CS 32 Fall '21 Final

Answer Key

Instructions

- Please read all the instructions very carefully.
- This exam is graded out of 90 points and you have 3 hours to finish the exam.
- Make sure that the handwriting in your solutions is legible.
- **For questions 2–9**, you can write *"I don't know but I promise I will learn"* and get 20% of the points.
- 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. (12 points) Total for Question 1: 12

   Mark whether each statement is True or False. If **false**, *briefly* state why.

   1. (T | F) Quicksort is always faster than bubble sort.
      False. Bubble sort takes $\Theta(n)$ time on a sorted list, quicksort takes $O(n)$.
   2. (T | F) A class `A` with only private constructors cannot be inherited from any other class not mentioned in the definition of `A`.
      True.
   3. (T | F) Threads have memory isolation (when a child thread gets created, it gets its own copy of the parents memory).
      False. Threads share memory, but get their own stack.
   4. (T | F) `jobs` shows all processes created during the current shell session, including the grandchildren (child processes of the commands run on shell).
      False. It shows only the processes directly launched from the shell.
   5. (T | F) Object slicing happens only when passing an object of a derived class by value.
      True. (Or, false, if you argue that assignment is not passing by value. You'll get 2/2 for that explanation too.)
   6. (T | F) Finding an element in an `std::set` takes $O(1)$ time.
      False. It takes $O(\log n)$ time.

2. Total for Question 2: 6

A common idiom in C++ is "private implementation" (pimpl): an interface class holds a pointer to its implementation so that if the implementation header changes, the modules depending on the interface don't need to be recompiled. The `Buffer` class below uses the pimpl idiom (so the actual class is behind the pointer `impl`).

```
// in buffer.h
class Buffer {
  BufferImpl * impl;
 public:
  // The big 3
  Buffer(Buffer& that);
  Buffer& operator = (Buffer& that);
  ~Buffer();

  // Some interface methods
  string toString() const;
  void append(const string&);
};

// in buffer.cpp
Buffer::Buffer() : impl(new BufferImpl) {}
```

(a) (6 points) Implement the copy constructor, copy assignment operator, and the destructor for `Buffer` class. You can assume that `BufferImpl` already implements these.

```
Buffer::Buffer(Buffer& that) : impl(new BufferImpl(that.impl)) {}

Buffer& Buffer::operator = (Buffer& that) {
  if (this == &that) return *this; // this is not necessary, but OK
  *impl = *that.impl;
  return *this;
}

Buffer::~Buffer() {
  delete impl;
}
```

3. (10 points) Fill in the blanks of the merge function below that implements the merge part of merge sort covered in class:

   See the algorithm in class for the solution, you will get the full grade even if you don't have the calls to `move`.

```
// Merge sorted subarrays [begin, middle) and [middle, end)
template<typename T>
void Merge(const typename vector<T>::iterator begin,
           const typename vector<T>::iterator middle,
           const typename vector<T>::iterator end) {
  // a temporary vector that will contain the sorted vector, we will move the
  // elements back after merging. The rest of the function will put the sorted
  // array into tmp
  vector<T> tmp;

  auto left = begin;
  auto right = middle;

  while (_________________ && ________________) {
    if (*left <= *right) {
      tmp.push_back(_________);
      ___________;
    } else {
      tmp.push_back(_________);
      ___________;
    }
  }

  while (left < _________) {
    tmp.push_back(___________);
    left++;
  }

  while (right < _________) {
    tmp.push_back(___________);
    right++;
  }

  // writes back the temporary vector
  auto j = begin;
  for (auto i = tmp.begin(); i != tmp.end(); ++i, ++j) {
    *j = move(*i);
  }
}
```

4. (6 points) The `HashMap` class below implements a hash table with chained hashing:

```
const size_t SIZE = ...;
// helper function that always returns a hash value < SIZE
template<T>
size_t hash(T& x) { ... }

template<Key, Value>
class HashMap {
  // the actual table
  vector<pair<Key, Value>> table[SIZE];
 public:
  void insert(Key k, Value v);
  // other methods ...
};
```

Implement the `insert` member function. If the key already exists, insert should update the value. Insert should not insert multiple copies of the same key.

```
template<Key, Value>
void HashMap::insert(Key k, Value v) {
  auto& bucket = table[hash(k)]; // table[hash(k) % SIZE] is also OK
  // a for loop instead of the call below is OK
  auto cell = std::find(bucket.begin(), bucket.end(),
                        [&k](auto p) { return p->first == k; });

  if (cell == bucket.end()) {
    // k is not in the map, insert (k, v)
    bucket.push_back({k, v});
  } else {
    // k is in the map, update its corresponding value
    cell->second = v;
  }
}
```

5. (12 points) Consider the class definitions below

```
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; }
 void f2() override { 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; }
};
```

Write the output of each snippet under the snippet. If there is a compile time error, write "Compiler error". If there is a run time error (including undefined behavior) write "Runtime error" after all the output before the error.

(a)
```
A * x = new C();
x->f1();
A y = *x;
y.f2();
delete x;
```

```
A.f1
A.f2
~C
~B
~A
~A
```

(b)
```
A * x = new B();
x->f1();
A& y = *x;
y->f2();
delete x;
```

```
Compile error
```

(c)
```
C * x = new C();
x->f3();
B y = x;
y->f1();
y->f2();
y->f3();
delete x;
```

```
Compile error
```

(d)
```
C * x = new C();
B& y = x;
y->f1();
y->f2();
y->f3();
delete x;
```

```
A.f1
B.f2
C.f3
~C
~B
~A
```

6. Total for Question 6: 10

(a) (4 points) Suppose, we have a C++ program like the following:

```
#include <iostream>
int main() { cout << "Hi!\n"; return 0; } // calls write() syscall to print the message
```

After compiling this program, when we run it like the following:

```
$ ./hello
Hi!
```

3 system calls happen: fork, exec, and write (to write the message "Hi!"). Describe the order of these calls, and the processes (parent vs child processes of the shell or this program, etc.) AND the programs (the shell's code, or the C++ program above) making these calls.

- The shell (the parent) calls fork, and create a child process
- The child process of the shell calls exec and replaces itself with the hello program
- The hello program (the child process) calls write to print the message

(b) (2 points) What happens to a process when we suspend it (say, via `Ctrl-Z`)? How is a suspended process different from a running process and a killed (terminated) process?

A suspended process still has associated data in the kernel, and its resources are not cleaned up by the operating system yet (unlike a killed process). It does not get scheduled unlike a running process.

(c) (4 points) Write the output of the following program to the space on the right. If two statements may happen simultaneously, write the line with the larger x value first.

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

using namespace std;

int x = 1; // pay attention here

void f(int n) {
  if (n == 1) {
    sleep(1);
    cout << "done, x = " << x << endl;
    return;
  }
  cout << "fork " << n << "," << x << endl;
  if (fork() == 0) {
    x++;
  }
  sleep(2*x);
  f(n-1);
}

int main(int argc, char **argv) {
  f(4);
  return 0;
}
```

```
forking, n = 4, x = 1
forking, n = 3, x = 1
forking, n = 3, x = 2
forking, n = 2, x = 1
forking, n = 2, x = 2
done, x = 1
forking, n = 2, x = 2
done, x = 2
forking, n = 2, x = 3
done, x = 2
done, x = 3
done, x = 2
done, x = 3
done, x = 3
done, x = 4
```

7. Total for Question 7: 12

(a) (4 points) What is deadlock? When does it happen? **Briefly** explain.

A deadlock is a situation where two threads are both waiting for a resource/lock held by the other so they cannot make process. For example, two threads T1 and T2 hold onto two resources R1 and R2 in a way that T1 is holding R1 and waiting for R2 to be released, and T2 is holding R2 and waiting for R1 to be released.

(b) (2 points) Give one reason why one would prefer creating threads over processes for concurrency or parallelism. Explain it in 1 sentence.

Possible answers:

- Threads share memory so they can be used to solve problems where they need to manipulate the same data structure (e.g. matrix multiplication).
- A thread is more lightweight than a process (no separate file handlers, code, heap etc.), so we can create more of it.

(c) (6 points) Write all possible outputs of the program below to the right of it.

```
#include <thread>
#include <mutex>
#include <iostream>

using namespace std;

int x = 0; // pay attention here
mutex m;

void f(int n) {
  m.lock();
  cout << "f(" << n << ")\n";
  m.unlock();
  if (n <= 2) {
    m.lock();
    x++;
    m.unlock();
    return;
  }
  thread b(f, n - 2);
  thread a(f, n - 1);
  a.join();
  b.join();
}

int main(int argc, char *argv[]) {
  f(4);
  cout << "x is " << x << endl;
  return 0;
}
----------------------
Answer:

f(4)  f(4)  f(4)  f(4)
f(2)  f(3)  f(3)  f(3)
f(3)  f(2)  f(2)  f(1)
f(1)  f(1)  f(2)  f(2)
f(2)  f(2)  f(1)  f(2)
```

8. (a) (7 points) Write the output of the following program:

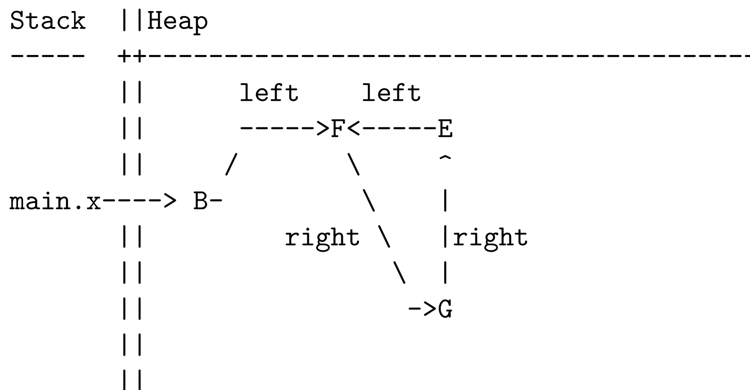
Total for Question 8: 14

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

struct Node {
  string name;
  shared_ptr<Node> left;
  shared_ptr<Node> right;
  Node(string name) : name(name) { cout << "created " << name << endl; }
  ~Node() { cout << "destroyed " << name << endl; }
};

shared_ptr<Node> buildGraph() {       // Write the output here
  auto a = make_shared<Node>("A");    //
  auto b = make_shared<Node>("B");    //  created A
  auto c = make_shared<Node>("C");    //  created B
  c->right = make_shared<Node>("D");  //  created C
  b->left = c->right;                 //  created D
  auto e = make_shared<Node>("E");    //  created E
  c->left = e;                        //  created F
  a->right = b->left;                 //  destroyed A
  a->left = b;                        //  created G
  e->left = make_shared<Node>("F");   //  destroyed C
  a = e->left;                        //  destroyed D
  a->right = make_shared<Node>("G");  //  created H
  a->right->right = e;                //  destroyed H
  b->left = e->left;                  //  destroyed B
  return b;                           //
}                                     //
                                      //
int main() {                          //
  auto x = buildGraph();              //
  /* DRAW MEMORY HERE */              //
  auto h = make_shared<Node>("H");    //
  h->left = x;                        //
  return 0;                           //
}
```

(b) (4 points) Draw a memory diagram showing which variables and objects point to other objects when the program reaches the line marked with `/* DRAW MEMORY HERE*/`.



(c) (3 points) What is one advantage of using `unique_ptr` over `shared_ptr`?

- `unique_ptr` does not do reference counting so it is faster.
- `unique_ptr` cannot create cycles (because there can be at most one owner), so it is guaranteed to not leak memory in the way that `shared_ptr` does.

9. Total for Question 9: 8

We covered many topics in this class. The topics asked below can be some C++ feature (some collections, standard library functions, virtual functions, etc.), a general design idea (like inheritance, polymorphism, exceptions, unit testing), a data structure or an algorithm, other concepts, or something specific from an assignment.

(a) (4 points) What is one topic you liked or found interesting in this class? Why? Elaborate in 2–3 sentences.

(b) (4 points) What is one topic you struggled with, maybe something that took a while to "click" or was difficult to implement? Why? Elaborate in 2–3 sentences.

Scratch Paper