University of Washington – Computer Science & Engineering

Spring 2019 Instructor: Justin Hsia 2019-05-10

CSE333 MIDTERM

Last Name:		
First Name:		
Student ID Number:		
Name of person to your Left | Right		
All work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CSE333 who haven't taken it yet. Violation of these terms could result in a failing grade. **(please sign)**		

Do not turn the page until 5:00.

Instructions

- This exam contains 10 pages, including this cover page. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The last page is a reference sheet. Please detach it from the rest of the exam.
- The exam is closed book (no laptops, tablets, wearable devices, or calculators). You are allowed one page (US letter, double-sided) of *handwritten* notes.
- Please silence and put away all cell phones and other mobile or noise-making devices. Remove all hats, headphones, and watches.
- You have 70 minutes to complete this exam.

Advice

- Read questions carefully before starting. Skip questions that are taking a long time.
- Read *all* questions first and start where you feel the most confident.
- Relax. You are here to learn.

Question	1	2	3	4	5	Total
Possible Points	19	15	25	31	13	**103**

Question 1: You MAKE Me Whole [19 pts]

Let `CFLAGS = -Wall -g -std=c11`. The symbol "`$^`" means all sources.

(A) Complete the corresponding directed acyclic graph for the `Makefile`. [5 pt]

```
may: april.o flowers.o
   gcc $(CFLAGS) -o may $^

april.o: april.c showers.h
   gcc $(CFLAGS) -c april.c

flowers.o: flowers.c showers.h sun.h
   gcc $(CFLAGS) -c flowers.c

soil.o: soil.c sun.h showers.h
   gcc $(CFLAGS) -c soil.c

garden: flowers.o soil.o
   gcc $(CFLAGS) -o garden $^

clean:
   rm -f *.o garden
```

(B) Starting with only the source files (`.c` and `.h`) and `Makefile`, we run "`make`" followed by "`make clean`". What happens to the following files? Use "**C**" for created, "**CD**" for created and then deleted, and "**U**" for untouched (*i.e.* unchanged or not created). [4 pt]

`may` ____ `april.o` ____ `soil.o` ____ `garden` ____

(C) Write out a new `all` target that builds all the non-phony targets with the *shortest* source list possible. [2 pt]

(D) Where should we put the `all` target in `Makefile`? [2 pt]

(E) [0] Assume `all` works properly. [1] We run "`make all`". [2] We modify `sun.h`. [3] What happens to the following files when we run "`make all`" again? Use "**M**" for modified and "**U**" for untouched. [4 pt]

`may` ____ `april.o` ____ `flowers.c` ____ `garden` ____

(F) The given `Makefile` above has a subtle mistake (besides no `all`). Describe the fix. [2 pt]

SID: _____

Question 2: Love Your Food (PRE)PROCESSOR [15 pts]

Suppose we have the following files:

food.h:

```
#ifdef SWITCH
#define FOOD(a) ((a>0)-0.5)*2*y;
#else
#define FOOD(a) a
#endif
typedef int num;
```

food.c:

```
#include <stdio.h>
#include "food.h"
#define x 3.5
int y = -7.5;
int main(int argc, char **argv) {
  printf("%d\n", (int) FOOD(x) );
  return 0;
}
```

(A) The header file is missing a header guard! Following the style guide for this class, what name should we use for the guard macro? [2 pt]

(B) If we compile with `gcc food.c`, what is output when we run `a.out`? [4 pt]

(C) Complete the result of `cpp -P -DSWITCH food.c` below. Ignore the output of the `#include <stdio.h>` directive. [6 pt]

```
int main(int argc, char **argv) {


}
```

(D) (Circle one) What will be happen when we try to compile `gcc -DSWITCH food.c` and run `a.out`? [3 pt]

compiler error	output -7	output 0	output 7	output 7.5

Question 3: SHELTER Me From The C And The Storm [25 pts]

We're writing software in C to help a local animal shelter track their current (*i.e.* unadopted) and former (*i.e.* adopted) residents. We will use the following typedef-ed structs:

```
typedef struct an {
  char *serial;     // unique ID (variable length) [Heap]
  uint8_t adopted;  // 0 - unadopted, 1 - adopted
} Animal;
```

```
typedef struct sh {
  Animal **residents;  // pointer to array of Animal pointers [Heap]
  uint32_t num_res;    // length of residents array
  char manager[7];     // manager's name
} Shelter;
```

(A) Draw a memory diagram for a small `Shelter Hsiadoption` that has two residents: an unadopted cat with serial number `"3DJc"` and an adopted dog with serial number `"xj1"`. The manager's name is `"Justin"`. **Internal character arrays should have individual elements drawn out explicitly, but pointed-to c-strings can be written as string literals. Don't forget to include variable/field names.** [8 pt]

(B) An implementation of `CloseShelter()` is below, which is supposed to clean up all of the Heap memory managed by a `Shelter` instance. Describe three errors below. [5 pt]

```
void CloseShelter(Shelter s) {
  for (int32_t i = 0; i < s.num_res; i++) {
    free(s.residents[i]->serial);
    free(s.residents[i]->adopted);
  }
}
```

Memory Error:
Memory Error:
Style Error:

(C) Below, complete the helper function `GenSerial()` that generates a new, random serial string of random length. Assume we have the following functions available to you: [9 pt]

```
int32_t randLen();  // returns a random int in the range of 1-10
char randChar();    // returns a random printable character
```

```
// Returns a random serial # and its length. Returns -1 on error.
int32_t GenSerial(char **serial) {




}
```

(D) Given a pointer `Animal *a = (Animal *) malloc(sizeof(Animal))`, set its fields to an unadopted animal and give it a serial using `GenSerial()`: [3 pt]

_____;

_____;

Question 4: Class DICTation [31 pts]

Abbrev: constructor (**ctor**), copy constructor (**cctor**), assignment (**op=**), destructor (**dtor**).
All code written for this question will be ***graded on style***.

```
struct KVPair {
  KVPair() = default;
  KVPair(string k, string v);
  KVPair(const KVPair &p) = delete;
  string key, value;
};  // struct KVPair

class Dict {
 public:
  Dict() : entries_(nullptr), size_(0) { }
  Dict(const Dict &d);  // DEEP copies data members
  Dict &operator=(const Dict &rhs);
  ...  // other methods that you will implement

 private:
  size_t size_;      // # of entries in dictionary
  KVPair *entries_;  // array of size_ entries [Heap]
};  // class Dict
```

(A) Given `KVPair p1` and `Dict d1`, will the following work? Answer "**Y**" or "**N**". [4 pt]

`KVPair p2;` _____ `Dict d2 = d1;` _____

`p1 = KVPair();` _____ `d1 = Dict(0,nullptr);` _____

(B) (Circle one) Which field is initialized first during the construction of a `Dict` object? [2 pt]

`key` `entries_` `size_` `value`

(C) Write out an *inline definition* of an accessor **`get_size()`** for `Dict`. [3 pt]

(D) *Briefly* argue whether or not we should define an accessor for `entries_` in `Dict`. [2 pt]

(E) `entries_` points to an array on the Heap. Define a `Dict` member method **`Push()`** for the implementation file (`.cc`) that adds a given `KVPair` to the end of `entries_`. [8 pt]

```
void Dict::Push(const KVPair &p) {









}  // many valid solutions exist
```

(F) The inline definition of the `Dict` destructor is given below: [3 pt]

```
~Dict() { delete[] entries_; }
```

(Circle one) Which destructor first *completes* during the destruction of a `Dict` object?

`key` `entries_` `size_` `value`

(G) (Circle one) What type of function should the following be? [2 pt]

```
Dict operator+(const Dict &a, const Dict &b) {
  Dict out;
  out.entries_ = new KVPair[a.size_ + b.size_];
  for (int i = 0; i < a.size_; i++)
    out.entries_[i] = a.entries_[i];
  for (int j = 0; j < b.size_; j++)
    out.entries_[j + a.size_] = b.entries_[j];
  return out;
}
```

non-friend + member | friend + member | non-friend + non-member | friend + non-member

(H) Assume that the `Dict` cctor (definition not shown) does a *deep* copy of data members. If `d1` and `d2` are both `Dicts` of size 1, how many times are each of the following invoked (count *both* `Dict` and `KVPair` methods) during **`d1 + d2`**? [7 pt]

ctor _____ cctor _____ op= _____ dtor _____

Question 5: The INs and OUTs [13 pts]

(A) *Briefly* explain why the C standard library file I/O functions are considered more **portable** than the POSIX library file I/O functions. [2 pt]

(B) Convert the following two lines of C code into their POSIX library equivalents. Do NOT add any other lines (*e.g.* error checking): [5 pt]

C Std Lib:
```
FILE *file = fopen("midterm.txt", "w");
size_t n = fwrite(buf, sizeof(long), 10, file);
```

POSIX: _____;

_____;

(C) When we find an *unrecoverable* error in the following function calls, do we need to close the associated file descriptor during our error handling? Answer "**Y**" for yes and "**N**" for no. [3 pt]

open ____ read ____ write ____ close ____

(D) For the following I/O function **return types**, what is the common indicator of an error? [3 pt]

FILE * _____

size_t _____

ssize_t _____

CSE 333 Reference Sheet (Midterm)

C Library Header – stdio.h

```
FILE          // type of object containing info to control a stream

FILE*  fopen (const char* filename, const char* mode);
int    fclose (FILE* stream);
int    fprintf (FILE* stream, const char* format, ...);
char*  fgets (char* str, int num, FILE* stream);
size_t fread (void* ptr, size_t size, size_t count, FILE* stream);
size_t fwrite (const void* ptr, size_t size, size_t count, FILE* stream);
void   perror (const char* str);
int    ferror (FILE* stream);       // returns non-zero if error on stream
```

C Library Header – stdlib.h

```
EXIT_SUCCESS  // success termination code
EXIT_FAILURE  // failure termination code

void*  malloc (size_t size);
void*  realloc (void* ptr, size_t size);  // change size of mem block *ptr
void   free (void* ptr);                  // does nothing when ptr = NULL
void   exit (int status);                 // terminate calling process
```

C Library Header – string.h

```
size_t strlen (const char* str);           // # of chars, not including '\0'

char*  strcpy (char* dst, const char* src);          // copy chars
char*  strcat (char* dst, const char* src);          // append chars
int    strcmp (const char* str1, const char* str2);  // compare strings
```

- Versions that take a third parameter `size_t num`: `strncpy()`,`strncat()`,`strncmp()`

C Library Header – math.h

```
INFINITY      // Infinity
NAN           // Not-A-Number

float  abs (float x);                // absolute value
float  pow (float base, float exp);  // base raised to the power exp
float  sqrt (float x);               // square root
float  ceil (float x);               // round up (towards +∞)
float  floor (float x);              // round down (towards -∞)
```

- All of these functions are overloaded to work with `double`, too

POSIX Library Headers – fcntl.h, unistd.h, dirent.h

```
O_RDONLY      // read-only flag
O_WRONLY      // write-only flag
O_RDWR        // read-write flag
O_APPEND      // append (add to end) flag
DIR           // type representing a directory stream

int     open (char* pathname, int flags, ...);          // open a file
int     close (int fd);                                 // close a file
ssize_t read (int fd, void* buf, size_t count);         // read from file
ssize_t write (int fd, const void* buf, size_t count);  // write to file

DIR*    opendir (const char* dirname);                  // open a directory
int     closedir (DIR* dirp);                           // close a directory
struct dirent* readdir (DIR* dirp);                     // read a directory
```

Error Library – errno.h

```
errno         // # of the last error, usually checked against defined consts

EACCES        // permission denied
EBADF         // bad file/directory descriptor
EFAULT        // bad address supplied
EINTR         // interrupted function
EISDIR        // is a directory
ENOTDIR       // is not a directory
```

C++ Memory Allocation

```
new           // allocate space for type, return pointer
new[]         // allocate space for array of type, return pointer
delete        // deallocate space indicated by pointer
delete[]      // deallocate space of array indicated by pointer
```

Format Specifiers

Specifier	Type
d / i	signed decimal integer
u	unsigned decimal int
x	unsigned hexadecimal integer
f	decimal floating point
c	character
s	string of characters
p	pointer address

Streams

<stdio.h>	POSIX	<iostream>
stdin	0	std::cin
stdout	1	std::cout
stderr	2	std::cerr