COSC1076 | Advanced Programming Techniques

Tutorial/Lab | Week 05

Overview

The week 05 tutorial/lab is to revise content from the last few weeks.

- Functions
- C-Style strings
- Pointers
- References
- Classes
- Allocating and De-allocating memory
- Object Ownership

Tutorial Questions

Look at the following C++ files that were discussed in the Week 04 Lecture.

VideoCharacter.h

```
#ifndef VIDEO_CHARACTER_H
#define VIDEO_CHARACTER_H

#include "Weapon.h"

#include <string>

class VideoCharacter {
public:
    VideoCharacter(std::string& name);
    VideoCharacter(VideoCharacter& other);
    ~VideoCharacter();

    std::string getName();

    void setLeftHand(Weapon* weapon);
    Weapon* getLeftHand();

private:
    std::string name;

    Weapon* leftHand;
};

#endif // VIDEO_CHARACTER_H
```

VideoCharacter.cpp

```cpp
#include "Weapon.h"
#include "VideoCharacter.h"

VideoCharacter::VideoCharacter(std::string& name) :
   name(name)
   leftHand(nullptr)
{}

VideoCharacter::VideoCharacter(VideoCharacter& other) :
    name(other.name),
    leftHand(other.leftHand)
{}

VideoCharacter::~VideoCharacter() {
}

std::string VideoCharacter::getName() {
   return name;
}

void VideoCharacter::setLeftHand(Weapon* weapon) {
   leftHand = weapon;
}

Weapon* VideoCharacter::getLeftHand() {
   return leftHand;
}
```

1. What is the purpose of the `#ifndef/#define/#endif` statements in the header file?
2. Why are objects passed as references in the copy constructor
3. Does the copy constructor conduct a *shallow* copy or *deep* copy?
4. Change the copy constructor to use the opposite of the method identified in the above question.
5. Is the `VideoCharacter` class the *owner* of the object stored in the field `leftHand`? Explain your answer
6. If the `setLeftHand` method *transfers* owners *to* the `VideoCharacter` class, re-write the `VideoCharacter` class to do this.

Lab Questions

It is a good idea to attempt the lab questions before coming to class. The lab might also be longer than you can complete in 2 hours. It is a good to finish the lab at home.

You should demonstrate your work to your tutor.

Exercises

1. Write the below function that is given a C-style string and computes the length of the string

```
int stringLength(char* string);
```

2. Write a C++ Program to test the function you wrote in the above question.
   (a) What steps should be in your program? For example, do you need a string? Where does that sting come from? Should you create memory for this string on the stack or the heap?
   (b) What are important test cases that you should consider? For example, should how long should the string be? What characters should be in the string? What are the edge cases, such as an "empty string"?

3. Write the below function that is given a C-style string in `src` and *copies* the contents of `src` into `dest`. You may *assume* that `dest` has enough memory allocated to fit all of the characters in `src`.

```
void copyString(char* src, char* dest);
```

4. Write a C++ Program to test the function you wrote in the above question. As with question 2, consider the types of things that your C++ program should do. However, you may need extra steps such as:
   (a) Did you set aside enough memory for `dest`?
   (b) Should the memory for `dest` be on the stack or the heap?
   (c) What happens if `src` is has a length of zero?

5. The below function uses the `VideoCharacter` class found in the above tutorial section for this week. It is very similar to the `copyString` function, however it uses an array of pointers to `VideoCharacter` objects. The function is given an *array of pointers* to video characters in `src`, and copies them into `dest`. It must also be told how many objects to copy, via the `length` parameter. This function should make a *deep* copy of the `src` array.

```
typedef VideoCharacter* VideoCharacterPtr;

void copyVCDeep(VideoCharacterPtr* src, VideoCharacterPtr* dest, int length);
```

6. The below C++ program can be used to start testing your above function. You can download this code from Canvas. The same code also provides a Makefile to help you build the code.
   Using this code, write some test code to make sure the *deep* copy has been correctly performed.

testCopy.cpp

```
#include "VideoCharacter.h"

#include <iostream>
#include <string>

#define EXIT_SUCCESS    0
#define LENGTH          2

typedef VideoCharacter* VideoCharacterPtr;
```

```
10
11 void copyVCDeep(VideoCharacterPtr* src, VideoCharacterPtr* dest, int length);
12
13 int main(void) {
14    VideoCharacterPtr* characters = new VideoCharacterPtr[LENGTH];
15
16    std::string name = "Scalan";
17    characters[0] = new VideoCharacter(name);
18
19    name = "Grog";
20    characters[1] = new VideoCharacter(name);
21
22    std::cout << characters[0]->getName() << std::endl;
23    std::cout << characters[1]->getName() << std::endl;
24
25    // COPY!
26    VideoCharacterPtr* duplicate = new VideoCharacterPtr[LENGTH];
27    copyVCDeep(characters, duplicate, LENGTH);
28
29    // TEST COPY!
30    // TODO
31
32    return EXIT_SUCCESS;
33 }
34
35 void copyVCDeep(VideoCharacterPtr* src, VideoCharacterPtr* dest, int length) {
36    // TODO
37 }
```