CSE 333 22au Midterm Exam 11/4/22

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`, `HashTable.h` and `HashTable_priv.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. 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. _____ / 16
2. _____ / 20
3. _____ / 20
4. _____ / 20
5. _____ / 20
6. _____ / 4

Question 1. (16 points) Preprocessor – Halloween edition. Suppose we have the following two C source files:

hdr.h

```
#ifndef HDR_H
#define HDR_H

#define Trick int
typedef int Treat;

#define MAGIC 42
#define WAND 1 + MAGIC

#endif
```

goblin.c

```
#include "hdr.h"

Trick goblin (Treat n) {
  int k = MAGIC;
  k += n + WAND;
  return k;
}
```

Show the result produced by the C preprocessor when it processes file `goblin.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 2. (20 points) Making things. Anya is the proud owner of an orchard and has written a program to predict the best times to harvest different kinds of fruits. Anya knows C but isn't very knowledgeable about build tools; that's where you come in. The program consists of the following files, with the following #includes to reference declarations in various header files:

main.c	fruit.h	fruit.c

`#include "fruit.h"`
`#include "seasons.h"`
`...`
	`...`	`#include "fruit.h"`
		`...`

seasons.c	seasons.h

`#include "seasons.h"`
`#include "fruit.h"`
`...`
	`...`

Anya has been retyping the following `gcc` commands to build the program

```
gcc -Wall -g -std=c17 -c fruit.c
gcc -Wall -g -std=c17 -c seasons.c
gcc -Wall -g -std=c17 -c main.c
gcc -Wall -g -std=c17 -o schedule main.o fruit.o seasons.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) (8 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 `schedule`?)

(continued on next page)

Question 2. (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 `schedule` 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	fruit.h	fruit.c

`#include "fruit.h"`
`#include "seasons.h"`
`...`
	`...`	`#include "fruit.h"`
		`...`

seasons.c	seasons.h

`#include "seasons.h"`
`#include "fruit.h"`
`...`
	`...`

Write the `Makefile` code below.

Question 3. (20 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 length of the longest bucket list in a `HashTable`. The specification of this function is:

```
// Return the largest size (# of items) of any bucket in the
// HashTable ht.
int HashTable_Max_Bucket_Size(HashTable* ht);
```

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 can remove them from the exam if you wish. 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 have any internal 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 10-12 lines long, but you don't need to match that.)

```
int HashTable_Max_Bucket_Size(HashTable* ht) {




}
```

Question 4. (20 points) Bugs 'R Us – a little debugging. The following C program reads a list of names (lines) from its input file and stores them in a linked list. Once it reaches the end of the file, it frees the linked list and terminates. It does compile successfully and run but, unfortunately, it contains some memory management errors, as shown in the valgrind output on the next page.

On the listing below, indicate where the memory management errors are located and show how to fix them. You should not change the program logic – it does whatever it does. You may assume that every input line has less than 100 characters. You should also ignore other errors such as failing to check for errors from `fopen` and other library functions. Just show changes needed to eliminate the errors reported by valgrind plus any other memory management bugs that might be present. Hint: recall that brief reference information about common C library functions is included at the end of the exam.

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

struct namelist {
  char* name;
  struct namelist* next;
};

struct namelist* create(char* string) {
  struct namelist* newNode =
           (struct namelist*)malloc(sizeof(struct namelist));
  char* s = (char*)malloc(strlen(string));
  strcpy(s,string);
  newNode->name = s;
  newNode->next = NULL;
  return newNode;
}

int main(int argc, char** argv) {
  char name[100];
  FILE* fp = fopen(argv[1],"r");
  struct namelist* list = NULL;
  while(fgets(name,100,fp)) {
    if(list == NULL)
      list = create(name);
    else {
      struct namelist* front = create(name);
      front->next = list;
      list = front;
    }
  }
  free(list);
  return 0;
}
```

(valgrind output on next page)

Question 4. (cont.) Here is the valgrind output produced when we executed the program on the previous page. Indicate the errors and show the corrections needed to fix them by writing on the program code on the previous page.

```
$ valgrind --leak-check=full ./memleak data.txt
==2472== Memcheck, a memory error detector
==2472== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2472== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==2472== Command: ./memleak data.txt
==2472==
==2472== Invalid write of size 1
==2472==    at 0x484A76E: strcpy (vg_replace_strmem.c:553)
==2472==    by 0x4011BA: create (namelist.c:16)
==2472==    by 0x40122B: main (namelist.c:28)
==2472==  Address 0x4a762f6 is 0 bytes after a block of size 6 alloc'd
==2472==    at 0x484486F: malloc (vg_replace_malloc.c:381)
==2472==    by 0x4011A3: create (namelist.c:15)
==2472==    by 0x40122B: main (namelist.c:28)
==2472==
==2472== Invalid write of size 1
==2472==    at 0x484A76E: strcpy (vg_replace_strmem.c:553)
==2472==    by 0x4011BA: create (namelist.c:16)
==2472==    by 0x40123D: main (namelist.c:30)
==2472==  Address 0x4a76396 is 0 bytes after a block of size 6 alloc'd
==2472==    at 0x484486F: malloc (vg_replace_malloc.c:381)
==2472==    by 0x4011A3: create (namelist.c:15)
==2472==    by 0x40123D: main (namelist.c:30)
==2472==
==2472==
==2472== HEAP SUMMARY:
==2472==     in use at exit: 567 bytes in 10 blocks
==2472==   total heap usage: 12 allocs, 2 frees, 4,679 bytes allocated
==2472==
==2472== 5 bytes in 1 blocks are definitely lost in loss record 1 of 7
==2472==    at 0x484486F: malloc (vg_replace_malloc.c:381)
==2472==    by 0x4011A3: create (namelist.c:15)
==2472==    by 0x40123D: main (namelist.c:30)
==2472==
==2472== 90 (16 direct, 74 indirect) bytes in 1 blocks are definitely lost in
loss record 6 of 7
==2472==    at 0x484486F: malloc (vg_replace_malloc.c:381)
==2472==    by 0x40118B: create (namelist.c:14)
==2472==    by 0x40123D: main (namelist.c:30)
==2472==
==2472== LEAK SUMMARY:
==2472==    definitely lost: 21 bytes in 2 blocks
==2472==    indirectly lost: 74 bytes in 7 blocks
==2472==      possibly lost: 0 bytes in 0 blocks
==2472==    still reachable: 472 bytes in 1 blocks
==2472==         suppressed: 0 bytes in 0 blocks
==2472== Reachable blocks (those to which a pointer was found) are not shown.
==2472== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==2472==
==2472== For lists of detected and suppressed errors, rerun with: -s
==2472== ERROR SUMMARY: 7 errors from 4 contexts (suppressed: 0 from 0)
```

Question 5. (20 points) C++ constructors and things. Here is a fairly small C++ class that looks a lot like some of the other small C++ classes we've seen before. Answer questions about this code on the next page.

```
#include <iostream>
#include <cstdlib>
using namespace std;
class Dbl {
public:
  // constructors and destructors
  Dbl() {cval_=0.0; cout<<"default("<<cval_<<")"<<endl;}
  Dbl(double n) {cval_=n; cout<<"ctor("<<cval_<<")"<<endl;}
  Dbl(const Dbl& n) {
    cval_ = n.cval_;
    cout << "copyctor(" << cval_ << ")" << endl;
  }
  ~Dbl() { cout << "dtor(" << cval_ << ")" << endl; }

  // assignment operators
  Dbl& operator=(const Dbl & other) {
    if (this != &other) {
      cval_ = other.cval_;
    }
    cout << "assign(" << other.cval_ << ")" << endl;
    return *this;
  }
  Dbl& operator+=(const Dbl& other) {
    cval_ += other.cval_;
    cout << "op+=(" << cval_ << ")" << endl;
    return *this;
  }
private:
  double cval_;
};

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

(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? (This program, and the code from the previous page that it uses, compiles and executes successfully.)

Hints: remember that variables declared in a function are initialized (constructed) in declaration order. Destruction order is the reverse of declaration/construction order. Anonymous temporary objects created by the compiler will be destroyed after they are last used, but the exact time that this is done is not guaranteed.

```
int main(int argc, char* argv[]) {
  Dbl a = 1.0;
  Dbl b(1.5);
  Dbl c = b;
  cout << "---" << endl;
  a = 1 + c;
  cout << "---" << endl;
  a = b = c += 2;
  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.