Create a "Boss" class which inherits from the "Armed Enemy" class. Implement an additional armour level field for the "Boss" class and provide suitable getter and setter methods. Create a "Boss" Object in the main function to test the functionality of your "Boss" class.

4. By taking advantage of "inheritance" and "polymorphism" we can group together objects of different classes and perform common operations on them. Make the "Enemy" class abstract. Create an array of 10 "ArmedEnemy" objects and a "Boss" object. With the help of "polymorphism", create a single array of references to the "ArmedEnemy" and "Boss" objects. Iterate

through the array decrementing the `hit_point` value of each reference; use a single function for decrementing which accepts a "Enemy" reference as its argument.

5. Create an abstract "Comparable" class. Any class which inherits from "Comparable" should implement a `compare_to` method which has the following header:

```
/** returns 1 if this class is greater than rhs, 0 if equal
* and (-1) if this class is less than rhs.
*/
int compare_to(const Comparable& rhs);
```

inheritance.cpp

Create a new Binary Search Tree class which stores "Comparable" objects rather than integers. Create some comparable objects and test your new Binary Search Tree class.

Lesson 11 - Templates

1. When does the Compiler usually perform compilation of a Template class?
2. Given how the compilation process is different for Templates, why might you place the Template's class method definitions in the header file with the class declaration?
3. Create a Matrix of characters using the template class in this lesson. Populate the matrix so it contains the characters:
   a, b, c,
   d, e, f,
   g, h, i
   I.E. the character at index [2][2] should contain 'e'.
4. Re-implement The Binary Search Tree class as a template type.
5. Create your own "stack" container as a template. Provide methods to push, pop, get the size and check if the stack is empty.

Lesson 12 - Operator Overloading

1. Overload the stream operator so that "ArmedEnemy" class can be printed to screen using `cout`, like the Matrix class; hint: use a `friend` function.
2. Provide implementations of "addition", "subtraction" and "equals" for the Matrix class.
3. Create a "Complex Number" class with the appropriate methods for "addition", "subtraction" etc and create a Matrix of "Complex Numbers".

Lesson 13 - Vectors (and the STL)

1. How does storage differ between a `vector` container and a `list` container? Given the way Caches reduce costly memory access, which container do you think might sometimes be better in terms of improving "cache hit rate"?
2. Create a vector of "ArmedEnemy" types by implementing the necessary code in "ArmedEnemy" so that it can be used with the vector template class.
3. Explore some of the other container classes in the STL. Create a "Stack" and a "Queue" of "ArmedEnemy" types, practise adding, removing and iterating through the elements as shown with the vector.
4. Create a vector of 10 "ArmedEnemy" types. Give each "ArmedEnemy" class in the vector a different value for its `hit_point` field. Use the appropriate "Algorithms" functionality in the STL to first randomly shuffle the "ArmedEnemy" members of the vector then sort them according to the value in their `hit_point` field.

Lesson 14 - Exception Handling

1. Explore the documentation for your compiler so that you know how to create programs which do not use the exception handling mechanism.

2. Change the name of the file "data.txt" and run the program, notice how the program responds as it now cannot locate the file.

3. Change the name of the file back to "data.txt". Amend line 39 so that the second argument of the `find_average` function is zero. Again, note the response of the divide by zero exception.

4. Create another data file called "data2.txt". In "data2.txt" place 6 random integers values as in "data.txt". Now amend the code so that it reads in both files and adds the values in "data.txt" and "data2.txt" before computing the average.

5. Create your own exception class which is thrown in the event that the `size` of the scores vector is less than 10 after the values have been read and stored in the scores vector. Amend the code to include a test for this exception and test that it works.

Lesson 15 - Threads

1. Create a "hello world" thread which executes a loop, printing to screen the message "Hello from Thread (handle)" 10 times. Extend the Thread class provided.

2. Create a thread-safe Binary Search Tree class which uses a single lock to ensure only one thread at a time can insert comparable objects (this is coarse-grained locking).

3. Create a thread-safe Binary Search Tree class which uses a lock-per-node so that threads can insert comparable objects concurrently (this is fine-grained locking).