CSC 209H5 S 2020 Midterm Test
Duration — 80 minutes
Aids allowed: none

Instructors: F. Alaca, I. Dema, A. Petersen

---

*Do **not** turn this page until you have received the signal to start.*
Please fill in the information below, **write your name on the back of the test**, and read the instructions below.
Good Luck!

This midterm consists of 6 questions on 18 pages (including this one), **plus a sheet for entering your answers to the multiple choice questions**. The questions are worth a total of 39 marks.

When you receive the signal to start, please make sure that your copy is complete.

Comments are not required, although they may help us mark your answers.

No error checking is required except where specifically requested.

You do not need to provide include statements for your programs.

If you use any space for rough work, indicate clearly what you want marked.

Question 1. [3 MARKS]

Basic shell usage.

Part (a) [3 MARKS]

Suppose that the following is a typescript of your terminal, in which you have compiled a program that you have written:

```
dovahkiin@dh2020pc02:~$ gcc -o myprog hello.c
dovahkiin@dh2020pc02:~$ cd dir1
dovahkiin@dh2020pc02:~/dir1$
```

Write **a single command** that you would type, after typing the commands shown above, which does the following:

- Executes the program that you just compiled, **without** changing directories.
- Sends the output of the `ls` command to your program, which reads it from standard input.
- Redirects the output of your program to a text file in the current directory named `output.txt`.

Question 2. [16 MARKS]

Multiple Choice (2 marks each)

Please indicate your selection by **shading in the corresponding bubbles on the last page of this test**.

2.1. Select the option below that is **NOT** equivalent to the following statement: `int *p = 20;`

A. `int* p = 20;`

B. `int * p = 20;`

C. `int *p; p = 20;`

D. `int *p; *p = 20;`

E. None of the above: All of the above statements are equivalent.

2.2. Suppose we have the following declarations at the start of a program. Select the statement below that does **NOT** assign the value `4` to the variable `q`.

```
int x = 4;
int *y = &x;
int **z = &y;

int q;
```

A. `q = x;`

B. `q = *(&x);`

C. `q = *y;`

D. `q = &(*x);`

E. `q = **z;`

2.3. Consider the program below, which has a function call missing from `main()`. Select the correct function call that would result in the program printing the string "Hello, world!".

```
void increment1(int count) {
   count++;
}

void increment2(int *count_ptr) {
   (*count_ptr)++;
}

int main(int argc, char **argv) {
    int count = 0;
    _____________________; // Correct function call (from the options below) goes here
    if(count == 1) printf("Hello, world!");
    return 0;
}
```

A. `increment1(&count);`

B. `increment1(*count);`

C. `increment2(count);`

D. `increment2(&count);`

E. `increment2(*count);`

2.4. Consider the program below, and select the correct output that matches what the program will print.

```
struct StudentNode {
    char *name;
    int num;
};

void updateStudent(struct StudentNode s) {
  s.num = s.num + 2;
  strcat(s.name, " v2");
}

int main() {
  struct StudentNode s1;
  s1.num = 12345;
  s1.name = malloc(256*sizeof(char));
  strcpy(s1.name, "Bob");
  updateStudent(s1);
  printf("%s; ", s1.name);
  printf("%d\n", s1.num);
}
```

A. `Bob; 12345`

B. `Bob; 12347`

C. `Bob v2; 12345`

D. `Bob v2; 12347`

E. This code will cause a segmentation fault, because `strcpy` is unsafe.

2.5. The `tee` program reads from `stdin` and copies the stream to **both** `stdout` and to the file specified by the command-line argument. The `sort` program sorts the lines of text contained in the file specified by the command-line argument, and outputs to `stdout`. The `head` program reads from `stdin` and outputs the first 10 lines to `stdout`. Consider the following shell command, and select the statement below that is **TRUE** after the command is run.

```
sort input.txt | tee output.txt | head
```

A. The contents of `input.txt` and `output.txt` must be identical.

B. Only the first 10 lines of `output.txt` will be displayed in the terminal.

C. The entire contents of `input.txt` followed by the first 10 lines of `output.txt` will be displayed in the terminal.

D. The entire contents of `output.txt` followed by the first 10 lines of `output.txt` will be displayed in the terminal.

E. The entire contents of both `input.txt` and `output.txt`, followed by the first 10 lines of `output.txt`, will be displayed in the terminal.

2.6. Select the statement that is **TRUE**, considering the program consisting of the following two source files. Assume that the program is compiled with `gcc -o hello hello.c main.c`.

```
/* Complete contents of main.c */
void hello(void);
int main(void) {
    hello();
    return 0;
}

/* Complete contents of hello.c */
#include <stdio.h>

void hello() {
    printf("Hello, world!\n");
}
```

A. The program will fail to compile, because `main.c` is missing the line `#include "hello.c"`.

B. The program will fail to compile, because `main.c` is missing the line `#include <stdio.h>`.

C. The program will fail to compile, because the `gcc` command is incorrect.

D. The program will compile successfully, but its behaviour is undefined (e.g., it may trigger a segmentation fault or print out garbage values).

E. The program will compile and print `Hello, world!` to the terminal.

2.7. Select the statement that is **TRUE**, considering the program below.

```
struct my_struct {
    char *name;
};

void array_boss(struct my_struct *s) {
  char new[4] = {'a', 'b', 'c', '\0'};
  s->name = new;
}

int main(void) {
  struct my_struct s1;
  s1.name = "Bob";
  array_boss(&s1);
  printf("%s\n", s1.name);
  /* Do other things, call some other functions... */
  return 0;
}
```

A. The program will fail to compile, because `array_boss()` assigns an array to a pointer.

B. The program will fail to compile, because `s1.name` in `main()` is a read-only string literal, which `array_boss()` attempts to overwrite.

C. The program will compile without errors, but a segmentation fault will be triggered when `main()` assigns the return value of `array_boss()` to `s1.name`, since the latter is a read-only string literal.

D. The program will compile without errors, but its behaviour is undefined, e.g., it may print `abc` or it may print other garbage values or result in other unpredictable behaviour.

E. The program will compile without errors, and will always print out `abc`.

2.8. Select the option that best describes the output of the program below.

```
#include <stdio.h>
int main()
{
    char c[6] = {'C', 'S', 'C', '2', '0', '9'};
    char *ptr = (char*)(c + 1);
    printf("%c %c\n", *(c+1), ptr[-1]);
    return 0;
}
```

A. `S 9`
B. `C S`
C. `S C`
D. The program will print garbage (undefined) values.
E. The program will crash due to a segmentation fault.

Question 3. [4 MARKS]

Arrays and Pointers

Consider the following program:

```
#define ARRAY_MAX 10
#include <stdio.h>
#include <stdlib.h>

struct array_list {
    int num_elements;
    int elements[ARRAY_MAX];
};

void array_juggler(struct array_list *p1, struct array_list *p2) {
    struct array_list *temp = p2;

    p2 = p1;
    p2->elements[2] = 10;
    p2->num_elements = 3;

    p1 = temp;
    p1 = malloc(sizeof(struct array_list));
    p1->elements[0] = 41;
    p1->num_elements = 1;
}

int main() {
    struct array_list a1, a2;

    a1.elements[0] = 42;
    a1.elements[1] = 4;
    a1.num_elements = 2;

    a2.elements[0] = 12;
    a2.elements[1] = 9;
    a2.elements[2] = 11;
    a2.num_elements = 3;

    array_juggler(&a1, &a2);

   // Continued on next page...

    printf("Contents of a1: ");
    for(int i = 0; i < ARRAY_MAX && i < a1.num_elements; i++)
        printf("%d ", a1.elements[i]);
    printf("\n");

    printf("Contents of a2: ");
    for(int i = 0; i < ARRAY_MAX && i < a2.num_elements; i++)
        printf("%d ", a2.elements[i]);
    printf("\n");

    return 0;
}
```

Print the **exact output** generated by running the above program:

Question 4. [5 MARKS]

Strings, Structs, and File I/O

The following questions use the `fentry` struct from Assignment 2:

```
typedef struct file_entry {
  char name[12];          // An empty name means the fentry is not in use.
  unsigned short size;
  short firstblock;       // A -1 indicates that no file blocks have been allocated.
} fentry;
```

Part (a) [2 MARKS]

Assume you have a pointer `fent_ptr` to an array of properly initialized `fentry` structs and an `int index` identifying a specific `fentry` to update. Given the `char` pointer `fname` that refers to a string, write a statement that safely copies the string into the filename field, ensuring that the resulting filename field contains a valid string.

Part (b) [3 MARKS]

Write code to open the binary file named "`simfs.fs`" from the current working directory. Then, starting from offset 128, write the `fentry` array of size `n` pointed to by `fent_ptr` into the file. You may assume that the binary file already contained at least 128 bytes. You may also assume that the `fentry` struct is not padded.

Question 5. [6 MARKS]

Memory Model

Part (a) [3 MARKS]

Write a main function that declares 3 strings. The first named `first` should be set to the value `"January"`, and be stored on the stack frame for `main`. `second` should be a string literal with the value `"February"`. `third` should have value `"March"` and be on the heap. You may omit `#include` statements.

Part (b) [3 MARKS]

Consider the code fragment below:

```
int a[10] = {1, 2, 3};
int *i;
i = malloc(sizeof(int)*10);
```

List each block of memory that is allocated by the code fragment above. Specify:

1. The **size** (in the form `N*sizeof(...)`) of each memory block that is allocated;
2. **Where** (e.g.,. stack, heap, or read-only data) each memory block is allocated; and
3. **When** (by referring to the appropriate line number) each memory block is allocated.

Question 6. [5 MARKS]

Consider the following code:

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

typedef struct item {
    int value;
    struct item *next;
} stack_type;

void display(stack_type* stack) {
    stack_type* top = stack;
    while (top != NULL) {
       printf("%d ", top->value);
       top = top->next;
    }
    printf("\n");
}

int main() {
    stack_type *s = NULL;
    push(&s, 2);
    push(&s, 3);
    display(s);
    return 0;
}
```

Write the complete the code for the function `push` so the program outputs `3 2`.

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

C function prototypes:

```
int fclose(FILE *stream)
char *fgets(char *s, int n, FILE *stream)
FILE *fopen(const char *file, const char *mode)
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
void free(void *ptr)
int fscanf(FILE *restrict stream, const char *restrict format, ...);
int fseek(FILE *stream, long offset, int whence)
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
void *malloc(size_t size);
void perror(const char *s)
int scanf(const char *restrict format, ...);
size_t strnlen(const char *s, size_t n)
char *strncat(char *dest, const char *src, size_t n)
int strncmp(const char *s1, const char *s2, size_t n)
char *strncpy(char *dest, const char *src, size_t n)
char *strstr(const char *haystack, const char *needle)
long strtol(const char *restrict_str, char **restrict_endptr, int base)
```

Excerpt from the fseek man page:

```
If whence  is  set to  SEEK_SET,  SEEK_CUR,  or  SEEK_END,  the offset is relative to the start of
the file, the current position indicator, or end-of-file, respectively.
```

Print your name in this box.

END OF EXAMINATION