**Solution for Homework 4**

**ECE 309 Fall 2019**

**Due: September 25, 2019**

Upload an electronic copy of your answers to Moodle under HW4.

*This is a shared google document. This means (1) it may change to clarify content, and (2) other people can view your comments on this file. If you have questions, you are encouraged to comment directly on this document, but* ***do not add your answers here****. Make a copy into your private Google Drive and then edit the document.*

*DO NOT ADD ANSWERS TO THE SHARED DOC! THAT’S CONSIDERED CHEATING!*

# 1. Object-Oriented Programming Concepts

(20 points) Give an example of C++ code for each concept below and briefly (in one comment/sentence) explain why it demonstrates the concept. You do not need to show a full implementation. Use comments to fill in gaps to avoid writing a lot of code.

1. A namespace.

|  |
| --- |
| namespace ece309 {  std::string class\_name = “ECE 309”;  }; |

1. A default class constructor that takes a parameter.

|  |
| --- |
| class A {  int x;  public:  A(int ax=0) { x = ax; }  }; |

1. A private class member.

|  |
| --- |
| class B {  private:  int x; // private member  }; |

1. Is-A relationship between classes.

|  |
| --- |
| class A {  };  class littleA : public A {  };  // littleA is a A |

# 2. Public and Private Access; References

[30 points/3 points each] Consider the following class declaration and variable definitions, then answer the following questions.

#include <stdio.h>

class A {

private:

class B {

public:

B() { x = 1; }

int x;

int& getx() { return x; }

void print() { printf(“%d”,x); }

};

private:

B b;

public:

B c;

void print() { b.print(); c.print(); }

int& getx() { return b.getx(); }

B& getb() { return b; }

B getCopy() { return b; }

};

Assume each code snippet in each row below runs independently in a main function. For each one, say whether its legal code or not. If it is, show the output and explain why it happens. If it’s illegal (e.g. syntax error), explain why. Try to do it without running the code, and then check and correct your answers as needed.

auto: the auto type specifier asks the compiler to figure out the type for us based on the initializer. If you can deduce the type of the initializer, you can assume the variable is the same type.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Code snippet** | **Errors?**  **(Yes or No)** | **Output or explanation of error.** |
| 1 | A a;  a.print(); | No | 11 |
| 2 | A::B b; | Yes | A::B is a private type. We can’t declare such an object. |
| 3 | A a;  a.b.x = -1;  a.print(); | Yes | A.b.x is private. |
| 4 | A a;  a.c.x = 10;  a.c.print(); | No | 110 |
| 5 | A a;  int& x = a.getx();  x = 21;  a.print(); | No | 211 |
| 6 | A a;  int& w = a.c.getx();  w = 19;  a.c.print(); | No | 19 |
| 7 | A a;  A::B c = a.c;  c.print(); | Yes | A::B is private, can’t declare A::B c. |
| 8 | A a;  int& w = a.c.getx();  int& x = a.getx();  w = 19;  x = w;  w = 20;  a.print(); | No | 1920 |
| 9 | A a;  auto &c = a.c;  c.getx() = 5;  c.print(); | No | 5 |
| 10 | A a;  auto b = a.getCopy();  a.getx() = 5;  b.print(); | No | 1 |

# 3. Pointers and References

[10 points] Compare and contrast pointers and references in C++.

1. [3 points] Describe something you can do to a pointer variable that cannot be done with a reference variable.

|  |
| --- |
| You can change the location of a pointer, but it’s impossible the change what a reference aliases.  int array[100];  // ptr is a pointer  int \*ptr = array;  ptr++; // points to new location in array;  // x is a reference  int &x = array[0]; // x is always first location in array, and can never change |

1. [7 points] Describe a circumstance in which it makes more sense to use a pointer than a reference and another one in which it makes more sense to use a reference. Use code snippets to make your case. (Hint: Your answer to part (a) may be a guide here.)

|  |  |
| --- | --- |
| **Reference is preferred** | **Pointer is preferred** |
| To “pass by reference” to a function, using a reference parameter is preferred over a pointer.  void foo(string &s) {  // pass s without copying it  // s can never be null.  }  void foo\_other(string \*s) {  // pass &s to avoid making a copy  // don’t forget to check for null!  }  Both mean about the same thing, but most programmers prefer to avoid pointers when possible, and leads to fewer mistakes.  Also, a pointer can be nullptr, but a reference cannot be. This simplifies our reasoning since we know the reference parameter can never be nullptr, but the pointer could be. | To traverse an array, a pointer is necessary.  int array[100];  // ptr is a pointer  int \*ptr = array;  while(ptr < &array[100])  ptr++; |

# 4. ZyLabs Problems

* [10 points] ZyLab 11.26. Implement a find function for the List class.
* [10 points] ZyLab 11.27. Impement a function to reverse a list.
* [20 points] ZyLab 11.28. Implement an iterator capable of reversal for a singly-linked list.
  + 5 points: your implementation obeys all the requirements specified in the ZyLab.
  + 15 points: as graded in the ZyLab.