EECS/ 381
Sample Exam Questions

These are questions from previous midterm and final exams.They are provided only as a study aid to illustrate the style of questions likely (but not guaranteed) to be on the exams. The topics mentioned in these questions may or may not have been covered by the time of the exam.

Do not ask us for answers to these questions! Their primary purpose is illustrate the kind of questions; they are not a study guide. If the question is about the covered content, work out the answer yourself for maximum benefit.

Very Short Answer Questions

Answer each question in a 2-3 sentences or the specified number of lines of code. Think the answer through before writing. Make sure you are actually answering the stated questions - no credit will be given for "memory dumps" that don't answer the question. If you have to write much at all, you've probably misunderstood the question or haven't figured out the desired answer. Stop and rethink it!

#1. The following function is supposed to allocate enough memory to hold a copy of a supplied C-string and return a pointer to the copy. What does the code actually do? How will it behave?

```
char * make_string(char * s)
{
    char * p = malloc(sizeof(strlen(s) + 1));
    strcpy(p, s);
    return p;
}
```

#2. In 2-3 sentences or phrases, write out the policy recommended in this course for what to put into .h and .cpp files so that it is possible to use functions, objects, and classes that are in the std namespace (e.g. cout and list)

#3. When is the copy constructor for a class called? A few phrases are enough.

Multiple Don't-Guess Questions

Draw a circle around *all* items labeled (a), (b), etc. that are correct about the described situation as presented in this course. There may be more than one correct answer! Each correctly circled one is worth +1 point, each incorrectly circled one is worth -1 point, so don't guess! If you are not *positive* that it is true, leave the item uncircled!

#1. Suppose you have a function prototype int foo(char *), and call it from main(), but forget to write the code that actually implements foo - but everything else in your program is correct. When will the error be detected, and what kind of error will it be?
(a). The compiler will report an "undefined function body" error.
(b). There will be an undefined error at run-time, because the compiler will generate code that branches to an undefined address; anything could happen, so it is undefined.
(c). Some very confusing compiler messages will result because the compiler will try to find all possible overloads of functions that take a char * argument, of which there are many, especially if the iostream header has been #included.
(d). The linker will report an "undefined symbol" error for "foo" and no executable will be built.

#2. On most machines, addresses have the identical binary format, no matter what kind of object they are the address of. Why then, is it necessary to declare a pointer variable with the type of the pointed-to object?
(a). It is a matter of readable programming style, but not required by the Standard.
(b). If the pointer is used to access members of a class or a struct, the compiler must be able to tell what class or struct is involved.
(c). If the pointer is modified using pointer arithmetic, the size of the pointed-to object must be known.
(d). The premise of the question is false; pointers of different types usually have different binary representations in the computer hardware.

Matching:

Fill in each blank with the single ID number from the item in the word and phrase list below that provides the best match. Each item should be used no more than once, and only one item may be put in a blank.

_____ auto_ptr approach to memory management
_____ encapsulation in C
_____ const
_____ const char * p;
_____ implements garbage collection
_____ function object
_____ dynamic_cast
_____ static member variable
_____ expands when filled at end
_____ delete this;
_____ Standard Library iterators
_____ string class object
_____ associative array implementation
_____ best done by object initialization
_____ char * const p
_____ C version of public access
_____ serious number-crunching only
_____ static_cast
_____ templates
_____ bad_alloc()
_____ vector, list, deque, map, set
_____ usually pointless in parameter list
_____ actually nonsensical in parameter list

Word and phrase list

1. allocation of resources
2. will not overflow on input
3. external function prototypes in header file
4. container classes
5. overloads operator()
6. instead of C cast
7. map form of binary tree
8. belongs to whole class
9. owns allocated object
10. overloads operator++
11. pointer to unalterable characters
12. safe way to downcast
13. internal linkage
14. Smart Pointer
15. protect me from myself
16. support generic programming
17. unalterable pointer to characters
18. Thing& const
19. suicidal object
20. valarray template
21. vector template
22. Thing const
23. Standard result of failed new

Problem 1. 30 points. To give you plenty of room to write, this question takes up the next two pages in addition to this one. Take a look now to see how this question works. The class declarations below, in a column labeled CODE, are the beginning of a program that continues with through the CODE column on the next two pages. In the right-hand column on the next two pages (labeled OUTPUT) is space for you to write the output produced by the program. To help you keep your place, lines of output starting with "***" are already shown, and they are lined up with the code statements that produce them. You will write in the remaining lines

of output between the ones already shown. If necessary to make your answer clear, draw an arrow showing which code statement produces which line(s) of output.

So study the class declarations and function definition below, then start on the next page, and write the rest of the output produced by the program in the OUTPUT column. The first constructor output is shown as an example.

The scoring will look at what you wrote between each pair of "***" lines. Lines containing wrong words, missing lines, extra lines, or lines out of order between "***" lines will be counted as incorrect. The number of incorrect lines will be subtracted from the number of correct lines, so guessing is not recommended.

CODE

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

class A {
public:
   A() {cout << "A ctor" << endl;}
   A(A& a) {cout << "A copy ctor" << endl;}
   virtual ~A() {cout << "A dtor" << endl;}
   virtual void f() {cout << "A f" << endl;}
};

class B : public A {
public:
   B() {cout << "B ctor" << endl;}
   ~B() {cout << "B dtor" << endl;}
   void f() {cout << "B f" << endl;}
};

class C : public B {
public:
   C() {cout << "C ctor" << endl;}
   ~C() {cout << "C dtor" << endl;}
   void f() {cout << "C f" << endl;}
};

class X {
public:
   X() {cout << "X ctor" << endl;}
   X(X& x) {cout << "X copy ctor" << endl;}
   ~X() {cout << "X dtor" << endl; }
   void f() {cout << "X f" << endl; }
 };

X foo(A& a, X x)
{
   cout << "*** in foo" << endl;
   A aa;
   aa = a;
   X xx(x);
   cout << "*** leaving foo" << endl;
   return xx;
}
```

CODE

```
int main() {
   cout << "*** starting main" << endl;
   A * a_ptr_to_a = new A;
   B * b_ptr_to_b = new B;
   C * c_ptr_to_c = new C;
   X x1;
   X * x_ptr_to_x = &x1;
   A * a_ptr_to_b = b_ptr_to_b;
   A * a_ptr_to_c = c_ptr_to_c;
   B * b_ptr_to_c = c_ptr_to_c;
```

OUTPUT

```
cout << "*** calls of f #1:" << endl;
a_ptr_to_a -> f();
b_ptr_to_b -> f();
c_ptr_to_c -> f();
x_ptr_to_x -> f();
```

```
cout << "*** calls of f #2:" << endl;
a_ptr_to_b -> f();
a_ptr_to_c -> f();
b_ptr_to_c -> f();
X x2;
```

```
cout << "*** calling foo" << endl;
x2 = foo( *a_ptr_to_a, x1 );
```

```
*** starting main
A ctor
*** calls of f #1:
*** calls of f #2:
*** calling foo
*** in foo
```

CODE	OUTPUT
	*** leaving foo
cout << "*** back in main" << endl; cout << "*** delete a_ptr_to_a:" << endl; delete a_ptr_to_a;	*** back in main *** delete a_ptr_to_a:
cout << "*** delete b_ptr_to_b:" << endl; delete b_ptr_to_b;	*** delete b_ptr_to_b:
cout << "*** delete b_ptr_to_c:" << endl; delete b_ptr_to_c;	*** delete b_ptr_to_c:
cout << "*** leaving main" << endl; return 0; }	*** leaving main

Problem 4. 31 points. True/false questions. **Circle** 'T' or 'F' based on the material presented in this course. The number of wrong answers will be subtracted from the number of right answers, so guessing is a bad idea.

T F Exceptions can be difficult to use correctly because when the stack is unwound, no destructors for local objects are executed; this results in memory leaks.

T F Encapsulation refers to enforcing a clear distinction between the internal implementation of a module and how it looks to the remainder of the program.

T F Using internal linkage in C is roughly equivalent to declaring "private" in C++.

T F When a library header file like <cmath> is included with #include in a source file, the definitions of all of the functions in the library become part of the translation unit for the source file.

T F If s is a C-string, the statement "strcpy(s+i, s);" may crash your program if i > strlen(s).

T F To invoke a virtual function for an object, your code must use a pointer of the same type as the type of the object when it was originally declared.

T F A linked list may require more memory than an array for the equivalent task.

T F Using a linked list will always produce faster performance than using an array.

T F To comply with the C++ Standard, if a member function is defined inside a class declaration, the compiler is required to generate code for it as an inline function.

T F Downcasting, casting a base-class pointer to a derived-class pointer, should be done with RTTI.

T F If a function was declared as " virtual void foo() = 0;" then it is not necessary to provide any definition of "foo" in the program.

T F C++ allows both object-oriented programming, and non-object-oriented programming.

T F The need to support generic programming is primarily an issue in languages that use dynamic typing.

T F Iterators allow the programmer to indicate a place in a container without having to know how the container is implemented.

T F Only one try/catch block is allowed in a program.

T F A binary tree data structure can be searched in logarithmic time like an ordered array data structure, but can usually be more quickly expanded than an array data structure.

T F The Standard Library algorithm templates are based on a common interface for iterators.

T F Static typing refers to assigning and enforcing variable types during compilation.

T F The ordered containers in the Standard Library rely on programmer-defined function pointers.

T F An important use of void * in C is to work around a lack of templates.

T F If you forget to define a function in your project, the compiler will produce an error message.

T F In a Standard implementation of C++, a program will fail safely if dynamic memory allocation fails.

T F A function object class overloads operator().

T F When compiling a .cpp file in a project with multiple source files, the C++ compiler compares the calls of functions with the actual function definitions in the relevant other .cpp file, in order to be sure that function arguments and parameters agree.

T F The "ownership" concept of a managed pointer involves keeping a count of the number of objects created.

T F Inheritance refers to how identical code can produce different effects depending on the actual type of the object being processed.

T F Managed pointers should not be used if exceptions might be thrown.

T F If your class requires a destructor to deallocate some resource, then it almost certainly needs a copy constructor and operator= definition as well.

T F The declaration "const Thing * p;" in a function means that no statement in that function can modify the Thing object that p points to.

T F Polymorphism refers to how structure and behavior can be shared between similar kinds of objects.

T F The function parameter declarations "Thing & const t" and "const Thing & t" are equivalent in effect.