



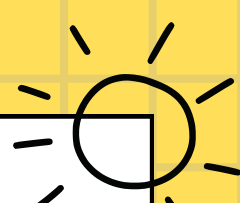


SRM

INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

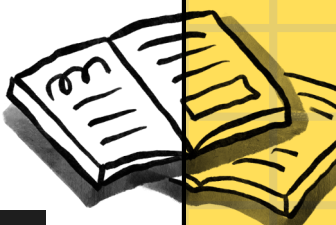
TIRUCHIRAPPALLI

DATA STRUCTURES AND ALGORITHMS

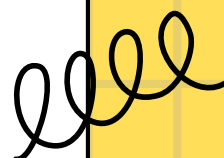
Name : Ullas Dewangan
Class : II, CSE-D
Register No. : RA2311003050094



Q.11 Mr. Ram is developing a software application to manage his inventory, which initially can hold a maximum of ten product packets. He allocates memory for ten packets at the start. However, as the market demand changes, he needs to dynamically update the allocated memory to store more or fewer packets without wasting memory or running out of space. Explain the concept of dynamic memory allocation and how Mr. Ram can effectively manage his inventory size using this concept.



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main() {
4     int initial_size = 10;
5     int new_size;
6     int *inventory;
7
8     inventory = (int *)malloc(initial_size * sizeof(int));
9     if (inventory == NULL) {
10         printf("Memory allocation failed!\n");
11         return 1;
12     }
13
14     for (int i = 0; i < initial_size; i++) {
15         inventory[i] = i + 1;
16     }
17
18     printf("Initial inventory:\n");
19     for (int i = 0; i < initial_size; i++) {
20         printf("Packet %d: %d\n", i + 1, inventory[i]);
21     }
22
23     printf("Enter new inventory size: ");
24     scanf("%d", &new_size);
25
26     inventory = (int *)realloc(inventory, new_size * sizeof(int));
27     if (inventory == NULL) {
28         printf("Memory reallocation failed!\n");
29         return 1;
30     }
31
32     if (new_size > initial_size) {
33         for (int i = initial_size; i < new_size; i++) {
34             inventory[i] = 0;
35         }
36     }
37     printf("Updated inventory:\n");
38     for (int i = 0; i < new_size; i++) {
39         printf("Packet %d: %d\n", i + 1, inventory[i]);
40     }
41     free(inventory);
42
43     return 0;
44 }
45
```



Explanation

1. Memory Allocation:

- malloc allocates memory for ten packets initially. If malloc fails, it returns NULL, and the program exits with an error message.
- realloc adjusts the size of the memory block to the new size specified by the user. It handles both increasing and decreasing the size of the memory block. If the reallocation fails, it returns NULL, and the program exits with an error message.

2. Initialization and Cleanup:

- The program initializes the inventory with example values and prints them.
- If the inventory size increases, the newly allocated memory is initialized to zero.
- After the inventory is updated, the program prints the new values and finally frees the allocated memory to avoid memory leaks.

Q.12 Mr. John is playing the game Subway Surfers. The game has a total of five treasures with different weights, that he needs to collect. Write a C program to count the total number of weights he collected from the treasures during the game.

```
1  #include <stdio.h>
2
3  int main() {
4      const int numberOfTreasures = 5;
5      int weights[numberOfTreasures];
6      int totalWeight = 0;
7
8      printf("Enter the weights of %d treasures:\n", numberOfTreasures);
9      for (int i = 0; i < numberOfTreasures; i++) {
10         printf("Weight of treasure %d: ", i + 1);
11         scanf("%d", &weights[i]);
12     }
13
14     for (int i = 0; i < numberOfTreasures; i++) {
15         totalWeight += weights[i];
16     }
17
18     printf("Total weight collected: %d\n", totalWeight);
19
20     return 0;
21 }
22
```




Explanation

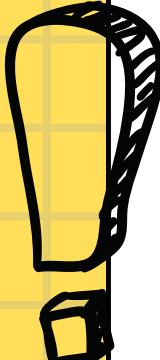
Data Structure:

- An array of integers (weights) is used to store the weights of the five treasures. This data structure is ideal because the number of treasures is fixed and known in advance, and arrays provide an easy way to index and access each treasure's weight.

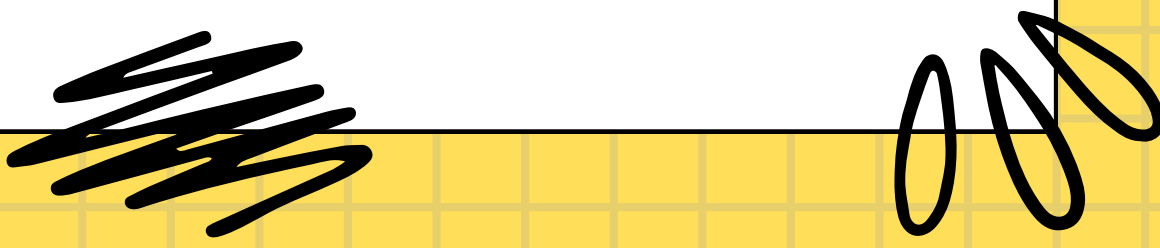
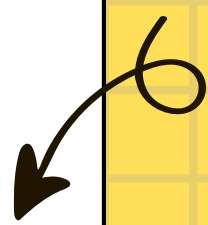
Functionality Implementation:

- The program first takes input for each treasure's weight and stores it in the weights array.
 - It then iterates through the array to sum up the weights.
 - Finally, it prints out the total weight collected.
- 

Q.13 Consider an object Shape that encompasses both a Square and a Rectangle as the data members. The Square object should have an attribute for its area, while the Rectangle object should have attributes for length and breadth. Identify the most suitable data structures for this scenario and write a C program to define the structure and demonstrate their usage.




```
1  #include <stdio.h>
2
3  struct Square {
4      int area;
5  };
6
7  struct Rectangle {
8      int length;
9      int breadth;
10 };
11
12 struct Shape {
13     struct Square square;
14     struct Rectangle rectangle;
15 };
16
17 int main() {
18     struct Shape shape;
19
20     shape.square.area = 25;
21     shape.rectangle.length = 10;
22     shape.rectangle.breadth = 5;
23
24     int perimeter = 2 * (shape.rectangle.length + shape.rectangle.breadth);
25     printf("Square Area: %d\n", shape.square.area);
26     printf("Rectangle Length: %d, Breadth: %d\n", shape.rectangle.length, shape.rectangle.breadth);
27     printf("Rectangle Perimeter: %d\n", perimeter);
28
29     return 0;
30 }
```





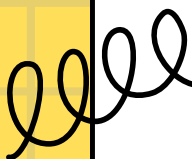
Explanation

Data Structure Definition:

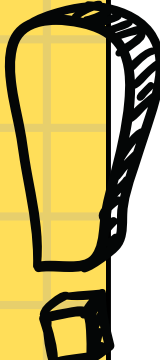
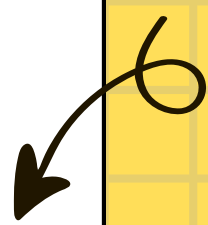
- Square Structure: The Square structure contains an integer area to represent the area of the square.
 - Rectangle Structure: The Rectangle structure has two integers, length and breadth, to represent the dimensions of the rectangle.
 - Shape Structure: The Shape structure contains a Square and a Rectangle, allowing it to represent both shapes in one entity.
- 

Operation:

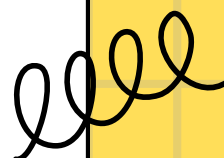
- The program demonstrates calculating the perimeter of the rectangle, using the values stored in the Shape structure. It then prints the area of the square and the perimeter of the rectangle.
-



Q.14 In a classroom, the teacher wants to create a list of students who have submitted their assignments. As students submit their work, the teacher needs to add each student's name to the list in the order of submission. Help the teacher by guiding them on how to use a proper data structure to insert each student's name into the list as they submit their assignment. Write a C program that uses an array to manage the list and demonstrates how to insert new student names into the array.



```
1  #include <stdio.h>
2  #include <string.h>
3
4  #define MAX_STUDENTS 100
5  #define NAME_LENGTH 50
6
7  int main() {
8      char studentList[MAX_STUDENTS][NAME_LENGTH];
9      int studentCount = 0;
10     char studentName[NAME_LENGTH];
11
12     while (studentCount < MAX_STUDENTS) {
13         printf("Enter student name (or 'exit' to stop): ");
14         gets(studentName);
15
16         if (strcmp(studentName, "exit") == 0) {
17             break;
18         }
19
20         strcpy(studentList[studentCount], studentName);
21         studentCount++;
22     }
23
24     printf("\nList of students who have submitted their assignments:\n");
25     for (int i = 0; i < studentCount; i++) {
26         printf("%d. %s\n", i + 1, studentList[i]);
27     }
28
29     return 0;
30 }
```

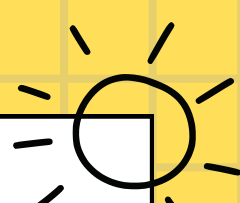




Q.15 Students need to check the availability of a book in the library based on its ID. Create an ordered list which will contain only book id. Implement a C program to search whether a particular book is available in the list or not.

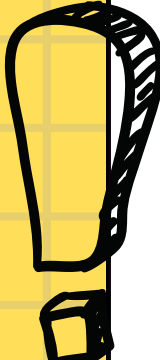
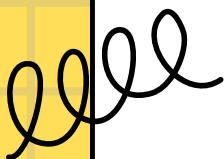

```
1  #include <stdio.h>
2  int binarySearch(int arr[], int size, int key) {
3      int low = 0, high = size - 1;
4      while (low <= high) {
5          int mid = (low + high) / 2;
6          if (arr[mid] == key) {
7              return mid;
8          }
9          if (arr[mid] < key) {
10             low = mid + 1;
11          } else {
12             high = mid - 1;
13          }
14      }
15      return -1;
16  }
17
18  int main() {
19      int bookList[] = {101, 203, 305, 407, 509}; // Ordered list of book IDs
20      int size = sizeof(bookList) / sizeof(bookList[0]);
21      int bookID;
22
23      printf("Enter the book ID to search: ");
24      scanf("%d", &bookID);
25
26      int result = binarySearch(bookList, size, bookID);
27
28      if (result != -1) {
29          printf("Book ID %d is available in the library.\n", bookID);
30      } else {
31          printf("Book ID %d is not available in the library.\n", bookID);
32      }
33
34      return 0;
35  }
```


Q.16 A = [3 4 2 4] B = [1 2 3 4]
Write a C program to perform Matrix addition for the above two matrices.



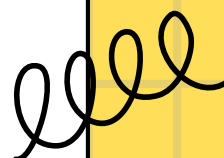
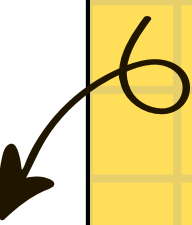
```
1  #include <stdio.h>
2
3  int main() {
4      int A[2][2] = {{3, 4}, {2, 4}};
5      int B[2][2] = {{1, 2}, {3, 4}};
6      int C[2][2];
7
8      for (int i = 0; i < 2; i++) {
9          for (int j = 0; j < 2; j++) {
10             C[i][j] = A[i][j] + B[i][j];
11         }
12     }
13
14     printf("Resultant Matrix after addition:\n");
15     for (int i = 0; i < 2; i++) {
16         for (int j = 0; j < 2; j++) {
17             printf("%d ", C[i][j]);
18         }
19         printf("\n");
20     }
21
22     return 0;
23 }
```

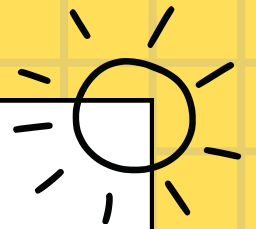
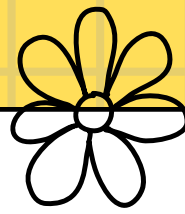


Q.17(a) Create a structure for student data base with following members (Register number, Name, Age, CGPA). Write a C program to perform the following operations (i) Get user input for 5 students record (ii) Find the student's name with greatest CGPA. (Note: Find appropriate data structure and develop a C program).



```
1  #include <stdio.h>
2  #include <string.h>
3  struct Student {
4      int regNumber;
5      char name[50];
6      int age;
7      float cgpa;
8  };
9  int main() {
10     struct Student students[5];
11     int n = 5, index = 0;
12     float maxCGPA = 0.0;
13     for (int i = 0; i < n; i++) {
14         printf("Enter details for student %d:\n", i + 1);
15         printf("Register Number: ");
16         scanf("%d", &students[i].regNumber);
17         printf("Name: ");
18         scanf("%s", students[i].name);
19         printf("Age: ");
20         scanf("%d", &students[i].age);
21         printf("CGPA: ");
22         scanf("%f", &students[i].cgpa);
23         printf("\n");
24     }
25     for (int i = 0; i < n; i++) {
26         if (students[i].cgpa > maxCGPA) {
27             maxCGPA = students[i].cgpa;
28             index = i;
29         }
30     }
31     printf("Student with the greatest CGPA:\n");
32     printf("Name: %s\n", students[index].name);
33     printf("Register Number: %d\n", students[index].regNumber);
34     printf("Age: %d\n", students[index].age);
35     printf("CGPA: %.2f\n", students[index].cgpa);
36     return 0;
37 }
```





Output:

```
Enter details for student 1:  
Register Number: Ra2311003050094  
Name: Age: 18  
CGPA: 9.12
```

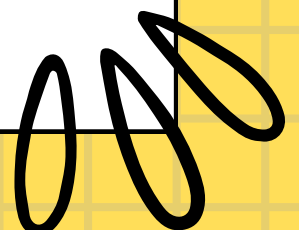
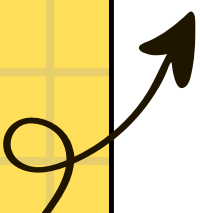
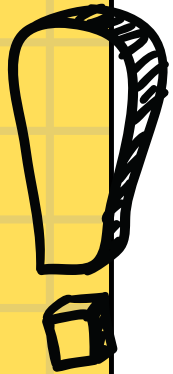
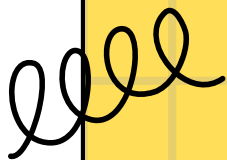
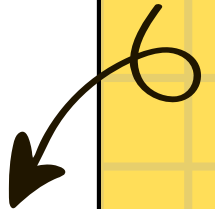
```
Enter details for student 2:  
Register Number: Ra2311003050089  
Name: Age: 18  
CGPA: 9.69
```

```
Enter details for student 3:  
Register Number: Ra2311003050085  
Name: Age: 17  
CGPA: 9.08
```

```
Enter details for student 4:  
Register Number: Ra2311003050143  
Name: Age: 19  
CGPA: 9.81
```

```
Enter details for student 5:  
Register Number: Ra2311003050174  
Name: Age: 18  
CGPA: 9.2
```

```
Student with the greatest CGPA:  
Name: Ra2311003050143  
Register Number: -552012504  
Age: 19  
CGPA: 9.81
```



Q.17 (b)

1. Illustrate how asymptotic notations can be used to analyze an algorithm? (6Marks)

2. (ii) Calculate time and space complexity for the following code (6Marks)

```
#include<stdio.h>
int main() {
    int i = 1, sum = 0, n;
    while (i<= n){
        i=i+1;
        sum = sum +i;
    }
}
```

(i) Asymptotic Notations for Analyzing an Algorithm (6 Marks)

Asymptotic notations are mathematical tools used to describe the behavior of an algorithm in terms of time and space as the input size grows. These notations allow us to analyze the efficiency and scalability of algorithms by focusing on their performance in the worst-case, best-case, and average-case scenarios.

The most commonly used asymptotic notations are:

1. Big O Notation (O):

- Represents the upper bound of an algorithm's running time. It gives the worst-case scenario, describing the maximum time an algorithm can take.
- Example: If an algorithm has a time complexity of $O(n^2)$, it means that the running time increases quadratically with the input size.

2. Omega Notation (Ω):

- Represents the lower bound of an algorithm's running time. It gives the best-case scenario, describing the minimum time an algorithm can take.
- Example: If an algorithm has a time complexity of $\Omega(n)$, it means that in the best case, the running time increases linearly with the input size.

3. Theta Notation (Θ):

- Represents the tight bound of an algorithm's running time. It gives both the upper and lower bounds, indicating the exact growth rate of the algorithm's running time.
- Example: If an algorithm has a time complexity of $\Theta(n \log n)$, it means that the running time grows at a rate proportional to $n \log n$ for all cases.

- Little o Notation (o):
 - Represents a stricter upper bound than Big O. It shows that an algorithm's growth rate is strictly less than the rate described by the function.
 - Example: If an algorithm has a time complexity of $o(n^2)$, it means that the running time grows slower than n^2 , but it doesn't exactly grow at n^2 .
 - Little Omega Notation (ω):
 - Represents a stricter lower bound than Omega. It shows that an algorithm's growth rate is strictly greater than the rate described by the function.
 - Example: If an algorithm has a time complexity of $\omega(n)$, it means that the running time grows faster than n .
- (ii) Time and Space Complexity of the Given Code

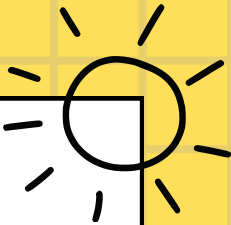
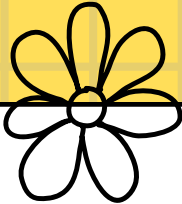

```
#include<stdio.h>int main() {
    int i = 1, sum = 0, n;
    while (i <= n) {
        i = i + 1;
        sum = sum + i;
    }
}
```

Time Complexity:



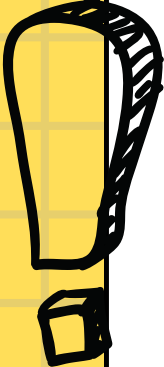
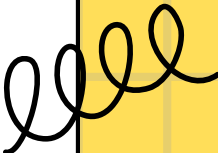
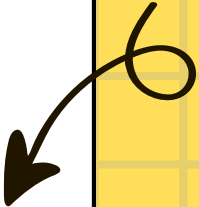
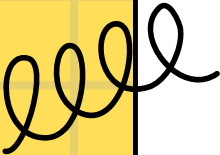

- Initialization:
 - The initialization of variables $i = 1$, $sum = 0$, and n takes constant time: $O(1)$.
- While Loop:
 - The while loop runs until i exceeds n . The loop's body executes for each value of i from 1 to n .
 - Each iteration of the loop has two operations:
 - $i = i + 1$; takes constant time: $O(1)$.
 - $sum = sum + i$; takes constant time: $O(1)$.
 - Therefore, the total time taken by the loop is the number of iterations multiplied by the time for each iteration, which is $O(n)$.
- Overall Time Complexity:
 - The overall time complexity of the code is $O(n)$.

Space Complexity:

- Variables:
 - The variables i , sum , and n are all integer variables that occupy a constant amount of space: $O(1)$.
- No Additional Data Structures:
 - The code does not use any additional data structures like arrays or dynamic memory allocation that would require extra space.
- Overall Space Complexity:
 - The overall space complexity of the code is $O(1)$, as the space required does not depend on the input size n .



Q.18(a) Given a number 'n', write an algorithm and the subsequent 'C' program to count the number of two-digit prime numbers in it when adjacent digits are taken. For example, if the value of 'n' is 114 then the two-digit numbers that can be formed by taking adjacent digits are 11 and 14. 11 is prime but 14 is not. Therefore print 1.



```
1  #include <stdio.h>
2  #include <string.h>
3
4  int isPrime(int num) {
5      if (num <= 1) return 0;
6      if (num <= 3) return 1;
7      if (num % 2 == 0 || num % 3 == 0) return 0;
8      for (int i = 5; i * i <= num; i += 6) {
9          if (num % i == 0 || num % (i + 2) == 0) return 0;
10     }
11     return 1;
12 }
13
14 int main() {
15     int n;
16     printf("Enter a number: ");
17     scanf("%d", &n);
18
19     char str[12];
20     sprintf(str, "%d", n);
21
22     int count = 0;
23     int length = strlen(str);
24
25     for (int i = 0; i < length - 1; i++) {
26         int num = (str[i] - '0') * 10 + (str[i + 1] - '0');
27         if (isPrime(num)) {
28             count++;
29         }
30     }
31
32     printf("Number of two-digit prime numbers: %d\n", count);
33
34     return 0;
35 }
```

1. Prime Checking Function (isPrime):

- This function determines if a number is prime. It efficiently checks for factors up to the square root of the number, improving performance over a naive approach.

2. Main Function:

- The number n is read and converted to a string for easy processing of digits.
- Adjacent pairs of digits are extracted and converted into two-digit numbers.
- Each two-digit number is checked for primality using the isPrime function.
- The count of prime numbers is maintained and printed.

Time Complexity

- Time Complexity: $O(m \cdot \sqrt{k})$ where m is the number of adjacent digit pairs and k is the maximum two-digit number (99).
- Space Complexity: $O(m)$, where m is the length of the string representation of the number.

Q.18(b) Demonstrate the usage of list and perform all the possible operation using array with suitable examples.

An array is a collection of elements of the same data type, stored at contiguous memory locations. Arrays allow us to perform various operations such as insertion, deletion, searching, and sorting.

1. Creation and Initialization

```
1  #include <stdio.h>
2
3  int main() {
4
5      int arr[10] = {10, 20, 30, 40, 50};
6
7      printf("Array elements: ");
8      for(int i = 0; i < 5; i++) {
9          printf("%d ", arr[i]);
10     }
11     printf("\n");
12
13     return 0;
14 }
15
```

Output:

```
Array elements: 10 20 30 40 50
```

2. Insertion

```
1  #include <stdio.h>
2
3  int main() {
4      int arr[10] = {10, 20, 30, 40, 50};
5      int n = 5;
6      int position = 2;
7      int value = 25;
8
9
10     for(int i = n; i > position; i--) {
11         arr[i] = arr[i - 1];
12     }
13
14
15     arr[position] = value;
16     n++;
17
18     printf("Array after insertion: ");
19     for(int i = 0; i < n; i++) {
20         printf("%d ", arr[i]);
21     }
22     printf("\n");
23
24     return 0;
25 }
26
```

Output:

```
Array after insertion: 10 20 25 30 40 50
```


3. Deletion

```
1  #include <stdio.h>
2
3  int main() {
4      int arr[10] = {10, 20, 30, 40, 50};
5      int n = 5;
6      int position = 2;
7
8
9      for(int i = position; i < n - 1; i++) {
10         arr[i] = arr[i + 1];
11     }
12
13     n--;
14     printf("Array after deletion: ");
15     for(int i = 0; i < n; i++) {
16         printf("%d ", arr[i]);
17     }
18     printf("\n");
19
20     return 0;
21 }
22
```

Output:

Array after deletion: 10 20 40 50

4. Traversal

```
1  #include <stdio.h>
2
3  int main() {
4      int arr[5] = {10, 20, 30, 40, 50};
5      int n = 5;
6
7      printf("Array elements: ");
8      for(int i = 0; i < n; i++) {
9          printf("%d ", arr[i]);
10     }
11     printf("\n");
12
13     return 0;
14 }
15
```

Output:

```
Array elements: 10 20 30 40 50
```