CSE 333 – Autumn 2013
Midterm

Please do not read beyond this cover page until told to start.

A question involving what could be either C or C++ is about C, unless it explicitly states that it is about C++. Code is considered to have compiled if no errors are raised; the code may generate warnings, but still compile.

We don't think the answer to any individual question needs to be very long. *Make sure you have at least read all the questions before spending too much time on any one.*

Name: _____

1. *[3 points]*
This code sometimes runs and produces a correct result and sometimes experiences a fatal runtime error.

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

int sum(int n) {
      if ( n <= 1 ) return 1;
      return n + sum(n-1);
}

int main(int argc, char *argv[]) {
      if ( argc != 2 ) exit(1);
      int n = atoi(argv[1]);
      printf("Sum(0..n) = %d\n", sum(n));
      return EXIT_SUCCESS;
}
```

Briefly explain exactly why some executions experience fatal errors.

If the command line argument is sufficiently large the recursion will be deep enough that it overflows the available stack space.
Note*: Many people answered that atoi will crash if its argument isn't a string representation of an integer. It doesn't (and I don't think you should expect it would – crashing is never a good option).*
Note: *Many people answered that the result is wrong if the argument is zero or less. That may be, but it's not a runtime error, it's just a bug (if that).*

2. *[3 points]*
Suppose you run the following code through the preprocessor when symbol _MY_H_ **is** defined:

```
#ifndef _MY_H_
int gInt = -2;
#include "B.h"
#endif // _MY_H_
```

We know that the preprocessor will drop all source lines between the `#ifndef` and its `#endif`, and so will not emit the line defining `gInt`. Although we haven't talked about it directly, we can also be certain that the preprocessor will not execute the `#include` directive, that is, will ignore preprocessor directives as well. Briefly explain how you can deduce, with certainty, that the preprocessor will not include `B.h` in the code above based on things you're already sure of.
I think a total of two people understood this question the way I intended it, so it wasn't graded as part of the midterm. It was possible to get some bonus points on it, but the entire exam is graded as though this question weren't here.
What I intended was that this be a question about ifdef guards. We know they work. We know they prevent double delcarations, but that's because they omit source lines. How do we know the

#include is not executed by the preprocessor? Because if it were some ifdef guarded .h files would put the preprocessor into a loop, and they don't – that's another of the purposes of the ifdef guards. If they don't, the preprocessor must be ignoring commands to itself whenever its also eliding source file lines.

3. *[8 points]*
   Given the following informal description of a function, design and then give C code for an implementation. Note that our description does not intend to specify the C interface – you should decide on the actual types used in implementing the function.
   The function takes two strings as inputs, and returns two values: a string that is the longest prefix of both input strings, and the length of that prefix. For example, if the two input strings are "lark" and "large", the function returns "lar" and 3. If the two strings are "abc" and "def" it returns "" and 0. The function has some way of indicating that an error has occurred, and so that it is unable to provide a result.
   Give the function's implementation here.

```
char * lcp(const char* s1, const char* s2, int *length) {
   char *result = (char*)malloc(strlen(s1) + 1);
   if ( result == NULL ) return NULL;
   for ( length=0;
         s1[length] && s1[*length] == s2[*length];
         (*length)++ ) {
     p[*length] = s1[*length];
   }
   result[*length] = '\0';
   return result;
}
```

   Many people typedef'ed a struct containing a char and an int, and returned one. That's a reasonable solution as well,.*

4. *[3 points]*
   Below is the complete contents of C source file q4.c. Circle all (and only) the lines of code that cause **compile time errors** when the file is compiled.

```
int main(int argc, char *argv[]) {
      char a[] = "initial string";
      char *b;
      a = "test";
      b = "test";
      *a = 'a';
      *b = 'b';
      a[4] = 'a';
      b[4] = 'b';
      return 0;
   }
```

   An array name cannot be assigned to. All the other statements compile without warning, although many of them are runtime errors.

*A string literal is a char *. A character array can be initialized from a string literal. String literals follow C string conventions, so are null terminated. Array notation is shorthand for pointer arithmetic and dereferencing (e.g., b[4] is the same as *(b+4)), so can be used even on char* variables.*

5. *[3 points]*
This code builds without warning but experiences a run time error.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void delete(void **p) {
  free(*p);
  *p = NULL;
}
int main(int argc, char *argv[]) {
  void *p = malloc(1024);
  strcpy(p, "literal string");
  printf("Memory contents = '%s'\n", (const char*)p);
  delete(p);  delete(&p);
  return EXIT_SUCCESS;
}
```

Fix the bug, either by describing the fix here or annotating the code above. DO NOT change the obvious intent of the code, just make what it is trying to do work.

The delete() function intends to both free the memory pointed at by some pointer and set the pointer to NULL. That requires that the argument be a pointer to the pointer. The call incorrectly passes the pointer to the memory to be freed.

6. *[3 points]*
This code seg faults. Briefly explain why.

```
#include <stdio.h>
#include <string.h>
#define LEN 4
char *truncate(char *str, unsigned int len) {
    if ( strlen(str) <= len ) return str;
    str[len] = '\0';
    return str;
}
int main(int argc, char *argv[]) {
  char *originalStr = "long string";
  printf("%s truncated to length %d is ", originalStr, LEN);
  printf("%s\n", truncate(originalStr,LEN));
  return 0;
}
```

The truncate() method tries to modify the memory pointed at by its str parameter. But in this code that parameter is the address of a string literal. String literals are stored in read-only memory, so the attempt to modify it (in the line str[len] = '\0';) results in a seg fault.

7. *[5 points]*
I'm considering building an application in C. As part of my design I'd like to implement a fixed capacity stack to hold elements of different types. I'd like my stack implementation to be generic – it can hold data of many different types, including some I haven't thought of yet. Finally, for debugging, I'd like to be able to print the contents of the stack, as part of the stack's functionality.
Briefly describe how I can achieve this. Be convincing. What information must be supplied by the client code? On what call(s) is that information supplied? How does my stack implementation keep track of the information it needs?

The original intention of this question was that you explain how a single stack could hold elements of many distinct types, all at the same time, and still print itself. It was graded without requiring the "simultaneous" part of different types, though.
Fully correct answers need to mention two specific things: void as the mechanism that allows the client to provide data of arbitrary type to the stack implementation, and the use of a callback, supplied by the client code, to print the data.*

8. *[4+2 = 6 points]*
Here is some **C++ code**. (It compiles and runs without error.)

```
#include <iostream>
using namespace std;
class Foo {
public:
  Foo(int x) { data = x; }
  Foo(const Foo &other) { data = other.data; }
  int value() const { return data; }
  Foo merge(const Foo second) const;
  Foo &reset();
private:
  int data;
};
Foo Foo::merge(const Foo second) const {
  Foo result(*this);
  result.data += second.data;
  return result;
}
Foo &Foo::reset() {
  data = 0;
  return *this;
}
int main(int argc, char *argv[]) {
  Foo first(2);
  Foo second(first);
  Foo third = first.merge(second);
  Foo fourth = third.reset();
  cout << fourth.value() << endl;
  return 0;
}
```

(A) Circle all the bits of code where it's possible a copy constructor might be invoked. (Be specific. For example, don't just circle all of main!)

There are six instances, highlighted above. (Note that, by default, the compiler will elide two of them).

.

(B) Explain why the following line of code, if inserted into main(), will not compile:

Foo fooArray[10];

For that declaration to work, there must be a default (no argument) constructor available for Foo. But, there is no explicitly provided default constructor and C++ won't synthesize one because there are explicitly provided constructors.

9. *[3 points]*

```
$ cat q9.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main(int argc, char *argv[]) {
  double radians = atof(argv[1]);
  printf("cosine(%lf) = %lf\n", radians, cos(radians));
  return 0;
}
$ gcc -Wall -std=gnu99 q9.c
/tmp/cckWxxhb.o: In function `main':
q9.c:(.text+0x39): undefined reference to `cos'
collect2: error: ld returned 1 exit status
```

What's wrong, and what do I need to do to fix it?

This is a linker error, meaning that the linker isn't finding an object file that defines the symbol cos. The solution is to provide such an object file. In this particular case, we need to tell the compiler to use the math library: gcc -Wall -std=gnu99 q9.c -lm

Note that this has nothing to do with .h files, in the sense that no .h file can fix the problem. .h files serve mainly to declare types, so that the compiler can type-check the code. This isn't a type error; there's a piece of code missing.

10. [4 points]

This code intends to print 0, 1, …, 4:

```
#include <stdio.h>
#include <stdlib.h>
int *createSequence(unsigned int N) {
  int result[N];
  int *result = (int*)malloc(N * sizeof(int));
  if ( result == NULL ) return NULL;
  for (int i=0; i<N; i++) {
    result[i] = i;
  }
  return result;
}
void printIntArray(int array[], int N) {
  for (int i=0; i<N; i++) { printf("%d ", array[i]); }
}
int main(int argc, char *argv[]) {
  int *seq = createSequence(5);
  if ( seq == NULL ) return 1;
  printIntArray(seq, 5);
  free(seq);
  return 0;
```

```
}
```

It builds, but when I run it it instead prints 0 0 0 0 4196100.
Fix the code (by writing on the source shown above). (You must solve all problems with it, so that it meets the correctness criteria we've required of exercise solutions.)

The problem is that createSequence is returning a pointer to a stack allocated variable. The memory it occupied is overwritten when printIntArray is called. The solution is inserted into the code above.