PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Applied Science and Engineering
Division of Engineering Science
FEBRUARY 2024

ESC 190 H1S

Duration: 1 hour and 50 minutes

Aids Allowed: Aid sheet distributed with the test paper.

Student Number: |_|_|_|_|_|_|_|_|_|_|

Given Name(s): _____

Family Name(s): _____

UTORid: _____

mail.utoronto.ca email: _____

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read *carefully* every reminder on this page.

- Fill out your name and student number above—do it now, don't wait!
- Fill out your student number on every odd-numbered page. Failure to do so will result in a 5% penalty.
- This test consists of 6 question on 14 page (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete.*
- Answer each question directly on the test paper, in the space provided. If you need more space for one of your solutions, use one of the "blank" pages and *indicate clearly the part of your work that should be marked.*
- Comments and docstrings are not required. Comments may help us mark your code. We mark your code and not your comments.
- You may use functions written to answer one question as part of the solution to another question.
- Write your answers in C (unless the question specifies otherwise). You may `#include` any header file that is available with a standard installation of `gcc` or is described in the C99 standard, **unless otherwise specified**. We will accept code that is correct under the C99 standard as well as any code that will run correctly when compiled with `gcc`.
- For full credit, your code should run correctly without crashing. We will not deduct points for memory leaks. Unless otherwise specified, you will not be penalized if your code runs but is not efficient.

MARKING GUIDE

Nº 1:	_____/15
Nº 2:	_____/ 5
Nº 3:	_____/ 7
Nº 4:	_____/10
Nº 5:	_____/10
Nº 6:	_____/ 3
TOTAL:	_____/50

Good Luck!

Question 1. [15 MARKS]

Part (a) [5 MARKS]

Complete this code so that the value of `a` becomes 42.

SOLUTION

```
void f(int *p)
{
    *p = 42;
}
```

MARKING SCHEME:

- **Function definition:** [1 mark]:
- **Correct parameter type** [2 marks]:
- **Correct dereferencing of pointer** [2 marks]:

```
// Complete here: write the function f
```

```
int main()
{
    int a = 0;
    int *p = &a;
    f(p); // a should now be 42
    return 0;
}
```

Part (b) [5 MARKS]

Write a function that computes the sum of the odd elements of an array of integers.

SOLUTION

```
int sum_odd(int *arr, int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        if (arr[i] % 2 != 0)
        {
            sum += arr[i];
        }
    }
    return sum;
}

// call: sum_odd(arr, n);
```

MARKING SCHEME:

- **Function signature and call** [1 mark]:
- **Loop** [2 marks]:
- **Implementation details** [2 marks]:

```
// Complete here: write the function sum_odd
```

```
int main()
{
    int arr[] = {4, 3, 2, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("%d\n", sum_odd(             )); // complete the call.
                                            // should print 9,
                                            // since 3 + 1 + 5 = 9
    return 0;
}
```

Part (c) [5 MARKS]

Complete this code so that the values of `arr` are printed in increasing order, separated by commas, but so that the values of `arr` remain unchanged. Your code should work for any values of elements of `arr`.

SOLUTION

```
void g(int *arr, int n)
{
    int *temp = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++)
    {
        temp[i] = arr[i];
    }
    qsort(temp, n, sizeof(int), cmp);
    for (int i = 0; i < n; i++)
    {
        printf("%d", temp[i]);
        if (i < n - 1){
            printf(", ");
        }
    }
    free(temp);
}
```

MARKING SCHEME:

- **Overall idea is correct** [2 marks]:
- **Overall correct idea AND array is unchanged at the end** [2 marks]:
- **Implementation details** [1 mark]:

```
// Complete here: write the function g
```

```
int main()
{
    int arr[] = {4, 2, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    g(arr, n); // should print out 2, 3, 4
               // but arr is still {4, 2, 3}
```

```
    return 0;
}
```

You may find the following useful:

```
int cmp(const void *a, const void *b) // don't need to copy this if used
{
    return (*(int*)a - *(int*)b);
}



int main()
{
    int arr[] = {4, 2, 3, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    qsort(arr, n, sizeof(int), cmp);
    return 0;
}
```

Question 2. [5 MARKS]

Write a function with the signature `int last_occurrence(char *str1, char *str2)` that takes in two C strings, `str1` and `str2`, and returns the location in `str1` where the *last* occurrence of `str2` starts. If `str2` is not found in `str1`, the function should return -1.

For example,

```
char str1[] = "EngSciEngSciTrackOne";
char str2[] = "EngSci";
int res = last_occurrence(str1, str2); // res should be 6, since the last
                                       // occurrence of "EngSci" starts at index 6

int last_occurrence(char *str1, char *str2)
{
```

SOLUTION

```
int last_occurrence(char *str1, char *str2)
{
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int res = -1;
    for (int i = 0; i < len1; i++)
    {
        if (strncmp(str1 + i, str2, len2) == 0)
        {
            res = i;
        }
    }
    return res;
}
```

MARKING SCHEME:

- **Overall idea is correct** [3 marks]:
- **Implementation details** [2 marks]:

Question 3. [7 MARKS]

Write a **recursive** function that takes in the strings `str1` and `str2` and returns the location in `str1` where the *first* occurrence of `str2` starts. If `str2` is not found in `str1`, the function should return -1.

You are **not** allowed to use any loops in your solution. You are **not** allowed to use any global variables. You are **not** allowed to use any external libraries.

You are allowed to use helper functions that you implement yourself, as long as they do not use loops, global variables, or external libraries.

You are allowed to define additional parameters for your function.

```
char str1[] = "aEngSci EngSci TrackOne";
char str2[] = "EngSci";
// the recursive function you define should return 1, since that is the
// index where the first occurrence of "EngSci" starts
```

SOLUTION

```
int first_occurrence(char *str1, char *str2, int i, int j)
{
    if (str1[i] == '\0' || str2[j] == '\0')
    {
        return -1;
    }
    if (str1[i] == str2[j])
    {
        if (str2[j + 1] == '\0')
        {
            return i - j;
        }
        return first_occurrence(str1, str2, i + 1, j + 1);
    }
    return first_occurrence(str1, str2, i + 1, 0);
}
```

MARKING SCHEME:

- **Overall idea is correct** [2 marks]:
- **Base case** [2 marks]:
- **Recursive step** [2 marks]:
- **Correct return value** [1 mark]:

Question 4. [10 MARKS]

You are given a file in the following format. On each line, the first number is a 10-digit student number. It is then followed by numbers representing marks. The number of marks is not fixed.

```
1234560789 25 75 50
9874512541 12 13 14 15 16
4564789145 7 6 5
```

Write a function that takes in the name of the file, and prints the student numbers of all the student(s) whose average mark is the highest. (There can be one or more such student).

You can assume all marks are whole numbers between 0 and 100, and that there are at most 10 marks per student. The student number and the marks are separated by a single space. The file is not sorted in any way. Assume at least one student is in the file, and assume that every student has at least one mark. Assume there are `"\n"`s after every line in the file. You cannot assume that the number of students is fixed.

You may find code from Q1 to be indirectly useful.

SOLUTION

```
// use while loops NOT strtok

typedef struct student{
    char student_number[11];
    int marks[10];
} student;

void read_in_student_from_str(char *str, student *s){
    int i = 0;
    while(str[i] != ' '){
        s->student_number[i] = str[i];
        i++;
    }
    s->student_number[i] = '\0';
    i++;
    int j = 0;
    while(str[i] != '\n'){
        char num[4];
        int k = 0;
        while(str[i] != ' ' && str[i] != '\n'){
            num[k] = str[i];
            k++;
            i++;
        }
        num[k] = '\0';
        s->marks[j] = atoi(num);
        j++;
        if(str[i] == ' '){
            i++;
        }
    }
}

void read_in_student_from_file(FILE *f, student *s){
```

```
    char str[100];
    fgets(str, 100, f);
    read_in_student_from_str(str, s);
}

void read_in_all_students_from_file(char *filename, student *s, int *num_students){
    FILE *f = fopen(filename, "r");
    int i = 0;
    while(!feof(f)){
        read_in_student_from_file(f, &s[i]);
        i++;
    }
    *num_students = i;
    fclose(f);
}

void highest_average(char *filename){
    student s[100];
    int num_students;
    read_in_all_students_from_file(filename, s, &num_students);
    int highest_average = 0;
    for(int i = 0; i < num_students; i++){
        int sum = 0;
        for(int j = 0; j < 10; j++){
            sum += s[i].marks[j];
        }
        int average = sum / 10;
        if(average > highest_average){
            highest_average = average;
        }
    }
    for(int i = 0; i < num_students; i++){
        int sum = 0;
        for(int j = 0; j < 10; j++){
            sum += s[i].marks[j];
        }
        int average = sum / 10;
        if(average == highest_average){
            printf("%s\n", s[i].student_number);
        }
    }
}
```

MARKING SCHEME:

- **Overall idea is OK (be generous here)** [2 marks]:
- **Read in one student from a string (mark details)** [2 marks]:
- **Read in all students from a file (mark details)** [2 marks]:
- **Find the highest average (mark details)** [1 mark]:
- **Put everything together (mark details)** [3 marks]:

Question 5. [10 MARKS]

A linked list structure is defined as follows:

```
typedef struct node{
    int data;
    struct node *next;
} node;

typedef struct LL{
    node *head;
} LL;
```

Part (a) [2 MARKS]

Write a function that returns the length of a linked list. **Solution**

```
int length(LL *l){
    int len = 0;
    node *n = l->head;
    while(n != NULL){
        len++;
        n = n->next;
    }
    return len;
}
```

MARKING SCHEME:

- **Correctly traverses the linked list** [1 mark]:
- **Returns the right thing** [1 mark]:

Part (b) [8 MARKS]

Write a function that returns the median data point of a linked list. You *must* use `qsort` to accomplish this. Assume the linked list has an add number of elements, and they are all distinct.

Solution

```
void linked_list_to_array(LL *l, int *p_arr){
    int len = length(l);
    *p_arr = (int *)malloc(len * sizeof(int));
    node *n = l->head;
    int i = 0;
    while(n != NULL){
        arr[i] = n->data;
        i++;
        n = n->next;
    }
}

int median(LL *l){
    int len = length(l);
    int *arr;
    linked_list_to_array(l, &arr);
```

```
    qsort(arr, len, sizeof(int), compare);
    ans = arr[len/2];
    free(arr);
    return ans;
}
```

MARKING SCHEME:

- **Correct idea (be generous here)** [2 marks]:
- **Correctly converts linked list to array** [2 marks]:
- **Correctly calls qsort** [2 marks]:
- **Correctly calculates median** [2 marks]:
- **Putting everything together** [2 marks]:

Question 6. [3 MARKS]

Write a function that returns a string that contains all possible strings that use characters from a given alphabet, of length k. The strings should be separated by a newline character. The function signature is:
`char *generate_strings(char *alphabet, int k)`

For example, `generate_strings("abc", 2)` should return the string: `"aa\nab\nac\nba\nbb\nbc\nca\ncb\ncc\n"`
The order of the strings inside the returned string doesn't matter.

Solution

```
char *generate_strings(char *alphabet, int k){
    int num_strings = pow(strlen(alphabet), k);
    char *ret = (char *)malloc(num_strings * (k+1) * sizeof(char));
    int i;
    for(i = 0; i < num_strings; i++){
        int j;
        int temp = i;
        for(j = 0; j < k; j++){
            ret[i*(k+1) + j] = alphabet[temp % strlen(alphabet)];
            temp /= strlen(alphabet);
        }
        ret[i*(k+1) + k] = '\n';
    }
    return ret;
}


int main(){
    char *s = generate_strings("abc", 5);
    printf("%s", s);
    free(s);
    return 0;
}
```

MARKING SCHEME:

- **Basic idea – only give if there is actual progress with the basic ideas in place** [1 mark]:
- **Substantial progress, but not quite there** [1 mark]:
- **Almost perfectly or perfectly correct answer** [1 mark]:

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.