CS 32 Fall ’21 Midterm 1

Name:

Perm #:

Instructions

- Please read all the instructions very carefully.
- This exam is graded out of 55 points.
- Make sure that the handwriting in your solutions is legible.
- You can write *“I don’t know but I promise I will learn”* and get 10% of the points. If the question has multiple parts, then you have to do this for the whole question.
- One double-sided cheat sheet is allowed, you need to turn it in with the exam itself.
- Course materials or computing devices are **not** allowed during the exam.
- Write your name and PERM # on top of each page.

1. (10 points) Mark whether each statement is True or False. If **false**, *briefly* state why.

   1. (T | F) Anonymous namespaces allow avoiding linker errors by telling a compiler to hide the definitions from other source files.

   2. (T | F) All members of a `class` are publicly-accessible by default.

   3. (T | F) `make` uses timestamps of the dependencies and the target to avoid re-building targets that are “up-to-date”.

   4. (T | F) If algorithm A has better (lower) worst-case complexity than algorithm B, then A also has a lower best-case complexity.

   5. (T | F) Inserting an element into `std::unordered_map` (hash table) takes O(n) time in the worst case.

2. Total for Question 2: 8

Given the following class definition,

```
class Foo {
public:
  int a;
  char b;
  short c;
  double d;
  char e;
  int f;
};
```

(a) (4 points) Draw out the memory structure g++/clang++ would allocate for an instance of `Foo`. You may assume that the word-size of the processor is 64-bits (8 bytes); so the memory structure should be same as what you would get on CSIL.

(b) (2 points) What is the size (in bytes) of an instance of `Foo`?

(c) (2 points) What can the programmer do to make instances of `Foo` smaller while having the same fields?

3. Total for Question 3: 10

(a) (6 points) Given the following namespace definitions and main function, write the output in the blank space to the right of each function call. If the line of code would yield a compiler error, simply write ERROR.

```
using namespace std;
namespace Foo {
namespace {
void f() { cout << "Foo::f\n"; }
}
void g() { cout << "Foo::g\n"; }
}
namespace Bar {
void f() { cout << "Bar::f\n"; }
void g() { cout << "Bar::g\n"; }
}
void g() { cout << "::g\n"; }
using namespace Foo;
using namespace Bar;
int main(int argc, char* argv[]) {
  f();      //  __________________
  g();      //  __________________
  Foo::f(); //  __________________
  Bar::f(); //  __________________
  Foo::g(); //  __________________
  ::g();    //  __________________
  return 0;
}
```

(b) (4 points) Suppose you are creating a linear algebra solver with three modules: I/O (input/output, `io.cpp`), and the solver algorithm itself (`solver.cpp`), and the main function (`main.cpp`). Fill in the blanks of the Makefile that describes how to build the targets `main` and `clean`, write the dependencies and the one line command to build the target. The `main` target should use object files, and make sure to use C++17. You can use g++, clang++, or $(CXX) as your compiler.

```
io.o: io.cpp solver.h
    # ...
solver.o: solver.cpp solver.h
    # ...
main.o: main.cpp solver.h
    # ...
# Fill the rest of the Makefile. The main program:
main:


# phony target to clean up all the generated files
clean:
```

4. (9 points) Implement the big three (copy constructor, assignment operator, and destructor) with the deep-copy idiom for the `Wrapper` class defined below. You can assume that `Widget` also implements the big three.

```
// Wrapper.h
#include "Widget.h"

class Wrapper {
  Widget * widget;
 public:
  int n;
  Wrapper() : widget(new Widget), n(0) {}
  const Widget & getWidget() const { return *widget; }
  // You will implement the member functions below
  Wrapper(const Wrapper&);
  Wrapper& operator = (const Wrapper&);
  ~Wrapper();
};
// ----------------------------------------------------
// Wrapper.cpp
#include "Wrapper.h"
// Your implementation goes below. You need to give the full function definitions, including
// both the body and the signature. You may assume that all necessary libraries are included.
```

5. Total for Question 5: 12

(a) (6 points) Sort the following array using quicksort, and write the intermediate states after each call to partition for a sub-array. Use the first element as the pivot, and underline it after placing it in the correct position. Assume that the recursive call for the left sub-array happens first. The first step is done for you:

4	-2	1	7	3	0	5	2	6
0	-2	1	2	3	$\underline{4}$	5	7	6

(b) (6 points) Complete the implementation of insertion sort below by filling in the blanks with valid expressions.

```
template<typename T>
void insertionSort(vector<T> & v) {
  size_t shiftIndex;
  for (size_t i = 1; i < v.size(); i++) {
    T itemToInsert = ____________;
    shiftIndex = i - 1;
    while (shiftIndex >= 0 && _______________________) {
      _________________ = ___________________;
      shiftIndex -= 1;
    }
    v[________________] = itemToInsert;
  }
}
```

6. Total for Question 6: 6

(a) (2 points) Explain one problem with open addressing (closed hashing) that chained hashing solves. How does chained hashing solve that problem?

(b) (4 points) Consider the following hash function:

x	$h(x)$
`bar`	3
`baz`	5
`corge`	10
`foo`	1
`quux`	7

Suppose we have an empty hash table with capacity 6. Insert the values in the following order, resolve collisions with linear probing, and show the state of the hash table after insertion. The first case is done for you.

Value to insert	0	1	2	3	4	5
`foo`		`foo`				
`bar`		`foo`				
`baz`		`foo`				
`quux`		`foo`				
`corge`		`foo`				

Scratch Paper

Scratch Paper