CSE 333 22sp Midterm Exam 5/6/22

Name _____ UW ID# _____
(please print legibly)

There are 6 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. However, you may have a single 5x8 notecard with any hand-written notes you wish on both sides.

There is a blank sheet of paper at the end with extra space for your answers if you need more room. It is after all the questions but before the detachable pages with reference information.

After the extra blank pages for answers, there is a sheet of paper containing assorted reference information (most of which you probably won't need) and copies of the `LinkedList.h` and `HashTable.h` header files from hw1, which should be useful for one of the questions. You can remove those pages from the exam if that is convenient.

Do not remove any pages from the middle of the exam.

If you do not 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. _____ / 20
3. _____ / 20
4. _____ / 20
5. _____ / 20
6. _____ / 4

Question 1. (16 points, 4 each part) Building things. Consider these two C files:

a.c

```
void f(char c);

int main(int argc, char**argv){
  f('x');
  return 0;
}
```

b.c

```
void f(int *p) {
  *p = 17;
}
```

(a) Why is the program made from `a.c` and `b.c` incorrect? What would you expect to happen if it is executed?

(b) Will `gcc -Wall -g -c a.c` and `gcc -Wall -g -c b.c` give an error or will they successfully produce `a.o` and `b.o` without complaint? (Note that `-c` implies that `gcc` will produce an output file named `x.o` from `x.c` if `-o` is not also given.)

(c) Will `gcc -Wall -g -o pgm a.c b.c` give an error or will it successfully produce an executable file `pgm` without complaint?

(d) How would you use standard C coding practices (using an extra file) to avoid the problems with this program (or at least detect them properly)? Give the contents of the extra file below and explain what modifications should be made to `a.c` and `b.c`, if any.

Question 2. (20 points) Making things. The K-Pop group, BTS, is working on building an application for keeping track of their schedule for their upcoming concert. Following good programming practices, they develop the following files:

main.c	schedule.h	schedule.c
`#include "schedule.h"` `...`	`...`	`#include "schedule.h"` `...`

Based on these files, one of the members, Suga, came up with the following commands to build the program based on their understanding of the code:

```
gcc -Wall -std=c17 -g -c main.o main.c schedule.h
gcc -Wall -std=c17 -g -c schedule.o schedule.c schedule.h
gcc -Wall -std=c17 -g -o scheduler main.o schedule.o
```

(a) (6 points) Draw the dependency diagram that these commands attempt to capture (i.e., which files depend on which other files to build the final target program `scheduler`?)

(b) (4 points) What are the technical error(s) in the above `gcc` commands, if any? Or are those commands ok as written?

(continued on next page)

Question 2. (cont.) (c) (10 points) Write the contents of a `Makefile` for this program that has the following properties:

- The command `make` should build the program `scheduler` from the source files.
- The command `make` should only recompile and link files that need to be rebuilt after any changes to source files. Existing `.o` files and other files should not be re-compiled or relinked if that is not needed to bring the program up to date.
- The command `make clean` should delete the executable file `scheduler` and all of the `.o` files in the current directory.

For reference, here is the table of file names from the previous page:

`main.c`	`schedule.h`	`schedule.c`
`#include "schedule.h"` `...`	`...`	`#include "schedule.h"` `...`

Write the `Makefile` code below.

Question 3. (20 points) HW1 linked lists and hash tables revisited. We would like to add a new public function to the HashTables we created in hw1 to retrieve a list of the keys found in the <key,value> pairs in a Hashtable. The specification of this function is:

```
// Return a pointer to a new LinkedList on the heap
// containing all of the keys found in the HashTable table.
// The order in which the keys appear in the resulting list
// is not specified.  The caller is responsible for freeing
// the returned LinkedList when they are finished with it.
LinkedList* HashTable_Keys(HashTable* table);
```

Your code should only use the public interface to the `LinkedList` and `HashTable` modules. Copies of the header files `LinkedList.h` and `HashTable.h` are included at the end of the exam, and you can remove them from the exam if you wish. Your code should allocate any data structures needed, but should not allocate more data than needed and should not have any internal memory leaks. Write your solution below. (Hints: you probably won't need nearly all of this space. The sample solution is about 10-12 lines long, but you don't need to match that. Also, iterators are your friend.)

```
LinkedList* HashTable_Keys(HashTable* table) {




}
```

Question 4. (20 points) Memory madness. Consider the following program which, as is traditional, does compile successfully, but, contrary to tradition, does not work and, in fact, generates a segfault when it is executed. Even if it didn't crash, it appears to leak memory. Header file `#includes` omitted from the code to save space.

```
// Capitalize first letter of word
void CapitalizeWord(char* word) {
  word[0] = toupper(word[0]);     // C library function
}

// Return a new c-string containing a copy of word but with
// each character in word duplicated (i.e., if the input ia "abc"
// return "aabbcc")
char* DoubleLetters(char* word) {
  int str_size = strlen(word) + 1;
  char* result = NULL;
  int i = 0;
  int j = 0;
  char c;
  // >>>draw memory diagram when execution reaches HERE<<<
  while (i < str_size) {
    c = word[i];
    result[j] = c;
    result[j + 1] = c;
    i++;
    j++;
  }
  return result;
}

// main program
int main(int argc, char* argv[]) {
  char cse[] = "cse333";
  printf("1. %s\n", cse);               // expected: "1. cse333"
  CapitalizeWord(cse);
  printf("2. %s\n", cse);               // expected: "2. Cse333"
  char* res_word = DoubleLetters(cse);
  printf("3. %s\n", res_word);     // expected: "3. CCssee333333"
  return EXIT_SUCCESS;
}
```

On the next page, (a) draw a memory diagram showing the situation when execution reaches the line marked "HERE" in `DoubleLetters`. Then answer part (b) giving the corrections needed in the code to fix any problems.

(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 `DoubleLetters`. Your diagram should have boxes showing the stack frames for all active functions. The stack frames should show values of all local variables. Draw an arrow from each pointer to the location that it references. If there is any data that is allocated on the heap, it 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) (10 points) What is wrong with this program and how should it be fixed? You should identify the bugs and write in the corrections needed on the code listing on the previous page. Cross out anything that should be deleted and add any corrected or new code needed. Be sure it is clear where any additional code or corrections should be inserted.

>>> write your corrections on the code on the previous page <<<

Question 5. (20 points) C++ constructors and things. Here is a fairly small C++ class based loosely on code from sections and from one of our C++ programming exercises. Answer questions about this code on the next page.

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

class Int {
public:
  // constructors and destructors
  Int()            {ival_=1; cout<<"default("<<ival_<<")"<<endl;}
  Int(const int n) {ival_=n; cout<<"ctor("<<ival_<<")"<<endl;}
  Int(const Int& n) {
    ival_ = n.ival_;
    cout << "copyctor(" << ival_ << ")" << endl;
  }
  ~Int() { cout << "dtor(" << ival_ << ")" << endl; }

  // assignment operators
  Int & operator=(const Int & other) {
    if (this != &other) {
      ival_ = other.ival_;
    }
    cout << "assign(" << other.ival_ << ")" << endl;
    return *this;
  }

  Int & operator+=(const Int & other) {
    ival_ += other.ival_;
    cout << "op+=(" << ival_ << ")" << endl;
    return *this;
  }

private:
  int ival_;
};

// additional overloaded operator
Int operator+(const Int & x, const Int & y) {
  cout << "op+" << endl;
  Int sum = x;
  sum += y;
  return sum;
}
```

(continued on next page)

Question 5. (cont.) Now suppose we execute this main program that uses the code from the previous page. What output is produced? (It, and the code from the previous page, compiles and executes successfully)

Hints: remember that variables in a function are initialized (constructed) in declaration order. Destruction order is the reverse of construction order.

```
int main(int argc, char* argv[]) {
  Int a;
  Int b(2);
  Int c = b;
  b = a;
  a = 3+b;
  return EXIT_SUCCESS;
}
```

Output:

Question 6. (4 free points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

(a) (2 points) What question were you expecting to appear on this exam that wasn't included?

(b) (2 points) Should we include that question on the final exam? (circle or fill in)

Yes

No

Heck No!!

$!@$^*% No !!!!!

Yes, yes, it *must* be included!!!

No opinion / don't care

None of the above. My answer is _____.

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.

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 may remove this page from the exam if you wish. **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. **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.