Department of Computer Science and Engineering
Class Test - 2, Section - 16
Subject : Programming and Data Structure (CS19003)

Date: 4th February 2022 Time: 9:30 AM to 12:00 Noon Marks: 100

Instructions for :

- Give the name of the programs files as <Your roll>_<test number>_<question number>.c. For example, 21XX12345_T1_Q1.c for question 1 of test 1 of a student with roll 21XX12345. **Follow the file naming convention strictly**.
- Apart from the main .c file for each program, you should also upload one additional temporary .c file for each program (such as when you have finished half of the code). The naming for the temporary file should be in the format <Your roll>_<test number>_<question number>_temp.c. For example, 21XX12345_T1_Q1_temp.c **Make sure that your main code do not deviate much from its temporary code for each program**.
- You should upload the main .c file and the temporary .c file individually to the Moodle course web page once you finish answering each question. No need to zip the files.
- The **deadline** to upload the programs is 12:00 Noon. Beyond that, submission will be closed (No extensions will be granted).

---

1. Write a C program that stores n integer values in a 1D array from the user and search for a key value in the array. If the key value is present in the array, shift the elements present on the right of the key by one position towards the left and finally store the key value at the end of the array. Otherwise, display "The key element not found!". The key should also be taken as input.

   Example 1:

   Input:
   ```
   Enter the value of n: 5
   Enter the values in the array:
   1
   6
   2
   7
   3
   Enter the key value: 6
   ```

   Output:
   ```
   1 2 7 3 6
   ```
   [15 Marks]

2. A set can be represented programmatically by a 1D array of elements, where no repetition is permitted. Now let us consider that there are n different sets, each having a maximum cardinality of m elements. This can be represented by an nXm 2D character array, where each row of the array represents one of n sets and each set (i.e each row) has a maximum size of m. Declare

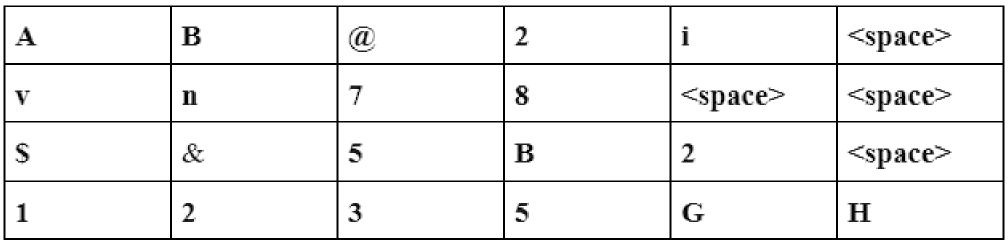
such a 2D character array taking the values of 'n' and 'm' from the keyboard. Hence fill up the array by filling up each set (i.e each row) with the respective set elements. You may consider that the valid set elements contain only printable ASCII characters, i.e A-Z, a-z, 0-9 and special characters. Note that two sets (i.e any two rows of the 2D array) may not be of the same size. Each set can contain any number of elements less than or equal to 'm' (the maximum cardinality of each set) **including zero elements** (the null set). The **space character** (ASCII value = 32) is used to denote an empty cell of the 2D array which contains no element. Thus, for each set (i.e each row), if the set contains less than 'm' elements, then all the cells in that row are filled with **space** after the end of that set (i.e after the last valid element of that set). Write the following C functions to perform the following operations on any two sets, where the 2D array A[][] denotes the collection of all the sets. These functions are **not recursive**.

- void Union (char A[ ][m], int i, int j): To display the union of two sets A[i] and A[j].
- void Intersection (char A[ ][m], int i, int j): To display the intersection of two sets A[i] and A[j].
- void Difference (char A[ ][m], int i, int j): To display the difference of two sets A[i] and A[j] (A[i] - A[j]).

In the main() function, the user should insert the values of n and m and fill up the 2D array with set elements. Following this, the user should insert the value of i and j representing the ith and jth set and call the respective functions as mentioned above to perform the set operations and display the output within that function itself. In case the output of the set operation results in a null set, print "**Null set**" as output for that particular function call.

Example:

The following 5 x 6 array represents 5 sets where each set can store at most 6 elements.



A	B	@	2	i	<space>
v	n	7	8	<space>	<space>
$	&	5	B	2	<space>
1	2	3	5	G	H

Now for example i = 1 (the 1st set) and j = 3 (the 3rd set), thus the output of the set operations will be,

Union: [**A, B, @, 2, i, $, &, 5**]

Intersection: [**B, 2**]

Difference: [**A, @, i**] **[25 Marks]**

3. Write a menu-driven C program that declares two structures named :

   a. Circle - having three integer members, radius r, x, and y, where (x,y) represents the center.

   b. Triangle - having *perpendicular*, *hypotenuse*, and *base* as the members of type integer

   Perform the following operations according to the functions as defined below.

   1. check_circle(struct Circle c1, struct Circle c2)- Check if the two circles intersect or touch each other or not. If they intersect then return 2. If both the circles touch each then return 1, otherwise 0.

      [**Formula**: Firstly, we calculate the distance between centers C1 and C2 as
      dist = $sqrt((x1 - x2)^2 +$ (y1 - y2)^2).

R1 and R2 are the radii of the circles.

Next, we check the following three conditions.

1. If dist == R1 + R2

   Circle A and B are touching each other.

2. If $dist > R1 + R2$

   Circle A and B are not touching each other.

3. If $dist < R1 + R2$

   Circles are intersecting each other.]

2. check_triangle(struct Circle c1, struct Triangle t1) - Firstly, check if the triangle is a right angle triangle or not. If it is not a right angle triangle, then returns 2. Otherwise, check if the area of the circle is equal to that of the incircle of the triangle or not after rounding off their respective values. If the area of the circle is equal to the area of the incircle, then returns 1, otherwise 0.

   [**Formula**: the area of the incircle will be $\pi * ((P + B – H)/2)^2$, where P, B, and H represent perpendicular, base, and hypotenuse respectively.]

   [**Display**: If 2 is returned, display "Not a right angle triangle" If 1 is returned, display "The area of the circle is equal to the area of the incircle of the triangle" If 0 is returned, display "The area of the circle is not equal to the area of the incircle of the triangle"]

   Your program must keep on running continuously until the user inputs 'n'. After each computation, prompt the user to continue again or not.

Example:

Input:

– – – – – – – – – – –MENU– – – – – – – – – – –

1. Check circles
2. Check triangle

Enter your choice: 1

Enter radius, x, and y coordinate values for the circle A: 4 2 3

Enter radius, x, and y coordinate values for the circle B: 7 15 19

Output:

Circle A and B are not touching each other.

Do you want to continue: y

Input:

– – – – – – – – – – –MENU– – – – – – – – – – –

1. Check circles
2. Check triangle

Enter your choice: 2

Enter the values of p, b, h of the triangle : 3 4 5

Enter the radius value of the circle: 3

Output:

The area of the circle is not equal to the area of the incircle of the triangle.

Do you want to continue: n **[25 Marks]**

4. Write a C program that performs the following operations. It prints 'n' rows of binary bit patterns ('0's and '1's) by following the rules as given below.

   (a) Consider that the first row always starts with value 0.

   (b) In each subsequent row, considering the previous row, replace each occurrence of 0 in the previous row with 01 in this row, and each occurrence of 1 in the previous row with 10 in this row. Continue like this for every subsequent row till you have 'n' rows. Consider the example given below to understand this concept clearly.

   Write a program to input the value 'n' and print the n different rows. You must use a **recursive C function** *int theithSymbol()* to generate each successive row from the previous one. You are free to use any additional data variables and arrays (1D array, 2D array etc) that you may wish. You are also free to choose whether you would print the i'th row within the recursive function, or you would print the i'th row in main() and the recursive function will only return the i'th row output to main() (say for example by filling up an array representing the i'th row). But you **must** ensure that the logic that you use to get each successive row from the previous one is recursive. The order of arguments of the recursive function is also your choice.

   Example:

   Input:

   Enter the value of n: 5

   Output:

   Value of Row 1: 0

   Value of Row 2: 01

   Value of Row 3: 0110

   Value of Row 4: 01101001

   Value of Row 5: 0110100110010110

   [Note how each row is being generated from the previous row by replacing 0 with 01 and 1 with 10. The first row contains 0 by default] **[35 Marks]**
