CSE 333 Final Exam December 13, 2017 Sample Solution

Question 1. (16 points) STL. Implement the template function `is_sorted`, below, so it returns `true` if its list argument is sorted in non-decreasing order, and returns `false` if not. Your function should work for any list whose elements can be compared using `operator<` (the usual minimal requirement for ordered values in STL containers). If the list is empty (contains no elements), the function should return `true`.

Examples: `is_sorted` should return `true` for the integer list { 1, 4, 7, 7, 10} and should return `false` for { 1, 4, 3, 7, 7, 10 }.

Restriction: The STL algorithms library contains a similar function. You may not use it in your solution. Your function should examine the elements in the list to determine if the list is or is not sorted.

Hints: Remember that if `x` is a STL container, `x.begin()` and `x.end()` are iterator values that might be useful. Variable declarations can often be simplified with `auto`. There is some reference information about the STL libraries at the end of the exam.

Write your code below. Assume that all necessary headers have been `#included`, but there is no `using namespace std;` directive (you can add one if you want to).

```
template <typename T>
bool is_sorted(std::list<T> lst) {
  if (lst.size() <= 1)
    return true;

  auto prev = lst.begin();
  auto curr = lst.begin();
  curr++;
  while (curr != lst.end()) {
    if (*curr < *prev) {
      return false;
    }
    prev++;
    curr++;
  }
  return true;
}
```

Note: Unlike a `vector`, a STL `list` does not overload `operator[]`, so subscripting cannot be used to directly access list elements.

Question 2. (24 points) And the point is? One of the summer interns left behind some C++ code that was supposed to implement circles containing a center point and a radius. The code is a bit odd, but we need to figure out what it does. Header guards are omitted to save space.

File Point.h:

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

class Point {
public:
  Point(): x_(0), y_(0) { cout << "Point()" << endl; }
  Point(const int x, const int y): x_(x), y_(y)
                        { cout << "Point(int,int)" << endl; }
  int x() {return x_;}
  int y() {return y_;}
private:
  int x_, y_;
};
```

File Circle.h:

```
#include "Point.h"
#include <iostream>
using namespace std;

class Circle {
public:
  Circle(): radius_(1) {
    center_ = new Point();
    cout << "Circle()" << endl;
  }
  Circle(const int radius, Point center): radius_(radius) {
    center_ = new Point(center);
    cout << "Circle(int,Point)" << endl;
  }
  ~Circle() {delete center_; cout << "~Circle()" << endl; }

  int radius() { return radius_; }
  bool isLarger(Circle other);
  void shift(Point newCenter);

private:
  int radius_;
  Point *center_;
};
```

(Code continued on the next page. You should remove this page from the exam. **Do not write anything on this page.** It will not be scanned for grading.)

Question 2. (cont.) More files:

File Circle.cc:

```
#include "Circle.h"
bool Circle::isLarger(Circle other) {
  return radius_ > other.radius();
}
void Circle::shift(Point p) {
  delete center_;
  center_ = new Point(p.x(), p.y());
}
```

File circletest.cc:

```
#include "Point.h"
#include "Circle.h"
#include <iostream>
int main() {
  Circle a = Circle(3, Point(1, 2));
  Circle b;
  std::cout << a.isLarger(b) << std::endl;
  Circle c = a;
  a.shift(Point(2, 2));
  return 0;
}
```

(a) (12 points) What output is produced when we compile and run this program? (It does run and terminate without crashing, although there may be some bugs). Note that the various constructors and destructors print messages when they are executed.

```
Point(int,int)
Circle(int,Point)
Point()
Circle()
1
~Circle()
Point(int,int)
Point(int,int)
~Circle()
~Circle()
~Circle()
```

(continued on next page)

Question 2. (cont.) When we ran this program using valgrind, it produced the following messages (program output omitted):

```
==9531== Invalid free() / delete / delete[] / realloc()
==9531==    at 0x4C2B18D: operator delete(void*) (vg_replace_malloc.c:576)
==9531==    by 0x400E7F: Circle::~Circle() (Circle.h:22)
==9531==    by 0x400C9C: main (circletest.cc:10)
==9531==  Address 0x5a19040 is 0 bytes inside a block of size 8 free'd
==9531==    at 0x4C2B18D: operator delete(void*) (vg_replace_malloc.c:576)
==9531==    by 0x400A9A: Circle::shift(Point) (Circle.cc:8)
==9531==    by 0x400C8B: main (circletest.cc:11)
==9531==  Block was alloc'd at
==9531==    at 0x4C2A203: operator new(unsigned long) (vg_replace_malloc.c:334)
==9531==    by 0x400E35: Circle::Circle(int, Point) (Circle.h:19)
==9531==    by 0x400BFB: main (circletest.cc:7)
==9531==
==9531== Invalid free() / delete / delete[] / realloc()
==9531==    at 0x4C2B18D: operator delete(void*) (vg_replace_malloc.c:576)
==9531==    by 0x400E7F: Circle::~Circle() (Circle.h:22)
==9531==    by 0x400CA8: main (circletest.cc:8)
==9531==  Address 0x5a19090 is 0 bytes inside a block of size 8 free'd
==9531==    at 0x4C2B18D: operator delete(void*) (vg_replace_malloc.c:576)
==9531==    by 0x400E7F: Circle::~Circle() (Circle.h:22)
==9531==    by 0x400C52: main (circletest.cc:9)
==9531==  Block was alloc'd at
==9531==    at 0x4C2A203: operator new(unsigned long) (vg_replace_malloc.c:334)
==9531==    by 0x400DBE: Circle::Circle() (Circle.h:15)
==9531==    by 0x400C07: main (circletest.cc:8)
==9531==
==9531== HEAP SUMMARY:
==9531==     in use at exit: 0 bytes in 0 blocks
==9531==   total heap usage: 3 allocs, 5 frees, 24 bytes allocated
==9531==
==9531== All heap blocks were freed -- no leaks are possible
==9531==
==9531== For counts of detected and suppressed errors, rerun with: -v
==9531== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Answer questions about this output on the next page. Remove this page from the exam. **Do not write anything on this page.** It will not be scanned for grading.

(continued on next page)

Question 2. (cont.) (b) (6 points) What error(s) in the original code caused the errors reported in the valgrind output on the previous page? (Hint: it might be useful to trace part of the execution of the program in addition to puzzling over the valgrind output.)

`Circle` does not contain a copy constructor, so the default (compiler-generated) copy constructor does a shallow copy of the `Circle` instance variables when we copy-construct a new `Circle` object from an existing one (either in a declaration or for function parameters or return values). When that happens, the `center_` instance variables in both `Circle`s reference the same `Point` object on the heap. When the `~Circle` destructors run, both of them will delete their `center_` variable, resulting in the double-delete errors reported by `valgrind`.

(c) (6 points) How should we fix the original code to eliminate these errors? Your change(s) should fix the problem(s) without changing the basic operation of the original code and should retain the original data representation and data structures of the objects to the extent possible. In other words, explain how to fix the problem(s) in the most straightforward, simplest way you can.

The simplest solution is to define a copy constructor for `Circle` that makes a copy of the `center_ Point` object. There are several ways to do this; here is one that we could add to the class declaration in `Circle.h`.

```
Circle(const Circle &other)
    : radius_(other.radius_),
      center_(new Point(*other.center_)) { }
```

Question 3. (32 points) The usual, demented, dreaded virtual function madness. Consider the following program, which, when appropriate code is inserted in the blank space in `main`, does compile and execute with no errors.

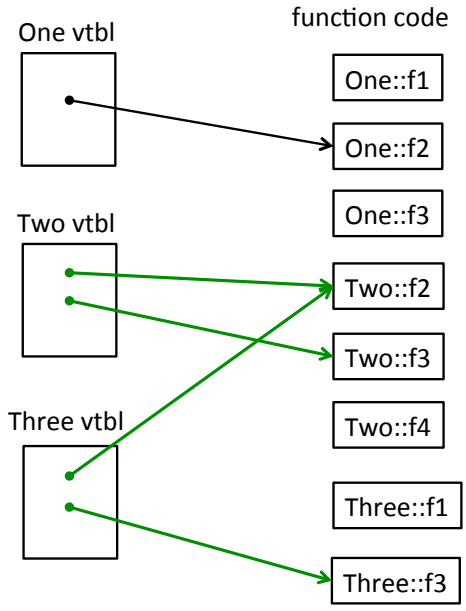
```
#include <iostream>
using namespace std;

class One {
public:
          void f1() { f3(); cout << "One::f1" << endl; }
  virtual void f2() {       cout << "One::f2" << endl; }
          void f3() {       cout << "One::f3" << endl; }
};

class Two: public One {
public:
          void f4() {       cout << "Two::f4" << endl; }
          void f2() { f1(); cout << "Two::f2" << endl; }
  virtual void f3() { f4(); cout << "Two::f3" << endl; }
};

class Three: public Two {
public:
  void f3() { f2(); cout << "Three::f3" << endl; }
  void f1() {       cout << "Three::f1" << endl; }
public:
};

int main() {
  [          ]
  return 0;
}
```

Remove this page from the exam, then answer questions about this code on the next pages. **Do not write anything on this page.** It will not be scanned for grading.

Question 3. (cont.) (a) (6 points) Complete the diagram below to show the layout of the virtual function tables for the classes given on the previous page. Be sure that the order of pointers in the virtual function tables is clear (i.e., which one is first, then next, etc.). One of the function pointers is already included to help you get started.

Notes: There are no pointers to non-virtual functions in vtables.

In derived classes, pointers for inherited functions must appear in the same order they do in base classes, followed by pointers to any new functions found in the derived class.



(b) (10 points) Now, for each of the following sequences of code, assume that we try to run the program with the given lines of code replacing the empty box in `main`. Either write the output that is produced when that program is executed, or, if an error occurs, give a concise description of the problem.

(i)
```
One *x = new Two();
x->f1();
```
One::f3
One::f1

(ii)
```
One *x = new Two();
x->f3();
```
One::f3

(iii)
```
Two *x = new Two();
x->f3();
```
Two::f4
Two::f3

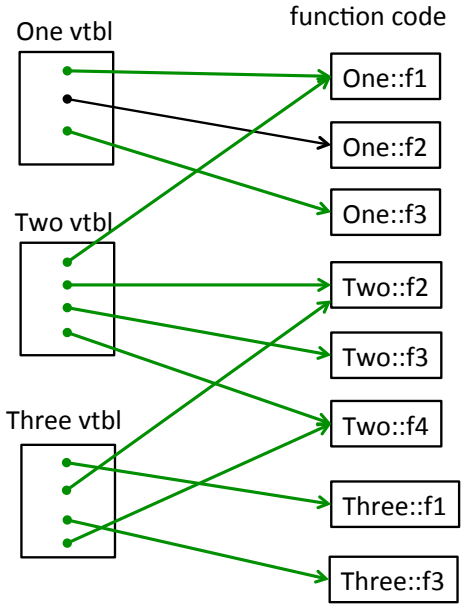
(iv)
```
One *x = new Three();
x->f4();
```
Compile error: no f4 in class One

(v)
```
Three *x = new Three();
x->f3();
```
One::f3
One::f1
Two::f2
Three::f3

(continued on next page)

Question 3. (cont.) (c) (6 points) Now, assume that we change the original code by writing the keyword `virtual` in front of every function declared in classes `One`, `Two`, and `Three`. Complete the vtable diagram below to show the vtable layouts for the three classes with this change. As before, be sure the order of pointers in each vtable is clear.

The vtable pointers in One can appear in any order. The pointers in the derived classes must be in the same order, followed by the new pointer to f4, which is added in class Two.



(d) (10 points) As before, show the output produced after changing all of the member functions in the classes to be `virtual` and then replacing the empty box in `main` with each of the following sequences of code. Either write the output that is produced when the program is executed, or, if an error occurs, give a concise description of the problem.

(i)
```
One *x = new Two();
x->f1();
```
Two::f4
Two::f3
One::f1

(ii)
```
One *x = new Two();
x->f3();
```
Two::f4
Two::f3

(iii)
```
Two *x = new Two();
x->f3();
```
Two::f4
Two::f3

(iv)
```
One *x = new Three();
x->f4();
```
Compile error: no f4 in class One

(v)
```
Three *x = new Three();
x->f3();
```
Three::f1
Two::f2
Three::f3

Question 4. (18 points) Too many things at once. Consider the following small program that uses pthreads. (Header files omitted to save space. This does compile and execute.)

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

int x = 0;
int y = 0;

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

int main() {
  pthread_t t1, t2, t3;
  int ignore;
  ignore = pthread_create(&t1, NULL, &worker, NULL);
  ignore = pthread_create(&t2, NULL, &worker, NULL);
  ignore = pthread_create(&t3, NULL, &worker, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  pthread_join(t3, NULL);
  printf("final x = %d, y = %d\n", x, y);
  return 0;
}
```

When we run this program it starts three threads and waits for them all to finish, and then prints the final value of the variables `x` and `y`.

(a) (6 points) What output would this program print if the threads do not interfere with each other (i.e., if, for example, the three threads were executed sequentially, one after the other, rather than running concurrently):

The simplest answer is to run the threads one after the other. There are other possible non-interfering sequences, of course.

x = 1, y = 1

x = 2, y = 3

x = 3, y = 6

final x = 3, y = 6

(continued on next page)

Question 4. (cont.) (b) (6 points) When the threads run concurrently, is it possible to get different output when the program is executed repeatedly? If it is, give three possible outputs that could be produced by the program. If there is only one or two possible outputs, write those and indicate that they are the only possible results.

(You should assume that the statements in each individual thread are executed in the order written, and not rearranged by the compiler or memory system to be executed out-of-order. You should also assume that the `printf` calls don't interfere with each other and that each line of output is printed correctly and separately from other output lines. If different executions lead to different outputs it is only because of the interaction between the threads as they run concurrently.)

One possibility is the sequential solution in the answer to the previous part:

```
x = 1, y = 1
x = 2, y = 3
x = 3, y = 6
final x = 3, y = 6
```

A few others:

```
x = 1, y = 1
x = 1, y = 1
x = 1, y = 1
final x = 1, y = 1
```

```
x = 1, y = 1
x = 3, y = 6
x = 2, y = 3
final x = 3, y = 6
```

```
x = 3, y = 3
x = 3, y = 6
x = 3, y = 9
final x = 3, y = 9
```

There are many more, either because the print statements run in different orders, or execution switches from one thread to another while a thread is in the middle of reading and updating a variable, etc.

(c) (6 points) Assuming that the threads are executed concurrently, as in part (b), what are the possible final values of variables x and y? Circle all that could possibly happen on some possible execution:

Possible final values for x:

0 (1 2 3) 4 5 6 7 8 9 10 or more

Possible final values for y:

0 (1 2 3 4 5 6 7 8 9) 10 or more

Some shorter questions…

Question 5. (6 points) Two ways to implement a concurrent server are to use threads and processes (fork).

(a) Give one distinct technical advantage of processes compared to threads.

The action of one process cannot interfere with another data belonging to another process since each process has its own private memory. (However, note that processes *can* share other state like open file descriptors when one process forks another, so they are not completely isolated.

(b) Give one distinct technical advantage of threads compared to processes.

Here are two basic answers:

Easier to share data among concurrent threads using shared memory

Cheaper to create than processes, and cheaper to switch from one thread to another in the same process during execution.

Question 6. (6 points) TCP and UDP are two transport-level protocols. TCP implements streams while UDP sends "datagrams" – i.e., messages like those sent by IP except that UDP allows messages of any length, which it splits into packets to send via IP. For each protocol, give an example of one user-level application that would be better suited to that protocol than to the other one. (Think about apps that run on your computer or phone.)

TCP:

Any application that would benefit from a reliable byte stream: web servers (http), email, file transfer, messaging/chat, etc.

UDP:

Applications where occasional packet loss is preferable to delays caused by retransmission of lost or delayed packets. Examples include live video or other media, skype, realtime online gaming audio/video (but not scores or other data must be updated without errors), etc.

More esoteric is complex applications where we are going to implement our own error-recovery and reliability protocols and do not want to incur the redundant overhead of having TCP also perform these functions.

Question 7. (6 points) C++ allows functions in classes to be either virtual, using dynamic dispatching to select methods (functions) at runtime, or non-virtual, which does not use the dynamic dispatch mechanism. In Java, all method calls are virtual and there are no non-virtual methods associated with objects. Give two technical reasons why it is sometimes useful in C++ programs to use non-virtual methods, even though this is rare:

(i)

Most useful: a class might want to guarantee that when one of its functions `f()` calls another of its functions `g()`, it will call the version of `g()` in its own class and not a different version, possibly added later, in some subclass. (Particularly common when designing class frameworks.)

(ii)

Less common: avoid the slight overhead of function calls involving indirect pointers through vtables.

One other uncommon reason is to avoid the presence of vtable pointers in objects to match the layout of non-C++ data structures defined in languages like C.

Question 8. (6 points) Describe one fundamental technical difference between a C++ reference and a C++ pointer (other than the obvious ones of the syntax used to declare or initialize them).

A few possibilities:

- **A pointer is an ordinary variable that can be re-assigned to contain the address of a different location in memory. A reference is an alias for a specific variable and cannot be re-bound to a different variable once it has been initialized.**
- **Pointers can be "dangling" if the data they point to is deallocated; it is much more difficult to create a "dangling" reference (although not impossible)**
- **Reference parameters can be used by overloaded operators like `operator+` to allow the operator to read the parameter data without having to make a copy. The same sort of syntactic overloading is not really possible using pointer parameters.**
- **Similar to the previous reason, to allow stream input functions to overload `operator>>` and store data in variables without having to use explicit pointers as parameters.**

Question 9. (6 points) In a C++ class, assignment (`operator=`) and the copy constructor are two distinct functions, even though both of them normally assign a new value to an object. What is the fundamental reason that these two operations are separate rather than just having a single "assignment/initialization" operation for C++ classes? i.e., what is the fundamental difference between assignment and copy construction?

Copy construction initializes a new, uninitialized collection of bytes that does not have a previously well-defined or initialized state.

Assignment replaces the existing value of an object with a different value. (This might, for example, involve deallocating existing resources like arrays to replace them with new data, but that is only one example of the kinds of differences potentially needed when we update existing variables instead of initializing newly-created ones.)

Question 10. (6 points) Why are there `weak_ptrs`? The main C++ smart pointers are `unique_ptr` and `shared_ptr`, which should be enough to handle most needs. But there is this third kind, `weak_ptr`. What is the primary reason it exists?

`weak_ptrs` allow us to build data structures that contain cycles and yet can still be deallocated by the reference counting strategy used by associated `shared_ptrs`. A `weak_ptr` can be used to break a cycle that would otherwise exist in a data structure using only `shared_ptrs`. If needed, the `weak_ptr` value can be converted to a `shared_ptr` if the referenced object still exists, and that `shared_ptr` value can then be used to access the object the `weak_ptr` was referencing.

Question 11. (6 points) Two of the functions used by a network server to establish connections with client programs are `accept()` and `listen()`. What do these two functions do and in which order should server code execute them?

A server must use `listen()` first to indicate that a socket is prepared to accept client connections on a particular port and IP accress.

After the listening socket is active, the server can use `accept()` to establish a connection to a client via a new file descriptor returned by `accept()`.

Question 12. (3 free points – all answers get the free points)

Who is the most awesome Star Wars character? Feel free to give your reason(s) and/or draw a picture to explain your choice.

Complicated this questions is, and difficult to answer….

☺

Congratulations on lots of great work this fall !!
Have a great holiday break and say hello when you get back !
The CSE 333 staff