**University of Washington – Computer Science & Engineering**

Spring 2018 Instructor: Justin Hsia 2018-05-04

# 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 12 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 | 6 | Total |
|---|---|---|---|---|---|---|---|
| Possible Points | 12 | 9 | 24 | 12 | 15 | 18 | **90** |

**Question 1:** You MAKE Me Whole [12 pts]

For the following questions, you may use the variable `CFLAGS = -Wall -g -std=c11`.

(A) We have a file `oneA.c` that includes `oneA.h`. Write a Makefile target to produce the executable `oneA`. [3 pt]

---

Recall that targets can execute multiple commands. The `touch` command updates the timestamp on a file to the current time (and creates the file if it did not previously exist).

(B) Draw out a corresponding directed acyclic graph for the Makefile on the left. [4 pt]

```
cse: cse.o engr.o
     gcc $(CFLAGS) -o cse *.o

cse.o: cse.c engr.h uw.h
     touch engr.o
     gcc $(CFLAGS) -c cse.c

engr.o: engr.c engr.h
     gcc $(CFLAGS) -c engr.c

uw.o: uw.c uw.h
     gcc $(CFLAGS) -c uw.c

clean:
     rm -f *.o *~ cse
```

(C) A likely dependency error should be apparent from part B. Describe the fix. [2 pt]

(D) Even with the dependency fix from part C applied, running `make clean` then `make` results in a linking error! *Briefly* describe why this happens. [3 pt]

SID: _____

**Question 2:** Trust the (PRE)PROCESS(OR) [9 pts]

Note: the `math.h` functions are relevant to this problem. Suppose we have the following files:

justin.h:

```
#ifndef SWITCH
#define ceil floor
#else
#define pow(x,y) 3*x
#endif
```

justin.c:

```
#include <stdio.h>
#include <math.h>
#include "justin.h"
int main(int argc, char **argv) {
  printf("%d\n", (int) pow(ceil(1.5),2) );
  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? [1 pt]

(B) If we compile with `gcc justin.c`, what is output when we run `a.out`? [3 pt]

(C) Show the result produced when we run `justin.c` through the C preprocessor with the `-DSWITCH` option enabled. Ignore the output of the `#include <stdio.h>` and `#include<math.h>` directives. [3 pt]

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




}
```

(D) (Circle one) What will be output when we run `a.out` after compiling with `gcc -DSWITCH justin.c`? [2 pt]

1 3 4 6 other

**Question 3:** It's Fight or FLIGHT [24 pts]

We're rewriting airport software to help the good folks at Sea-Tac keep track of flights. We will use the following typedef-ed structs:

```
typedef struct t {
  size_t hr;   // 0-23
  size_t min;  // 0-59
} Time;
```

```
typedef struct f {
  char *dest;  // destination airport
  Time dep;    // departure time
  Time arr;    // arrival time
} Flight;
```

```
typedef struct a {
  char *name;
  Flight *flights;  // address of array of flights
  size_t num_f;     // number of flights in array
  struct a *next;
} Airport;          // node in linked list of airports
```

Each airport node holds a pointer to an array of `Flight`s on the Heap and the length of that array is stored in `num_f`. **Pointers `name` and `dest` should also point to the Heap.** Assume we have the code shown below:

```
// Creates a new Airport (name: copied from argument, flights: NULL,
// num_f: 0, next: NULL) on the Heap and then pushes it to the front
// of our linked list of airports.
Airport *MakeAirport(char *name);

// Takes the provided Flight data and stores it in the flights array
// of the specified airport.  Needs to update flights and num_f.
void AddFlight(Airport *a, char *dest, Time dep, Time arr);
```

```
Airport *head = NULL;

int main(int argc, char **argv) {
  Time t1 = {10,40}, t2 = {12,42};
  head = MakeAirport("SEA");
  AddFlight(head, "SFO", t1, t2);
  head = MakeAirport("SFO");
  return EXIT_SUCCESS;
}
```

(A) Draw a memory diagram for our linked list of airports before `main()` returns: [9 pt]

(B) Complete the implementation of `AddFlight()`. Assume `stdio.h`, `stdlib.h`, and `string.h` are included. Assume arguments are valid, but check for other errors. [15 pt]

```
void AddFlight(Airport *a, char *dest, Time dep, Time arr) {
  // make more space for larger array









  // allocate space for destination name




  // copy data into appropriate fields



  // update number of flights

}
```

**Question 4:** FILE This One Away [12 pts]

(A) Name one *major* difference between the C standard library file I/O functions and the POSIX library file I/O functions. [2 pt]

(B) Complete the program below that will write each command-line argument (*not* including the executable name) to a separate line *at the end* of the specified file. You shouldn't need to write more than ~15 lines. Hint: POSIX discouraged. [10 pt]

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

int main(int argc, char **argv) {
   char *filename = "file.txt";









  return EXIT_SUCCESS;  // be consistent!
}
```

**Question 5:** Class Is Now In Session STUDENTS [15 pts]

```
using namespace std;
class Student {
 public:
  Student(size_t id, string name, size_t years);
  void birthday() { age_++; }
  Student &operator=(const Student &rhs) = delete;
 private:
  size_t sid_;   // student ID number
  string name_;
  size_t age_;
};  // class Student
```

(A) Complete the definition of the 3-argument constructor below using an *initializer list*: [3 pt]

```
Student(size_t id, string name, size_t years)

```

(B) Given `Student` instances `Alice` and `Bob`, will the following work? Answer Y/N. [2 pt]

`Student Carol;` _____ `Student Dan(Bob);` _____

`Student Eve = Alice;` _____ `Alice = Bob;` _____

(C) If we add `Student() { }` as a `public` member, what are the following values in a default constructed `Student` instance? *Be precise, if possible!* [2 pt]

`sid_` _____ `name_` _____

(D) Write out an inline definition of an accessor function for `name_` *using good style.* [3 pt]

(E) We want to add an *operator* (which one?) to compare two `Student`'s by their unique ID numbers. Write out the declaration and definition *using good style!* [5 pt]

**Decl:**

**Defn:**

**Question 6:** An EXAM Within An Exam [18 pts]

Abbrev: constructor (**ctor**), copy constructor (**cctor**), assignment (**op**=), destructor (**dtor**).

Below is a *modified* version of `class MultChoice` from section, now renamed `MC` for brevity:

```
class MC {
 public:
  MC() : resp_(' ') { }
  MC(char resp) : resp_(resp) { }
  char get_resp() const { return resp_; }
  bool Compare(MC mc) const;  // do the fields match?

 private:
  char resp_;  // response: 'A','B','C','D', or 'E'
};  // class MC
```

Answer the following questions based on the class definition above and the code below:

```
#define QS 2
MC key[QS] = {'D', 'A'};  // this works

size_t Score(const MC *ans) {
  size_t score = 0;
  for(int i = 0; i < QS; i++) {
    if(ans->Compare(key[i])) score++;
    ans++;
  }
  return score;
}

int main(int argc, char **argv) {
  MC myAns[QS];
  myAns[0] = MC('B');
  myAns[1] = MC('A');
  std::cout << "Score: " << Score(myAns) << std::endl;
  return 0;
}
```

(A) Could the following variables be passed as an argument to `Score`? Answer Y/N. [2 pt]

`MC *a` _____ `MC * const b` _____

`const MC *c` _____ `const MC * const d` _____

(B) When the program is executed, how many times are each of the following invoked? [6 pt]

ctor _____ cctor _____ op= _____ dtor _____

Using `class MC` would get tedious quickly. We want to write a wrapper class called `Exam` that stores an array of `MC` instances *on the heap*. We turn it into a class template:

```
template <int QS> class Exam {
 public:
  Exam();
  Exam(std::string name, MC *mcs);
  ~Exam();
  MC get_question(size_t num) const;
  void change_resp(size_t num, char resp);
  size_t Score(Exam<QS> &key) const;       // score exam against key
  std::string name;                        // name of the exam

 private:
  MC *mcs_;                                // array of MCs on heap
};  // class Exam
```

(C) The destructor should clean up as necessary. Complete its definition below: [1 pt]

```
template <int QS> Exam<QS>::~Exam() {


}
```

(D) The default constructor should set `name` to an empty string and `mcs_` to the address of an appropriately-sized array. Write out its definition: [3 pt]

(E) Write out the definition of the member function `Score`. [4 pt]

(F) A cheater got a copy of the answer key using the `Exam` synthesized default copy constructor! What happens to our answer key at the end of the program? [2 pt]

# THIS PAGE PURPOSELY LEFT BLANK

# 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
size_t read (int fd, void* buf, size_t count);         // read from file
size_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 |