

CSE 333 Final Exam 6/12/13

Name _____

There are 13 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, open mind.

If you don't remember the exact syntax for something, make the best attempt you can. We will make allowances when grading.

Don't be alarmed if there seems to be more space than is needed for your answers – we tried to include more than enough blank space.

Relax, you are here to learn.

Please wait to turn the page until everyone is told to begin.

Score _____ / 100

1. _____ / 18

8. _____ / 6

2. _____ / 12

9. _____ / 6

3. _____ / 16

10. _____ / 6

4. _____ / 6

11. _____ / 6

5. _____ / 7

12. _____ / 4

6. _____ / 6

13. _____ / 1

7. _____ / 6

CSE 333 Final Exam 6/12/13

Question 1. (18 points) A little C++ hacking. On the following page, implement function `intersect`. The input to the function is two lists of strings that are sorted in ascending order and do not contain duplicates. The function should replace the original contents of the output list with copies of all the strings that appear in both of the original lists, and the words should appear in the output list in the same sorted order they were found in the original lists.

For example, if we initially have

```
list<string> fruit = {"apple", "banana", "cherry", "lemon",  
                    "mango", "orange", "watermelon"};  
list<string> citrus = {"grapefruit", "lemon", "lime", "orange"};  
list<string> result = {"asparagus", "broccoli"};
```

then after the function call `intersect(fruit, citrus, &result)`, the list `result` should contain `{"lemon", "orange"}`.

For full credit, your function must scan the input lists only once, i.e., it should run in linear time proportional to the lengths of the input lists.

Hints and reference information:

- Remember that a STL `list` is a linked list underneath, so you must use iterators to scan it; you can't use `[]` subscripts to access individual list elements.
- If `lst` is a STL `list`, then `lst.begin()` and `lst.end()` return iterator values of type `list<...>::iterator` 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 sequential STL containers, including `list`:
 - `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
 - `c.push_back(x)` – copy `x` to end of `c`
 - `c.push_front(x)` – copy `x` to front of `c`

Note for the curious: this problem deliberately avoided using templates because they introduce some technical complications in variable declarations that seemed best omitted.

(write your answer on the next page – you may remove this page from the exam if you wish)

CSE 333 Final Exam 6/12/13

Question 1. (cont.) Complete the definition of function `intersect` below. Some useful `#include` directives as well as a `using` directive have been provided for convenience. You should add any additional libraries or code that you need (although the sample solution only needed the ones given here).

```
#include <string>
#include <iostream>
#include <list>

using namespace std;

// replace previous contents of ans with list of elements
// that appear in both lst1 and lst2.
// pre: lst1 and lst2 elements are sorted in ascending
//      order and the lists contain no duplicates.
void intersect(const list<string> &lst1,
              const list<string> &lst2,
              list<string> *ans) {

}

}
```

CSE 333 Final Exam 6/12/13

Question 2. (12 points) A debugging exercise. Below, and continued on the next page is the valgrind output from a test run of the searchshell program from homework 2. Although the code seems to “work” in that it produces the right answer(s), the valgrind output suggests there are problems. What seems to be wrong and what is the evidence for your conclusions?

```
bash-4.2$ valgrind --leak-check=full ./searchshell test_tree/enron_email/
==10186== Memcheck, a memory error detector
==10186== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==10186== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==10186== Command: ./searchshell test_tree/enron_email/
==10186==
Indexing 'test_tree/enron_email/'
==10186== Invalid write of size 1
==10186==   at 0x4A09440: strncpy (mc_replace_strmem.c:476)
==10186==   by 0x401733: DTRegisterDocumentName (doctable.c:77)
==10186==   by 0x401495: HandleFile (filecrawler.c:181)
==10186==   by 0x4013D8: HandleDir (filecrawler.c:147)
==10186==   by 0x4011BA: CrawlFileTree (filecrawler.c:77)
==10186==   by 0x400C60: main (searchshell.c:46)
==10186== Address 0x4c8d3ca is 0 bytes after a block of size 26 alloc'd
==10186==   at 0x4A0881C: malloc (vg_replace_malloc.c:270)
==10186==   by 0x401705: DTRegisterDocumentName (doctable.c:75)
==10186==   by 0x401495: HandleFile (filecrawler.c:181)
==10186==   by 0x4013D8: HandleDir (filecrawler.c:147)
==10186==   by 0x4011BA: CrawlFileTree (filecrawler.c:77)
==10186==   by 0x400C60: main (searchshell.c:46)
==10186==
```

(valgrind output continued on the next page)

CSE 333 Final Exam 6/12/13

Question 2. (cont.) Remainder of the valgrind output.

```
==10186== Invalid read of size 1
==10186==   at 0x4A09194: strlen (mc_replace_strmem.c:403)
==10186==   by 0x401851: DTRegisterDocumentName (doctable.c:106)
==10186==   by 0x401495: HandleFile (filecrawler.c:181)
==10186==   by 0x4013D8: HandleDir (filecrawler.c:147)
==10186==   by 0x4011BA: CrawlFileTree (filecrawler.c:77)
==10186==   by 0x400C60: main (searchshell.c:46)
==10186== Address 0x4c8d3ca is 0 bytes after a block of size 26 alloc'd
==10186==   at 0x4A0881C: malloc (vg_replace_malloc.c:270)
==10186==   by 0x401705: DTRegisterDocumentName (doctable.c:75)
==10186==   by 0x401495: HandleFile (filecrawler.c:181)
==10186==   by 0x4013D8: HandleDir (filecrawler.c:147)
==10186==   by 0x4011BA: CrawlFileTree (filecrawler.c:77)
==10186==   by 0x400C60: main (searchshell.c:46)
==10186==
enter query:
xyzyy
enter query:
==10186==
==10186== HEAP SUMMARY:
==10186==   in use at exit: 178,112 bytes in 7,980 blocks
==10186== total heap usage: 2,733,651 allocs, 2,725,671 frees, 57,300,024 bytes allocated
==10186==
==10186== 80,216 (24 direct, 80,192 indirect) bytes in 1 blocks are definitely lost in loss record
11 of 12
==10186==   at 0x4A0881C: malloc (vg_replace_malloc.c:270)
==10186==   by 0x4035FF: AllocateHashTable (HashTable.c:55)
==10186==   by 0x4015E0: AllocateDocTable (doctable.c:42)
==10186==   by 0x401130: CrawlFileTree (filecrawler.c:71)
==10186==   by 0x400C60: main (searchshell.c:46)
==10186==
==10186== 97,896 (24 direct, 97,872 indirect) bytes in 1 blocks are definitely lost in loss record
12 of 12
==10186==   at 0x4A0881C: malloc (vg_replace_malloc.c:270)
==10186==   by 0x4035FF: AllocateHashTable (HashTable.c:55)
==10186==   by 0x4015CF: AllocateDocTable (doctable.c:41)
==10186==   by 0x401130: CrawlFileTree (filecrawler.c:71)
==10186==   by 0x400C60: main (searchshell.c:46)
==10186==
==10186== LEAK SUMMARY:
==10186==   definitely lost: 48 bytes in 2 blocks
==10186==   indirectly lost: 178,064 bytes in 7,978 blocks
==10186==   possibly lost: 0 bytes in 0 blocks
==10186==   still reachable: 0 bytes in 0 blocks
==10186==   suppressed: 0 bytes in 0 blocks
==10186==
==10186== For counts of detected and suppressed errors, rerun with: -v
==10186== ERROR SUMMARY: 1978 errors from 4 contexts (suppressed: 2 from 2)
bash-4.2$
```

CSE 333 Final Exam 6/12/13

Question 3. (16 points) Virtual madness. Consider the following program, which compiles and executes without errors.

(See the rest of the question on the next page. You may remove this page for reference.)

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

class Uno {
public:
    void first() {second();third(); cout << "Uno::first" << endl;}
    void second() { third(); cout << "Uno::second" << endl; }
    virtual void third() { cout<< "Uno::third" << endl; }
};

class Due : public Uno {
public:
    void first() { second(); cout << "Due::first" << endl; }
    virtual void fourth() {second();cout << "Due::fourth" << endl;}
};

class Tre : public Due {
public:
    void second() { first(); cout << "Tre::second" << endl; }
    virtual void third() { cout << "Tre::third" << endl; }
};

int main(void) {

    Uno * one = new Uno();
    one->first();

    cout << "-----" << endl;

    Uno * ein = new Due();
    ein->first();

    cout << "-----" << endl;

    Due * two = dynamic_cast<Due *>(ein);
    two->first();

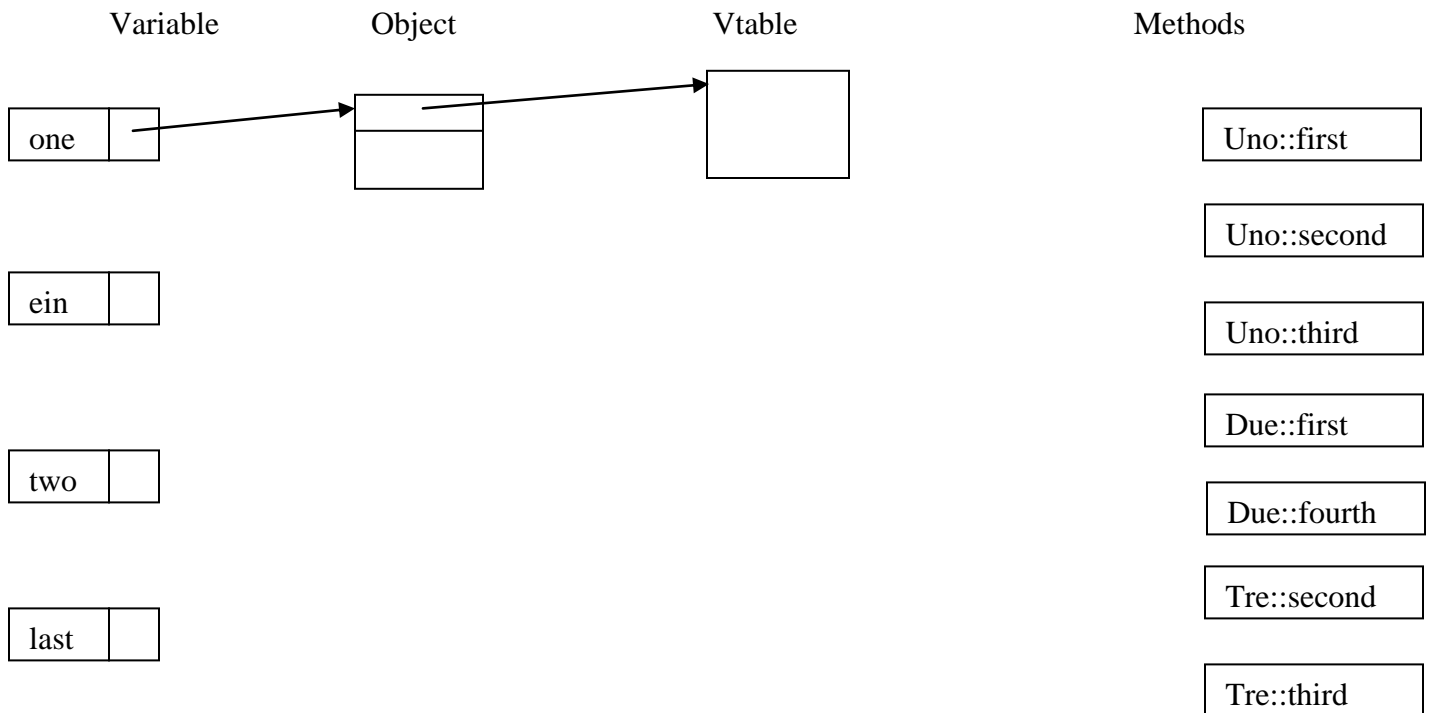
    cout << "-----" << endl;

    Due * last = new Tre();
    last->fourth();

    /* draw diagram at this point*/
    return 0;
}
```

CSE 333 Final Exam 6/12/13

Question 3. (cont) (a) (8 points) Draw a diagram showing all of the variables, objects, vtables, and methods they point to when execution reaches the end of main, right before executing the return statement. Note that you can draw this diagram without tracing execution of the method calls. The diagram only depends on the variable declarations and object creation code (e.g., new). To save time, we've filled in part of the drawing, including the first variable (one), the object and vtable it refers to, and the method bodies for all three classes (but we haven't included any necessary vtable pointer(s) to methods).



(b) (8 points) What does this program print when it is executed?

CSE 333 Final Exam 6/12/13

Question 4. (6 points) When we were exploring architectures for concurrent servers, one of our example servers created a new process to handle each incoming request. The code used a “double-fork” trick to create a grandchild process to actually handle each request. A sketch of the code, with error handling omitted, looked something like this:

```
// Loop forever, accepting a connection from a client and processing
// queries arriving over it.
while (1) {
    int client_fd = client request;
    pid_t pid = fork(); // create child process

    if (pid > 0) {
        // I'm the parent
        int stat_loc;
        pid_t res = wait(&stat_loc);
        close(client_fd);
        break; // handle next request
    } else if (pid == 0) {
        // I'm the child. Fork a grandchild and exit.
        pid = fork();
        if (pid > 0) {
            // I'm the child. Exit!
            exit(EXIT_SUCCESS);
        } else if (pid == 0) {
            // I'm the grandchild. Handle the client connection.
            HandleClient(...);
        }
    }
}
```

So the question is, why bother with all of this complexity? Why create a grandchild process to handle the request – why not just have the child process handle it?

(Brief answers that explain key technical points succinctly make graders happy.)

CSE 333 Final Exam 6/12/13

Question 5. (7 points) Fun with concurrency!! Consider the following C program:

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

int x = 0;

void * thread_worker(void * ignore) {
    x = x + 1;
    printf("x=%d\n", x);
    return NULL;
}

int main() {
    pthread_t t1, t2;
    int ignore;
    ignore = pthread_create(&t1, NULL, &thread_worker, NULL);
    ignore = pthread_create(&t2, NULL, &thread_worker, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("final x=%d\n", x);

    return 0;
}
```

For each of the following output sequences, circle “yes” if it could be produced by some possible execution of the above program and circle “no” if it could never happen under any circumstances. (Hint: at least one of these sequences is possible.)

YES	NO	x=0 x=1 final x=1
YES	NO	x=1 x=1 final x=1
YES	NO	x=1 x=2 final x=1
YES	NO	x=1 x=1 final x=2
YES	NO	x=1 x=2 final x=2
YES	NO	x=2 x=1 final x=2
YES	NO	x=2 x=2 final x=2

CSE 333 Final Exam 6/12/13

Some short answer questions. Please help us give you the most points by being accurate, brief but complete, and to the point.

Question 6. (6 points) As part of opening a connection to a client, a TCP server must use both the `accept()` and `listen()` functions. What do these two functions do, and in what order should the server code execute them?

Question 7. (6 points) Most of our network programming used TCP sockets. Another protocol built on top of IP is UDP. What is the key difference between UDP and TCP, and what sort of applications is UDP suited for compared to TCP? (a specific example of two and why would be helpful)

CSE 333 Final Exam 6/12/13

Question 8. (6 points) Suppose we have two C++ classes related by inheritance:

```
class Base { ... };  
class Derived : public Base { ... }
```

The subclass, `Derived`, can have additional methods and fields (instance variables) in addition to the ones it inherits from `Base`. In most of the code we've written using inheritance, we've used pointers and pointer assignments such as the following:

```
Derived * d = new Derived();  
Base * b = d;
```

But what about object value assignment? Is the following code legal (i.e., will it compile)? What happens when it is executed, if it does compile?

```
Derived dobj = ... ; // initialization omitted  
Base bobj = dobj;
```

Question 9. (6 points) There are several kinds of *smart pointers* in the C++ library. One kind is weak pointers (`weak_ptr`). These are similar to `shared_ptr`, but not quite the same. Why do we have both kinds? What problem is solved by `weak_ptr`s?

CSE 333 Final Exam 6/12/13

Question 10. (6 points) The following function unlinks the first node from a single-linked list and returns the pointer to that node in addition to updating the head of the list:

```
struct node {
    int val;
    struct node * next;
};

// remove the first node from the list starting at head and
// return a pointer to the unlinked node. Return NULL if
// the list is empty initially (head == NULL).
struct node * unlink(struct node ** head) {
    if (head == NULL)
        return NULL;
    struct node * tmp = *head;
    *head = (*head)->next;
    return tmp;
}
```

Are there any potential problems using this function to manage linked lists in a multi-threaded program? If so, what might go wrong? How could we fix it?

CSE 333 Final Exam 6/12/13

Question 11. (6 points) Orders of magnitude. Circle the time that is the closest to the typical time needed for the following operations. (msec = milliseconds, μ sec = microseconds, nsec = nanoseconds)

(a) Time for a modern laptop processor to add two numbers

100 msec 10 msec 1 msec 100 μ sec 10 μ sec 1 μ sec 100 nsec 10 nsec 1 nsec

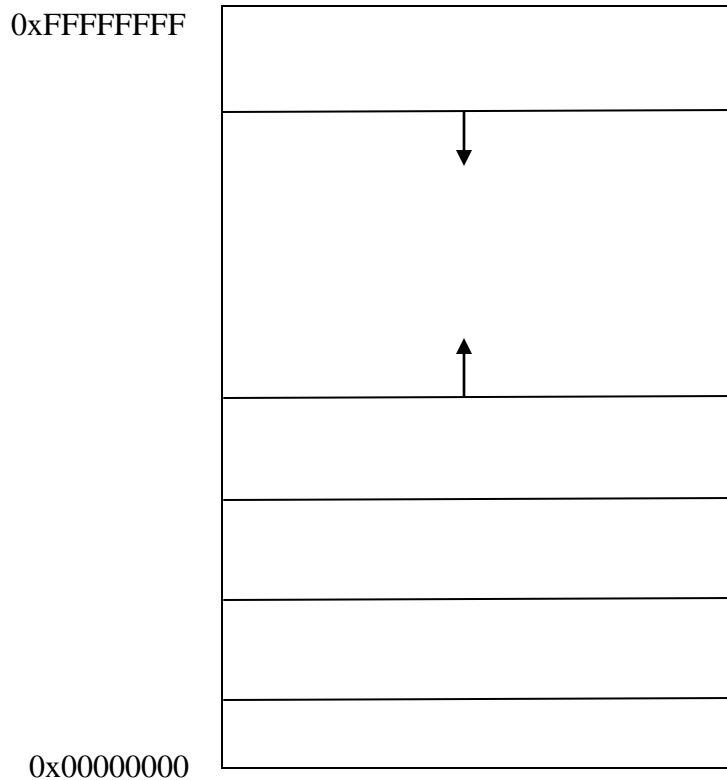
(b) Time to access main (not cache) memory on a modern laptop

100 msec 10 msec 1 msec 100 μ sec 10 μ sec 1 μ sec 100 nsec 10 nsec 1 nsec

(c) Time to read or write one record off of a rotating hard disk drive

100 msec 10 msec 1 msec 100 μ sec 10 μ sec 1 μ sec 100 nsec 10 nsec 1 nsec

Question 12. (4 points) Had to ask this one. Below is the standard diagram of the memory address space for a process. Label the boxes.



CSE 333 Final Exam 6/12/13

Question 13. (1 point – circle the letter of your choice)

The **BEST** thing about C++ is:

- (a) simple stream I/O (`cout/cin` instead of `printf/scanf`)
- (b) type-safe `new` and `delete` operators
- (c) it still requires manual memory management and it's possible to have dangling pointers, memory leaks, and all the other goodies – none of that wimpy garbage collection stuff.
- (d) references as well as pointers
- (e) smart pointers – not just the old, dumb ones
- (f) a much bigger library (STL, collections, iterators, etc.)
- (g) classes and objects
- (h) dynamic *and* static method dispatch
- (i) destructors
- (j) multiple inheritance
- (k) operator overloading
- (l) the many meanings of `const`
- (m) the many meanings of `static`
- (n) templates
- (o) function overloading
- (p) namespaces
- (q) You forgot the best one!! It's really _____ !!!
- (r) It's all great!!!!!!!!!!!!!!
- (s) C++ is the most loathsome programming language ever invented. There's nothing good about it except that maybe by knowing it I'll be able to get a job at Google. Even then I hope never to see it again.
- (t) I don't care – just give me my free point
- (u) I do care, but don't have time to think about it – just give me my free point