CMPSC 32 F18
Object Oriented Design and Implementation

Final 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 52 POINTS.

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

Question 1 (7 points) Write whether each statement is True or False. If False, briefly state why (1 point each)

a. Multiple threads in a single application can execute concurrently on a single-core processor.

b. The `std::set` container cannot contain duplicate values.

c. Class constructors can be virtual.

d. A derived class must override an inherited virtual function.

e. C++ directly communicates with a computer system's hardware.

f. A thread shares the same memory space as the process in which it was created.

g. A process shares the same memory space as the parent process that created it.

Question 2 (10 points)

a. Briefly explain / describe a **deadlock** in a multi-threaded application.

b. Briefly explain the situation when a class' copy constructor is called vs. when a class' assignment operator is called.

c. Briefly explain what an **atomic transaction** is. What mechanism discussed in lecture can be used to ensure the OS performs an atomic transaction within a multi-threaded application.

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

Values / Iterations	15	10	2	5	9
i = 1					
i = 2					
i = 3					
i = 4					

Question 3 (8 points)

Write the output for the following application. **For each line of output, write a comment next to each line stating which process or thread outputs the line (read below for thread notation)**. You may assume that:

- Processes are defined in the order the OS created them. For example, **P1** is the process the OS created when executing the application, **P2** is the second process the OS created, etc. When forking a process, you may assume the parent process has priority and executes a statement before a child process.

- Each process has a main thread, and any thread may create more threads. Threads are defined in the order the OS created them and are associated with a specific process. For example, if P1's main thread created a child / subthread, then this is denoted as **[P1, T1]**. If P1 (or a thread in P1) creates another thread, then this thread is denoted as **[P1, T2]**. Any output that the main thread produces is simply denoted with just the process (for example: **P1**). Use this notation in your comments when stating which thread outputs a line to the console.

- The OS executes one instruction from each active main thread or subthread in the order that the processes were created. For example, if there are three active processes and two active subthreads in P1, then the OS executes one instruction from P1, then one instruction from [P1, T1], then one instruction from [P1, T2], then one instruction from P2, then one instruction from P3, and then one instruction from P1, etc.

- The time for the OS to create and start executing a thread is less than 1ms.

- any instruction (except sleep) is executed in less than 1 ms.

```
#include <iostream>
#include <unistd.h>
#include <thread>

using namespace std;

void f(bool value) {
	sleep(1);
	cout << "f()" << endl;
	if (value) {
		thread b(f, false);
		b.detach();
		cout << "b was detached" << endl;
	}
}

int main() {
	pid_t x = fork();
	cout << "after fork1" << endl;
	if (x == 0) {
		cout << "x == 0" << endl;
		thread a(f, true);
		a.join();
	}
	sleep(5);
	cout << "end" << endl;
}
```

Question 4 (12 points)

Given the definition of a MinHeap class containing `int` values:

```
class MinHeap {
private:
       int* heapArray;
       int size;
       const static int CAPACITY = 100;
       void heapify(int index);
public:
       MinHeap();
       int removeMin();
       void insert(int e);
       int getSize() { return size; }
       void printHeap();
};
```

a. Write the definition of the `insert` method for the MinHeap class. You may assume all necessary libraries are included and the underlying `heapArray` is not full before executing the insert method.

```
void MinHeap::insert(int e) {



}
```

b. Given the following array containing the current state of a minHeap, draw the resulting minHeap tree.

```
heapArray: [4,7,10,15,20,11,12,16]
```

c. After **each** of the following three minHeap method calls below, write the resulting array that satisfies the MinHeap property in the box to the right of the method call.

`removeMin()`

`removeMin()`

`insert(5)`

Question 5 (9 points)

Use the following class definitions and various definitions for function `f()` 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; }
};

int main() {
	f();
}
```

a.

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

b.

```
void f() {
	A* a = new B();
	a->f1();
	a->f2();
	delete a;
}
```

c.

```
void f() {
	A* a = new A();
	a->f1();
	a->f2();
	delete a;
}
```

d.

```
void f() {
	int a[3] = {5,6,7};
	try {
		for (int i = 0; i < 3; i++) {
			cout << a[i] << endl;
			if (a[i] % 2 == 0)
				throw A();
		}
	} catch (C b) {
		b.f3();
	}
	cout << "after catch" << endl;
}
```

Question 6 (6 points)

Assume you have a chained HashTable implementation that stores an array of vectors similar to Lab04. Each vector element contains a `std::pair` where the first int represents a unique key in the hash table and the second int represents some value. The hash table structure is defined as:

```
vector<pair<int, int> > hashTable[100];
```

Also assume that the simple hash function is defined as:

```
int hashFunction(int key) {
       return key % 100;
}
```

Write a function `keyExists` that returns true if the key already exists in the HashTable structure and returns false otherwise.

```
bool keyExists(vector<pair<int, int> > table[], int key) {
```