CSE 333 19wi Midterm Exam Feb. 15, 2019

Name _____ UW ID# _____

There are 8 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.

There are two pages of reference information at the end of the exam that may be useful for some of the questions.

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 some 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. _____ / 16
2. _____ / 12
3. _____ / 15
4. _____ / 16
5. _____ / 16
6. _____ / 12
7. _____ / 12
8. _____ / 1

Note: Please write your answers only on the specified pages. Reference pages and pages with only questions and explanations will not be scanned for grading, and you should remove them from the exam.

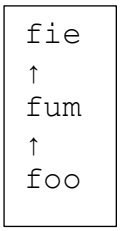
Please write only on the front of each page.

There is an extra blank page after the last question if you need additional space for one or more answers. That page will be graded if it is not blank.

Question 1. (16 points) Build tools and `make`. Suppose we have the following `Makefile` in the current directory.

```
rain.o: rain.h rain.c
     gcc -Wall -g -c rain.c
snow.o: snow.h snow.c rain.h
     gcc -Wall -g -c snow.c
forecast: forecast.o rain.o snow.o
     gcc –Wall –g –o forecast forecast.o rain.o snow.o
forecast.o: forecast.c rain.h snow.h
     gcc -Wall -g -c forecast.c
```

(a) (6 points) In the space below, draw a dependency graph (diagram) showing the dependencies between the files as specified by the above Makefile. Your drawing should have an arrow from each file to the files that it depends on, as in the diagram to the right, where `foo` depends on `fum`, which depends in turn on `fie`. Hint: if the `gcc -c` option is present but no `-o` option is given, the default output file name for `gcc -c xyz.c` is `xyz.o`.



(continued on next page)

Question 1. (cont.) (b) (3 points) If we just type "`make`" in the directory containing this `Makefile`, what happens? Assume that all of the source files (`.c` and `.h`) are present in the directory along with the `Makefile`, but that none of the other files (`.o`, etc.) named in the `Makefile` are present when we run make.

Now suppose we've been working on this project and want to add a new source file `wind.c` along with an associated header file `wind.h`. File `wind.c` only includes the `wind.h` header; it does not use any of the others. Files `forecast.c` and `rain.c` have been modified to use the new functions defined in these new files, with the appropriate `#includes` and calls to the new functions.

(c) (4 points) On the previous page, write in the modifications needed to the `Makefile` so these new files are included in the program and are correctly compiled (and recompiled) as needed when the program is built.

(c) (3 points) Once the new `wind.h` and `wind.c` files have been tested and the `Makefile` updated in our working copy of the program, we need to be sure to update the `git` repository. Write the necessary `git` commands below to update the repository to reflect these additions and changes.

Question 2. (12 points) Preprocessor – more madness than usual. We have the following three C files written by someone who clearly doesn't quite understand things:

foo.h:

```
#ifndef _FOO_H_
#define _FOO_H_
int THE_ANSWER = 42;
#define BAR(x) x * THE_ANSWER
int bar(int);
#endif  // _FOO_H_
```

foo.c:

```
#include "foo.h"
int bar(int x) {
  return x * THE_ANSWER;
}
```

main.c:

```
#include <stdio.h>
#include "foo.c"  // ☜☜ !!!!
int main(int argc,
         char **argv) {
  printf("%d\n", bar(2));
  printf("%d\n", BAR(1 + 1));
  return 0;
}
```

(a) (9 points) Show the result produced by the C/C++ preprocessor when it processes file `main.c` (i.e., if we were compiling this file, what output would the preprocessor send to the C++ compiler that actually translates the program to machine code?). You should ignore the `#include <stdio.h>` directive since that includes library declarations that we do not have access to. Write the rest of the preprocessor output below.

(b) (3 points) What happens when we try to compile and execute `main.c` by itself using `gcc -Wall -g -std=c11 -o main main.c` (i.e., does it compile, and, if so, what happens when it runs, etc.)? If there is an error, explain the problem briefly.

Question 3. (15 points) Nodes in a binary search tree of C strings can be defined as follows:

```
typedef struct bst_node {
  char *str;               // heap-allocated string for this node
  struct bst_node *left;   // left subtree with strings < str
  struct bst_node *right;  // right subtree with strings > str
} BST_Node, *BST_NodePtr;
```

Complete the implementation of `insert` on the next page so that `insert(str,r)` adds a copy of `str` to the BST with root `r` if `str` is not already present in the tree, and returns a pointer to the root of the modified tree. If `str` is already in the tree, `insert` should just return the pointer to the root of the original tree. For example, if we have a BST whose root node is `words`, the following code adds `"pqr"` to this BST if `"pqr"` is not already present.

```
words = insert("pqr", words);
```

Note that the tree (`words` in this example) may be empty (`NULL`) initially. When new strings are added, the code should preserve the binary search tree property that left subtrees have values less than their parent node, and right subtree values are greater. For full credit your answer should only traverse the necessary nodes in the tree.

You may assume that the original tree is properly formed, in particular, each node has a non-`NULL str` pointer and the string it points to is a properly '\0'-terminated array of characters (a C string) allocated on the heap. Also assume that all necessary library header files have already been `#include`d.

Hints: there are pages at the end of the exam with reference information that might be useful. If need to use `malloc` you may assume that it always succeeds and you do not need to check for errors.

Hint: Hint: Hint: recursion is your friend (yes, it really is!), as is your experience with BSTs in CSE143.

Write your answer on the next page and **remove this page from the exam. This page will not be scanned for grading.**

(continued on next page)

Question 3 (cont.) Write your implementation of function `insert` below.

```
// insert string s into the BST with root r if not present
// and return a pointer to the root of the updated tree
BST_NodePtr insert(char *s, BST_NodePtr r) {



}
```

Question 4. (16 points) Memory madness. Consider the following program which, as traditional, does compile and execute successfully.

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

void confuse(int* p, int* q, int n) {
  *q = n-3;
  *p = n+1;
  ////HERE////
  printf("confuse: %d %d %d\n", *p, *q, n);
}

void strange(int* x) {
  int *a = (int*)malloc(sizeof(int));
  *a = 10;
  printf("before: %d %d\n", *x, *a);
  confuse(x, a, *a);
  printf("after: %d %d\n", *x, *a);
  free(a);
}

int main(int argc, char* argv[]) {
  int p;
  p = 2;
  strange(&p);
  printf("done: %d\n", p);
  return 0;
}
```

Answer questions about this program on the next page and **remove this page from the exam. This page will not be scanned for grading.**

(continued on next page)

Question 4. (cont.) (a) (10 points) Draw a boxes 'n arrows diagram showing state of memory when control reaches the comment containing `////HERE////` in the middle of function `confuse`. Your diagram should have three boxes showing the stack frames for functions `main`, `strange`, and `confuse`. The stack frames should show values of all local variables. Draw an arrow from each pointer to the location that it references. Data that is allocated on the heap should be drawn in a separate area, since it is not part of any function stack frame After drawing your diagram, be sure to answer part (b) at the bottom of the page.

(b) (6 points) What output does this program produce when it is executed?

Question 5. (16 points) Consider the following C++ program, which contains a class that represents a 4-digit PIN number for a bank ATM account.

```
#include <iostream>

class PinNumber{
public:
  PinNumber(const int first_digit, const int second_digit,
            const int third_digit, const int fourth_digit) {
    pin_ = new int[4];
    pin_[0] = first_digit;
    pin_[1] = second_digit;
    pin_[2] = third_digit;
    pin_[3] = fourth_digit;
  }
  ~PinNumber() { delete[] pin_; }
  // returns true if the other PinNumber is equal to this one
  bool equals(const PinNumber other) {
    return this->pin_[0] == other.pin_[0]
      && this->pin_[1] == other.pin_[1]
      && this->pin_[2] == other.pin_[2]
      && this->pin_[3] == other.pin_[3];
  }
private:
  int *pin_;
};

int main () {
  PinNumber* pin1 = new PinNumber(1, 2, 3, 4);
  PinNumber pin2(3, 4, 5, 6);
  bool equivalent = pin1->equals(pin2);
  if (equivalent) {
    std::cout << "Access given" << std::endl;
  } else {
    std::cout << "Access denied" << std::endl;
  }
  PinNumber pin3(11, 12, 13, 14);
  pin3 = pin2;
  if (pin2.equals(pin3)) {
    std::cout << "pins match" << std::endl;
  } else {
    std::cout << "something broken" << std::endl;
  }
  delete pin1;
}
```

Please write your answer on the next page and **remove this page from the exam. This page will not be scanned for grading.**

(continued on next page)

Question 5. (cont) (a) (6 points) This program compiles successfully, but when we try to run it there are a bunch of runtime errors. What's wrong? Give a brief, but precise technical description of the problem(s) and the reason(s) for the crash(es).

(b) (10 points) What needs to be done to fix the problem(s)? If it involves adding or changing any code, write any new code below and precisely describe any other changes needed.

Question 6. (12 points) Constructor madness. Consider the following (very unusual) C++ program which does compile and execute successfully. At the bottom of this page, write the output produced when it is executed.

Hints: remember that member variables are initialized in declaration order. Destruction order is the reverse of construction order. The body of a constructor runs after its initializer list. Last, notice that this program consists of a single object declaration and then termination, but that triggers a whole bunch of constructor and destructor calls.

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

class foo {
public:
  foo()                              { cout << "p"; }  // ctor
  foo(int i)                         { cout << "a"; }  // ctor(int)
  foo(int i, int j)                  { cout << "h"; }  // ctor(2int)
  foo(const foo &rhs)                { cout << "o"; }  // cctor
  foo &operator=(const foo &rhs)     { cout << "f"; return *this; }
  ~foo()                             { cout << "s"; }
};
class bar {
public:
  bar(): foo_(new foo())             { cout << "g"; }   // ctor
  bar(int i): foo_(new foo(i))       { cout << "p"; }   // ctor(int)
  bar(const bar &rhs)                { cout << "r"; }   // cctor
  bar &operator=(const bar &rhs)     { cout << "m"; return *this; }
  ~bar()                             { cout << "e"; delete foo_; }
private:
  foo *foo_;
  foo otherfoo_;
};
class baz {
public:
  baz(int a,int b,int c): foo_(b,c), bar_(a)
                                     { cout << "i"; }  // ctor(3int)
  baz(const baz &rhs)                { cout << "d"; }  // cctor
  baz &operator=(const baz &rhs)     { cout << "l"; return *this; }
  ~baz()                             { cout << "n"; }
private:
  foo foo_;
  bar bar_;
};
int main() {
  baz b(1,2,3);
  return 0;
}
```

Output: _____

Question 7. (12 points) A little programming including a function template. We're working on a C++ program and have discovered a function that needs to be implemented. The function is declared as follows in a header file:

```
// Given an array a with *len elements sorted in non-decreasing
// order, insert x into the array and increment *len so that the
// updated array a[0..*len-1] is sorted in non-decreasing order.
// Assume that the array is large enough to hold the new element.
void InsertSorted(int a[], uint32_t *len, int x);
```

So, for example, if an array `nums` contains `{-2,5,7}` and `size` contains 3 (the number of array elements), then calling `InsertSorted(nums, &size, 5);` should insert the new value in the proper place so that `nums` contains `{-2,5,5,7}`and `size` should be increased to 4.

For this problem, implement this function, with the following modification: change the function to be a template so that it will work with array element values of any type, provided that the array elements and the value `x` can all be compared to each other using the < (less than) operator. (i.e., `x` and the array elements will have the same type which could be anything that can be compared with < like `int`, `string`, `double`, etc.). Write your solution below. You can assume that any necessary library files have already been `#include`d. There is more room on the next page if you need it.

(additional room for your answer on the next page if needed)

Question 7. (cont.) Additional space for your answer if needed.

Question 8. (1 free point) (All reasonable answers receive the point. All answers are reasonable as long as there is an answer. ☺)

Draw a picture of something you did to pass the time while UW was closed because of the winter weather the last couple of weeks.

Extra space for answers, if needed. Please be sure to label which question(s) are answered here, and be sure to put a note on the question page so the grader will know to look here.

Reference information. Here is a collection of information that might, or might not, be useful while taking the test. You can remove this page from the exam if you wish.

Please do not write on this page. It will not be scanned for grading.

Memory management (<stdlib.h>)

- void * malloc(size_t size)
- void free(void *ptr)
- void * calloc(size_t number, size_t size)
- void * realloc(void *ptr, size_t size)

Strings and characters (<string.h>, <ctype.h>)

Some of the string library functions:

- char* strncpy(*dest*, *src*, *n*), copies exactly *n* characters from *src* to *dst*, adding '\0's at end if the '\0' at the end of the string *src* is found before *n* chars copied.
- char* strcpy(*dest*, *src*)
- char* strncat(*dest*, *src, n*), Appends the first *n* characters of *src* to *dst*, plus a terminating null-character. If the length of the C string in *src* is less than *n*, only the content up to the terminating null-character is copied.
- char* strcat(*dest*, *src*)
- int strncmp(*string1*, *string2*, *n*), <0, =0, >0 if compare <, =, >
- int strcmp(*string1*, *string2*)
- char* strstr(*string, search_string*)
- int strnlen(*s, max_length*), # characters in *s* not including terminating '\0'
- int strlen(*s*)
- Character tests: isupper(*c*), islower(*c*), isalpha(*c*), isdigit(*c*), isspace(*c*)
- Character conversions: toupper(*c*), tolower(*c*)

Files (<stdio.h>)

Some file functions and information:

- Default streams: stdin, stdout, and stderr.
- FILE* fopen(*filename*, *mode*), modes include "r" and "w"
- char* fgets(*line, max_length, file*), returns NULL if eof or error, otherwise reads up to max-1 characters into buffer, including any \n, and adds a \0 at the end
- size_t fread(buf, 1, count, FILE* f)
- size_t fwrite(buf, 1, count, FILE* f)
- int fprintf(format_string, data…, FILE *f)
- int feof(*file*), returns non-zero if end of *file* has been reached
- int ferror(FILE* f), returns non-zero if the error indicator associated with f is set
- int fputs(*line, file*)
- int fclose(*file*)

A few printf format codes: %d (integer), %c (char), %s (char*)

More reference information, C++ this time. You can also remove this page from the exam. **Please do not write on this page.** It will not be scanned for grading.

C++ strings

If `s` is a string, `s.length()` and `s.size()` return the number of characters in it. Subscripts (`s[i]`) can be used to access individual characters. The usual comparison operators can be used to compare strings.

C++ STL

- If `lst` is a STL `vector`, then `lst.begin()` and `lst.end()` return iterator values of type `vector<...>::iterator`. STL `lists` and `sets` are similar.
- A STL `map` is a collection of `Pair` objects. If p is a `Pair`, then `p.first` and `p.second` denote its two components. If the `Pair` is stored in a `map`, then `p.first` is the key and `p.second` is the associated value.
- If `m` is a `map`, `m.begin()` and `m.end()` return iterator values. For a `map`, these iterators refer to the `Pair` objects in the `map`.
- 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 STL containers (lists, maps, sets, etc.):
  - `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
- Additional operations on `vectors`:
  - `c.push_back(x)` – copy x to end of c
- Some additional operations on `maps`:
  - `m.insert(x)` – add copy of `x` to `m` (a key-value pair for a `map`)
  - `m.count(x)` – number of elements with key `x` in `m` (0 or 1)
  - `m[k]` can be used to access the value associated with key `k`. If `m[k]` is read and has never been accessed before, then a <key,value> `Pair` is added to the map with `k` as the key and with a value created by the default constructor for the value type (0 or `nullptr` for primitive types).
- Some additional operations on `sets`
  - `s.insert(x)` – add `x` to `s` if not already present
  - `s.count(x)` – number of copies of `x` in `s` (0 or 1)
- You may use the C++11 `auto` keyword, C++11-style `for`-loops for iterating through containers, and any other features of standard C++11, but you are not required to do so.