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

Spring 2018 Instructor: Justin Hsia 2018-05-04

# CSE333 MIDTERM

| | |
|---|---|
| Last Name: | **Perfect** |
| First Name: | **Perry** |
| Student ID Number: | 1234567 |
| Name of person to your Left \| Right | Samantha Student \| Larry Learner |
| 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]

```
oneA: oneA.c oneA.h
      gcc -Wall -g -std=c11 -o oneA oneA.c
      OR
      gcc $(CFLAGS) -o oneA oneA.c
```

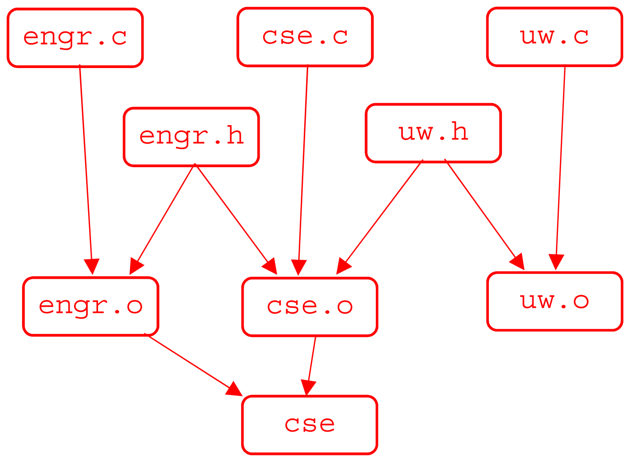
---

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
```

Note: direction of arrows didn't matter as long as consistent.



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

Add `uw.o` to the source list in the `cse` target.

(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]

`make clean` removes all object files. In addressing the target `cse`, we first run the commands in the `cse.o` target, which creates an *empty*, but "updated" `engr.o` file. Therefore, we don't run the `engr.o` target commands and there is an error when linking to the empty `engr.o` file.

## 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]

JUSTIN_H_

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

Prints the value of `(int) pow(floor(1.5),2) = 1^2 = 1`.

1

(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) {

  printf("%d\n", (int) 3*ceil(1.5) );
  return 0;

}
```

(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]

The `int` cast applies first to the `3`, which is then multiplied by `2.0` because `ceil()` returns a `float/double`. Interpreting a floating point number as an `int` (`%d` specifier) will cause `printf()` to print a weird number.

## 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;
```

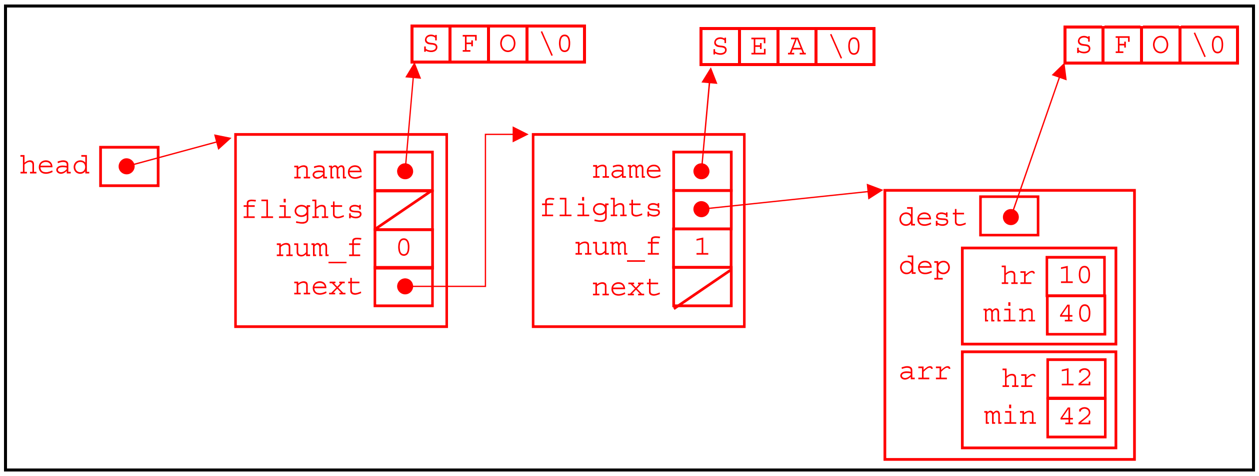
```
typedef struct a {
  char *name;
  Flight *flights;  // address of array of flights
  size_t num_f;     // number of flights in array
  struct a *next;
} Airport;          // node in linked list of airports
```

Each airport node holds a pointer to an array of `Flights` 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
  // realloc works like malloc if a->flights == NULL
  a->flights = (Flight*) realloc(a->flights,
                               (a->num_f+1)*sizeof(Flight) );
  // check for realloc error
  if (a->flights == NULL) {
    perror("flight malloc/realloc failed");
    exit(EXIT_FAILURE);
  }



  // malloc space for destination name
  a->flights[a->num_f].dest = (char*) malloc( (strlen(dest)+1)
                                            * sizeof(char) );
  // check for malloc error
  if (a->flights[a->num_f].dest == NULL) {
    perror("dest malloc failed");
    exit(EXIT_FAILURE);
  }

  // copy data into appropriate fields
  strcpy(a->flights[a->num_f].dest,dest);
  a->flights[a->num_f].dep = dep;
  a->flights[a->num_f].arr = arr;


  // update number of flights
  a->num_f++;

}
```

## 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]

- Buffered vs. non-buffered
- `FILE*` vs. file descriptors (`int`)
- Less error handling in POSIX
- etc.

(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";

  FILE *file = fopen(filename, "a");  // or "ab"
  if (file == NULL) {
    perror("fopen error");
    return EXIT_FAILURE;    // or exit(EXIT_FAILURE)
  }
  for (int i = 1; i < argc; i++) {
    fprintf(file, "%s\n", argv[i]);
    if (ferror(file)) {
      perror("fprintf error");
      fclose(file);         // optional, but good to have
      return EXIT_FAILURE;  // or exit(EXIT_FAILURE)
    }
  }
  fclose(file);







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

SID: ________________

## 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)
  : sid_(id), name_(name), age_(years) { }
```

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

```
        Student Carol;  N (def ctor)        Student Dan(Bob);  Y (cctor)
 Student Eve = Alice;  Y (cctor)                Alice = Bob;  N (op=)
```

(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_` garbage/unknown        `name_` `std::string("")`

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

```
std::string get_name() const { return name_; }  OR
const std::string &get_name() const { return name_; }
```

(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:**

```
friend bool operator<(const Student &lhs, const Student &rhs);
  OR
bool operator<(const Student &rhs) const;
```

**Defn:**

```
bool operator<(const Student &lhs, const Student &rhs) {
  return lhs.sid_ < rhs.sid_;
}
  OR
bool Student::operator<(const Student &rhs) const {
  return sid_ < rhs.sid_;
}
```

## 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 - ctor (x2)

size_t Score(const MC *ans) {
  size_t score = 0;
  for(int i = 0; i < QS; i++) {        // in loop (x2), non-ref param
    if(ans->Compare(key[i])) score++; //  to Compare invokes cctor
    ans++;
  }
  return score;
}

int main(int argc, char **argv) {
  MC myAns[QS];          // default ctor (x2)
  myAns[0] = MC('B');    // ctor, then op=
  myAns[1] = MC('A');    // ctor, then op=
  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` | __**Y**__ | `MC * const b` | __**Y**__ |
| `const MC *c` | __**Y**__ | `const MC * const d` | __**Y**__ |

The pointer address is copied into parameter, so all are allowed (assuming variables were assigned to properly).

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

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ctor | __**6**__ | cctor | __**2**__ | op= | __**2**__ | dtor | __**8**__ |

dtor = ctor + cctor (nothing is dynamically allocated, so no memory leaks).

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() {
  delete[] mcs_;
}
```

(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]

```
template <int QS> Exam<QS>::Exam() : name("") {

  mcs_ = new MC[QS];

}
```

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

```
template <int QS>
size_t Exam<QS>::Score(Exam<QS> &key) const {
  size_t score = 0;
  for (int i = 0; i < QS; i++) {
    if ( mcs_[i].Compare(key.mcs_[i]) ) {  // op== not defined
      score++;
    }
  }
}
```

(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]

**Double delete.** The cctor does a shallow copy of the pointer `mcs_`, so two different `Exam` objects point to the same array on the Heap. When they are destructed, we delete the same address twice.