University of Washington – Computer Science & Engineering

Spring 2019 Instructor: Justin Hsia 2019-05-10

CSE333 MIDTERM

Last Name:	**Perfect**	
First Name:	**Perry**	
Student ID Number:	1234567	
Name of person to your Left | Right	Samantha Student	Samantha Student
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.

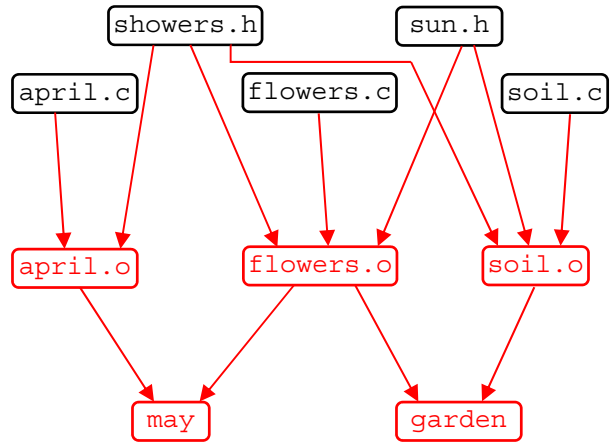
Question	1	2	3	4	5	Total
Possible Points	19	15	25	31	13	**103**

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` **_C_** `april.o` **_CD_** `soil.o` **_U_** `garden` **_U_**

`make` runs the first target in the Makefile (`may`), which doesn't reach `soil.o` or `garden`. `make clean` removes all object files and `garden` (never built), but not `may`.

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

```
all: may garden
```

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

At the top/beginning of the Makefile.

(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` **_M_** `april.o` **_U_** `flowers.c` **_U_** `garden` **_M_**

Follow the DAG from `sun.h` to see that `may` and `garden` are affected, but `april.o` is not. None of the source files (`flowers.c`) are affected by the commands in this Makefile.

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

Need to include the executable `may` in the `rm` command under the phony target `clean`.

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]

FOOD_H_

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

`SWITCH` is not defined, so it prints the value of `(int) 3.5`

3

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

```
typedef int num;
int y = -7.5;

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

  printf("%d\n", (int) ((3.5>0)-0.5)*2*y; );
  return 0;
}
```

(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

Notice the extra semicolon! Even without that, the `(int)` cast would round `((3.5>0)-0.5) = 0.5` down to `0`, resulting in `0*2*-7 = 0` being printed.

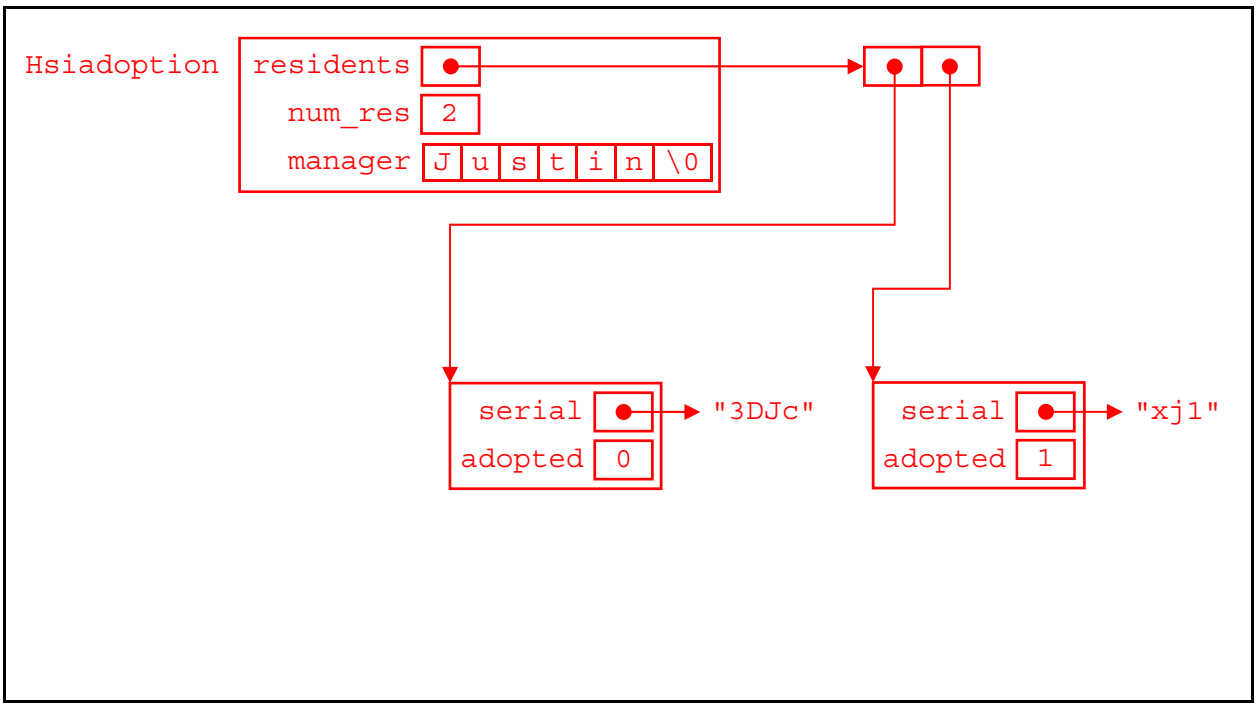
Question 3: SHELTER Me From The C And The Storm [25 pts]

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 Errors:	memory leak of `s.residents` memory leak of `s.residents[i]` invalid free of `s.residents[i]->adopted` (not malloc-ed)
Style Error:	the `Shelter` should be passed as a pointer to avoid struct copying a variety of other style issues were given partial and full credit

(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) {
  // generate random serial length
  int32_t len = randLen();

  // allocate space for c-string (including null terminator)
  *serial = (char *) malloc((len+1)*sizeof(char));

  // error checking for failed allocation
  if (*serial == NULL)
    return -1;

  // assign random characters
  for (int32_t i = 0; i < len; i++)
    (*serial)[i] = randChar();

  // add null terminator to end c-string
  (*serial)[len] = '\0';

  return len;
}
```

(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]

```
__GenSerial(&a->serial)_____________________;
__a->adopted = 0____________________________;
```

In real code, we should have error checked the return value of `GenSerial` (for `-1`), but this was overlooked for the purposes of this exam (only 2 lines given).

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;` **Y** (def ctor) `Dict d2 = d1;` **Y** (cctor)

`p1 = KVPair();` **Y** (synth op=) `d1 = Dict(0,nullptr);` **N** (2-arg ctor)

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

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

Data members are constructed/initialized in the order they are defined.

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

```
size_t get_size() const { return size_; }
```

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

No. Returning a copy of `entries_` will allow outside access to modify the contents of `*entries_`.

(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) {
  KVPair *old = entries_;
  entries_ = new KVPair[size_ + 1];  // def ctor
  for (int i = 0; i < size_; i++)
    entries_[i] = old[i];            // op=
  entries_[size_] = p;               // op=
  size_++;                           // increase size by one
  delete[] old;                      // clean up old memory
}  // 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)

During the deletion of `entries_`, each `KVPair` in the array gets destructed. Data members are destructed in the *reverse* order of definition.

(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)

The function prototype takes 2 parameters, so it must be non-member. But it directly accesses private members, so it has to be a `friend` function.

(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 __5__ cctor __1__ op= __4__ dtor __3__

ctor: `out` (1, `Dict`), `new` (2, `KVPair`), `return` (2, `KVPair` during cctor of `Dict`).
op=: for-loop assignments (2, `KVPair`), `return` (2, `KVPair` during cctor of `Dict`).
cctor: `return` (1, `Dict`). dtor: `return` (1, `Dict` and 2, `KVPair` during `Dict` 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]

The C standard library is specified as part of the C programming language, and is therefore found in *every* implementation of C on any system. The POSIX library is defined for just Unix-like variants. On systems without POSIX, the C standard library file I/O functions will invoke the appropriate other library functions (*e.g.* Windows API) instead.

(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:
```
int fd = open("midterm.txt", O_WRONLY);
ssize_t n = write(fd, buf, 10*sizeof(long));
```

(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 **N** read **Y** write **Y** close **N**

On `open`, nothing to close! On `close`, likely won't work on repeated attempts.

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

FILE * **NULL**

size_t **0**

ssize_t **-1**