The University of Calgary

ENCM 339 – Programming Fundamentals
Fall 2017

Instructor: M. Moussavi

Wednesday November 1st From 7:00 to 9:00 PM

Location for Lecture 02: ST 148
Note: This exam is ONLY for students registered in L02 taught by Dr. Moussavi
MWF 12:00 to 12:50 PM

The First Letter of your Last Name: ☐

Please Print your Name – (First Name and Last Name): ________________________________

Your Signature: ______________________________

Please also write your name and your U of C Student ID on the last page

A reference page to some of the C-library file I/O functions is given on the last page.

Instructions

- To prevent distracting your fellow students, you may not leave the exam room during the last ten minutes of the exam.
- No electronic calculators or computers may be used during the test.
- This is a closed-book test. You must not refer to books or notes during the test.
- Write legibly! What cannot be read will not be marked.
- If you remove the staple from the paper, write your name at the top of every loose page.
- Attempt all questions.
- Get the easiest marks first! If you find a particular question difficult, return to it after completing easier questions.
- If you write any rough work on the question paper, cross it out with a large "X" and write, "Rough work" beside it. Rough work will not be marked.
- Follow the directions for each problem carefully.

EXAMINATION RULES AND REGULATIONS

STUDENT IDENTIFICATION

All candidates for final examinations are required to place their University of Calgary I.D. cards on their desks for the duration of the examination. (Students writing mid-term tests can also be asked to provide identity proof.) Students without an I.D. card who can produce an acceptable alternative I.D., e.g., one with a printed name and photograph, are allowed to write the examination.

A student without acceptable I.D. will be required to complete an Identification Form. The form indicates that there is no guarantee that the examination paper will be graded if any discrepancies in identification are discovered after verification with the student's file. A Student who refuses to produce identification or who refuses to complete and sign the Identification Form is not permitted to write the examination.

EXAMINATION RULES

(1) Students late in arriving will not normally be admitted after one-half hour of the examination time has passed.

(2) No candidate will be permitted to leave the examination room until one-half hour has elapsed after the opening of the examination, nor during the last 15 minutes of the examination. All candidates remaining during the last 15 minutes of the examination period must remain at their desks until their papers have been collected by an invigilator.

(3) All inquiries and requests must be addressed to supervisors only.

(4) Candidates are strictly cautioned against:

(a) speaking to other candidates or communicating with them under any circumstances whatsoever;

(b) bringing into the examination room any textbook, notebook or memoranda not authorized by the examiner;

(c) making use of calculators and/or portable computing machines not authorized by the instructor;

(d) leaving answer papers exposed to view;

(e) attempting to read other student's examination papers.

The penalty for violation of these rules is suspension or expulsion or such other penalty as may be determined.

(5) Candidates are requested to write on both sides of the page, unless the examiner has asked that the left hand page be reserved for rough drafts or calculations.

(6) Discarded matter is to be struck out and not removed by mutilation of the examination answer book.

(7) Candidates are cautioned against writing in their answer book any matter extraneous to the actual answering of the question set.

(8) The candidate is to write his/her name on each answer book as directed and is to number each book.

(9) A candidate must report to a supervisor before leaving the examination room.

(10) Answer books must be handed to the supervisor-in-charge promptly when the signal is given. Failure to comply with this regulation will be cause for rejection of an answer paper.

(11) If during the course of an examination a student becomes ill or receives word of a domestic affliction, the student should report at once to the supervisor, hand in the unfinished paper and request that it be cancelled. If physical and/or emotional ill health is the cause, the student must report at once to a physician/counsellor so that subsequent application for a deferred examination is supported by a completed Physician/Counsellor Statement form. Students can consult professionals at University Health Services or University Counselling Services during normal working hours or consult their physician/counsellor in the community.

Should a student write an examination, hand in the paper for marking, and later report extenuating circumstances to support a request for cancellation of the paper and for another examination, such a request will be denied.

(12) Smoking during examinations is strictly prohibited.

RO 94-01

<u>Please Notice:</u> To answer the questions in this exam, you can always make the following assumptions:

- All necessary header files are included.
- Size of integer data type is 4 bytes, and size of double data type and pointers are 8 bytes.

Also, here are some ASCII values: `'A' == 65`, `'a' == 97`, and `'0' (zero) == 48`, if you need.

<u>SECTION 1 - Multiple Choice Questions (1 mark each) – Total 10 marks:</u>

1. What is the output of the following code segment?

```
char course[] = "XNCM019F2016";
char* sp = &*(course + 1);
while(sp[1] != '2'){
    printf("%c", sp[1]);
    sp = sp + 1 ;
}
```

```
a. NCM019F
b. NCM019F
```
`c. CM019F`
```
d. This code produces a compilation error
e. None of the above.
```

2. What is the value of `y` after this code segment is implemented?

```
unsigned long int y;
const char* s= "012345678";
const char* sp = &s[10];
y = s - sp;
```

`a. garbage`
```
b. -9
c. 10
d. None of the above.
```
Note: It is illegal to assign a negative value to n unsigned integer. The answer is garbage because of writing a negative number (-10) into an unsigned long integer.

3. What if anything is wrong with the following code segment?

Line 1	`char *code = "ASCII";`
Line 2	`char* s;`
Line 3	`s = code;`
Line 4	`code[1] = s[+1];`
Line 5	`s[1] = *code;`

`a. There will be a runtime error(s) in this code segment`
```
b. There is a compilation error on the fourth line of this code segment
c. There is a compilation error on the fifth line of the code segment
d. None of the above
```

4. What is the output of the following C program:

```
#define WINDOWS
int main()
{
   char platform[10];
#ifndef MAC
   strcpy(platform, "LINUX");
#elif !defined(LINUX)
   strcpy(platform,"MAC");
#else
   strcpy(platform, "WINDOWS")
#endif
   printf("%s", platform);
  return 0;
}
```

```
a. MAC
b. WINDOW
```
`c. LINUX`
```
d. LINUXMAC
e. None of the above
```

5. Consider the following definition of `struct point` and the next code segment in C:

```
typedef struct Point{double x, y;}Point;
 Point x = {.y = 200};
 Point y = {.x = 900};
 Point z = y = x;
```

What is the value of `z.x` and `z.y;`

a. z.x is 0.0 and z.y is 200.0
b. z.x is 900.0 and z.y is 200.0
c. z.x is 900.0 and z.y is 900.0
d. z.x is 200.0 and z.y is 200.0
e. None of the above are correct answers.

6. Consider the following C code segment:

```
1  char *p;
2  p = (char*) malloc(20);
3  assert(p != NULL);
4  p += 3;
5  free(p);
```

a. There is an illegal operation on line 2.
b. There is an illegal operation on line 4.
c. There is an illegal operation on line 5.
d. There is no illegal operation(s) in this code.

7. What is the output of the following code segment:

```
char s1[] = "ABCDEF";
char* p = s1;
while(*p){
    (*p)++;
    p++;
}
printf("%s\n", s1);
```

Output is:
a. ABCDEF
b. BCDEFG
c. FEDCBA
d. None of the above

8. Consider the following C code segment:

```
1  int *p;
2  int n = 5;
3  p = (int*) malloc(n);
4  assert(p != NULL);
5  p[3] = 1000;
```

a. There is an illegal operation on line 3.
b. There is an illegal operation on line 4.
c. There is an illegal operation on line 5.
d. There is no illegal operation in this code.

9. What is the output of the following C program?

```
void foo() {
  static int s = 5;
  printf("%d", ++s);
}
```

```
int main() {
  for(int i =0; i < 3; i++)
     foo();
  return 0;
}
```

a. 678
b. 555
c. 567
d. 666
e. None of the above

10. What is the output of the following code segment?

```
char s1[20] = "Barlow";
char s2[]= "Victoria";
const char* p = "ABC";
printf("\n%2d %2d %2d %2d  %2d", (int)sizeof(s1), (int)sizeof(s2),
           (int)sizeof(int), (int)sizeof(p), (int) sizeof(*p));
```

a. 7 9 4 8 4
b. 20 9 4 8 1
c. 20 3 4 8 1
d. 6 8 4 8 1
e. 20 9 4 8 4
f. None of the above

SECTION 2 – Short Answer Questions – Total 6 marks:

In this section you will answer three questions.

1. **(2 marks)** Write a simple macro in C, called `RATIO`, that receives two numeric arguments, `x` and `y`, and returns the ratio of the smaller of the two over the greater one, or returns 1 if they have the same values.

   See the following examples that use macro `RATIO`:

```
printf("%f", RATIO(5, 10)); // prints 0.500000
printf("%f", RATIO(10, 5)); // prints 0.500000
printf("%f", RATIO(5, 5));  // prints 1.000000
```

Write your macro in the following space:

```
#define RATIO(x, y) ((x) <= (y) ? (double)(x) / (y) : (double)(y) / (x))
```

2. **(2 marks)** What is the output of the following program assuming that text file `numbers.txt` is located in the same working directory and contains the values in the following box:

```
126  456
333  abc  900
```

```
#include <stdio.h>

int main() {
    FILE* fp;
    int a[5] = {-1, -2, -33, 4, 500};
    char filename[20] = "numbers.txt";

    fp = fopen(filename, "r");
    if(fp == NULL){
        printf("File not found.");
        exit(1);
    }

    a[4] = fscanf(fp, "%d%d%d%d", &a[0], &a[1], &a[2], &a[3]);
    printf("%d  %d  %d  %d  %d", a[0], a[1], a[2], a[3], a[4]);

    return 0;
}
```

Write your answer in the following space:

```
126  456  333  4  3
```

3. **(2 marks)** Complete the following partial definition of the function `my_strlen,` using pointer arithmetic, so that it returns the length of its argument `str`:
Please deduct one mark for any mistake up to 2 marks.

```
int my_strlen(const char* str) {
      const char* p = str;
      while(*p) {

            ______p++___________________;

      } // end of while


      return ___P - str__;
```

SECTION 3 – Functions – 22 marks

In this section you will write four functions, and you are expected to consider an efficient code, when applicable.

Part a (4 marks): In the following space write the definition of a function that matches the following function prototype and its interface comment.

```
void write_data_squares(double * data, int n, char *filename);
/* REQUIRES: data points to an array of double with n elements, filename points to a
 * valid C-string that represents a file name.
 * PROMISES: It opens a text file with the given filename and writes the values of array
 * data and their squares into the file.(one value and its square per line, separated
 * with a single space. For example: 5.00 25.00). Don't worry about number of decimal places)*/
```

Deduct one mark for each error up to maximum of 4 marks

```
void write_data_squares(double * data, int n, char *filename) {
    FILE *fp = fopen(filename,"w");
    if(fp == NULL) {
        fprintf(stderr, "cannot be oppened...");
        exit(1);
    }

    int i = 0;
    while(i < n) {
        fprintf(fp,"%f     %f\n",  data[i], data[i]*data[i] );
        i++;
    }

    fclose(fp);
}
```

Part b (6 marks): In the following space write the definition of function called `all_prime_numbers`.

```
int all_prime_numbers (int* x, int n);
/*REQUIRES: x points to an integer array with n elements.
* PROMISES: returns true, if all the values in x[0], x[1], ..., x[n-2], and x[n-1] are
* prime numbers. Otherwise it returns false.
* For your information a prime number is a number that is greater than 1 and only
* divisible by 1 and itself. Here are a few prime number: 2 3 5 7 11 13 17 19
*/
// Solution #1 - There are many different ways to solve this problem. This is a good
// solution. Efficiency (speed and space) is number one concern in algorithm development
int all_prime(int a[], int n){

    for(int i = 0; i < n; i++){
        if(a[i] <= 1)
            return 0;
        for(int j = 2; j <= a[i]/2; j++) // note the last element must
            if(a[i]% j == 0)
                return 0;
    }
    return 1;
}
//-------------------------------------------------------------------------------------
Solution #2 - Another good solution
int all_prime3 (int a[], int n){

    for(int i = 0; i < n; i++){
        if(a[i] <= 1)
            return 0;
        int limit = a[i]/2;
        if(limit == 2) return 0;

        for(int j = 2; j < limit; j++)
            if(a[i]% j == 0)
                return 0;
    }
    return 1;
}

//-------------------------------------------------------------------------------------
// Solution #2 – Less Efficien. As value of n increase the impact of efficiency becomes
// more significant.
int all_prime(int a[], int n){

    for(int i = 0; i < n; i++){
        if(a[i] <= 1)
            return 0;
        for(int j = 2; j <a[i]; j++)
            if(a[i]% j == 0)
                return 0;
    }
    return 1;
}
```

Part c (6 marks) – In the following space write the definition of a function that matches the given function prototype and its interface comment. **Note:** You are **NOT** allowed to use any of the C library functions in your solution:

```
int largest_digit(char *S);
/* REQUIRES: S points to the beginning of a valid C-string.
 * PROMISES: if any character within the string pointed by S, is not a digit
 * returns -1. Otherwise returns the position (index number) of the largest digit in the
 * string. If there are more than one occurrence of a digit that is the largest in the
 * string, returns the position of the first one. SEE THE EXAMPLES:
 * largest_digit ("8012970");// should return 4, the index number of '9'
 * largest_digit ("8801270");//should return 0, the index number of the first occurrence of'8'
 * largest_digit ("8F2"); // should return -1 because the second character is 'F' */
```

<u>**One Possible Solution:**</u>

```
int largest_digit(const char* s) {
    int pos = 0;
    char largest = s[0];

    for(int i = 0; s[i] != '\0'; i++) {
        if(s[i] < '0' || s[i] > '9') return -1;

        if(s[i] > largest) {
            pos = i;
            largest = s[i];
        }
    }
    return pos;
}
```

Part d (6 marks): Complete the definition of a function `extract_digits` that receives a pointer to a valid C-string, called `str`, and a pointer to a **character-array** called `arr`. If any character in `str` is a digit (`'0' to '9'`), the function copies them into `arr`, and adds a `'\0'` to make it a valid C-string. The function should return the number of digits copied into `arr`.
EXAMPLE: If `str` points to string `"AB98x7$"`, characters stored in `arr[0]`, `arr[1]`, and `arr[2]`, `arr[3]` after a call to this function should be respectively: `'9'`, `'8'`, `'7'`, and `'\0'`, and the return value should be 3.
Note1: If there are no digits in `str`, function should return zero and `arr[0]` should be `'\0'`.
Note2: You are not allowed to call any C library function in this function.
<u>You can assume that number of elements in `arr` is always greater than the length of `str` (no error checking is required).</u>

<u>**One possible solution:**</u>

```
int extract_digits(const char* str, char *arr) {
    int counter = 0;
    for(int i = 0 ; str[i] != '\0'; i++) {
         if((str[i]) >= '0' && str[i] <= '9') {
           arr[counter] = str[i];
           counter++;
         }

    }
    arr[counter] = '\0';
    return counter;
}
```

SECTION 4 – C Structures – Total 12 marks

Consider the definition of the following C structures and the given C program.

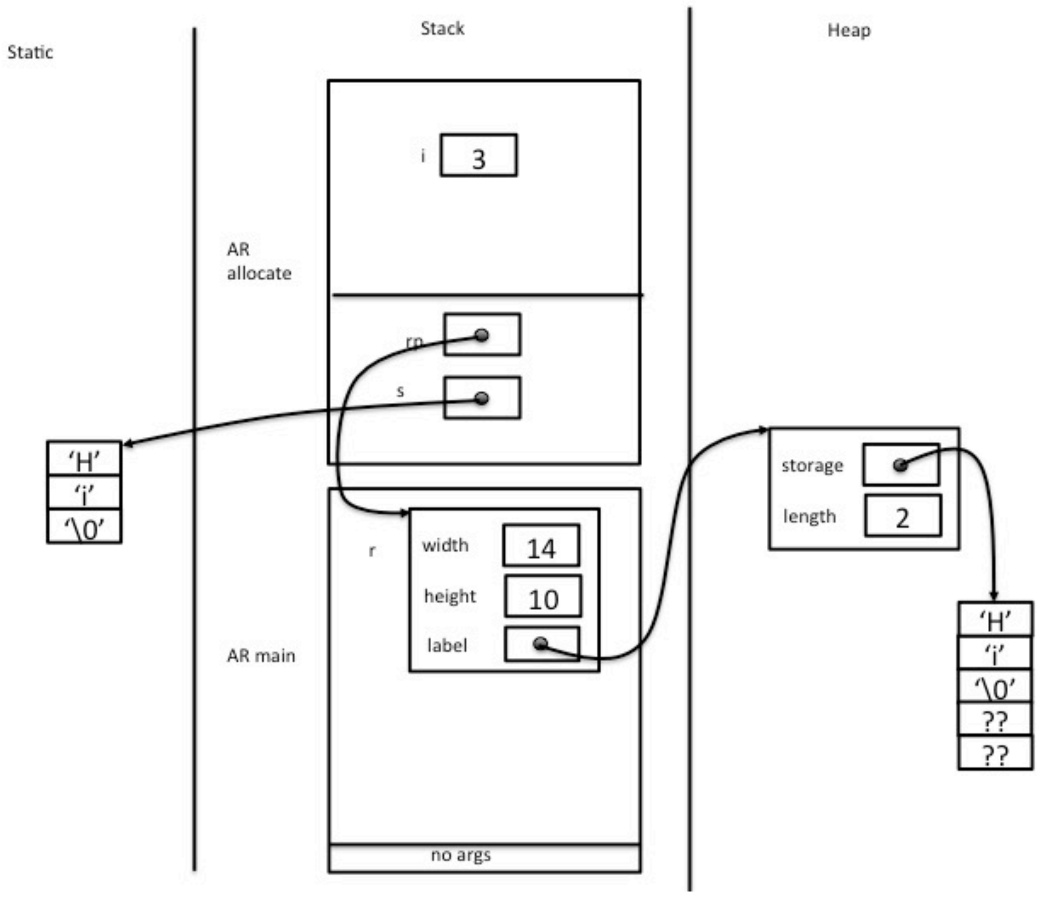
```
#define SIZE 5
typedef struct String{
    char* storage;
    int length;
} String;
```

```
typedef struct Rectangle{
    double width, height;
    String *label;
}Rectangle;
```

```
void allocate(Rectangle* rp, const char *s){
  int i;
  rp -> label = (String*) malloc(sizeof(String));
  rp->label->storage = (char*) malloc(SIZE);

  for(i = 0; i <= strlen(s); i++)
      rp->label->storage[i] = s[i];

  rp->label->length = i-1;
    // Point one
}
```

```
int main()
{
  Rectangle r;
  r.width = 14;
  r.height = 10;
  allocate(&r , "Hi");

  return 0;
}
```

Part I (10 marks) - In the following space, draw an AR diagram for **point one** in the function `allocate`:



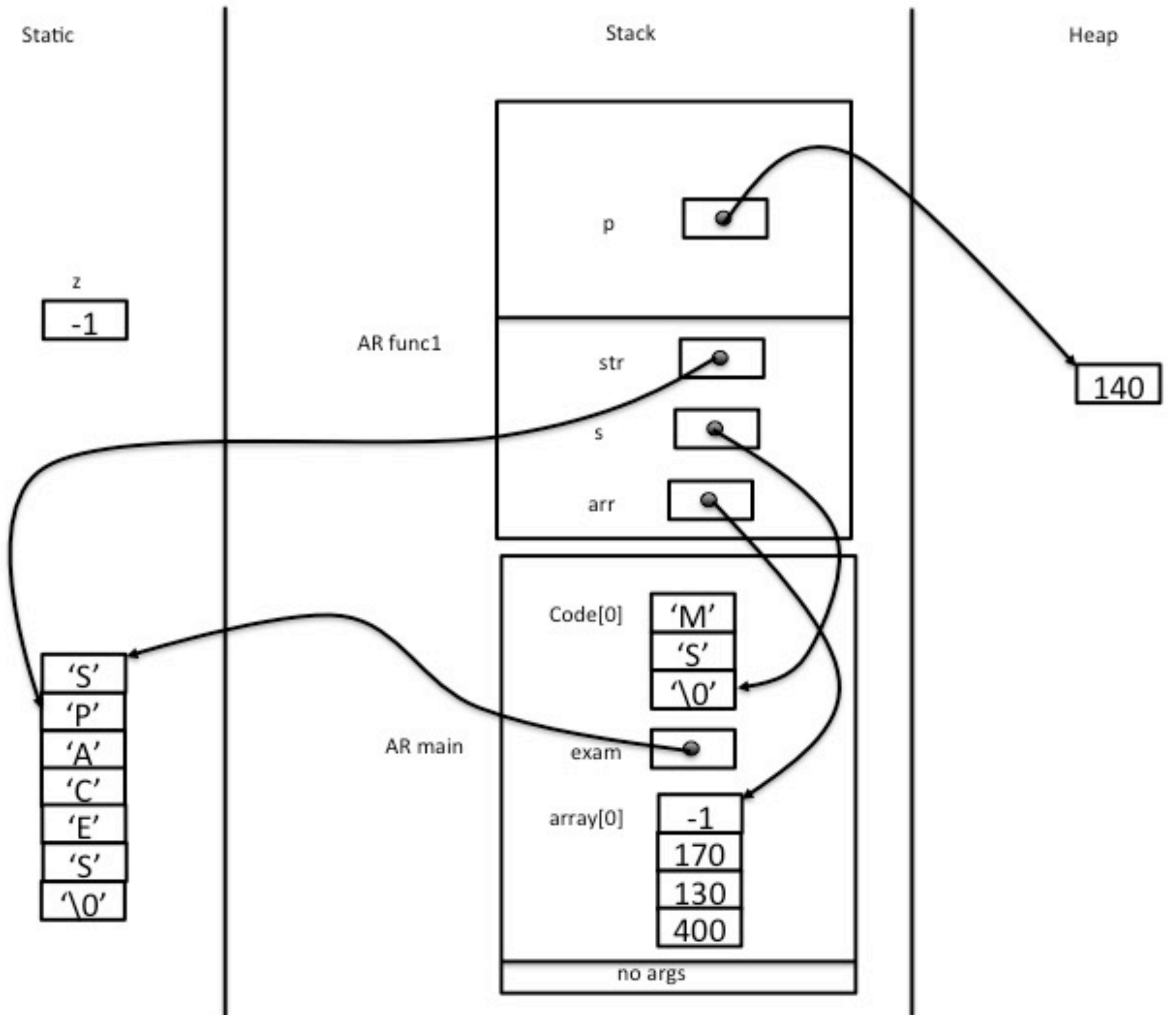
Part II (2 marks) - Complete the two missing lines in the following definition of the function `deallocate`, that is supposed to free (de-allocate) any memory that is dynamically allocated by function `allocate` in part I.

```
void deallocate(Rectangle* rp){
    free(rp ->label ->storage); // also ok if: free((char*)rp ->label ->storage);
    free(rp -> label);          // also ok if: free((String*)rp -> label);
}
```

SECTION 5 – Arrays and Pointers (10 marks):
Consider the definition of the following program and draw a memory diagram for point one:

```
void func1(const char *str, char s[], int arr[]);

int z = -1;

int main(void){
    char code[] = "MA";
    const char* exam = "SPACES";
    int array[] = {140, 170, 130, 400};
    func1(&exam[1] , code + 2, array);
    return 0;
}

void func1(const char *str, char s[], int arr[]) {
    s[z] = *(str-1);

    int *p = malloc(sizeof (int));
    *p = *arr;
    *arr = z;
    // point one
}
```

Draw the diagram for point one in the following space:



Reference to a few C library file I/O functions:

```
FILE * fopen ( const char * filename, const char * mode );
```
Opens the file whose name is specified in the parameter *filename* and associates it with a stream that can be identified in future operations by the FILE pointer returned. If the file is successfully opened, the function returns a pointer to a FILE object that can be used to identify the stream on future operations.
Otherwise, a null pointer is returned.

```
int fprintf ( FILE * stream, const char * format, ... );
```
Writes the C string pointed by *format* to the *stream*. If *format* includes *format specifiers* (subsequences beginning with %), the additional arguments following *format* are formatted and inserted in the resulting string replacing their respective specifiers.

```
int fscanf (FILE * stream, const char * format, ...);
```
Reads data from the *stream* and stores them according to the parameter *format* into the locations pointed by the additional arguments. On success, the function returns the number of items of the argument list successfully filled. This count can match the expected number of items or be less. If end of file is reached, returns EOF.

```
int fclose ( FILE * stream );
```
Closes the file associated with the *stream* and disassociates it.

Your Name: **Your Student ID Number:**

Please don't write anything in this box – this box is for marking

Section 1	Section 2	Section 3	Section 4	Section V	Total
/10	/6	/22	/12	10	/60