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.

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Points</td>
<td>19</td>
<td>10</td>
<td>24</td>
<td>32</td>
<td>19</td>
<td>104</td>
</tr>
</tbody>
</table>
**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]

<table>
<thead>
<tr>
<th>winter: rain.o snow.o clouds.o</th>
<th>rain.h</th>
<th>clouds.h</th>
<th>cold.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc $(CFLAGS) $^</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>snow: snow.o</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc $(CFLAGS) -o snow $^</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rain.o: rain.c rain.h clouds.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc $(CFLAGS) -c rain.c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clouds.o: clouds.c clouds.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc $(CFLAGS) -c clouds.c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>snow.o: snow.c clouds.h rain.h cold.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc $(CFLAGS) -c snow.c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clean:</th>
</tr>
</thead>
<tbody>
<tr>
<td>rm -f rain.o clouds.o winter snow</td>
</tr>
</tbody>
</table>

(B) Starting with only the source files (.c and .h) and Makefile, what should happen to the following files if we run “make” followed by “make clean”? Use “C” for created, “CD” for created and then deleted, and “U” for untouched (i.e. unchanged or not created).  [4 pt]

| rain.o ____ | clouds.o ____ | snow.o ____ | winter ____ |

(C) Do we need a phony all target in Makefile? Briefly justify your response.  [2 pt]

| Yes / No    |


| rain.c ____ | clouds.o ____ | snow.o ____ | snow ____ |

(E) Assuming that the two executables do different things, it turns out that there is something inherently wrong with our project setup that will cause 1 of 2 possible compilation errors. Identify the compilation errors and which target will cause them. **Hint:** what does every C executable need?  [4 pt]

<table>
<thead>
<tr>
<th>Possible error:</th>
<th>Target:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible error:</td>
<td>Target:</td>
</tr>
</tbody>
</table>
Question 2: PREPROCESS This! [10 pts]

Suppose we have the following files:

twoface.h:
```c
#ifdef DSWITCH
#define FACE(f) NULL
#else
#define FACE(f) (f * -2)
typedef int my_type;
#endif
```
twoface.c:
```c
#include <stdio.h>
#define f 2.0
#include "twoface.h"
int main(int argc, char** argv) {
    printf("%ld\n", (long) FACE(f));
    return 0; // EXIT_SUCCESS
}
```

(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) Complete the result of `cpp -P -DSWITCH twoface.c` below. Ignore the output of the `#include <stdio.h>` directive. [5 pt]

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

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

<table>
<thead>
<tr>
<th>compiler error</th>
<th>output 0</th>
<th>output 4</th>
<th>output -4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 3: ORDER Up [24 pts]

We’re writing C software for restaurants to track orders using the following typedef-ed struct:

```
#define NUM_MENU_ITEMS 3
typedef struct order_st {
    int table;       // table number
    char* server;    // name of server
    int orders[NUM_MENU_ITEMS]; // # of each menu item ordered
    struct order_st* next;    // next order in linked list
} Order;
```

// order of 3 of menu item #0 for table 333, served by Justin
Order example = {333, "Justin", {3, 0, 0}, NULL};

We use Order* head to track all orders and Order* curr to track the current order. Assume both are defined in main. Because we cannot predict how many orders we will get, Orders must be allocated individually on the heap.

(A) Draw a memory diagram for a small linked list of two orders. The first order is for table 3, served by "Andrew", and is for 1 of menu item #1. The second (and current) order is for table 7, served by "Cheng", and is for 2 of menu item #0 and 4 of menu item #2.

Character arrays can be written as string literals. Don’t forget to include variable and field names. [8 pt]
(B) Below, complete the helper function `CreateOrder()` that generates a new, empty order (i.e., 0 quantity of all menu items) with some specified field values. Assume that *server doesn’t need to be deep-copied. NUM_MENU_ITEMS is #define-d.  [8 pt]

```c
// Returns a pointer to an empty order, or NULL on error.
Order* CreateOrder(int table, char* server) {
    Order* order = (Order*) malloc(sizeof(Order));
    if (order != NULL) {
        order->table = table;
        order->server = server;
        for (int i = 0; i < NUM_MENU_ITEMS; i++) {
            order->orders[i] = 0;
        }
        order->next = NULL;
    }
    return order;
}
```

(C) Recall that head and curr are local pointers in main. We are writing `AddOrder` that takes a specified heap-allocated `Order` (e.g. the return value from `CreateOrder`) and adds it to the end of the head list. If either head or curr is NULL, then they need to be updated to point to this new `Order`, meaning we may need to update the values of both head and curr in this function. Following good style guidelines, propose a suitable declaration:  [4 pt]

```
AddOrder(
```

(D) If we want to create a module for our `Order` system, indicate which file the following would go in (checkmark):  [4 pt]

<table>
<thead>
<tr>
<th></th>
<th>Order.h</th>
<th>Order.c</th>
<th>Restaurant.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order typedef from problem description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateOrder() definition from part B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateOrder() declaration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>main()</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 4: Time to Get in SHAPE  [32 pts]

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

```
struct Point {
    Point() : x(0), y(0) { }
    Point(int x, int y) : x(x), y(y) { }
    int x, y;
}; // struct Point

class Shape {
public:
    Shape() : num_pts_(1), points_(new Point) { }
    Shape(const Shape& s); // DEEP copies data members
    Shape& operator=(const Shape& rhs); // DEEP copies
    ... // other methods mentioned in this question
private:
    Point* points_; // array of num_pts_ points [Heap]
    size_t num_pts_; // # of points in shape
    uint8_t color[3]; // RGB values of shape color
}; // class Shape
```

(A) Do we need accessor methods for Point? Briefly explain why or why not. [2 pt]

(B) Write out a line of code that will disable the ctor inside the definition Point. [2 pt]

(C) What does a default Shape describe? [2 pt]

(D) The member function Area returns the area of the Shape as a double. Propose a suitable function signature (for the implementation file): [3 pt]

```
{ .................................................. }
```

(E) The member function ChangeColor sets the Shape’s color to specified red, green, and blue values. Propose a suitable function signature (for the implementation file): [3 pt]

```
{ .................................................. }
```
points_ points to an array on the heap. Define a Shape member function **Union()** that *appends* the points from a second Shape to points_ in this. Don’t worry about duplicate points or self-unions. [10 pt]

```cpp
void Shape::Union(const Shape& s) {
    // many valid solutions exist
}
```

The inline definition of the Shape destructor is given below, but leads to a memory error in our code! *Briefly* describe the issue and the fix (which may not be in the dtor): [4 pt]

```
Shape() { delete[] points_; }
```

**Issue:**

**Fix:**

Assume that the Shape **ctor** (definition not shown) does a deep copy of data members. If s is a Shape with 2 points, how many times are each of the following invoked (count both Shape and Point methods) during the execution of the friend non-member function **Reverse(s)**? [6 pt]

```cpp
Shape Reverse(const Shape& s) {
    Shape out = s;
    for (size_t i = 0; i < s.num_pts_; i++) {
        out.points_[i] = s.points_[s.num_pts_-1-i];
    }
    return out;
}
```

**ctor** ______  **cctor** ______  **op=** ______  **dtor** ______
**Question 5:** INPUT and OUTPUT and ERRORS, oh my! [19 pts]

(A) Assume that the C std lib is using an internal write buffer of **1024 bytes** and we are trying to write 2048 bytes total in **256-byte chunks**. Assuming that all writes are successful (i.e. no partial writes or errors), how many system calls do we invoke using C std lib vs. POSIX? [4 pt]

<table>
<thead>
<tr>
<th>write()</th>
<th>fwrite()</th>
</tr>
</thead>
</table>

(B) Name a C function that we have used in this class that fits the descriptions: [4 pt]

- Part of the C standard library, but doesn’t invoke a system call.
- A POSIX system call that doesn’t have a C std lib equivalent.

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

**POSIX:**
```c
int fd = open("midterm.txt", O_RDONLY);
ssize_t n = read(fd, buf, 333*sizeof(int32_t));
```

**C Std Lib:**

```c
____________________________________________________________
____________________________________________________________
```

(D) Before exiting/terminating a C program, name the three categories of **resources** that we have seen in this class that we need to make sure are cleaned up/closed: [3 pt]

|       |       |       |

(E) **Briefly** describe in what situations you prefer to use **perror** instead of **fprintf** to **stderr**. [3 pt]

|       |       |       |
CSE 333 Reference Sheet (Midterm)

C Library Header – stdio.h

```c
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);
```

C Library Header – stdlib.h

```c
EXIT_SUCCESS // success termination code
EXIT_FAILURE // failure termination code
void* malloc (size_t size);
void* calloc (size_t num, size_t size); // zero-initialized block
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

```c
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

```c
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 +\infty)
float floor (float x); // round down (towards -\infty)
```

- 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

```c
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

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>d / i</td>
<td>signed decimal integer</td>
</tr>
<tr>
<td>u</td>
<td>unsigned decimal int</td>
</tr>
<tr>
<td>x</td>
<td>unsigned hexadecimal integer</td>
</tr>
<tr>
<td>f</td>
<td>decimal floating point</td>
</tr>
<tr>
<td>c</td>
<td>character</td>
</tr>
<tr>
<td>s</td>
<td>string of characters</td>
</tr>
<tr>
<td>p</td>
<td>pointer address</td>
</tr>
</tbody>
</table>

### Streams

<table>
<thead>
<tr>
<th>Streams</th>
<th>&lt;stdio.h&gt;</th>
<th>POSIX</th>
<th>&lt;iostream&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdin</td>
<td>0</td>
<td>std::cin</td>
<td></td>
</tr>
<tr>
<td>stdout</td>
<td>1</td>
<td>std::cout</td>
<td></td>
</tr>
<tr>
<td>stderr</td>
<td>2</td>
<td>std::cerr</td>
<td></td>
</tr>
</tbody>
</table>