Indian Institute of Technology Kharagpur EXAMINATION ANSWERSCRIPT	
	Stamp/Signature of the Invigilator

Spring Semester 2025-26 – Mid Sem Exam	SEMESTER (Spring)

Roll Number										Section		Name	
Subject Number	C	S	1	0	0	0	3		Subject Name	Programming and Data Structures			
Department/Centre/School										Additional Sheets			

Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession; even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, the exchange of these items or any other papers (including question papers) is not permitted.
6. Write on both sides of the answer-script and do not tear off any page. **Use designated places of the answer-script for rough work**. Report to the invigilator if the answer-script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the Examination Hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer-script to the invigilator. **In any case, you are not allowed to take away the answer-script with you**. After the completion of the examination, do not leave your seat until the invigilators collect all the answer-scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging information with others or any such attempt will be treated as '**unfair means**'. Don't adopt unfair means and also don't indulge in unseemly behavior.

Violation of any of the above instructions may lead to severe punishment.

Signature of the Student

To be Filled by the Examiner

Question Number	1	2	3	4	5	6	7	8	9	10	Total
Marks Obtained											

Marks Obtained (in words)	Signature of the Examiner	Signature of the Scrutineer

ADDITIONAL INSTRUCTIONS

1. **No clarification will be provided during the exam. If you make any ADDITIONAL assumptions in a question, write it clearly beside the question.**

2. **Write your answers only within the boxes provided (for Question 1) and in the blanks provided (for Questions 2 to 7). Answers written anywhere else will not be evaluated.**

3. **Do rough work only in the spaces clearly marked for rough work. Do not do rough work anywhere else.**

4. **Write final answers with pen only. No answers written in pencil will be evaluated.**

DO NOT WRITE ANYTHING IN THIS PAGE

Q1. Answer the following questions as directed. Write the answer inside the box provided below each question. (8 × 1 = 8)

(a) What is the value of the following expression if *x* = 5, *y* = 2, and *z* = 5? *x, y, z* are all integer variables.

```
x + z*-y/(x%2)
```

ANS: -5

(b) Let *i* be an integer variable. How many times will the loop condition (*i* < 100) be checked in the following C code fragment?

```
for (i = 0; i < 100; i = i + 3)
        printf ("%d", i++);
```

ANS: 26

(c) Which of the following logical expressions is/are equivalent to the expression

```
( !((x <= y) && (y <= z)) )
```

(*x*, *y* and *z* are variables)? Write only the correct choice no(s) (A, B, C, D) in the space provided.

(A) `((x > y) && (y > z))`
(B) `((x > y) || (y > z))`
(C) `( !(x <= z) )`
(D) `(x >= z)`

ANS: **B**

(d) Let *m* and *n* be two integer variables, If m = 0 and n = 13643, what will be the value of *m* after the following C code fragment is executed?

```
while (n > 0)
{
      m = (m*10) + (n%10);
      n = n/10;
}
```

ANS: 34631

(e) Let i, j, p be integer variables. Suppose that all of them are initialized to 0. What will be the value of p after the following C code fragment is executed?

```
while ( i < 10 ) {
   for ( j = 0; j < 5; ++j )
      if ( i != j ) ++p;
   ++i;
}
```

ANS: 45

(f) What will be the value of A[4] after the following code fragment is executed?

```
int A[5] = {1, 6, 8, 4, 5}, i, j, temp;
for (i = 0; i < 5; i++)
   for (j = i + 1; j < 5; j++)
      if (A[i] > A[j]) {
         temp = A[i];
         A[i] = A[j];
         A[j] = temp;
      }
```

ANS: 8

(g) What will be printed after the following code fragment is executed?

```
int change(int i) {
   i = i + 2; return i;
}
int main() {
   int i = 0, j = 0;
   while (i < 10)  i += change(j);
   printf("i=%d, j=%d\n", i, j);
   return 0;
}
```

ANS: i=10, j=0

(h) Consider the following function.

```
int g( int n )
{
        if (n < 7) return n;
        return g(n/2);
}
```

What will be the value returned by the call `g(346)`?

ANS: 5

GRADING GUIDELINE:

For all parts EXCEPT (g): 1 mark if correct, 0 otherwise (binary)

For part (g): 0.5 for value of i, 0.5 for value of j. "i=" and "j=" are ignored.

Q2. Given a sequence **S** of **distinct positive integers**, a *maximal increasing subsequence* of **S** is defined as a subsequence **R** of **consecutive** integers in **S** such that all of the following three conditions are true:

(i) the first integer in **R** is either the first integer in **S**, or is less than the integer just before it in **S**,
(ii) the last integer in **R** is either the last integer in **S**, or is greater than the integer just after it in **S**, and
(iii) all integers in **R** are in ascending order of value.

The length of a maximal increasing subsequence is the number of integers in it. For example, for the sequence **S** = <6 7 2 29 17 5 9 11 16 1 3 4>, the maximal increasing subsequences are: <6, 7> (of length 2), <2, 29> (of length 2), <17> (of length 1), <5, 9, 11, 16> (of length 4) and <1, 3, 4> (of length 3).

The following C program reads a sequence of **distinct positive integers** from the keyboard until the user types 0. It counts the lengths of all maximal increasing subsequences in the sequence of positive integers entered, and prints the maximum length among them. Thus, if the sequence shown above is entered, the answer should be 4 (the length of the subsequence <5, 9, 11, 16>). Assume that the user enters at least one positive integer before entering 0. Complete the program by filling in the blanks. (6)

```
#include <stdio.h>
int main ( )
{
          int prevno, curno, curlength, maxlength;
          scanf ("%d", &prevno);   scanf ("%d", &curno);

          curlength = maxlength = 1;                          [A]
          while ( curno != 0 ) {

                    if (curno > prevno) {                     [B]
                              curlength++;
                    } else {
                              if (curlength > maxlength ) {   [C]
                                        maxlength = curlength;
                              }
                              curlength = 1 ;                 [D]
                    }
                    prevno = curno ;                          [E]

                    scanf("%d", &curno );                     [F]
          }
          if ( curlength > maxlength ) maxlength = curlength;
          printf ("Maximum length is %d\n", maxlength) ;
          return 0;
}
```

GRADING GUIDELINES:

For all blanks except [A]: 1 mark if fully correct, 0 otherwise (binary)

For blank [A], ok if two separate assignments statements are written, one for curlength and one for maxlength. Also, ½ mark is given if (i) curlength is correctly initialized to 1 but maxlength is not initialized or wrongly initialized, OR (ii) both curlength and maxlength are initialized but wrongly (for understanding that both have to be initialized).

For blanks [B] and [C], other equivalent variations (such as “prevno < curno” in [B] and “maxlength <= curlength” in [C] etc. etc.) are accepted.

Q3. The following program counts the number of unique (distinct) integers present in a 1-d array holding positive integers only. For example, for an array {1,2,3,4,5,4,3,2} as input, the output is 5 (as there are five distinct integers, 1, 2, 3, 4, and 5, in the array). Complete the program by filling in the blanks. (5)

```
#include <stdio.h>
int main(){
        int arr[100];
        int unique_cnt=0, arr_size, i, j;
        /* Enter number of elements */
        scanf("%d", &arr_size);
        /* Enter the elements */
        for (i = 0; i < arr_size; i++) scanf("%d", &arr[i]);          [A]
        /* Count and print the number of unique elements */
        for (i = 0; i < arr_size; i++ ){
                if (arr[i] != -9999){

                        unique_cnt++;                                  [B]

                        for (j = i+1 ; j < arr_size; j++){             [C]

                                if ( arr[j] == arr[i]){                [D]

                                        arr[j] = -9999 ;               [E]
                                }
                        }
                }
        }
        printf("Count of unique numbers : %d", unique_cnt);
        return 0;
}
```

GRADING GUIDELINES:

For blank [A]: 1 mark if fully correct, ½ mark if "&" is missed but otherwise correct, 0 in all other cases

For blanks [B] and [E] each: 1 mark if fully correct, 0 otherwise (binary)

For blank [C]: 1 mark if fully correct, ½ marks if "<= arrsize" or "< arrsize – 1", 0 in all other cases.

For blank [D]: 1 mark if fully correct, ½ mark if "=" used instead of "==", 0 in all other cases.

Equivalent variations in blanks such as

[B] ("++unique_cnt", "unique_cnt +=1", "unique_cnt = unique_cnt + 1") etc.

[C] ("arr_size > j") etc.

[D] ("arr[i] == arr[j]") etc.

are accepted.

Q4. The following program reads a string (of length at most 50 characters) in the array S. Assume that the string entered contains only alphabetic letters, in either lowercase or uppercase (i.e. only characters in 'a' to 'z', and 'A' to 'Z'). The program then forms a new null-terminated string in the array R that contains all lowercase letters from the alphabet in order that are **not** present in S (in either lowercase or uppercase). The program finally prints R. For example, if S = "aBg", then R = "cdefhijklmnopqrstuvwxyz" (i.e. all lowercase letters in the alphabet other than 'a', 'b', and 'g', in the order in which they appear in the alphabet). Similarly, if S = "AbraCADAbRA", then R = "efghijklmnopqstuvwxyz" (i.e. all lowercase letters in the alphabet other than 'a', 'b', 'r', 'c', and 'd', in the order in which they appear in the alphabet). You **cannot** use any string functions or ascii code values of any character. (5)

```
#include <stdio.h>
int main()
{
        char S[100], R[100], temp;
        int i, j, X[26];
        scanf("%s", S);
        /* X will keep track if a letter is present in S */
        for(i = 0; i < 26; i++) X[i] = 0;
        /* Loop over S to search for letters and update X */

        for ( i = 0; S[i] != '\0'; i++ ) {                  [A]
                if (S[i] >= 'A' && S[i] <= 'Z')

                        temp = 'a' + S[i] - 'A';            [B]
                else
                        temp = S[i];

                X[ temp - 'a'] = 1;                         [C]
        }
        /* Write to R */
        j = 0;
        for ( i = 0; i < 26; i++ )
                if (X[i] == 0) {R[ j++ ] = 'a' + i;}         [D]

        R[j] = '\0';                                        [E]
```

```
                printf("R = %s\n", R);
        }
```

GRADING GUIDELINES:

For each blank: 1 mark if fully correct, 0 otherwise (binary).

For blank [D], ok if the j++ is done in a separate statement after assigning R[j].

For [B], [C], [D], 0 given if ascii code is used. This includes using the difference of two ascii codes in [B] (using 32 instead of 'a' – 'A'), as the difference can be computed only if the exact ascii codes are used. 1 mark (overall) is given at teacher's discretion if all three used ascii code values but are otherwise correct. (all copies checked by the same teacher for uniformity).

½ mark deducted if double-quote is used instead of single-quote for characters.

Equivalent variations wherever correct are accepted.

Q5. Consider a **recursive** function print_numbers() that takes the following three parameters: N (a positive integer), d1 (a single digit integer between 0 and 9), and d2 (a single digit integer between 0 and 9). Assume that d1 is not equal to d2. The function will recursively print all integers between 1 and N (including 1 and N) that contains only d1 and d2 (one or both, a digit can come more than once also in the number) in its digits. Some example sets of parameters and the corresponding output printed by the function are shown below.

Parameters: N = 15, d1= 1, d2= 5	Output: 15, 11, 5, 1
Parameters: N = 50, d1= 0, d2= 1	Output: 11, 10, 1
Parameters: N = 100, d1= 1, d2= 2	Output: 22, 21, 12, 11, 2, 1

Fill in the blanks below so that the function works correctly. (5)

```
void print_numbers(int N, int d1, int d2)
{
        int status_flag = 1;
        int temp = N, digit;
        if (N > 0) {
           while (temp > 0 && status_flag ==1 ) {

              digit = temp%10;                          [A]

              if (digit != d1 && digit != d2 )          [B]  [C]
                   status_flag = 0;

              temp = temp/10;                           [D]
           }
```

```
                if ( status_flag == 1 )                    [E]
                      printf("%d ", N);

                print_numbers(N - 1, d1, d2);              [F]
            }
    }
```

GRADING GUIDELINES:

For blanks [A], [D], [E] each: 1 mark if fully correct, 0 otherwise (binary)

For blank [B] and [C] each: 0.5 for each blank if correct, 0 otherwise (binary)

For blank [F]: 0.5 for calling print_numbers() fully correctly, 0.5 for passing correct parameters in correct order (binary).

For blank [E], Variations like "status_flag != 0" or "status_flag" are also ok.

Q6. Consider the mathematical function myFunc() which is recursively defined as follows: (5)

```
myFunc(m, n) = m,     if m = n
myFunc (m, n) = m,    if n = 1
myFunc (m, n) = 1,    if n = 0
myFunc (m, n) = myFunc (m-1, n) + myFunc (m-1, n-1), otherwise
```

Fill in the blanks below so that the recursive function myFunc() correctly implements (i.e., returns the correct value of) the mathematical function above.

```
int myFunc (int m, int n)                                   [A]
{
      if ( m==n || n==1 ) return( m );                      [B] [C]

      if ( n==0 ) return (1);                               [D]

      else return (myFunc(m-1,n) + myFunc(m-1,n-1));        [E]
}
```

GRADING GUIDELINES:

For each blank: 1 mark if fully correct, 0 otherwise (binary).

½ mark is deducted in each blank (subject to a maximum of 1 for the whole question) if "=" is used instead of "=="

Q7. Fill in the blanks so that the following C program prints a pyramid of digits (as shown below) for a given number of lines n (assume 0 < n < 10) to be read from keyboard. For example, for n = 5, the pyramid of 5 lines should be printed as (there are two blank spaces separating two consecutive numbers):

```
            1
         2  3  2
      3  4  5  4  3
   4  5  6  7  6  5  4
5  6  7  8  9  8  7  6  5
```

(6)

```
#include <stdio.h>
void printPattern(int n) {
        int i, j  ;
        /* for each line to print do */
        for (i = 1; i <= n; i++) {
                /* get to the start point to print. The printf has 3 spaces */

                for (j = i; j < n; j++) printf("   ");                      [A]
                /* print numbers in increasing order */

                for (j = 0;j < i; j++) printf(" %2d", i + j);               [B]  [C]
                /* print numbers in decreasing order */

                for (j = j-2; j >= 0; j--) printf(" %2d", i + j);           [D] [ E]

                printf("\n");                                              [F]
        }
        return;
}

int main() {
        int n;
        scanf("%d",&n);

        printPattern(n);                                                   [G]
        return 0;
}
```

GRADING GUIDELINES:

For blanks [F], [G] each: 1 mark if fully correct, 0 otherwise (binary)

Blanks [A],]B], and [D] has too many variations possible and blanks [C] and [E] depend on what is used in the for loops. But some things are constant in all the variations. For blank [A], the for loop has to go exactly (n – i) times. For blank [B] and [C], the loops have to go i times and (i – 1) respectively (or (i - 1) times and i times, depending on which loop prints the middle element). Grading guideline is as follows:

For blank [A]: 1 mark if the loop goes (n – i) times, 0 otherwise

For blank [B]: 1 mark if the loop goes i times (or (i – 1) times if [C] goes i times), 0 otherwise

For blank [D]: 1 mark if the loop goes (i – 1) times (or i times if [B] goes (i – 1) times), 0 otherwise

For blanks [C] and [E] each: 0.5 marks if fully correct, 0 otherwise (binary). Correct or not is determined by teacher based on the answers in [B] and [D] (all copies checked by the same teacher for uniformity).

SPACE FOR ROUGH WORK

SPACE FOR ROUGH WORK