CMPSC 32 S18
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 94 POINTS.

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

Question 1 (8 points)

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

a. The *initialized data section* in a .o file contains the sizes of the other sections in the .o file.

b. Any type in C++ can be thrown and caught using the try / catch mechanism.

c. The size of an empty class with no members is 1 byte.

d. You must write a template class' methods in a .h file where the class is defined.

e. The `ps` Unix command will only show active processes running on the current terminal application.

f. Random Access Memory (RAM) is an example of volatile storage in a computer.

g. Mergesort is more efficient than Insertionsort in all cases.

h. It is possible to make constructors virtual.

Question 2 (9 points)
For the following code, write the resulting output:

```
#include <iostream>
using namespace std;

class A {};
class B : public A {};
class C : public A {};
class D : public B {};

void X(int value) throw (A,B,C,D) {
      if (value == 1) { throw B(); }
      else if (value == 2) { throw C(); }
      else if (value == 3) { throw D(); }
      else if (value == 4) { throw A(); }
}
void Y(int value) {
      try { X(value); }
      catch (D e) { cout << "Caught Y:D" << endl; }
      catch (B e) { cout << "Caught Y:B" << endl; }
}
void Z(int value) {
      try { Y(value); }
      catch (D e) { cout << "Caught Z:D" << endl; }
      catch (C e) { cout << "Caught Z:C" << endl; }
      catch (A e) { cout << "Caught Z:A" << endl; }
}

int main() {
      int array[] = {4,3,2,1};
      for (size_t i = 0; i < 4; i++) {
            cout << array[i] << endl;
            Z(array[i]);
      }
      cout << "end of main" << endl;
}
```

Question 3 (18 points)

a. Describe how UNIX executes the command "pwd" on the terminal application **with respect to processes and memory** using `fork` and `exec`.

b. Briefly define the differences and relationship between a *process* and a *thread*.

c. Briefly define the two steps in the C++ linking process.

d. Briefly describe the relationship between a device driver and the Operating System.

Question 4 (10 points)

For the MaxHeap implementation covered in class, the `removeMax()` method removes and returns the root element of the heap (organized as an array called `heapArray` containing the number of elements represented with the class member `size`) while reorganizing the heap in order to maintain the heap property. Given the `removeMax()` method definition, complete the `heapify()` method in the space below.

```
int MaxHeap::removeMax() throw (HeapEmptyException) {
      if (size <= 0) throw HeapEmptyException();
      if (size == 1) {
            size--;
            return heapArray[1];
      }
      int index = 1;
      int max = heapArray[index];
      heapArray[index] = heapArray[size];
      size--;
      heapify(index);
      return max;
}

void MaxHeap::heapify(int index) {
// Complete this method.









}
```

Question 5 (9 points)
For the quicksort implementation covered in class, complete the algorithm by filling in the blanks with the proper expression or values.

```
void partition(int a[], size_t size, size_t& pivotIndex) {
      int pivot = a[0];        // choose 1st value for pivot
      size_t left = 1;         // index just right of pivot
      size_t right = size - 1; // last item in array
      int temp;

      while (left <= right) {

            while (_____________________________________________________) {

                  left++;
            }

            while (_____________________________________________________) {

                  right--;
            }
            if (left < right) {
                  temp = a[left];
                  a[left] = a[right];
                  a[right] = temp;
            }
      }

      pivotIndex = _____________________________;

      temp = a[0];
      a[0] = a[pivotIndex];
      a[pivotIndex] = temp;
}

void quicksort(int a[], size_t size) {
      size_t pivotIndex;      // index of pivot
      size_t leftSize;        // num elements left of pivot
      size_t rightSize;       // num elements right of pivot

      if (size > 1) {

            partition(_______________,_______________,_______________);

            leftSize = ____________________________________________;

            rightSize = ____________________________________________;

            quicksort(_____________________________,_____________________________);


            quicksort(_____________________________,_____________________________);
      }
}
```

Question 6 (10 points)

A partial class definition for a `HashTable` implemented with chained hashing is given below. This HashTable implementation creates an array of vectors containing Entry elements. Assuming the maintenance of `HashTable` is implemented correctly, complete the methods for the "Big Three" (destructor, copy constructor, and assignment operator) using methods in the `HashTable` class below. Note you must handle self-assignment correctly.

```
class Entry {
public:
       int key; string value;
};
class HashTable {
public:
       HashTable();
       ~HashTable();
       HashTable(const HashTable &orig);
       HashTable & operator=(const HashTable &right);
       vector<Entry> getEntry(int index) const { return table[index]; }
       int getSize() const { return size; }
       static const int CAPACITY = 100;
       // ... other methods...
private:
       int size;
       vector<Entry>* table;
};
HashTable::HashTable() {
       size = 0;
       table = new vector<Entry>[CAPACITY];
}
HashTable::~HashTable() {   // complete this method



}
HashTable & HashTable::operator=(const HashTable &right) { // complete this method









}
HashTable::HashTable(const HashTable &orig) { // complete this method









}
```

Question 7 (10 points)

Write the output for the following application using `fork` system calls. **For each line of output, write a comment next to each line stating which process outputs the line**. 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.
- the OS executes one line from each active process in the order that the processes were created. For example, if there are three active processes, then the OS executes one instruction from P1, then one instruction from P2, then one instruction from P3, and then one instruction from P1, etc.
- any instruction can be executed in less than 1 ms.

```
#include <unistd.h>
#include <iostream>
using namespace std;
int main() {
	cout << "before fork1" << endl;
	pid_t x = fork();
	cout << "after fork1" << endl;
	if (x == 0) {
		cout << "before fork2" << endl;
		pid_t y = fork();
		cout << "after fork2" << endl;
		if (y == 0) {
			cout << "y == 0" << endl;
		}
	}
	sleep(1);
	cout << "end of main" << endl;
	return 0;
}
```

Question 8 (7 points)

For parts **a – d**, draw the resulting MinHeap tree after the insert / remove statements. Note that **each part is dependent on the previous part(s)**. Clearly label the final tree structure for each part.

a.

```
minHeap.insert(7);
minHeap.insert(20);
minHeap.insert(-5);
minHeap.insert(5);
minHeap.insert(10);
```

b.

```
minHeap.removeMin();
minHeap.removeMin();
```

c.

```
minHeap.insert(1);
minHeap.insert(15);
minHeap.insert(-20);
```

Question 9 (13 points)

Use the following code segment to write the output for parts **a – d**. Clearly label each part.

```
class A {
public:
	A() {}
	~A() { cout << "A::~A" << endl; }
	void function1() { cout << "A::f1" << endl; }
	virtual void function2() = 0;
};
class B : public A {
public:
	B() {}
	virtual ~B() { cout << "B::~B" << endl; }
	virtual void function2() { cout << "B::f2" << endl; }
	void function3() { cout << "B::f3" << endl; }
};
class C : public B {
public:
	C() {}
	~C() { cout << "C::~C" << endl; }
	void function3() { cout << "C::f3" << endl; }
};
```

a.

```
int main() {
	A* a1 = new B();
	A* a2 = new C();
	a1->function1();
	a1->function2();
	a2->function1();
	a2->function2();
	delete a1;
	delete a2;
}
```

b.

```
int main() {
	B* b1 = new B();
	B* b2 = new C();
	b1->function1();
	b1->function2();
	b1->function3();
	delete b1;
	delete b2;
}
```

c.

```
int main() {
	C* c1 = new C();
	c1->function1();
	c1->function2();
	c1->function3();
	delete c1;
}
```

d.

```
void d() {
	C c1;
	B b1 = c1;
	b1.function3();
}
int main() {
	d();
}
```

Scratch Paper (Do not detach)