CMPSC 32 W19
Object Oriented Design and Implementation

Midterm 2 Examination

Please state your answers as clearly as possible. This exam not only tests your understanding of the material, but also how well you can convey your understanding to us. Remember that you are solely responsible for the answers to the questions, therefore, please refrain from consulting with your class peers.

Please write all your answers **LEGIBLY** and **CLEARLY**. If we cannot decipher your answers, you will not receive credit.

No electronic devices are allowed during the exam (calculators, cell phones, laptops, etc.).

READ all questions carefully before attempting to answer. If there are any ambiguities in the statement of questions, please ask us. **You may assume that each problem is correct and solvable unless the question specifically asks about errors.**

THE GRADE IN THIS EXAM IS A TOTAL OF 51 POINTS.

Name (as it would appear on the official course roster)	Umail Address
	@umail.ucsb.edu

Question 1 (6 points)
Write whether each statement is True or False. If False, briefly state why (1 point each)

a. An abstract class cannot be instantiated on the stack or the heap.

b. Non-virtual destructors will result in a compilation error.

c. Polymorphism only applies to objects created on the heap.

d. A subclass' constructor must call a base class' default constructor in its initialization list.

e. Mergesort and Quicksort have the same best-case running time.

f. Constructors can be virtual.

Question 2 (5 points)

Show the state of the **entire** array for each iteration of insertionsort's algorithm as discussed in lecture in the table below. The sorted elements should be in ascending order (from least to greatest).

Values / Iterations	20	15	3	8	11
i = 1					
i = 2					
i = 3					
i = 4					

Question 3 (11 points)

a. Write a function `arrayToMap` that takes an array of strings and returns a `std::map<int, string>` such that the values in the map are the string values in the array of strings, and the keys in the map are the corresponding array indices of the string values. You may assume all necessary libraries have been included in your program and your solution must be syntactically correct in order to receive full credit.

```
map<int, string> arrayToMap(string arr[], int arrSize) {




}
```

b. Write a function `printMap` that takes a `std::map<int, string>` and prints each key / value pair in the form of `key:value` (one pair per line). Your solution **must use** iterators to traverse the map and access the values. You may assume all necessary libraries have been included in your program and your solution must be syntactically correct in order to receive full credit.

```
void printMap(map<int, string> x) {




}
```

Question 4 (10 points)

Use the following class definitions and various definitions for `main()` to answer parts **a – d**. You may assume that each part executes independently of the other parts. If the segment of code results in a compilation error, briefly state why. If the segment of code has a runtime error, write the output up to the point of the error and briefly state why the runtime error occurred.

```
class A {
public:
      A() {}
      virtual ~A() { cout << "~A" << endl; }
      void f1() { cout << "A.f1" << endl; }
      virtual void f2() { cout << "A.f2" << endl; }
};

class B : public A {
public:
      B() {}
      virtual ~B() { cout << "~B" << endl; }
      virtual void f2() { cout << "B.f2" << endl; }
      virtual void f3() = 0;
};

class C : public B {
public:
      C() {}
      ~C() { cout << "~C" << endl; }
      virtual void f1() { cout << "C.f1" << endl; }
      void f3() { cout << "C.f3" << endl; }
};

void f1(A& a) { a.f1(); a.f2(); }

void f2(A a) { a.f1(); a.f2(); }
```

a.

```
int main () {
    A* a = new B();
    f1(*a);
    f2(*a);
    return 0;
}
```

b.

```
int main () {
    A* a1 = new C();
    f1(*a1);
    f2(*a1);
    return 0;
}
```

c.

```
int main() {
    int a[3] = {1,2,3};
    try {
        for (int i = 0; i < 3; i++) {
            cout << a[i] << endl;
            if (a[i] % 2 == 0)
                throw A();
        }
    } catch (C c) {
        c.f3();
    }
    cout << "after catch" << endl;
}
```

d.

```
int main() {
    A* a = new C();
    a->f1();
    a->f2();
    delete a;
}
```

Question 5 (11 points)

a. Complete the table below by writing the O-notation for the average and worse-case scenarios for lookup and deletion operations in a `std::map` and a `std::unordered_map.`

	unordered_map		**map**	
	Average	*Worst-Case*	*Average*	*Worst-Case*
Lookup				
Deletion				

b. **Briefly** describe open-address hashing and explain the two main problems with this implementation as discussed in lecture.

c. **Briefly** describe the process of how exceptions are passed along functions in the stack. State what happens when an exception is not caught.

Question 6 (8 points)
For the mergesort implementation covered in class, complete the algorithm by filling in the blanks with the proper expression or values.

```
void merge(int a[], size_t leftArraySize, size_t rightArraySize) {
      int* tempArray;         // tempArray to hold sorted elements
      size_t copied = 0;      // num elements copied to tempArray
      size_t leftCopied = 0;  // num elements copied from leftArray
      size_t rightCopied = 0; // num elements copied from rightArray


      tempArray = new int[_____________________________________________];

      while ((leftCopied < leftArraySize) && (rightCopied < rightArraySize)) {

            if (a[_____________________] < ___________________________________) {

                  tempArray[copied++] = a[leftCopied++];
            } else {
                  tempArray[copied++] = (a + leftArraySize)[rightCopied++];
            }
      }

      while (_____________________________ < _____________________________) {
            tempArray[copied++] = a[leftCopied++];
      }

      while (_____________________________ < _____________________________) {
            tempArray[copied++] = (a + leftArraySize)[rightCopied++];
      }

      for (int i = 0; i < leftArraySize + rightArraySize; i++) {

            a[_____________________] = tempArray[_____________________________];

      }
      delete [] tempArray;
}

void mergesort(int a[], size_t size) {
      size_t leftArraySize;
      size_t rightArraySize;

      if (_____________________________ > 1) {

            leftArraySize = ____________________________________;

            rightArraySize = ____________________________________;

            mergesort(a, _____________________________________);

            mergesort((a + leftArraySize), ____________________________________);

            merge(a, _____________________, _____________________);
      }
}
```

Scratch Page