CSE 333 Final Exam Sample Solution 8/22/14

Question 1. (20 points) A bit of C++ hacking – STL version. The C++ 11 standard library contains a container `unordered_set<T>` that stores sets of elements with no duplicates. Internally the set elements are stored using a hash table, so there is no guarantee about the order in which the elements are retrieved by an iterator.

On the next page, give the full implementation of a function `intersect` that returns a new `unordered_set` that is the intersection of its two arguments. For example, if we execute the following code:

```
unordered_set<int> s1 { 22, -5, 333, 17, -12, 42 };
unordered_set<int> s2 { 42, 5, 333, 9, 17, 0 };
unordered_set<int> result = intersect(s1, s2);
```

the variable `result` should contain the unordered collection of values { 17, 333, 42 }, i.e., only values that appear in both of the argument sets. The `intersect` function **must not** make any changes to either of its arguments. Your function will need one `unordered_set` variable to accumulate the result, but it may not use any other STL containers, C++ arrays, or other complex data structures.

Your solution only needs to work with argument and result values of type `unordered_set<int>`, i.e., sets of integers. You need to provide proper declarations for the parameter and result types for the function, as well as the code to implement it. You may, if you wish, use a function template so it will work with any element type, but there is no extra credit for doing so. Your code should be written in good style.

Some reference information: `unordered_set` has the same interface as `set`, except it does not guarantee the order in which values are stored or are retrieved by an iterator.

- The default (0 argument) constructor creates an empty `unordered_set`. Copy constructors and initializer lists can also be used to create new sets.
- As with any STL container, if `s` is a `unordered_set`, `s.begin()` and `s.end()` return iterator values that might be useful. If `it` is an iterator, then `*it` can be used to reference the item `it` currently points to, and `++it` will advance `it` to the next item, if any.
- Some useful operations on all STL containers, including `unordered_set`:
  - `c.clear()` – remove all elements from `c`
  - `c.size()` – return number of elements in `c`
  - `c.empty()` – true if number of elements in `c` is 0, otherwise false
- Some additional operations on unordered_sets:
  - `s.insert(x)` – add `x` to `s` if not already present
  - `s.count(x)` – number of copies of `x` in `s` (0 or 1)

You are free to use the C++11 `auto` keyword, C++11-style `for`-loops for iterating through containers, and any other features of standard C++11, but you are not required to use these.

Write your answer on the next page. You may remove this page for reference while working if you wish.

Question 1. (cont.) Write your implementation of function `intersect` below.

This solution defines a template. A non-template solution would omit the first line and would have `int` in place of `T` in the various `unordered_set` declarations.

```
template <class T>
unordered_set<T> intersect(const unordered_set<T> &s1,
                           const unordered_set<T> &s2) {
  unordered_set<T> result;
  for (auto val: s1) {
    if (s2.count(val) > 0) {
      result.insert(val);
    }
  }
  return result;
}
```

Many solutions used a pair of nested loops to iterate through the entire contents of one set repeatedly for every single element in the other set. Since we didn't realize we needed to disallow that in the question, we gave credit for answers that did it correctly – even though that changes something that should require only $O(n)$ time into an $O(n^2)$ operation.

Question 2. (18 points) C++ programming – with templates this time. Write a C++ template for a function `max` that has two parameters, an array `a` and an integer `n`. Function `max` should return the largest value found in the first `n` elements of `a`. For example, if we execute the following statements,

```
int ints[5] = { 22, -5, 333, 17, 42 };
string strings[4] = { "groucho", "harpo", "chico", "zeppo" };
cout << "largest int is " << max(ints, 5) << endl;
cout << "largest string is " << max(strings, 3) << endl;
```

the output should be (note that only 3 elements of the string array are examined):

```
largest int is 333
largest string is harpo
```

Your code should work on any array type whose elements can be compared with '<'. You may not use any additional arrays, STL containers, or other complex data structures. You should assume `n>=1`, i.e., the array has at least one element. Use good style.

```
// return the largest element in a[0..n-1]
// pre: n >= 0
template<class T>
T max(T a[], int n) {
  T largest = a[0];
  for (int k = 1; k < n; k++) {
    if (largest < a[k]) {
      largest = a[k];
    }
  }
  return largest;
}
```

Note: Several answers initialized `largest` (or the equivalent variable) to 0 or `nullptr`, or left it uninitialized. None of those are correct for arbitrary types. Even if `T` is `int`, 0 is the wrong initial value if the array contains only negative numbers.

The question stated that elements of the array could be compared with <, which is the same as the typical minimum requirement for elements stored in ordered STL containers. But that means that the solution cannot use >, >=, ==, or other relations. They are not necessarily available.

Question 3. (15 points) Dynamic dispatch. The following program prints `howle rorwl`. Add the keyword `virtual` in appropriate places so that the program will print `hello world` instead. You may not make any other changes to the code.

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

class A {
public:
        void m1() { cout << "h"; }
virtual void m2() { cout << "o"; }
virtual void m3() { cout << "w"; }
};
class B : public A {
public:
virtual void m1() { cout << "r"; }
        void m2() { cout << "e"; }
        void m3() { cout << "l"; }
};
class C : public B {
public:
        void m1() { cout << "w"; }
        void m2() { cout << "o"; }
        void m3() { cout << "d"; }
};
int main() {
  A* ab = new B();
  B* b = new B();
  B* bc = new C();
  A* ac = new C();
  ac->m1();
  ab->m2();
  ab->m3();
  b->m3();
  bc->m2();
  cout <<" ";
  bc->m1();
  ac->m2();
  b->m1();
  ab->m3();
  bc->m3();
  cout << endl;
  return 0;
}
```

This solution shows the minimum number of "`virtual`"s needed to cause the program to print "hello world". It would be fine to put `virtual` in front of all of the other methods in classes `B` and `C`, although it would not change the output. The correct output is not produced if `virtual` is placed in front of `m1` in class `A`.

Question 4. (15 points) Pointy Things. Consider the following program:

```
 1  #include <memory>
 2  #include <stdlib.h>
 3  #include <iostream>
 4  using namespace std;
 5
 6  class Thing {
 7  private:
 8    int id_;
 9  public:
10    Thing(int n): id_(n) {}
11    ~Thing() { cout << "destruct " << id_ << endl; }
12  };
13
14  void DoingThings(shared_ptr<Thing> &p){
15    Thing* b = new Thing(2);
16    unique_ptr<Thing> c(new Thing(3));
17    c.reset(new Thing(4));
18    cout << "new scope" << endl;
19    {
20      unique_ptr<Thing> d(new Thing(5));
21      shared_ptr<Thing> e = p;
22    }
23    cout << "got here" << endl;
24  }
25
26  int main(){
27    shared_ptr<Thing> a(new Thing(1));
28    DoingThings(a);
29    unique_ptr<Thing> g(new Thing(6));
30    cout << "another scope" << endl;
31    {
32      shared_ptr<Thing> j = a;
33      Thing* i = g.release();
34      delete i;
35    }
36    cout << "almost done" << endl;
37  }
```

Answer questions about this program on the next page. You may detach this page for convenience if you wish.

Question 4. (cont.) (a) (11 points) What output does the above program produce when it is executed?

```
destruct 3
new scope
destruct 5
got here
destruct 4
another scope
destruct 6
almost done
destruct 1
```

(b) (4 points) Are there any memory leaks, dangling pointers, or other memory management errors in this program? If so, describe the errors and where they are in the code. (You can give a brief verbal description; you don't need to annotate the code unless you want to.) If a `Thing` object is leaked, include its `id_` number in your description; if there are dangling pointers or other problems, identify the specific pointer(s) and describe the problem(s) – *briefly*!

There is one memory leak. `Thing(2)`, allocated in line 15 (variable `b`), is never deleted.

Question 5. (16 points) Games. Here is a sketch of a program that plays a game with a user using standard console input and output. The details of the game don't matter for this question. We assume that I/O operations are done with POSIX read/write functions called from inside the `GetUserMove` and `WriteComputerMove` functions.

```
// Read the next user move from the given input file descriptor
// and return a numeric code describing the move.  0 means quit;
// non-zero is a specific move.
int GetUserMove(int input_file_descriptor) { ... }

// Given a code describing the last user input move, compute a
// response and write it to the given output file descriptor
void WriteComputerMove(int user_input,
                       int output_file_descriptor) { ... }

// play a game using standard console I/O until the user quits.
const int STDIN_FD = 0;
const int STDOUT_FD = 1;

int main() {
  while (1) {
    int user_move = GetUserMove(STDIN_FD);
    if (user_move == 0)
      break;  // game over
    WriteComputerMove(user_move, STDOUT_FD);
  }
  return 0;
}
```

We would like adapt this code to create a "game server" to play this game with a network user using the same `GetUserMove` and `WriteComputerMove` functions and basic game loop shown above. You need to write server pseudo-code to create a connection to a client machine, read input from the client and write answers back, and close the connection and terminate when the user ends the game (i.e., when `GetUserMove` returns 0). The server only needs to handle one client and should shut down when the game with that single client is done – i.e., you don't need to handle multiple clients.

The pseudo-code should have a similar level of detail as the code above, but it should be clear where sockets are set up, which file descriptors are used at various points in the code, and so forth. But you don't need to worry about all of the grubby details of the exact socket parameter lists. The next page summarizes many networking functions, and you will want to use several of them in your answer.

Question 5. (cont.) Reference information:

Here is a sketch of some TCP, IP, and POSIX I/O functions and data structures that may be useful in your solution. You do not need to provide detailed values for all arguments as you would in a real program, but you do need to keep track of file descriptors and sketch when address information is created and used, so it is clear how connections are set up and which descriptors are used for input and output. You should assume that all functions succeed on the very first try and you do not need to check for errors.

You will not need to use all of these functions in your answer.

If it matters, you can use port 1337 for your server, but any number is fine.

- struct addrinfo – address data structure; appears as *address_info* in several function descriptions, below
- int **accept**(*file_descriptor*, ...), // returns a *file_descriptor*
- **bind**(*file_descriptor*, *address_info*)
- **close**(*file_descriptor*)
- **connect**(*file_descriptor*, *address_info*);
- **freeaddrinfo**(struct addrinfo **result*)
- **getaddrinfo**(NULL, *port*, struct addrinfo * *hints*, struct addrinfo ***result*)
- **listen**(*file_descriptor*)
- int **read**(*file_descriptor*, *buffer*) // returns # bytes read or error
- int **socket**(*address_info*) // returns a *file_descriptor*
- int **write**(*file_descriptor*, *buffer*) // returns # bytes written or error

Write the pseudo-code for your game server on the next page. You can detach this page, too, for reference if you want.... ☺

Question 5. (cont.) Write your game server pseudo-code below.

The main thing we were looking for here was the correct sequence of calls to set up the server and accept a client connection. Answers that did that correctly and kept track of the file descriptors received full credit.

```
struct addrinfo hints, *rp;
getaddrinfo(NULL, 1337, &hints, &rp);
int listen_fd = socket(rp...);
bind(listen_fd, rp...);
freeaddrinfo(rp);
listen(listen_fd);
int client_fd = accept(listen_fd, ...)
while (1) {
  int user_move = GetUserMove(client_fd);
  if (user_move == 0)
    break;  // game over
  WriteComputerMove(user_move, client_fd);
}
close(listen_fd);  // could be done earlier
close(client_fd);
```

Question 6. (15 points) Doing (too?) many things at once? The following C++ code is part of a class that implements a stack of integers using a linked list representation.

```
 1  class Stack {
 2   public:
 3    // construct new, empty stack
 4    Stack() : top_(nullptr) { }
 5
 6    // push n onto top of stack
 7    void push(int n) {
        lock_.Acquire();    // added
 8      top_ = new Node(n, top_);
        lock_.Release();    // added
 9    }
10
11    // return top value from stack and remove it
12    // pre: stack is not empty
13    int pop() {
        lock_.Acquire();    // added
14      int topval = top_->val_;
15      Node *p = top_;
16      top_ = top_->next_;
17      delete p;
        lock_.Release();    // added
18      return topval;
19    }
20
21   private:
22    // stack representation is a linked list of Nodes
23    class Node {
24     public:
25      int val_;     // integer value in this Node
26      Node *next_;  // Node below this one, nullptr if none
27      // convenience constructor
28      Node(int value, Node* next): val_(value),next_(next) {}
29    };
30    Lock lock_;   // lock variable for this Stack -- added
31    Node* top_;   // ptr to top Node; nullptr if empty stack
32  };
```

Other operations, including the destructor, are omitted to save space.

Answer questions about this class on the next page. You may want to leave this page in the exam to hand in with your answer to part (b).

Question 6. (cont.) (a) (8 points) This code is not safe to use in a multi-threaded, concurrent program. Give a concrete example that explains a problem that can occur if this code is used in a multi-threaded program. (There may be many possible examples, you just need to give one.)

Any situation that involves two threads sharing a `Stack` where at least one of them modifies it is probably unsafe, most likely because of concurrent operations on `top_`. A very simple example is two threads pushing values onto the same stack. Suppose two threads are interleaved like this, using a shared stack `s`:

Thread 1	**Thread 2**
call s.push(1);	
read top_ into register r1	
	call s.push(2);
	read top_ into register r2
	execute top_ = new Node(2, r2);
execute top_ = new Node(1,r1)	

The first thread will overwrite the value of top_ stored by the second thread. The value 2 will be lost from the stack, and we have a memory leak – the node containing val_ = 2 is never deleted.

Note that it is *not* a problem if the order in which two threads operate on a shared stack is different in different executions. If, for example, thread 1 pushes 1 and thread 2 pushes 2, it is permissible for either thread to go first, and there is no way to predict whether 1 or 2 will be on top of the stack after both operations finish.

(b) (7 points) Assume that we have a class Lock with the following operations:

```
Lock lock;          // construct a new Lock variable that is unlocked
lock.Acquire();     // delay thread until lock is available then acquire the lock
lock.Release();     // release (unlock) the lock.  If other threads are waiting
                    // for the lock to be available, one of the pending Acquire
                    // operations will succeed and that thread will run
```

In the code for class `Stack` on the previous page, show where to insert one or more `Lock` variables and `Acquire` and `Release` operations so the `Stack` class can be used safely by concurrent, multi-threaded programs. Or you can describe the changes below, but be specific and be sure it is crystal clear what changes should be made and where.

See code inserted on the previous page. The key idea is that each `Stack` should have an associated `Lock` variable that is acquired and released when an operation is performed.

Question 7. (1 free point) The very best telephone of all time is (circle):

a) Palm Treo

b) Nokia 9000

c) Motorola TAC

d) BlackBerry

e) iPhone

f) Nokia Windows phone

g) Samsung Galaxy

h) Motorola Android

i) Western Electric 500 

j) Generic flip-phone

k) You are clueless! The correct answer is _____!!

l) I don't care, but please give me my free point!

m) I do care, but I'm not telling. Give me my free point anyway!!

n) Other: **Tin-can telephone**