Name: _____

CS 211
Spring 2004
Midterm

1) Using classic compact "what, me worry?" C style, define the C string function `strcat( s1, s2 )` which copies the characters in `s2` onto the end of `s1`. For example, if `s1 = "abc"` and `s2 = "def"` then afterwards `s1 = "abcdef"`. (5 points)

```
char * strcat( char *s1, char *s2 )
{
  char *r = s1;
  while (*s1) s++;
  while (*s1++ = *s2++);
  return r;
}
```

Comment: strtcat() returns the concatenated string, not void.

Comment: Need to save start of s1 so we can return it. Or we could increment r and return s1.

Comment: Skip to the end of s1.

Comment: Same loop as strcpy(). No [] or separate index variable needed. That's the point (in C) or pointers.

Several people wrote s1[i] = s2[i++]. This is wrong. Depending on the compiler, either the left or right side might be evaluated first, leading to different results.

2) Using reference parameters, not explicit pointers, define a function `swap` to swap two integers. Show how `swap` would be called to exchange a[i] and a[j]. (3 points)

```
void swap( int &x, int &y )
{
  int temp = x;
  x = y;
  y = temp;
}
```

Comment: No dererference operators. No need for a separate assignment statement to set temp. Only one temp var needed (some people used two).

Sample call: `swap( a[i], a[j] );`

Comment: No address-of operators.

3) The following C string code is very broken. Circle every mistake, and describe what's wrong and what might happen if this code is run. Be specific! Don't bother fixing. (5 points)

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

#include <string.h>
using std::strcat;

char * getResponse();

int main()
{
      char *response = getResponse();
      cout << response << endl;
}

char * getResponse()
{
      char name[20];
      char response[40];
      cout << "Enter name: ";
      cin >> name;
      strcat( response, "Hello, " );
      strcat( response, name );
      return response;
}
```

Comment: <cstring>. <string> is not correct – that defines the C++ string class.

Comment: input could overflow name buffer

Comment: response never initialized to empty string. To fix, either put '\0' in response[0] or use strcpy().

Comment: Name could easily overflow response buffer

Comment: returns a pointer to a local variable, which will be garbage after the function exits.

The buffer overflows mean that a hacker could attack a machine running this code.

Fortunately (!), not initializing response and returning a pointer to it means this program will probably crash with a segmentation fault before that happens.

Many people thought `cout << response` was wrong in some way. It's fine.

It's also legal (but bad style) to omit `return 0` in `main()`.

Almost everyone missed returning a pointer to a local variable. It's very important to know why this is a major error. It can be fixed (sort of) by making response a static variable, but that leads to string sharing that is almost always a bad idea.

4) Deitel gives this algorithm for shuffling an array of N cards: Make an array A of N empty slots. For each of the N cards, choose a slot in A at random until you find an empty one, and put the card there.

   Explain why this algorithm is incredibly inefficient. Be specific. Use an example (3 points).

   As the slots get filled up, it will spend more and more time generating random numbers that access filled slots. At the end, for example, it will keep guessing until it randomly finds the one free slot. This gets worse and worse as N gets larger. For example, with a million cards, at the end it would be randomly trying to find 1 slot out of a million.

5) On the following pages, implement the classes `Book` and `BookList` so that code like the following will work. The `Book` constructor should default both title and author to empty strings, and store them as instances of `string` internally. The `BookList` constructor should take any integer N, and internally create an array of `Book`'s. Follow best practices, e.g., make everything `const` that can be, minimize the amount of implementation code in the header file, and so on.

```
#include <iostream>
using std::cout;
using std::endl;
using std::ostream;

#include "booklist.h"

int main()
{
	BookList books(2);
	books.setBook(0, Book("C++", "Deitel"));
	books.setBook(1, Book("ICBR", "Riesbeck"));
	for (int i = 0; i < books.size(); ++i)
	{
		cout << i << ": " << books.getBook(i)<< endl;
	}
	return 0;
}
```

prints

```
0: C++ by Deitel
1: ICBR by Riesbeck
```

a) Write a single complete header file, `booklist.h`, for `Book` and `BookList`, with `#include`'s, `using` declarations, and header guard. (15 points)

```
#ifndef BOOKLIST_H
#define BOOKLIST_H

#include <iostream>
using std::ostream;

#include <string>
using std::string;

class Book {
public:
  Book(const string &t = "", const string &a = "");
  string getTitle() const;
  string getAuthor() const;
private:
  string myTitle;
  string myAuthor;
};

class BookList {
public:
  BookList( int n );
  ~BookList();
  int size() const;
  Book getBook( int i ) const;
  void setBook( int i, const Book &b );

private:
  int mySize;
  Book *myBooks;
};

ostream &operator<< (ostream &out, const Book &b);

#endif
```

Comment: header guard

Comment: ostream needed for operator<< declaration, but cout, cin, etc are NOT needed

Comment: string needed for title and author variables

Comment: Default parameters must be specified in the header, so the compiler knows. This also gives us the default constructor needed for the new[] call in BookList::BookList().

Comment: const member functions. Omitting these means operator<< has to be a friend and makes Book less useful.

Comment: setAuthor() and setTitle() OK but not needed.

Comment: We need a destructor to deallocate the memory allocated by the constructor.

Comment: string is NOT a good return type. It avoids needing to overload operator<< but makes getBook() useless for getting actual Book data.

Comment: more const member functions. It's not necessary to const primitive types like int.

Comment: This will be an array but we don't know the size until construction.

Comment: ostream reference returned

Comment: operators must be declared in header so compiler knows about them. Does NOT need to be a friend if it uses public accessors. Takes const Book reference. It was OK to omit getTitle() and getAuthor() and make operator<< a friend, but in any real code you'd need these accessors anyway.

Getting the header file correct is harder than the code!

Many people were pretty bad about using const and const references.

It is very bad to say `using namespace std;` in a header file. This would cause any code using the header to get EVERY name in std, which could easily cause name conflicts. It's OK, but not recommended, in .cpp files.

Some people made BookList::size static, which isn't appropriate at all. Each BookList is a different size.

b) Write the implementation file `booklist.cpp` for `Book` and `BookList`. Don't forget to overload `operator<<`. (10 points)

```
#include <iostream>
using std::ostream;

#include <string>
using std::string;

#include "booklist.h"

Book::Book(const string &t, const string &a)
 : myTitle(t), myAuthor(a)
{}

string Book::getTitle() const { return myTitle; }

string Book::getAuthor() const { return myAuthor; }

ostream &operator<< (ostream &out, const Book &b)
{
      return out << b.getTitle() << " by " << b.getAuthor();
}

BookList::BookList( int n )
 : mySize(n), myBooks(new Book[n])
{}

BookList::~BookList() { delete[] myBooks; }

int BookList::size() const { return mySize; }

Book BookList::getBook( int i ) const { return myBooks[i]; }

void BookList::setBook( int i, const Book &b )
{ myBooks[i] = b; }
```

Comment: no need for cout, cin, or endl since we don't use them here.

Comment: It can be argued that you know that these headers are already included by booklist.h because the function signatures use ostream and string. But it's clearer to be explicit.

Comment: Initialization lists used.

Comment: Legal since << returns the stream.

Comment: Initialization lists again.

Comment: Critical to use delete[] not delete.

Comment: Checking the validity of i would be good.

Some people tried to allocate the Book array in a loop like this:

```
for (int i = 0; i < n; ++i) {
  books[i] = new Book;
}
```

This does NOT work because `books` has not been allocated any space so `books[i]` is accessing some random location in memory.

It was not OK to make an array of some fixed maximum size.