INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Stamp / Signature of the Invigilator

EXAMINATION: Mid Semester (Autumn 2024-25)	**Answer all (Total marks: 60, Duration: 2 hours)**

Roll Number										**Section**		**Name**	

Subject Number	C	S	1	0	0	0	3	**Subject Name**	*Programming and Data Structures*

Department / Center of the Student		**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, exchange of these items or any other papers (including question papers) is not permitted.
6. Write on the answer-script and do not tear off any page. **For rough work, use last page(s) of the answer script and white spaces around the questions.** 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 the 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'**. Do not adopt unfair means and do not indulge in unseemly behavior.

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

Signature of the Student

To be filled in by the examiner

Question No.:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Marks Obtained											
Question No.:	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**Total**
Marks Obtained											

Marks obtained (in words)	**Signature of the Examiner**	**Signature of the Scrutineer**

1. The end of this question paper is marked by —— **END** ——
2. Answer all questions.
3. Write your answers in the specified **blanks**, using **pen only**.
4. You may do **Rough Work** in the designated areas or any white space, as long as it does not encroach or interfere with your answers.
5. If you use **Extra Sheets** for rough work, do not submit them.

---

1. Answer the following questions. **$1 \times 10 = 10$ marks**

   (a) Which one among the following data types takes the least space in memory and which one the most?
   `char`, `int`, `long int`, `float`, `long double`
   Answer: `char`, `long double`

   (b) `float z = 0.1225e2;`
   What's the smallest positive integer that should be multiplied with `z` so that the result is an integer?
   Answer: 4

   (c) Write the value of the expression `a == ((a*b+1 < c)&&(a+b==c))`, where `a = 1, b = 2, c = 3`.
   Answer: 0

   (d) Write the value of the expression `(a&b)&(c&17)`, where `a = 1, b = 2, c = 3`.
   Answer: 0

   (e) Write the value of the expression `(a|b)&(c|17)`, where `a = 1, b = 2, c = 3`.
   Answer: 3

   (f) Write the tab character and the null character.
   Answer: `'\t '`, `'\0'`

   (g) Which of the following is not a function of `string.h`?
   `strlen`, `strcpy`, `strman`, `strncmp`
   Answer: All except `strman`

   (h) How many element-to-element comparisons are enough to find the largest among 1024 numbers?
   Answer: 1023

   (i) How many multiplications are enough to compute the value of x^{1024}, where x is a real number in $[0, 1]$ (without using the math library)?
   Answer: 10

   (j) Which one runs faster between the recursive function and the iterative function for computing the value of $n!$? (n is a large positive integer)
   Answer: iterative

2. For the following mathematical expressions, write their corresponding expressions in C language. For example, the expression $x^2 + 0.1x$ is written as `x*x + 0.1*x` in C language. Assume that $x > 0$ and is declared as a variable of type `double`. You may use functions from the `math.h` library. **$1 \times 5 = 5$ marks**

   (a) $1.23x + x/1.23$ **Answer:** `1.23 * x + x / 1.23;`

   (b) $x^{1.23} + (1.23)^x$ **Answer:** `pow(x, 1.23) + pow(1.23, x);`

   (c) $\sqrt{1 + \sqrt{x}}$ **Answer:** `sqrt(1 + sqrt(x));`

   (d) $\log(x) + e^x + e^{-x}$ **Answer:** `log(x) + exp(x) + exp(-x);`

   (e) $(\sin(x))^x + \cos(x)$ **Answer:** `pow(sin(x), x) + cos(x);`

3. A *subarray* is a contiguous portion of an array; i.e., it consists of elements from the original array that are adjacent to each other and appear in the same order as they do in the original array. Here is a function to find all contiguous subarrays of an array `a[]` that sum up to a given target sum `t`. For example, with `t` = 6, the array `[-2 3 5 1 3]` has two subarrays: `[-2, 3, 5]` and `[5, 1]`. Fill up the blanks. **1 × 5 = 5 marks**

```
void findSubarraysWithSum(int a[], int n, int t) {
  for (int start = 0; start < n; start++) { // n = number of elements
    int current_sum = 0;
    for (int end = ___________________; end < n; end++) {

      current_sum += _________________;

      if (_____________________________) {
        printf("Subarray: [");
        for (int i = start; _______________; i++) {

          printf("%d", _____________);
          if (i < end) printf(", ");
        }
        printf("]\n");
      }
    }
  }
}
```

Answer — **Full code:** `contiguousSubarraysWithTargetSum.c`

```
start
a[end]
current_sum == t
i <= end
a[i]
```

Space for rough work

4. Two numbers are *co-primes* if their greatest common divisor (GCD) is 1. Here is a code that takes a positive integer n as input and prints each pair of co-primes that occur in $[2, n]$, exactly once.
For example, if $n = 3$, it prints `(2, 3)`;
if $n = 4$, it prints `(2, 3)` and `(3, 4)`;
if $n = 5$, it prints `(2, 3)`, `(2, 5)`, `(3, 4)`, `(3, 5)`, and `(4, 5)`;
if $n = 12$, it prints `(2, 3)`, `(2, 5)`, `(2, 7)`, `(2, 9)`, `(2, 11)`, `(3, 4)`, ..., `(11, 12)`.
Fill up the blanks. **$1 \times 7 = 7$ marks**

```c
#include <stdio.h>

int gcd(int a, int b) {
  int temp;
  while (b != 0) {

    ____________________________;

    ____________________________;

    ____________________________;
  }
  return a;
}

int main() {
  int n, i, j;
  printf("Enter a positive integer n: ");
  scanf("%d", &n);

  for (i = 2; _________________; i++){

    for (j = _____________; _____________; j++){

      if (___________________){ // function call

        printf("(%d, %d) ", i, j); // Co-prime pairs
  }}}
  return 0;
}
```

Answer — **Full code:** `co-primes.c`

```
temp = b;
b = a % b;
a = temp;
i <= n
j = i + 1
j <= n
gcd(i, j) == 1
```

Space for rough work

5. Three or more points are said to be *collinear* if they all lie on the same straight line. Here is a code that takes as input an integer n, followed by the coordinates of n points, and checks if all points are collinear. Since all points have integer coordinates, the code doesn't require `math.h`. It includes a user-defined function `areCollinear` that checks if a triangle formed by any three points has a non-zero area. Note that the function `areCollinear` calls another function `twiceArea`, which returns zero if and only if the three points taken as argument are collinear.

   The `main()` function handles the input and calls `areCollinear` to determine if all points are collinear. Fill up the blanks. **$1 \times 8 = 8$ marks**

```
#include <stdio.h>

/* returns twice the area of the triangle
   with vertices (x1,y1),(x2,y2),(x3,y3) */
int twiceArea(int x1, int y1, int x2, int y2, int x3, int y3) {
  return x1*(y2-y3)+x2*(y3-y1)+x3*(y1-y2);
}

int areCollinear(int n, int x[], int y[]) {

  for (int i = 0; _________________________; i++) {

    for (int j = _____________ ; j < _____________; j++) {

      for (int k = _____________; k < _____________; k++) {

        int t = twiceArea(x[i], y[i], x[j], y[j], x[k], y[k]);

        if (_____________________) return 0; // Points are not collinear
      }
    }
  }
  return 1; // All points are collinear
}

int main() {
  int n;
  printf("Enter the number of points: ");
  scanf("%d", &n);
  int x[n], y[n]; // arrays to store the coordinates of the points

  printf("Enter the coordinates of the points:\n");
  for (int i = 0; i < n; i++){

    scanf("%d %d", _____________, _____________);
  }
  if (areCollinear(n, x, y))
    printf("All points are collinear.\n");
  else
    printf("The points are not all collinear.\n");
  return 0;
}
```

Answer — **Full code:** `collinearity-nPoints.c`

```
i < n - 2
j = i + 1; j < n - 1
k = j + 1; k < n
t != 0
&x[i], &y[i]
```

Space for rough work

6. An *alphanumeric string* is made of English letters and digits. Let `s[]` and `t[]` be two alphanumeric strings of length `m` and `n` respectively, with $\texttt{m} \geq \texttt{n}$. The string `t[]` is said to be a *substring* of the string `s[]` if and only if there exists an integer `k`, where $0 \leq k \leq m - n$, such that `t[0]` = `s[k]`, `t[1]` = `s[k+1]`, ..., `t[m-1]` = `s[k+m-1]`. A string is *palindromic* if it reads the same forward and backward. For example, `abba` is even-length palindromic, while `ababa` is odd-length palindromic. The following code takes an alphanumeric string `s[]` as input and finds its longest palindromic substring `t[]`. Fill up the blanks. **$1 \times 10 = 10$ marks**

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

void findLongestPalindrome(char s[], char t[]) {
  int n = strlen(s), maxLength = 1, start = 0, low, high;
  for (int i = 1; i < n; i++) {
    // Find the longest even-length palindrome centered at s[i-1] and s[i]
    low = i - 1; high = i;

    while (_____________ >=0  && ____________ < n && ____________________) {

      if (_____________________> maxLength) {
        start = low;
        maxLength = high - low + 1;
      }
      low--; high++;
    }

    // Find the longest odd-length palindrome centered at s[i]
    low = i - 1; high = i + 1;

    while (_____________ >=0 && _____________ < n && ____________________) {

      if (____________________________> maxLength) {
        start = low;
        maxLength = high - low + 1;
      }
      low--; high++;
    }
  }

  // Copy the longest palindromic substring to t[]
  strncpy(t, s + start, maxLength);

  t[maxLength] = _________________;
}

int main() {
  char s[101], t[101];
  printf("Enter an alphanumeric string (max 100 characters): ");
  scanf("%s", s);
  findLongestPalindrome(____________________);
  printf("Longest palindromic substring: %s\n", _______);
  return 0;
}
```

Answer **Full code:** `longestPalindromicSubstring.c`

```
low >= 0 && high < n && s[low] == s[high]
high - low + 1 > maxLength
low >= 0 && high < n && s[low] == s[high]
high - low + 1 > maxLength
'\0'
s, t
t
```

Space for rough work

7. The following code uses recursion to generate and print each n-digit number (as a string) consisting of the digits 1, 2, and 3 (as characters). Assume that $1 \leq n \leq 10$.
For example, if $n = 2$, it generates $11, 12, 13, 21, 22, 23, 31, 32, 33$;
if $n = 3$, it generates $111, 112, 113, 121, \ldots, 323, 331, 332, 333$.
Fill up the blanks. **10 marks**

```
#include <stdio.h>

void generateNumbers(char number[], int n, int current_digit) {
  if (current_digit == n) { // Base case

    number[n] = _____________;  // Null-terminate the string [1 mark]
    printf("%s\n", number);
    return;
  }

  for (int i = 1; _____________; __________) { // 1.5 + 1.5 marks
    number[current_digit] = i + '0';  // Convert digit to character

    ____________________________________________________; // recursive call
  }                                                       // [3 marks]
}

int main() {
  int n;
  char number[11];  // Array to store the n-digit number as a string
  printf("Enter the value of n (1 to 10): ");

  scanf("%d", ____________);  // 1 mark

  generateNumbers(number, _______, ___________________________); // 2 marks
  return 0;
}
```

Answer — **Full code:** generateNumbersDigits-123.c

```
'\0'
i <= 3; i++
generateNumbers(number, n, current_digit + 1)
&n
n, 0
```

Space for rough work

8. The following code dynamically allocates memory for an array of integers. Initially, it allocates for 10 elements. It allows the user to enter nonzero elements into the array, and stops entering when the user enters `0`. If the array becomes full and the user wants to add more, it resizes the array by doubling its size using `realloc`. Fill up the blanks. **$1 \times 5 = 5$ marks**

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

int main() {
  int *arr;
  int size = 10; // Initial size of the array
  int count = 0; // Number of elements in the array
  int input;

  // Dynamically allocate memory for the array
  arr = (int *)malloc(size * sizeof(int));

  if (__________________________) {

    printf("Memory allocation failed.\n"); return 1;
  }

  printf("Enter integers (enter 0 to stop):\n");
  while (1) {
    scanf("%d", &input);
    if (input == 0)
      break;
    if (_____________________________________) { // resize it
      size *= 2;
      arr = (int *)realloc(_____________________________________);

      if (___________________________________) {

        printf("Memory reallocation failed.\n"); return 1;
      }
    } // resizing completed

    arr[count] = input; count += _________________;
  } // end while
  free(arr);
  return 0;
}
```

Answer — **Full code:** `reallocDynMem1D.c`

```
arr == NULL
count == size
arr, size * sizeof(int)
arr == NULL
1
```

END

Space for rough work