CSE 333 24wi Midterm Exam 2/9/24

Name _____ UW Netid: _____@uw.edu
(please print legibly)

There are 7 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`, `HashTable.h` and `HashTable_priv.h` header files from hw1, which should be useful for one of the questions. You should remove these reference pages from the exam and use them during the exam. They will not be scanned for grading, so do not write answers on them.

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. _____ / 18
2. _____ / 16
3. _____ / 18
4. _____ / 20
5. _____ / 20
6. _____ / 6
7. _____ / 2

Question 1. (18 points) Making things. We're working on a new app that recommends recipes based on information about things stored in the fridge. So far, we've got the following files with these #includes to reference declarations in various header files:

main.c	recipe.h	recipe.c

`#include "recipe.h"`
`...`
	`...`	`#include "recipe.h"`
		`#include "fridge.h"`
		`...`

fridge.c	fridge.h
`#include "fridge.h"` `...`	`...`

We've been retyping the following `gcc` commands to build the program:

```
gcc -Wall -g --std=c17 -c fridge.c
gcc -Wall -g --std=c17 -c recipe.c
gcc -Wall -g --std=c17 -c main.c
gcc -Wall -g --std=c17 -o recipe main.o recipe.o fridge.o
```

Hint: recall that if we compile `foo.c` with the `-c` option and do not specify the output file name (no `-o` option), the output file created will be named `foo.o`, as in the `gcc` commands above.

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

(continued on next page)

Question 1. (cont.) (b) (12 points) Write the contents of a `Makefile` for this program that has the following properties:

- The command `make` should build the program `recipe` from the source files using the `gcc` options `-Wall -g --std=c17`.
- 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.
- You do not need to include anything else – no `clean` target or anything not mentioned above.

For reference, here are the tables of file information from the previous page:

main.c	recipe.h	recipe.c

`#include "recipe.h"`
`...`
	`...`	`#include "recipe.h"`
		`#include "fridge.h"`
		`...`

fridge.c	fridge.h
`#include "fridge.h"` `...`	`...`

Write the `Makefile` code below.

Question 2. (16 points) Preprocessor. Suppose we have the following two C source files:

hdr.h

```
#ifndef HDR_H_
#define HDR_H_

#define dbl double
typedef long int longint;
#define IT 42
#define THAT 12 + IT
#define OTHER IT + THING

#endif
```

prepro.c

```
#include "hdr.h"

#define THING 333

int fun(dbl x, longint n) {
  int k = x + THING;
  double y = THAT + IT;
  n += 12;
  return n + OTHER;
}
```

Show the result produced by the C preprocessor when it processes file `prepro.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?)

Question 3. (18 points) HW1 linked lists and hash tables. We would like to add a new public function to the `HashTable`s we created in hw1 to report the total number of buckets in a hash table, and the number of those buckets that contain no entries. The specification of this function is:

```
// Return the total number of buckets in HashTable ht in
// *nbuckets, and return the number of buckets that have no
// elements (are empty) in *nempty.
void HashTable_Bucket_Info(HashTable* ht,
                           int *nbuckets, int *nempty);
```

Give an implementation of this function as it would appear in `HashTable.c`. Your code should only use the public interface to the `LinkedList` module since the `HashTable` implementation is a client of that module. Copies of the header files `LinkedList.h`, `HashTable.h`, and `HashTable_priv.h` are included at the end of the exam, and you should remove them from the exam to use for this question. They will not be scanned for grading. If needed, your code may allocate new data structures while it is executing, but should not allocate more data than needed and should not cause any memory leaks. You may assume that any linked list or hash table functions that you call will succeed, and you do not need to check for errors when you do that.

Write your solution below. (Hints: you probably won't need all of this space. The sample solution is about 8-10 lines long, but you don't need to match that.)

```
void HashTable_Bucket_Info(HashTable* ht,
                           int *nbuckets, int *nempty) {




}
```

Question 4. (20 points) Pointer trickery. Here is a small C program that compiles and executes without errors, except that it leaks memory, which is not relevant for this question. Answer questions about it on the next page.

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

typedef struct pr {
  int x, y;
} Pair;

int sum(Pair p, Pair *q) {
  int val = p.x + p.y;
  q->y = 2;
  ///// draw memory diagram when control reaches here /////
  return val;
}

int alloc(Pair *p, Pair** q) {
  *q = (Pair *) malloc(sizeof(Pair));
  **q = *p;
  p->x = 1;
  int n = sum(*p, *q);
  return n;
}

int main(int argc, char **argv) {
  Pair p = {17, 42};
  Pair *ptr;
  int val;
  val = alloc(&p, &ptr);
  printf("%d %d\n", p.x, p.y);
  printf("%d %d\n", ptr->x, ptr->y);
  printf("%d\n", val);
  return EXIT_SUCCESS;
}
```

(continued on next page)

Question 4. (cont.) (a) (14 points) In the space below, draw a careful diagram showing the state of memory when execution reaches the point in function `sum` labeled "draw memory diagram when control reaches here". Be sure to carefully distinguish between local variables on the stack to the left and values allocated on the heap. Also be sure you clearly show the difference between pointers to structs and struct values that have x and y components. Local variables for each function should be drawn in a separate box for each function labeled with the function name. When you're done, be sure to answer the "what does it print" part of the question at the bottom.

Stack (local variables)	Heap (dynamic storage – malloc)

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

Question 5. (20 points) C++ constructors and things. Here's a small C++ program that compiles and executes without errors (#includes omitted to save space).

```
class Toy {
 public:
  Toy() : kind_("ball")     { cout << "ball" << endl;}
  Toy(string t) : kind_(t)  { cout << kind_ << endl;}
  Toy(const Toy &other) : kind_(other.kind_)
                            { cout << "another " << kind_ << endl;}
  Toy &operator=(const Toy &other) {
    if (this != &other) {
      kind_ = other.kind_;
    }
    cout << "change to " << kind_ << endl;;
    return *this;
  }
  void dbl() { kind_ = kind_ + " " + kind_;
               cout << "double " << kind_ << endl; }
  ~Toy()  { cout << "gone " << kind_ << endl; }
 private:
  string kind_;
};

int main() {
  Toy mine;
  Toy yours("car");
  Toy* scooter = new Toy("scooter");
  mine = *scooter;
  delete scooter;
  Toy thing(yours);
  yours.dbl();
  return EXIT_SUCCESS;
}
```

What output does this program produce when it is executed? (It does compile and run without errors.)

Question 6. (6 points) Recall the small C++ string class from lecture. `Str.h` defines the class as follows:

```
class Str {
 public:
  Str();                // default constructor
  Str(const char *s);   // c-string constructor
  Str(const Str &s);    // copy constructor
  ~Str();               // destructor

  // operations
  int length() const;             // = length of this string
  char * c_str() const;           // = new char* copy of this string
  Str &operator=(const Str &s);   // assignment

  // stream output
  friend std::ostream &operator<<(std::ostream &out, const Str &s);

 private:
  char *st_;  // c-string on heap with data bytes terminated by '\0'
};
```

The assignment operator for this class is a bit complicated, reallocating the heap array that belongs to the `Str` each time we need to update a `Str` object:

```
Str & Str::operator=(const Str &s) {
  if (this == &s) {
    return *this;
  }
  delete [] st_;
  st_ = new char[strlen(s.st_)+1];
  strcpy(st_, s.st_);
  return *this;
}
```

One of our colleagues has suggested that the code would be much more efficient if we simply updated the `st_` instance variable to point to the data array belonging to the other string, which contains the characters we want to assign. In other words, replace the entire body of the `operator=` function with the following two lines:

```
  st_ = s.st_
  return *this;
```

The question is, would this be a good idea and work properly? Answer yes or no and give a precise technical reason for your answer. (Hint: think about the other operations, constructors, and destructor in the class and how they need to use the `st_` variable.)

Question 7. (2 free points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

(a) (1 point) What question were you expecting to appear on this exam that wasn't included?

(b) (1 point) 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, and the operator + can be used to concatenate 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.