RAJALAKSHMI ENGINEERING COLLEGE

### An Autonomous Institution, Affiliated to Anna University Rajalakshmi Nagar, Thandalam – 602 105



**DEPARTMENT OF COMPUTER SCIENCE AND BUSINESS SYSTEMS**

|  |
| --- |
| **CB23231**  **DATA STRUCTURES AND ALGORITHMS**  **(*Regulation 2023*)** |
| **LAB RECORD** |

2023-27

2

CSBS-B



2116231401112

Subramaniam R

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

**RAJALAKSHMI NAGAR, THANDALAM – 602 105 BONAFIED CERTIFICATE**

NAME REGISTER NO.

ACADEMIC YEAR **2023-24** SEMESTER **II** BRANCH B.E/B.Tech

This Certification is the bonafide record of work done by the above student

in the **CS23231-DATA STRUCTURES AND ALGORITHM -** Laboratory during the year

2023 – 2024.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on

Internal Examiner External Examiner

**LESSON PLAN**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Course Code** | **Course Title**  **(Laboratory Integrated Theory Course)** | **L** | **T** | **P** | **C** |
| **CB23231** | **DATA STRUCTURES AND ALGORITHMS** | **2** | **1** | **4** | **5** |

|  |  |  |
| --- | --- | --- |
| **LIST OF EXPERIMENTS** | | |
| **Sl. No** | **Name of the experiment** | **No of Hours** |
| 1. | Implementations of Stack and Queue | **6** |
| 2. | Applications of Stack and Queue (Tower of Hanoi, Infix, Postfix and Prefix Conversion, Expression Evaluation) | **6** |
| 3. | Implementations of Singly, Doubly and Circular List | **6** |
| 4. | Polynomial Manipulations | **6** |
| 5. | Implementations Binary Search Tree and AVL Tree (All Operations) | **6** |
| 6. | Implementation of BFS and DFS | **6** |
| 7. | Linear and Binary Search Operations | **6** |
| 8. | Implementation of Bubble Sort, Quick and Merge Sort | **6** |
| 9. | Implementation of Hashing Techniques | **6** |
| 10. | Mini Projects Snakes Game Sudoku  Travel Planner  Cash Flow Minimiser  Text Editor Cut, Copy, Paste | **6** |

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No.** | **Name of the Experiment** | **Expt. Date** | **Page No** | **Faculty Sign** |
|  | | | | |
| **Week 1** |  | | | |
| 1.a | Vote Casting |  | 2-3 |  |
| 1.b | Prime Number Determination of Quadrant |  | 4-5 |  |
| 1.c | Temperature State Check |  | 6-7 |  |
| 1.d | Transpose of a Matrix |  | 8-11 |  |
|  | | | | |
| **Week 2** |  | | | |
| 2.a | The Largest Two Numbers In A Given Array using Loop |  | 12-15 |  |
| 2.b | Structure :Details Of The Student |  | 16-19 |  |
| 2.c | Structure :Details Of The Student with dynamic  memory allocation for storing |  | 20-24 |  |
|  | | | | |
| **Week 3** |  | | | |
| 3.a | Create a union named 'Data' that can hold an integer, float, and character. |  | 20-26 |  |
| 3.b | Union For Representing Different Shapes (circle, square, rectangle) |  | 27-32 |  |
|  | | | | |
| **Week 4** | **Implementations of Stack and Queue** | | | |
| 4.a | Implement a stack - push, pop, and peek operations |  | 33-36 |  |
| 4.b | Implement a queue using an array |  | 37-40 |  |
|  | | | | |
| **Week 5** | **Applications of Stack and Queue** | | | |
| 5.a | Postfix/ Prefix Expression Using A Stack. |  | 41-44 |  |
| 5.b | Tower of Hanoi |  | 45-48 |  |
| 5.c | The Evaluation Of Infix Expressions Using A Stack |  | 49-52 |  |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No.** | **Name of the Experiment** | **Expt. Date** | **Page No** | **Faculty Sign** |
|  |  | | | |
| **Week 6** | **Implementations of Singly, Doubly and Circular List** | | | |
| 6.a | Implementation of operations on Singly Linked List. |  | 53-57 |  |
| 6.b | Doubly Linked List |  | 58-63 |  |
| 6.c | Circular Linked List |  | 64-68 |  |
|  | | | | |
| **Week 7** | **Polynomial Manipulations** | | | |
| 7 | 1. Polynomial Addition 2. Polynomial Subtraction 3. Polynomial Multiplication |  | 69-74 |  |
|  | | | | |
| **Week 8** | **Implementations Binary Search Tree and AVL Tree (All Operations)** | | | |
| 8.a | Tree Traversal |  | 75-78 |  |
| 8.b | Implementation of Binary Search tree |  | 79-83 |  |
| 8.c | Implementation of AVL Tree |  | 84-88 |  |
|  | | | | |
| **Week 9** | **Implementation of BFS and DFS** | | | |
| 9.a | Graph Representation |  | 89-92 |  |
| 9.b | BFS Implementation |  | 93-95 |  |
| 9.c | DFS Implementation |  | 96-99 |  |
|  | | | | |
| **Week 10** | **Linear and Binary Search Operations** | | | |
| 10.a | Implement a function for linear search |  | 100-104 |  |
| 10.b | Implement a function for binary search |  | 105-109 |  |
|  | | | | |
| **Week 11** | **Implementation of Bubble Sort, Quick and Merge Sort** | | | |
| 11 | To implement the same using following sorting techniques.   1. Quick Sort 2. Merge Sort 3. Bubble Sort |  | 110-114 |  |
| **Week 12** | **Implementation of Hashing Techniques** | | | |
| 12 | Hash table and perform collision resolution using the following techniques.   1. Open addressing 2. Closed Addressing 3. Rehashing |  | 115-120 |  |
|  | | | | |
| **Week 13** | **Mini Projects** | | | |
| 13 | * Snakes Game * Sudoku * Travel Planner * Cash Flow Minimiser Text Editor Cut, Copy, Paste |  | 121- |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No.** | **Name of the Experiment** | **Expt. Date** | **Page No** | **Faculty Sign** |
|  | | | | |
| **Week 14** |  | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | | | | |
| **Week 15** |  | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  | | | | |
| **Week 16** |  | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | | | | |
| **Week 17** |  | | | |
|  |  |  |  |  |
|  |  |  |  |  |

# Note: Students have to write the Algorithms at left side of each problem statements.

**Decision Making**

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

1

**Ex. No.: 1. a Date:**

### Vote Casting

**Write a C program to read the age of a candidate and determine whether he is eligible to cast his/her own vote.**

**Sample Input**

Test Data: 21

**Sample Output**

Congratulation! You are eligible for casting your vote

**Aim:**

To write a C program that reads the age of a candidate and determines whether he or she is eligible to cast a vote

**Algorithm:**

2

1. Start.

2. Declare an integer variable to store the age.

3. Prompt the user to enter the candidate's age.

4. Read the input value and store it in the variable.

5. Check if the age is greater than or equal to 18.

• If true, print "Congratulations! You are eligible for casting your vote."

• If false, print "You are not eligible for casting your vote."

6. End

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Ex. No.: Date:**



#include <stdio.h>

int main() {

// Variable to store the age of the candidate

int age;

// Prompt the user to enter the candidate's age

printf("Enter your age: ");

// Read the input age from the user

scanf("%d", &age);

// Check if the candidate is eligible to vote

if (age >= 18) {

// Candidate is eligible to vote

printf("Congratulations! You are eligible for casting your vote.\n");

} else {

// Candidate is not eligible to vote

printf("You are not eligible for casting your vote.\n");

}

return 0;

}

**Result:**

Based on the input, the program will check if the age is 18 or more. If so, it will print a congratulatory message indicating that the user is eligible to vote. Otherwise, it will inform the user that they are not eligible to vote.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

3

**Ex. No.: 1. b Date:**

### Prime Number Determination of Quadrant

**Write a C program to accept a coordinate point in an XY coordinate system and determine in which quadrant the coordinate point lies.**

**Sample Input:**

**7 9**

**Sample Output:**

**The coordinate point (7,9) lies in the First quadrant. Aim:**

To write a C program that accepts a coordinate point in an XY coordinate system and determines in which quadrant the point lies.

**Algorithm:**

1. Start.

2. Declare two integer variables to store the coordinates (x and y).

3. Prompt the user to enter the x and y coordinates.

4. Read the input values and store them in the variables.

5. Determine the quadrant based on the values of x and y:

• If x > 0 and y > 0, the point lies in the First quadrant.

• If x < 0 and y > 0, the point lies in the Second quadrant.

• If x < 0 and y < 0, the point lies in the Third quadrant.

• If x > 0 and y < 0, the point lies in the Fourth quadrant.

• If x == 0 and y == 0, the point is at the origin.

• If x == 0 and y != 0, the point lies on the Y-axis.

• If y == 0 and x != 0, the point lies on the X-axis.

6. Print the result.

7. End.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

4

### Program:

#include <stdio.h>

int main() {

// Variables to store the coordinates

int x, y;

// Prompt the user to enter the coordinates

printf("Enter the coordinates (x y): ");

// Read the input coordinates from the user

scanf("%d %d", &x, &y);

// Determine the quadrant or axis in which the point lies

if (x > 0 && y > 0) {

printf("The coordinate point (%d,%d) lies in the First quadrant.\n", x, y);

} else if (x < 0 && y > 0) {

printf("The coordinate point (%d,%d) lies in the Second quadrant.\n", x, y);

} else if (x < 0 && y < 0) {

printf("The coordinate point (%d,%d) lies in the Third quadrant.\n", x, y);

} else if (x > 0 && y < 0) {

printf("The coordinate point (%d,%d) lies in the Fourth quadrant.\n", x, y);

} else if (x == 0 && y == 0) {

printf("The coordinate point (%d,%d) is at the Origin.\n", x, y);

} else if (x == 0) {

printf("The coordinate point (%d,%d) lies on the Y-axis.\n", x, y);

} else if (y == 0) {

printf("The coordinate point (%d,%d) lies on the X-axis.\n", x, y);

}

return 0;

}

**Result:**

Based on the input, the program will determine the quadrant in which the point lies and print the appropriate message.

5

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Ex. No.: 1. c Date:**

**Temperature State Check**

**Write a C program to read temperature in centigrade and display a suitable message according to the temperature state below:**

**Temp < 0 then Freezing weather Temp 0-10 then Very Cold weather Temp 10-20 then Cold weather Temp 20-30 then Normal in Temp Temp 30-40 then Its Hot**

**Temp >=40 then Its Very Hot .**

**Sample Input: 42**

**Sample Output: Its very hot.**

**Aim:**

To write a C program that reads temperature in centigrade and displays a suitable message according to the temperature state

**Algorithm:**

6

1. Start.

2. Declare a float variable to store the temperature.

3. Prompt the user to enter the temperature in centigrade.

4. Read the input value and store it in the variable.

5. Determine the temperature state based on the value:

• If temperature < 0, print "Freezing weather."

• If 0 <= temperature < 10, print "Very Cold weather."

• If 10 <= temperature < 20, print "Cold weather."

• If 20 <= temperature < 30, print "Normal in Temp."

• If 30 <= temperature < 40, print "Its Hot."

• If temperature >= 40, print "Its Very Hot."

6. End.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

### Program:

#include <stdio.h>

int main() {

// Variable to store the temperature in centigrade

float temperature;

// Prompt the user to enter the temperature

printf("Enter the temperature in centigrade: ");

// Read the input temperature from the user

scanf("%f", &temperature);

// Determine and display the temperature state

if (temperature < 0) {

printf("Freezing weather.\n");

} else if (temperature < 10) {

printf("Very Cold weather.\n");

} else if (temperature < 20) {

printf("Cold weather.\n");

} else if (temperature < 30) {

printf("Normal in Temp.\n");

} else if (temperature < 40) {

printf("Its Hot.\n");

} else {

printf("Its Very Hot.\n");

}

return 0;

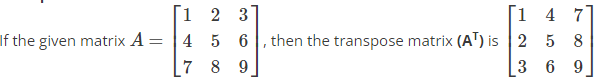
}

**Result:**

7

Based on the input, the program will display a suitable message indicating the temperature state.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College



**Ex. No.: 1. d Date:**

**Transpose of a Matrix**

**The** transpose of a matrix **is found by interchanging its rows into columns or columns into rows. The transpose of the matrix is denoted by using the letter “**T**” in the superscript of the given matrix.**

**For** example**, if “A” is the given matrix, then the transpose of the matrix is represented by** A’ **or** AT**.**

**Sample Input:**

**Sample Output:**

**Enter the order of the matrix 3 3**

**Enter the coefficients of the matrix 1 2 3**

**4 5 6**

**7 8 9**

**The given matrix is 1 2 3**

**4 5 6**

**7 8 9**

**Transpose of matrix is**

**1 4 7**

**2 5 8**

**3 6 9**

**Aim:**

To write a C program that reads a matrix from the user, computes its transpose, and prints both the original matrix and its transpose.

**Algorithm:**

1. Start.

2. Declare variables for the order of the matrix (rows and columns).

3. Prompt the user to enter the order of the matrix.

4. Read the order of the matrix.

5. Declare a 2D array to store the matrix elements.

6. Prompt the user to enter the elements of the matrix.

7. Read the elements of the matrix and store them in the 2D array.

8. Compute the transpose of the matrix by interchanging rows and columns.

9. Print the original matrix.

10. Print the transpose of the matrix.

11. End.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

8

### Program:

#include <stdio.h>

int main() {

int rows, cols;

// Prompt the user to enter the order of the matrix

printf("Enter the order of the matrix (rows and columns): ");

scanf("%d %d", &rows, &cols);

// Declare a 2D array to store the matrix

int matrix[rows][cols];

int transpose[cols][rows];

// Prompt the user to enter the elements of the matrix

printf("Enter the coefficients of the matrix:\n");

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

scanf("%d", &matrix[i][j]);

}

}

// Compute the transpose of the matrix

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

transpose[j][i] = matrix[i][j];

}

}

// Print the original matrix

printf("The given matrix is:\n");

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

// Print the transpose of the matrix

printf("Transpose of the matrix is:\n");

for (int i = 0; i < cols; ++i) {

for (int j = 0; j < rows; ++j) {

printf("%d ", transpose[i][j]);

}

printf("\n");

}

return 0;

}

9

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Output :**

Enter the order of the matrix (rows and columns): 3 3

Enter the coefficients of the matrix:

1 2 3

4 5 6

7 8 9

The given matrix is:

1 2 3

4 5 6

7 8 9

Transpose of the matrix is:

1 4 7

2 5 8

3 6 9

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

10

**Result:**

This program reads the dimensions and elements of a matrix from the user, calculates the transpose of the matrix, and then prints both the original matrix and the transposed matrix.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

11

12

|  |  |
| --- | --- |
| **Ex. No.: 2 a** | **Date:** |
| **To Calculate The Largest Two Numbers In A Given Array using Loop.** | |
| **Problem Statement:**  To calculate the largest two numbers in a given Array.  **Problem Description**  We have to write a program in C such that the program will read the elements of a one- dimensional array, then compares the elements and finds which the largest two elements are in a given array.  **Sample Input1 :**  If we are entering 5 elements (N = 5), with array element values as 2,4,5,8 and 7 then,  **Sample Input2 :**  If we are entering 6 elements (N = 6), with array element values as 2,1,1,2,1 and 2 then,  **Sample Output 1:**  The FIRST LARGEST = 8 THE SECOND LARGEST = 7  **Sample Output 1:**  The FIRST LARGEST = 2 THE SECOND LARGEST = 1  **Aim:**  To write a C program that reads the elements of a one-dimensional array and finds the largest two elements in the given array.  **Algorithm:**  1. Start.  2. Declare an integer variable N for the number of elements in the array.  3. Declare an integer array to store the elements.  4. Prompt the user to enter the number of elements (N).  5. Read the number of elements.  6. Prompt the user to enter the elements of the array.  7. Read the elements of the array and store them in the array.  8. Initialize two variables firstLargest and secondLargest to store the two largest elements.  9. Traverse the array to find the largest and the second largest elements:  • If an element is greater than firstLargest, update secondLargest to firstLargest and firstLargest to the current element.  • Else if the element is greater than secondLargest and not equal to firstLargest, update secondLargest to the current element.  10. Print the largest and the second largest elements.  11. End. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Program:**

#include <stdio.h>

int main() {

int N;

// Prompt the user to enter the number of elements

printf("Enter the number of elements in the array: ");

scanf("%d", &N);

// Declare an array to store the elements

int arr[N];

// Prompt the user to enter the elements of the array

printf("Enter the elements of the array:\n");

for (int i = 0; i < N; ++i) {

scanf("%d", &arr[i]);

}

// Initialize the first and second largest elements

int firstLargest, secondLargest;

if (N >= 2) {

firstLargest = secondLargest = arr[0];

for (int i = 1; i < N; ++i) {

if (arr[i] > firstLargest) {

secondLargest = firstLargest;

firstLargest = arr[i];

} else if (arr[i] > secondLargest && arr[i] != firstLargest) {

secondLargest = arr[i];

}

}

// If all elements are the same, there won't be a second largest distinct element

if (firstLargest == secondLargest) {

printf("All elements are the same or there is no distinct second largest element.\n");

} else {

printf("The FIRST LARGEST = %d\n", firstLargest);

printf("The SECOND LARGEST = %d\n", secondLargest);

}

} else {

printf("The array should have at least two elements.\n");

}

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

13

**Output 1**

Enter the number of elements in the array: 5

Enter the elements of the array:

2 4 5 8 7

The FIRST LARGEST = 8

The SECOND LARGEST = 7

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

14

**Output 2**

Enter the number of elements in the array: 6

Enter the elements of the array:

2 1 1 2 1 2

The FIRST LARGEST = 2

The SECOND LARGEST = 1

### Result:

This program handles finding the two largest elements in the array, including checking for the edge case where all elements are the same or there is no distinct second largest element.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

15

|  |  |
| --- | --- |
| **Ex. No.:2.b** | **Date:** |
| **Structure :Details Of The Student** | |
| **Write a program that defines a structure representing a student with fields for name, roll number, and marks in three subjects. Implement functions to:**  **Read details of 'n' students.**  **Calculate and display the total marks and average marks of each student. Find and display the details of the student with the highest total marks.**  **Sample Input :**  Enter the number of students: 3 Enter details for student 1: Name: vv  Roll Number: 21  Marks in three subjects: 54 55 76  **Sample Output:**  Student Details:  Student 1:  Name: vv  Roll Number: 21  Total Marks: 185  Average Marks: 61.67  Student with highest total marks: Name: ss  Roll Number: 23  Total Marks: 221  Average Marks: 73.67  **Aim:**  To write a C program that defines a structure representing a student with fields for name, roll number, and marks in three subjects, and implements functions to:  1. Read details of 'n' students.  2. Calculate and display the total marks and average marks of each student.  3. Find and display the details of the student with the highest total marks.  **Algorithm:**  1. Define a structure Student with fields for name, roll number, marks in three subjects, total marks, and average marks.  2. Implement a function readStudentDetails to read the details of a student.  3. Implement a function calculateTotalAndAverageMarks to calculate the total and average marks of a student.  4. Implement a function displayStudentDetails to display the details of a student.  5. Implement a function findStudentWithHighestMarks to find and return the student with the highest total marks.  6. In the main function:  • Prompt the user to enter the number of students.  • Read and store the details of each student.  • Calculate the total and average marks for each student.  • Display the details of each student.  • Find and display the details of the student with the highest total marks. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

16

**Program:**

17

#include <stdio.h>

#include <string.h>

// Define a structure to represent a student

struct Student {

char name[100];

int rollNumber;

int marks[3];

int totalMarks;

float averageMarks;

};

// Function prototypes

void readStudentDetails(struct Student\* student);

void calculateTotalAndAverageMarks(struct Student\* student);

void displayStudentDetails(const struct Student\* student);

struct Student findStudentWithHighestMarks(struct Student students[], int n);

int main() {

int n;

// Prompt the user to enter the number of students

printf("Enter the number of students: ");

scanf("%d", &n);

// Declare an array to store the students

struct Student students[n];

// Read details for each student

for (int i = 0; i < n; ++i) {

printf("Enter details for student %d:\n", i + 1);

readStudentDetails(&students[i]);

calculateTotalAndAverageMarks(&students[i]);

}

// Display details of each student

printf("\nStudent Details:\n");

for (int i = 0; i < n; ++i) {

printf("Student %d:\n", i + 1);

displayStudentDetails(&students[i]);

printf("\n");

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

### Program:

// Find and display the student with the highest total marks

struct Student topStudent = findStudentWithHighestMarks(students, n);

printf("Student with highest total marks:\n");

displayStudentDetails(&topStudent);

return 0;

}

// Function to read details of a student

void readStudentDetails(struct Student\* student) {

printf("Name: ");

scanf("%s", student->name);

printf("Roll Number: ");

scanf("%d", &student->rollNumber);

printf("Marks in three subjects: ");

for (int i = 0; i < 3; ++i) {

scanf("%d", &student->marks[i]);

}

}

// Function to calculate total and average marks of a student

void calculateTotalAndAverageMarks(struct Student\* student) {

student->totalMarks = 0;

for (int i = 0; i < 3; ++i) {

student->totalMarks += student->marks[i];

}

student->averageMarks = student->totalMarks / 3.0;

}

// Function to display details of a student

void displayStudentDetails(const struct Student\* student) {

printf("Name: %s\n", student->name);

printf("Roll Number: %d\n", student->rollNumber);

printf("Total Marks: %d\n", student->totalMarks);

printf("Average Marks: %.2f\n", student->averageMarks);

}

// Function to find the student with the highest total marks

struct Student findStudentWithHighestMarks(struct Student students[], int n) {

struct Student topStudent = students[0];

for (int i = 1; i < n; ++i) {

if (students[i].totalMarks > topStudent.totalMarks) {

topStudent = students[i];

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

18

}

}

return topStudent;

}

**Output**

Enter the number of students: 3

Enter details for student 1:

Name: vv

Roll Number: 21

Marks in three subjects: 54 55 76

Enter details for student 2:

Name: ss

Roll Number: 23

Marks in three subjects: 70 75 76

Enter details for student 3:

Name: rr

Roll Number: 22

Marks in three subjects: 60 65 64

Student Details:

Student 1:

Name: vv

Roll Number: 21

Total Marks: 185

Average Marks: 61.67

Student 2:

Name: ss

Roll Number: 23

Total Marks: 221

Average Marks: 73.67

Student 3:

Name: rr

Roll Number: 22

Total Marks: 189

Average Marks: 63.00

Student with highest total marks:

Name: ss

Roll Number: 23

Total Marks: 221

Average Marks: 73.67

**RESULT:**

This program successfully reads the details of multiple students, calculates and displays their total and average marks, and identifies the student with the highest total mark

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

19

|  |  |  |
| --- | --- | --- |
| CB23231 - DATA STRUCTURES AND ALGORITHMS  **Ex. No.:2.C** | | |
|  | **Structure :Details Of The Student with dynamic memory allocation for storing** |  |
| **Write a program that defines a structure representing a student with fields for name, roll number, and marks in three subjects. Implement functions to:**  **Read details of 'n' students.**  **Calculate and display the total marks and average marks of each student. Find and display the details of the student with the highest total marks**  **Extend the above program to include dynamic memory allocation for storing details of students.**  **Sample Input :**  Enter the number of students: 3 Enter details for student 1: Name: vv  Roll Number: 23  Marks in three subjects: 67 78 89 Enter details for student 2: Name: ii  Roll Number: 11  Marks in three subjects: 99 89 78 Enter details for student 3: Name: ss  Roll Number: 18  Marks in three subjects: 23 34 56  **Sample Output :**  Student Details:  Student 1:  Name: vv  Roll Number: 23  Total Marks: 234  Average Marks: 78.00  Student 2:  Name: ii  Roll Number: 11  Total Marks: 266  Average Marks: 88.67  Student 3:  Name: ss  Roll Number: 18  Total Marks: 113  Average Marks: 37.67  Student with highest total marks: Name: ii  Roll Number: 11  Total Marks: 266  Average Marks: 88.67  Department of Computer Science and Business Systems, Rajalakshmi Engineering College  Department of Computer Science and Business Systems, Rajalakshmi Engineering College |
| 20 |
|  |

**Aim:**

To write a C program that defines a structure representing a student with fields for name, roll number, and marks in three subjects, and implements functions to:

1. Read details of 'n' students using dynamic memory allocation.

2. Calculate and display the total marks and average marks of each student.

3. Find and display the details of the student with the highest total marks.

**Algorithm:**

1. Define a structure Student with fields for name, roll number, marks in three subjects, total marks, and average marks.

2. Implement a function readStudentDetails to read the details of a student.

3. Implement a function calculateTotalAndAverageMarks to calculate the total and average marks of a student.

4. Implement a function displayStudentDetails to display the details of a student.

5. Implement a function findStudentWithHighestMarks to find and return the student with the highest total marks.

6. In the main function:

• Prompt the user to enter the number of students.

• Dynamically allocate memory to store the details of each student.

• Read and store the details of each student.

• Calculate the total and average marks for each student.

• Display the details of each student.

• Find and display the details of the student with the highest total marks.

• Free the dynamically allocated memory.

**Program**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define a structure to represent a student

struct Student {

char name[100];

int rollNumber;

int marks[3];

int totalMarks;

float averageMarks;

};

// Function prototypes

void readStudentDetails(struct Student\* student);

void calculateTotalAndAverageMarks(struct Student\* student);

void displayStudentDetails(const struct Student\* student);

struct Student findStudentWithHighestMarks(struct Student students[], int n);

int main() {

int n;

printf("Enter the number of students: "); // user to enter the number of students

scanf("%d", &n);

Department of Computer Science and Business Systems, Rajalakshmi Engineering College 21

// Dynamically allocate memory to store the students

struct Student\* students = (struct Student\*)malloc(n \* sizeof(struct Student));

if (students == NULL) {

printf("Memory allocation failed\n");

return 1;

}

// Read details for each student

for (int i = 0; i < n; ++i) {

printf("Enter details for student %d:\n", i + 1);

readStudentDetails(&students[i]);

calculateTotalAndAverageMarks(&students[i]);

}

// Display details of each student

printf("\nStudent Details:\n");

for (int i = 0; i < n; ++i) {

printf("Student %d:\n", i + 1);

displayStudentDetails(&students[i]);

printf("\n");

}

// Find and display the student with the highest total marks

struct Student topStudent = findStudentWithHighestMarks(students, n);

printf("Student with highest total marks:\n");

displayStudentDetails(&topStudent);

// Free the dynamically allocated memory

free(students);

return 0;

}

// Function to read details of a student

void readStudentDetails(struct Student\* student) {

printf("Name: ");

scanf("%s", student->name);

printf("Roll Number: ");

scanf("%d", &student->rollNumber);

printf("Marks in three subjects: ");

for (int i = 0; i < 3; ++i) {

scanf("%d", &student->marks[i]);

}

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

## 22

// Function to calculate total and average marks of a student

void calculateTotalAndAverageMarks(struct Student\* student) {

student->totalMarks = 0;

for (int i = 0; i < 3; ++i) {

student->totalMarks += student->marks[i];

}

student->averageMarks = student->totalMarks / 3.0;

}

// Function to display details of a student

void displayStudentDetails(const struct Student\* student) {

printf("Name: %s\n", student->name);

printf("Roll Number: %d\n", student->rollNumber);

printf("Total Marks: %d\n", student->totalMarks);

printf("Average Marks: %.2f\n", student->averageMarks);

}

// Function to find the student with the highest total marks

struct Student findStudentWithHighestMarks(struct Student students[], int n) {

struct Student topStudent = students[0];

for (int i = 1; i < n; ++i) {

if (students[i].totalMarks > topStudent.totalMarks) {

topStudent = students[i];

}

}

return topStudent;

}

**Dep**artment of Computer Science and Business Systems, Rajalakshmi Engineering College

23

**Output:**

Student Details:

Student 1:

Name: vv

Roll Number: 23

Total Marks: 234

Average Marks: 78.00

Student 2:

Name: ii

Roll Number: 11

Total Marks: 266

Average Marks: 88.67

Student 3:

Name: ss

Roll Number: 18

Total Marks: 113

Average Marks: 37.67

Student with highest total marks:

Name: ii

Roll Number: 11

Total Marks: 266

Average Marks: 88.67

### Result:

This program dynamically allocates memory to store student details, reads and processes the information, and identifies the student with the highest total marks. After processing, it frees the allocated memory to prevent memory leaks.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

24

|  |  |
| --- | --- |
| **Ex. No.: 3.a** | **Date:** |
| **Union** | |
| **Create a union named 'Data' that can hold an integer, float, and character. Write a program to demonstrate the use of this union by accepting input values for each type and displaying them.**  **Sample Input :**  Enter an integer value: 16 Enter a float value: 12.84 Enter a character value: AV **Sample Input :**  Values stored in the union: Integer value: 12.84  Float value: 12.84 Character value: A  **Aim:**  To demonstrate the use of a union in C, let's create a union named Data that can hold an integer, a float, and a character. The program will accept input values for each type, store them in the union, and display them.  **Algorithm:**  1. Start.  2. Define a union named Data with members for an integer, a float, and a character.  3. Declare a variable data of type Data.  4. Accept an integer input from the user and store it in data.integer.  5. Accept a float input from the user and store it in data.floating.  6. Accept a character input from the user and store it in data.character.  7. Display the values stored in the union:  • Print the integer value stored in data.integer.  • Print the float value stored in data.floating.  • Print the character value stored in data.character.  8. End. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

25

**Program:**

#include <stdio.h>

// Define a union named 'Data' that can hold an integer, float, and character

union Data {

int integer;

float floating;

char character;

};

int main() {

union Data data;

// Accept input for an integer value

printf("Enter an integer value: ");

scanf("%d", &data.integer);

// Accept input for a float value

printf("Enter a float value: ");

scanf("%f", &data.floating);

// Accept input for a character value

printf("Enter a character value: ");

scanf(" %c", &data.character);

// Display the values stored in the union

printf("\nValues stored in the union:\n");

printf("Integer value: %d\n", data.integer);

printf("Float value: %f\n", data.floating);

printf("Character value: %c\n", data.character);

return 0;

}

**Result:**

This program demonstrates the use of a union by accepting input values for an integer, a float, and a character, storing them in the union, and displaying them. Since a union can only hold one value at a time, storing a new value will overwrite the previous value

26

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

|  |  |
| --- | --- |
| **Ex. No.:3.b** | **Date:** |
| **Union For Representing Different Shapes (circle, square, rectangle)** | |
| **Develop a program that defines a union for representing different shapes (circle, square, rectangle) with appropriate fields for each. Implement functions to:**  **Calculate and display the area of each shape. Determine and print the largest area among the shapes.**  **Input Format**  Enter the number of shapes: 3 Enter type of shape (0 for circle, 1  for square, 2 for rectangle) for shape 1: 0  Enter radius of circle: 33  Enter type of shape (0 for circle, 1 for square, 2 for rectangle) for shape 2: 2  Enter length and width of rectangle: 45 67  Enter type of shape (0 for circle, 1 for square, 2 for rectangle) for shape 3: 1  Enter side length of square: 23  **Output Format**  Areas of shapes:  Area of circle 1: 3421.19  Area of square 2: 2025.00  Area of rectangle 3: 0.00  Largest area among the shapes: 3421.19  **Aim:**  To develop a program that defines a union for representing different shapes (circle, square, rectangle) and implements functions to calculate and display the area of each shape, as well as determine and print the largest area among the shapes:  **Algorithm:**  **1.** Start  2. Define a union named Shape with appropriate fields for representing different shapes:  • For a circle, include a field for radius.  • For a square, include a field for side length.  • For a rectangle, include fields for length and width.  3. Define a structure named Circle with a field for the radius.  4. Define a structure named Square with a field for the side length.  5. Define a structure named Rectangle with fields for the length and width.  6. Define a function calculateArea that takes a union Shape as input and returns the calculated area based on the type of shape (circle, square, or rectangle).  7. Define a function displayAreas that takes an array of unions shapes and its size as input, calculates and displays the area of each shape.  8. Define a function findLargestArea that takes an array of unions shapes and its size as input, determines the largest area among the shapes, and returns it.  9. In the main function: a. the user to enter the number of shapes. b. Dynamically allocate memory to store the shapes. c. Read details for each shape: - For a circle, prompt the user to enter the radius. - For a square, prompt the user to enter the side length. - For a rectangle, prompt the user to enter the length and width. d. Calculate and display the area of each shape using the displayAreas function. e. Determine and print the largest area among the shapes using the findLargestArea function. f. Free dynamically allocated memory.  10. End. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

27

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Define constants for shape types

#define CIRCLE 0

#define SQUARE 1

#define RECTANGLE 2

// Define a structure for a circle

struct Circle {

double radius;

};

// Define a structure for a square

struct Square {

double sideLength;

};

// Define a structure for a rectangle

struct Rectangle {

double length;

double width;

};

// Define a union for representing different shapes

union Shape {

struct Circle circle;

struct Square square;

struct Rectangle rectangle;

};

// Function to calculate the area of a circle

double calculateCircleArea(struct Circle c) {

return M\_PI \* c.radius \* c.radius;

}

// Function to calculate the area of a square

double calculateSquareArea(struct Square s) {

return s.sideLength \* s.sideLength;

}

// Function to calculate the area of a rectangle

double calculateRectangleArea(struct Rectangle r) {

return r.length \* r.width;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

28

### Program

// Function to calculate and display the area of each shape

void displayAreas(union Shape shapes[], int numShapes) {

printf("Areas of shapes:\n");

for (int i = 0; i < numShapes; i++) {

double area;

switch (shapes[i].circle.radius) {

case CIRCLE:

area = calculateCircleArea(shapes[i].circle);

printf("Area of circle %d: %.2f\n", i + 1, area);

break;

case SQUARE:

area = calculateSquareArea(shapes[i].square);

printf("Area of square %d: %.2f\n", i + 1, area);

break;

case RECTANGLE:

area = calculateRectangleArea(shapes[i].rectangle);

printf("Area of rectangle %d: %.2f\n", i + 1, area);

break;

}

}

}

// Function to find the largest area among the shapes

double findLargestArea(union Shape shapes[], int numShapes) {

double largestArea = 0.0;

for (int i = 0; i < numShapes; i++) {

double area;

switch (shapes[i].circle.radius) {

case CIRCLE:

area = calculateCircleArea(shapes[i].circle);

break;

case SQUARE:

area = calculateSquareArea(shapes[i].square);

break;

case RECTANGLE:

area = calculateRectangleArea(shapes[i].rectangle);

break;

}

if (area > largestArea) {

largestArea = area;

}

}

return largestArea;

}

29

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

int main() {

int numShapes;

// the user to enter the number of shapes

printf("Enter the number of shapes: ");

scanf("%d", &numShapes);

// Dynamically allocate memory to store the shapes

union Shape\* shapes = (union Shape\*)malloc(numShapes \* sizeof(union Shape));

if (shapes == NULL) {

printf("Memory allocation failed\n");

return 1;

}

// Read details for each shape

for (int i = 0; i < numShapes; i++) {

int shapeType;

printf("Enter type of shape (0 for circle, 1 for square, 2 for rectangle) for shape %d: ", i + 1);

scanf("%d", &shapeType);

switch (shapeType) {

case CIRCLE:

printf("Enter radius of circle: ");

scanf("%lf", &shapes[i].circle.radius);

break;

case SQUARE:

printf("Enter side length of square: ");

scanf("%lf", &shapes[i].square.sideLength);

break;

case RECTANGLE:

printf("Enter length and width of rectangle: ");

scanf("%lf %lf", &shapes[i].rectangle.length, &shapes[i].rectangle.width);

break;

default:

printf("Invalid shape type\n");

free(shapes);

return 1;

}

}

// Display the area of each shape

displayAreas(shapes, numShapes);

// Find and print the largest area among the shapes

double largestArea = findLargestArea(shapes, numShapes);

printf("\nLargest area among the shapes: %.2f\n", largestArea);

// Free dynamically allocated memory

free(shapes);

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

30

**Input**

Enter the number of shapes: 3

Enter type of shape (0 for circle, 1 for square, 2 for rectangle) for shape 1: 0

Enter radius of circle: 33

Enter type of shape (0 for circle, 1 for square, 2 for rectangle) for shape 2: 2

Enter length and width of rectangle: 45 67

Enter type of shape (0 for circle, 1 for square, 2 for rectangle) for shape 3: 1

Enter side length of square: 23

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

31

Areas of shapes:

Area of circle 1: 3421.19

Area of rectangle 2: 3015.00

Area of square 3: 529.00

Largest area among the shapes: 3421.19

**Result:**

32

Above program that defines a union for representing different shapes (circle, square, rectangle) and implemented functions to calculate and display the area of each shape, as well as output and print the largest area among the shapes

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

|  |  |
| --- | --- |
| **Ex. No. : 4.a** | **Date :** |
| **Implement a stack - push, pop, and peek operations** | |
| Implement a stack using an array in C program. Provide functions/methods for push, pop, and peek operations. Provide the code and demonstrate its usage with examples  **Output Format**  Top element: 40  Popped element: 40  Top element after pop: 30  **Aim:**  To implement a stack using an array in C and provide functions for push, pop, and peek operations.  **Algorithm:**  1. Initialize Stack:  • Define a structure for the stack.  • Initialize the stack with a maximum size and an array to hold elements.  • Initialize the top of the stack to -1 (indicating an empty stack).  2. Push Operation:  • Check if the stack is full.  • If not full, increment the top and add the element to the stack.  3. Pop Operation:  • Check if the stack is empty.  • If not empty, return the element at the top and decrement the top.  4. Peek Operation:  • Check if the stack is empty.  • If not empty, return the element at the top without removing it. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

33

**Program:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct {

int items[MAX];

int top;

} Stack;

// Function to initialize the stack

void initialize(Stack \*s) {

s->top = -1;

}

// Function to check if the stack is empty

int isEmpty(Stack \*s) {

return s->top == -1;

}

// Function to check if the stack is full

int isFull(Stack \*s) {

return s->top == MAX - 1;

}

// Function to add an element to the stack

void push(Stack \*s, int newItem) {

if (isFull(s)) {

printf("Stack is full. Cannot push %d\n", newItem);

return;

}

s->items[++(s->top)] = newItem;

}

// Function to remove an element from the stack

int pop(Stack \*s) {

if (isEmpty(s)) {

printf("Stack is empty. Cannot pop\n");

return -1;

}

return s->items[(s->top)--];

}

// Function to get the top element of the stack without removing it

int peek(Stack \*s) {

if (isEmpty(s)) {

printf("Stack is empty. Cannot peek\n");

return -1;

}

return s->items[s->top];

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

34

35

**Program:**

int main() {

Stack s;

initialize(&s);

// Demonstrating the usage of the stack

push(&s, 10);

push(&s, 20);

push(&s, 30);

push(&s, 40);

printf("Top element: %d\n", peek(&s));

printf("Popped element: %d\n", pop(&s));

printf("Top element after pop: %d\n", peek(&s));

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Program:**

Top element: 40

Popped element: 40

Top element after pop: 30

### Result:

The stack operations (push, pop, and peek) using an array in C, showing the top element before and after a pop operation.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

36

|  |  |
| --- | --- |
| **Ex. No. :4.b** | **Date :** |
|  | **Implement a queue using an array** |
| Implement a queue using an array in C programming language. Include functions/methods for enqueue, dequeue, and peek operations. Provide the code implementation and demonstrate its usage with examples.  **Sample Output**  Dequeued element: 10  Dequeued element: 20  Front element: 30  **Aim:**  To implement a stack using an array in C and provide functions for push, pop, and peek operations.  **Algorithm:**  1. Initialize Queue:  • Define a structure for the queue.  • Initialize the queue with a maximum size and an array to hold elements.  • Initialize front and rear to -1 (indicating an empty queue).  2. Enqueue Operation:  • Check if the queue is full.  • If not full, increment rear and add the element to the queue.  • If the queue was initially empty, set front to 0.  3. Dequeue Operation:  • Check if the queue is empty.  • If not empty, return the element at front and increment front.  • If after the operation the queue becomes empty, reset front and rear to -1.  4. Peek Operation:  • Check if the queue is empty.  • If not empty, return the element at front without removing it. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

37

**Program:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct {

int items[MAX];

int front;

int rear;

} Queue;

// Function to initialize the queue

void initialize(Queue \*q) {

q->front = -1;

q->rear = -1;

}

// Function to check if the queue is empty

int isEmpty(Queue \*q) {

return q->front == -1;

}

// Function to check if the queue is full

int isFull(Queue \*q) {

return q->rear == MAX - 1;

}

// Function to add an element to the queue

void enqueue(Queue \*q, int newItem) {

if (isFull(q)) {

printf("Queue is full. Cannot enqueue %d\n", newItem);

return;

}

if (isEmpty(q)) {

q->front = 0;

}

q->items[++(q->rear)] = newItem;

}

// Function to remove an element from the queue

int dequeue(Queue \*q) {

if (isEmpty(q)) {

printf("Queue is empty. Cannot dequeue\n");

return -1;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

38

39

### Program:

int item = q->items[q->front];

if (q->front == q->rear) {

q->front = q->rear = -1;

} else {

q->front++;

}

return item;

}

// Function to get the front element of the queue without removing it

int peek(Queue \*q) {

if (isEmpty(q)) {

printf("Queue is empty. Cannot peek\n");

return -1;

}

return q->items[q->front];

}

int main() {

Queue q;

initialize(&q);

// Demonstrating the usage of the queue

enqueue(&q, 10);

enqueue(&q, 20);

enqueue(&q, 30);

enqueue(&q, 40);

printf("Dequeued element: %d\n", dequeue(&q));

printf("Dequeued element: %d\n", dequeue(&q));

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

40

**Program:**

printf("Front element: %d\n", peek(&q));

return 0;

}

**OUTPUT:**

Dequeued element: 10

Dequeued element: 20

Front element: 30

### Result:

### The queue operations (enqueue, dequeue, and peek) using an array in C, showing the elements being dequeued and the front element after dequeuing.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

|  |  |
| --- | --- |
| **Ex. No. :5.a** | **Date :** |
| **Postfix/ Prefix Expression Using A Stack.** | |
| Write a program in C programming language to evaluate a postfix/ prefix expression using a stack. Explain the postfix/ prefix evaluation algorithm and provide the code implementation with examples.  **Sample Output**  Postfix expression evaluation: 35 Prefix expression evaluation: 35 **Aim:**  To evaluate a postfix and prefix expression using a stack in C programming language**.**  **Algorithm:**  Postfix Evaluation:  1. Initialize an empty stack.  2. Scan the postfix expression from left to right.  3. For each character:  • If the character is an operand, push it onto the stack.  • If the character is an operator, pop the required number of operands from the stack, perform the operation, and push the result back onto the stack.  4. The final result will be at the top of the stack.  Prefix Evaluation:  1. Initialize an empty stack.  2. Scan the prefix expression from right to left.  3. For each character:  • If the character is an operand, push it onto the stack.  • If the character is an operator, pop the required number of operands from the stack, perform the operation, and push the result back onto the stack.  4. The final result will be at the top of the stack. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

41

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

typedef struct {

int items[MAX];

int top;

} Stack;

// Function to initialize the stack

void initialize(Stack \*s) {

s->top = -1;

}

// Function to check if the stack is empty

int isEmpty(Stack \*s) {

return s->top == -1;

}

// Function to check if the stack is full

int isFull(Stack \*s) {

return s->top == MAX - 1;

}

// Function to push an element onto the stack

void push(Stack \*s, int newItem) {

if (isFull(s)) {

printf("Stack is full. Cannot push %d\n", newItem);

return;

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

42

**Program**

}

s->items[++(s->top)] = newItem;

}

// Function to pop an element from the stack

int pop(Stack \*s) {

if (isEmpty(s)) {

printf("Stack is empty. Cannot pop\n");

return -1;

}

return s->items[(s->top)--];

}

// Function to evaluate a postfix expression

int evaluatePostfix(char \*exp) {

Stack s;

initialize(&s);

for (int i = 0; exp[i]; i++) {

if (isdigit(exp[i])) {

push(&s, exp[i] - '0');

} else {

int val2 = pop(&s);

int val1 = pop(&s);

switch (exp[i]) {

case '+': push(&s, val1 + val2); break;

case '-': push(&s, val1 - val2); break;

case '\*': push(&s, val1 \* val2); break;

case '/': push(&s, val1 / val2); break;

}

}

}

return pop(&s);

}

// Function to evaluate a prefix expression

int evaluatePrefix(char \*exp) {

Stack s;

initialize(&s);

int length = strlen(exp);

for (int i = length - 1; i >= 0; i--) {

if (isdigit(exp[i])) {

push(&s, exp[i] - '0');

} else {

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

43

**Program:**

int val1 = pop(&s);

int val2 = pop(&s);

switch (exp[i]) {

case '+': push(&s, val1 + val2); break;

case '-': push(&s, val1 - val2); break;

case '\*': push(&s, val1 \* val2); break;

case '/': push(&s, val1 / val2); break;

}

}

}

return pop(&s);

}

int main() {

char postfixExp[] = "53+82-\*";

char prefixExp[] = "-+5382";

int postfixResult = evaluatePostfix(postfixExp);

int prefixResult = evaluatePrefix(prefixExp);

printf("Postfix expression evaluation: %d\n", postfixResult);

printf("Prefix expression evaluation: %d\n", prefixResult);

return 0;

}

**OUTPUT**

Postfix expression evaluation: 35

Prefix expression evaluation: 35

### Result:

The evaluation of both postfix and prefix expressions using a stack in C, showing the correct results for the provided expressions.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

44

|  |  |
| --- | --- |
| **Ex. No. :5.b** | **Date :** |
| **Tower of Hanoi** | |
| **Objective:**  To understand and implement the Tower of Hanoi problem using stack data structure in C.  **Problem Statement:**  The Tower of Hanoi is a classic problem that involves moving disks from one peg to another, subject to the following constraints:   1. Only one disk can be moved at a time. 2. Each move consists of taking the top disk from one stack and placing it on top of another stack. 3. No disk may be placed on top of a smaller disk.   Implement the Tower of Hanoi problem using the stack data structure. You are required to implement the following functions:  void towerOfHanoi(int numDisks, char source, char auxiliary, char destination): A recursive function to solve the Tower of Hanoi problem. It takes the number of disks (numDisks), and characters representing the source, auxiliary, and destination pegs (source, auxiliary, destination). void displayMove(char source, char destination): A function to display the move of disks from the source peg to the destination peg.  Instructions:  Implement the towerOfHanoi function using recursion to solve the Tower of Hanoi problem.  In the towerOfHanoi function, use stacks to represent the pegs. You can implement a stack using an array or linked list.  Implement the displayMove function to display the move of disks from one peg to another. Test your implementation with different numbers of disks.  **Sample Input/Output:**  Enter the number of disks: 3  Moves to solve the Tower of Hanoi with 3 disks: Move disk 1 from A to C  Move disk 2 from A to B Move disk 1 from C to B Move disk 3 from A to C Move disk 1 from B to A Move disk 2 from B to C Move disk 1 from A to C **Aim:**  To implement the Tower of Hanoi problem using a stack data structure in C and understand the recursive solution for the problem.  **Algorithm:**  1. Initialize Stacks:  • Define a stack structure to represent the pegs.  • Initialize stacks for source, auxiliary, and destination pegs. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

45

2. Display Move:

• Implement the displayMove function to print the movement of disks from the source peg to the destination peg.

3. Recursive Tower of Hanoi:

• Implement the towerOfHanoi function using recursion.

• If there is only one disk, move it directly from the source peg to the destination peg and display the move.

• For more than one disk:

• Recursively move the top 𝑛−1n−1 disks from the source peg to the auxiliary peg.

• Move the nth disk from the source peg to the destination peg.

• Recursively move the 𝑛−1n−1 disks from the auxiliary peg to the destination peg.

4. Main Function:

• Take the number of disks as input.

• Initialize the pegs.

• Call the towerOfHanoi function to solve the problem.

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Stack {

int \*items;

int top;

int max\_size;

} Stack;

// Function to initialize the stack

void initialize(Stack \*s, int max\_size) {

s->items = (int \*)malloc(max\_size \* sizeof(int));

s->top = -1;

s->max\_size = max\_size;

}

// Function to check if the stack is empty

int isEmpty(Stack \*s) {

return s->top == -1;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

46

### Program

// Function to push an element onto the stack

void push(Stack \*s, int newItem) {

if (s->top == s->max\_size - 1) {

printf("Stack is full. Cannot push %d\n", newItem);

return;

}

s->items[++(s->top)] = newItem;

}

// Function to pop an element from the stack

int pop(Stack \*s) {

if (isEmpty(s)) {

printf("Stack is empty. Cannot pop\n");

return -1;

}

return s->items[(s->top)--];

}

// Function to display the move

void displayMove(char source, char destination, int disk) {

printf("Move disk %d from %c to %c\n", disk, source, destination);

}

// Recursive function to solve Tower of Hanoi problem

void towerOfHanoi(int numDisks, Stack \*source, Stack \*auxiliary, Stack \*destination, char s, char a, char d) {

if (numDisks == 1) {

int disk = pop(source);

push(destination, disk);

displayMove(s, d, disk);

return;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

47

**Program:**

towerOfHanoi(numDisks - 1, source, destination, auxiliary, s, d, a);

int disk = pop(source);

push(destination, disk);

displayMove(s, d, disk);

towerOfHanoi(numDisks - 1, auxiliary, source, destination, a, s, d);

}

int main() {

int numDisks;

printf("Enter the number of disks: ");

scanf("%d", &numDisks);

Stack source, auxiliary, destination;

initialize(&source, numDisks);

initialize(&auxiliary, numDisks);

initialize(&destination, numDisks);

for (int i = numDisks; i >= 1; i--) {

push(&source, i);

}

printf("Moves to solve the Tower of Hanoi with %d disks:\n", numDisks);

towerOfHanoi(numDisks, &source, &auxiliary, &destination, 'A', 'B', 'C');

// Free allocated memory

free(source.items);

free(auxiliary.items);

free(destination.items);

return 0;

}

**OUTPUT:**

Enter the number of disks: 3

Moves to solve the Tower of Hanoi with 3 disks:

Move disk 1 from A to C

Move disk 2 from A to B

Move disk 1 from C to B

Move disk 3 from A to C

Move disk 1 from B to A

Move disk 2 from B to C

Move disk 1 from A to C

**Result**:

The solution to the Tower of Hanoi problem using a stack data structure in C, showing the sequence of moves required to transfer the disks from the source peg to the destination peg following the problem constraints.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

48

|  |  |
| --- | --- |
| **Ex. No. :5.c** | **Date :** |
| **The Evaluation Of Infix Expressions Using A Stack** | |
| **To understand and implement the evaluation of infix expressions using a stack data structure in C.**  **Problem Statement:**  Infix expressions are mathematical expressions where operators are placed between operands. For example, 2 + 3 \* 4-5  In order to evaluate infix expressions, we need to follow the rules of operator precedence and associativity.  Implement a C program to evaluate infix expressions using a stack data structure. You are required to implement the following functions:   1. int evaluateInfix(char\* expression): A function that evaluates an infix expression and returns the result. The input expression is provided as a string. 2. int precedence(char operator): A function that returns the precedence of an operator. Operators with higher precedence should have a higher return value. 3. int applyOperator(int operand1, char operator, int operand2): A function that applies the given operator to the two operands and returns the result.   Instructions:   * 1. Implement the evaluateInfix function using a stack to evaluate the infix expression. You can use two stacks - one for operands and one for operators.   2. Implement the precedence function to determine the precedence of operators according to mathematical rules.   3. Implement the applyOperator function to apply the given operator to the two operands and return the result.   4. Test your implementation with different infix expressions.   **Sample Input/Output:**  Enter the infix expression: (2 + 3) \* 4 - 5 Result of infix expression evaluation: 15 **Aim:**  To understand and implement the evaluation of infix expressions using a stack data structure in C.  **Algorithm:**  1. Initialize Stacks:  • Use two stacks: one for operands and one for operators.  2. Precedence Function:  • Implement the precedence function to determine the precedence of operators according to mathematical rules.  3. Apply Operator Function:  • Implement the applyOperator function to apply the given operator to two operands and return the result.  4. Evaluate Infix Function:  • Implement the evaluateInfix function to evaluate the infix expression using stacks:  • Scan the expression from left to right.  • For each character in the expression:  • If it is an operand, push it onto the operand stack.  • If it is an operator, pop from the operator stack and operand stack to evaluate the expression and push the result back to the operand stack based on precedence.  • Handle parentheses by pushing them onto the operator stack or evaluating the expression inside the parentheses when encountering a closing parenthesis.  • After scanning the expression, apply the remaining operators in the stack. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

49

**Program:**

**#**include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

typedef struct {

int items[MAX];

int top;

} Stack;

// Function to initialize the stack

void initialize(Stack \*s) {

s->top = -1;

}

// Function to check if the stack is empty

int isEmpty(Stack \*s) {

return s->top == -1;

}

// Function to check if the stack is full

int isFull(Stack \*s) {

return s->top == MAX - 1;

}

// Function to push an element onto the stack

void push(Stack \*s, int newItem) {

if (isFull(s)) {

printf("Stack is full. Cannot push %d\n", newItem);

return;

}

s->items[++(s->top)] = newItem;

}

// Function to pop an element from the stack

int pop(Stack \*s) {

if (isEmpty(s)) {

printf("Stack is empty. Cannot pop\n");

return -1;

}

return s->items[(s->top)--];

}

// Function to get the precedence of operators

int precedence(char operator) {

switch (operator) {

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

50

### Program:

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return 0;

}

}

// Function to apply an operator to two operands

int applyOperator(int operand1, char operator, int operand2) {

switch (operator) {

case '+':

return operand1 + operand2;

case '-':

return operand1 - operand2;

case '\*':

return operand1 \* operand2;

case '/':

return operand1 / operand2;

case '^':

return (int)pow(operand1, operand2);

default:

return 0;

}

}

/ / Function to evaluate an infix expression

int evaluateInfix(char\* expression) {

Stack operands, operators;

initialize(&operands);

initialize(&operators);

for (int i = 0; i < strlen(expression); i++) {

if (expression[i] == ' ')

continue;

if (isdigit(expression[i])) {

int value = 0;

while (i < strlen(expression) && isdigit(expression[i])) {

value = (value \* 10) + (expression[i] - '0');

i++;

}

i--;

push(&operands, value);

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

51

52

**Program:**

} else if (expression[i] == '(') {

push(&operators, expression[i]);

} else if (expression[i] == ')') {

while (!isEmpty(&operators) && operators.items[operators.top] != '(') {

int operand2 = pop(&operands);

int operand1 = pop(&operands);

char operator = pop(&operators);

push(&operands, applyOperator(operand1, operator, operand2));

}

pop(&operators); // Remove the '('

} else {

while (!isEmpty(&operators) && precedence(operators.items[operators.top]) >= precedence(expression[i])) {

int operand2 = pop(&operands);

int operand1 = pop(&operands);

char operator = pop(&operators);

push(&operands, applyOperator(operand1, operator, operand2));

}

push(&operators, expression[i]);

}

}

while (!isEmpty(&operators)) {

int operand2 = pop(&operands);

int operand1 = pop(&operands);

char operator = pop(&operators);

push(&operands, applyOperator(operand1, operator, operand2));

}

return pop(&operands);

}

int main() {

char expression[MAX];

printf("Enter the infix expression: ");

fgets(expression, MAX, stdin);

expression[strcspn(expression, "\n")] = '\0';

int result = evaluateInfix(expression);

printf("Result of infix expression evaluation: %d\n", result);

return 0;

}

OUTPUT:

Enter the infix expression: (2 + 3) \* 4 - 5

Result of infix expression evaluation: 15

**Result:**

The evaluation of an infix expression using a stack data structure in C, correctly handling operator precedence and parentheses to produce the correct result.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

|  |  |
| --- | --- |
| **Ex.No: 6.a** | **Date :** |
| **Implementation the following operations on Singly Linked List.** | |
| Write a C program to implement the following operations on Singly Linked List.   1. Insert a node in the beginning of a list. 2. Insert a node after P 3. Insert a node at the end of a list 4. Find an element in a list 5. FindNext 6. FindPrevious 7. isLast 8. isEmpty 9. Delete a node in the beginning of a list. 10. Delete a node after P 11. Delete a node at the end of a list 12. Delete the List   **Sample Output:**   * 1. Insert at beginning   2. Insert after a node   3. Insert at end   4. Find an element   5. Find next   6. Find previous   7. Check if last   8. Check if empty   9. Delete from beginning   10. Delete after a node   11. Delete from end   12. Delete the list   13. Display   14. Exit   Enter your choice: 1  **Aim:**  To implement a Singly Linked List and perform various operations on it.  **Algorithm:**  1. Create a structure for the node of the linked list.  2. Implement functions for each operation:  • Insert at beginning: Create a new node and add it to the beginning of the list.  • Insert after a node: Create a new node and add it after a specified node.  • Insert at end: Create a new node and add it to the end of the list.  • Find an element: Traverse the list to find a specified element.  • Find next: Return the next node of a specified node.  • Find previous: Return the previous node of a specified node.  • Check if last: Check if a node is the last node in the list.  • Check if empty: Check if the list is empty.  • Delete from beginning: Remove the first node from the list.  • Delete after a node: Remove the node after a specified node.  • Delete from end: Remove the last node from the list.  • Delete the list: Remove all nodes from the list.  • Display: Print the elements of the list.  3. Create a menu-driven program to perform these operations. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

53

**Program:**

#include <stdio.h>

#include <stdlib.h>

// Structure for a node

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Function to insert a node at the beginning of the list

void insertAtBeginning(Node\*\* head, int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

}

// Function to insert a node after a specified node

void insertAfter(Node\* prev, int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = prev->next;

prev->next = newNode;

}

// Function to insert a node at the end of the list

void insertAtEnd(Node\*\* head, int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

\*head = newNode;

} else {

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

// Function to find an element in the list

Node\* findElement(Node\* head, int data) {

while (head != NULL) {

if (head->data == data) {

return head;

}

head = head->next;

}

return NULL;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

54

55

**Program:**

// Function to find the next node of a specified node

Node\* findNext(Node\* node) {

return node->next;

}

// Function to find the previous node of a specified node

Node\* findPrevious(Node\* head, Node\* node) {

if (node == head) {

return NULL;

}

Node\* temp = head;

while (temp->next != node) {

temp = temp->next;

}

return temp;

}

// Function to check if a node is the last node in the list

int isLast(Node\* node, Node\* head) {

while (head->next != NULL) {

head = head->next;

}

return head == node;

}

// Function to check if the list is empty

int isEmpty(Node\* head) {

return head == NULL;

}

/

// Function to delete a node from the beginning of the list

void deleteFromBeginning(Node\*\* head) {

Node\* temp = \*head;

\*head = (\*head)->next;

free(temp);

}

// Function to delete a node after a specified node

void deleteAfter(Node\* prev) {

Node\* temp = prev->next;

prev->next = temp->next;

free(temp);

}

// Function to delete a node from the end of the list

void deleteFromEnd(Node\*\* head) {

if (\*head == NULL) {

return;

}

if ((\*head)->next == NULL) {

free(\*head);

\*head = NULL;

return;

}

Node\* temp = \*head;

while (temp->next->next != NULL) {

temp = temp->next;

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

56

**Program:**

}

free(temp->next);

temp->next = NULL;

}

// Function to delete the entire list

void deleteList(Node\*\* head) {

Node\* temp = \*head;

while (temp != NULL) {

Node\* next = temp->next;

free(temp);

temp = next;

}

\*head = NULL;

}

// Function to display theelements of the list

void display(Node\* head) {

while (head != NULL) {

printf("%d -> ", head->data);

head = head->next;

}

printf("NULL\n");

}

int main() {

Node\* head = NULL;

int choice, data;

while (1) {

printf("\n1. Insert at beginning\n2. Insert after a node\n3. Insert at end\n4. Find an element\n5. Find next\n6. Find previous\n7. Check if last\n8. Check if empty\n9. Delete from beginning\n10. Delete after a node\n11. Delete from end\n12. Delete the list\n13. Display\n14. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data: ");

scanf("%d", &data);

insertAtBeginning(&head, data);

break;

case 2:

printf("Enter data and node data: ");

scanf("%d%d", &data, &data);

Node\* node = findElement(head, data);

if (node != NULL) {

insertAfter(node, data);

} else {

printf("Node not found\n");

}

break;

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Program:**

case 3:

printf("Enter data: ");

scanf("%d", &data);

insertAtEnd(&head, data);

break;

case 4:

printf("Enter data: ");

scanf("%d", &data);

Node\* found = findElement(head, data);

if (found != NULL) {

printf("Element found\n");

} else {

printf("Element not found\n");

}

break;

case 5:

printf("Enter node data: ");

scanf("%d", &data);

Node\* next = findNext(findElement(head, data));

if (next != NULL) {

printf("Next node: %d\n", next->data);

} else {

printf("No next node\n");

}

break;

case 6:

printf("Enter node data: ");

scanf("%d", &data);

Node\* prev = findPrevious(head, findElement(head, data));

if (prev != NULL) {

printf("Previous node: %d\n", prev->data);

} else {

printf("No previous node\n");

}

break;

case 7:

printf("Enter node data: ");

scanf("%d", &data);

Node\* last = findElement(head, data);

if (isLast(last, head)) {

printf("Last node\n");

} else {

printf("Not last node\n");

}

break;

case 8:

if (isEmpty(head)) {

printf("List is empty\n");

} else {

printf("List is not empty\n");

}

break;

case 9:

deleteFromBeginning(&head);

break;

case 10:

printf("Enter node data: ");

scanf("%d", &data);

Node\* delAfter = findElement(head, data);

if (delAfter != NULL && delAfter->next != NULL) {

deleteAfter(delAfter);

} else {

printf("No node to delete after\n");

}

break;

case 11:

deleteFromEnd(&head);

break;

case 12:

deleteList(&head);

break;

case 13:

display(head);

break;

case 14:

exit(0);

default:

printf("Invalid choice\n");

break;

}

}

return 0;

}

### Result:

### The program will display a menu with various options to perform operations on a Singly Linked List. The user can choose an option and enter the required data to perform the operation. The program will display the result of the operation and return to the menu to perform another operation. The user can exit the program by choosing option 14.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

57

|  |  |
| --- | --- |
| **Ex. No. : 6.b** | **Date :** |
| **Doubly Linked List** | |
| **Write a C p r o g r a m t o i m p l e m e n t t h e f o l l o w i n g o p e r a t i o n s o n D o u b l y L i n k e d L i s t .**   1. **Insertion** 2. **Deletion** 3. **Search** 4. **Display Sample Output**   . Insert at beginning   1. Insert at end 2. Delete from beginning 3. Delete from end 4. Search 5. Display 6. Exit   Enter your choice: 6 Doubly linked list: NULL   1. Insert at beginning 2. Insert at end 3. Delete from beginning 4. Delete from end 5. Search 6. Display 7. Exit   Enter your choice: 3 List is empty  **Aim:**  To implement operations on a doubly linked list including insertion, deletion, search, and display.  **Algorithm:**  1. Initialization:  • Define a structure for the doubly linked list nodes.  2. Insertion:  • Insert at the beginning.  • Insert at the end.  3. Deletion:  • Delete from the beginning.  • Delete from the end.  4. Search:  • Search for an element in the list.  5. Display:  • Display the elements of the list. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

58

**Program:**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a doubly linked list node

typedef struct Node {

int data;

struct Node\* prev;

struct Node\* next;

} Node;

// Function to create a new node

Node\* createNode(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (!newNode) {

printf("Memory allocation failed\n");

return NULL;

}

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

// Function to insert a node at the beginning of the list

void insertAtBeginning(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (!newNode) return;

if (\*head != NULL) {

(\*head)->prev = newNode;

}

newNode->next = \*head;

\*head = newNode;

}

// Function to insert a node at the end of the list

void insertAtEnd(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (!newNode) return;

if (\*head == NULL) {

\*head = newNode;

return;

}

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

59

**Program:**

// Function to delete a node from the beginning of the list

void deleteFromBeginning(Node\*\* head) {

if (\*head == NULL) {

printf("List is empty\n");

return;

}

Node\* temp = \*head;

\*head = (\*head)->next;

if (\*head != NULL) {

(\*head)->prev = NULL;

}

free(temp);

}

// Function to delete a node from the end of the list

void deleteFromEnd(Node\*\* head) {

if (\*head == NULL) {

printf("List is empty\n");

return;

}

Node\* temp = \*head;

if (temp->next == NULL) {

free(temp);

\*head = NULL;

return;

}

while (temp->next != NULL) {

temp = temp->next;

}

temp->prev->next = NULL;

free(temp);

}

// Function to search for an element in the list

Node\* search(Node\* head, int data) {

Node\* temp = head;

while (temp != NULL) {

if (temp->data == data) {

return temp;

}

temp = temp->next;

}

return NULL;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

60

// Function to display the elements of the list

void displayList(Node\* head) {

if (head == NULL) {

printf("Doubly linked list: NULL\n");

return;

}

Node\* temp = head;

printf("Doubly linked list: ");

while (temp != NULL) {

printf("%d <-> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

// Main function with menu-driven interaction

int main() {

Node\* head = NULL;

int choice, data;

Node\* result;

while (1) {

printf("\n1. Insert at beginning\n");

printf("2. Insert at end\n");

printf("3. Delete from beginning\n");

printf("4. Delete from end\n");

printf("5. Search\n");

printf("6. Display\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

insertAtBeginning(&head, data);

break;

case 2:

printf("Enter data to insert at end: ");

scanf("%d", &data);

insertAtEnd(&head, data);

break;

case 3:

deleteFromBeginning(&head);

break;

case 4:

deleteFromEnd(&head);

break;

case 5:

printf("Enter data to search: ");

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

61

**Program:**

scanf("%d", &data);

result = search(head, data);

if (result != NULL) {

printf("Node with data %d found\n", data);

} else {

printf("Node with data %d not found\n", data);

}

break;

case 6:

displayList(head);

break;

case 7:

exit(0);

default:

printf("Invalid choice, please try again\n");

}

}

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

62

OUTPUT:

1. Insert at beginning

2. Insert at end

3. Delete from beginning

4. Delete from end

5. Search

6. Display

7. Exit

Enter your choice: 1

Enter data to insert at beginning: 10

1. Insert at beginning

2. Insert at end

3. Delete from beginning

4. Delete from end

5. Search

6. Display

7. Exit

Enter your choice: 6

Doubly linked list: 10 <-> NULL

1. Insert at beginning

2. Insert at end

3. Delete from beginning

4. Delete from end

5. Search

6. Display

7. Exit

Enter your choice: 3

List is empty

1. Insert at beginning

2. Insert at end

3. Delete from beginning

4. Delete from end

5. Search

6. Display

7. Exit

Enter your choice: 7

**Result:**

The program is executed, it provides a menu-driven interface for performing operations on a doubly linked list.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

63

64

|  |  |
| --- | --- |
| **Ex. No. :6.c** | **Date :** |
|  | **Circular Linked List** |
| Write a C program to implement a Circular Linked List with the following operations: Insertion:   1. Insert a node at the beginning of the circular list. 2. Insert a node at the end of the circular list. 3. Insert a node after a specific node in the circular list. Deletion:    1. Delete a node from the beginning of the circular list.    2. Delete a node from the end of the circular list.    3. Delete a node after a specific node in the circular list. Search and Display:       1. Search for a given element in the circular list.       2. Display the elements of the circular list. 4. Insert at beginning 5. Insert at end 6. Insert after a specific node 7. Delete from beginning 8. Delete from end 9. Delete after a specific node 10. Search for an element 11. Display the circular list 12. Exit   Enter your choice: 1  Enter data to insert at beginning: 23  **Aim:**  To implement a Circular Linked List in C and perform various operations such as insertion, deletion, search, and display.  **Algorithm:**  1. Initialization:  • Define a structure for the circular linked list nodes.  2. Insertion:  • Insert at the beginning.  • Insert at the end.  • Insert after a specific node.  3. Deletion:  • Delete from the beginning.  • Delete from the end.  • Delete after a specific node.  4. Search and Display:  • Search for an element in the list.  • Display the elements of the list. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Program:**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a circular linked list node

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Function to create a new node

Node\* createNode(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (!newNode) {

printf("Memory allocation failed\n");

return NULL;

}

newNode->data = data;

newNode->next = newNode; // Initialize next as circular

return newNode;

}

// Function to insert a node at the beginning of the circular list

void insertAtBeginning(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (!newNode) return;

if (\*head == NULL) {

\*head = newNode;

} else {

Node\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

newNode->next = \*head;

temp->next = newNode;

\*head = newNode;

}

}

// Function to insert a node at the end of the circular list

void insertAtEnd(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (!newNode) return;

if (\*head == NULL) {

\*head = newNode;

} else {

Node\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = \*head;

}

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

65

**Program**

// Function to insert a node after a specific node in the circular list

void insertAfter(Node\* head, int key, int data) {

Node\* temp = head;

do {

if (temp->data == key) {

Node\* newNode = createNode(data);

if (!newNode) return;

newNode->next = temp->next;

temp->next = newNode;

return;

}

temp = temp->next;

} while (temp != head);

printf("Node with data %d not found\n", key);

}

// Function to delete a node from the beginning of the circular list

void deleteFromBeginning(Node\*\* head) {

if (\*head == NULL) {

printf("List is empty\n");

return;

}

Node\* temp = \*head;

if (temp->next == \*head) {

free(temp);

\*head = NULL;

} else {

Node\* last = \*head;

while (last->next != \*head) {

last = last->next;

}

last->next = temp->next;

\*head = temp->next;

free(temp);

}

}

// Function to delete a node from the end of the circular list

void deleteFromEnd(Node\*\* head) {

if (\*head == NULL) {

printf("List is empty\n");

return;

}

Node\* temp = \*head;

if (temp->next == \*head) {

free(temp);

\*head = NULL;

} else {

Node\* prev = NULL;

while (temp->next != \*head) {

prev = temp;

temp = temp->next;

}

prev->next = \*head;

free(temp);

}

66

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

// Function to delete a node after a specific node in the circular list

void deleteAfter(Node\* head, int key) {

Node\* temp = head;

do {

if (temp->data == key) {

Node\* delNode = temp->next;

if (delNode == head) {

printf("Cannot delete the head node after itself\n");

return;

}

temp->next = delNode->next;

free(delNode);

return;

}

temp = temp->next;

} while (temp != head);

printf("Node with data %d not found\n", key);

}

// Function to search for a given element in the circular list

Node\* search(Node\* head, int key) {

Node\* temp = head;

do {

if (temp->data == key) {

return temp;

}

temp = temp->next;

} while (temp != head);

return NULL;

}

// Function to display the elements of the circular list

void displayList(Node\* head) {

if (head == NULL) {

printf("Circular linked list: NULL\n");

return;

}

Node\* temp = head;

printf("Circular linked list: ");

do {

printf("%d -> ", temp->data);

temp = temp->next;

} while (temp != head);

printf("(head)\n");

}

// Main function with menu-driven interaction

int main() {

Node\* head = NULL;

int choice, data, key;

Node\* result;

while (1) {

printf("\n1. Insert at beginning\n");

printf("2. Insert at end\n");

printf("3. Insert after a specific node\n");

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

67

printf("4. Delete from beginning\n");

printf("5. Delete from end\n");

printf("6. Delete after a specific node\n");

printf("7. Search for an element\n");

printf("8. Display the circular list\n");

printf("9. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

insertAtBeginning(&head, data);

break;

case 2:

printf("Enter data to insert at end: ");

scanf("%d", &data);

insertAtEnd(&head, data);

break;

case 3:

printf("Enter the key after which to insert: ");

scanf("%d", &key);

printf("Enter data to insert after node %d: ", key);

scanf("%d", &data);

insertAfter(head, key, data);

break;

case 4:

deleteFromBeginning(&head);

break;

case 5:

deleteFromEnd(&head);

break;

case 6:

printf("Enter the key after which to delete: ");

scanf("%d", &key);

deleteAfter(head, key);

break;

case 7:

printf("Enter data to search: ");

scanf("%d", &data);

result = search(head, data);

if (result != NULL) {

printf("Node with data %d found\n", data);

} else {

printf("Node with data %d not found\n", data);

}

break;

case 8:

displayList(head);

break;

case 9:

exit(0);

default:

printf("Invalid choice, please try again\n");

}

}

return 0;

}

**OUTPUT:**

1. Insert at beginning

2. Insert at end

3. Insert after a specific node

4. Delete from beginning

5. Delete from end

6. Delete after a specific node

7. Search for an element

8. Display the circular list

9. Exit

Enter your choice: 1

Enter data to insert at beginning: 23

1. Insert at beginning

2. Insert at end

3. Insert after a specific node

4. Delete from beginning

5. Delete from end

6. Delete after a specific node

7. Search for an element

8. Display the circular list

9. Exit

Enter your choice: 8

Circular linked list: 23 -> (head)

**Result:**

The program is executed, it provides a menu-driven interface for performing operations on a circular linked list

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

68

|  |  |
| --- | --- |
| **Ex. No. : 7** | **Date :** |
| **P o l y n o m i a l M a n i p u l a t i o n s** | |
| **Write a C p r o g r a m t o i m p l e m e n t t h e f o l l o w i n g o p e r a t i o n s o n S i n g l y L i n k e d L i s t .**   1. **Polynomial Addition** 2. **Polynomial Subtraction** 3. **Polynomial Multiplication Sample Out:**   Polynomial 1: 3x^2 + 4x^1 + 5x^0 Polynomial 2: 2x^3 + 3x^2 + 1x^1  Polynomial Addition: 2x^3 + 6x^2 + 5x^1 + 5x^0 Polynomial Subtraction: -2x^3 + 0x^2 + 3x^1 + 5x^0  Polynomial Multiplication: 6x^5 + 9x^4 + 3x^3 + 8x^4 + 12x^3 + 4x^2 + 10x^3 + 15x^2  + 5x^1  **Aim:**  To implement polynomial addition, subtraction, and multiplication using a singly linked list in C.  **Algorithm**  1. Initialization:  • Define a structure for a polynomial term.  • Define functions to create and insert nodes into the polynomial linked list.  2. Polynomial Operations:  • Addition:  • Traverse both polynomials and add corresponding terms.  • Subtraction:  • Traverse both polynomials and subtract corresponding terms.  • Multiplication:  • Multiply each term of the first polynomial by each term of the second polynomial and sum the results for terms with the same power. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

69

**Program:**

#include <stdio.h>

#include <stdlib.h>

typedef struct PolyNode {

int coeff;

int exp;

struct PolyNode\* next;

} PolyNode;

// Function to create a new polynomial term node

PolyNode\* createNode(int coeff, int exp) {

PolyNode\* newNode = (PolyNode\*)malloc(sizeof(PolyNode));

newNode->coeff = coeff;

newNode->exp = exp;

newNode->next = NULL;

return newNode;

}

// Function to insert a node in the polynomial linked list

void insertNode(PolyNode\*\* poly, int coeff, int exp) {

PolyNode\* newNode = createNode(coeff, exp);

newNode->next = \*poly;

\*poly = newNode;

}

// Function to display a polynomial

void displayPoly(PolyNode\* poly) {

PolyNode\* temp = poly;

while (temp != NULL) {

printf("%dx^%d", temp->coeff, temp->exp);

temp = temp->next;

if (temp != NULL)

printf(" + ");

}

printf("\n");

}

// Function to add two polynomials

PolyNode\* addPolynomials(PolyNode\* poly1, PolyNode\* poly2) {

PolyNode\* result = NULL;

PolyNode\* temp1 = poly1;

PolyNode\* temp2 = poly2;

while (temp1 != NULL && temp2 != NULL) {

if (temp1->exp == temp2->exp) {

insertNode(&result, temp1->coeff + temp2->coeff, temp1->exp);

temp1 = temp1->next;

temp2 = temp2->next;

} else if (temp1->exp > temp2->exp) {

insertNode(&result, temp1->coeff, temp1->exp);

temp1 = temp1->next;

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

70

} else {

insertNode(&result, temp2->coeff, temp2->exp);

temp2 = temp2->next;

}

}

while (temp1 != NULL) {

insertNode(&result, temp1->coeff, temp1->exp);

temp1 = temp1->next;

}

while (temp2 != NULL) {

insertNode(&result, temp2->coeff, temp2->exp);

temp2 = temp2->next;

}

return result;

}

// Function to subtract two polynomials

PolyNode\* subtractPolynomials(PolyNode\* poly1, PolyNode\* poly2) {

PolyNode\* result = NULL;

PolyNode\* temp1 = poly1;

PolyNode\* temp2 = poly2;

while (temp1 != NULL && temp2 != NULL) {

if (temp1->exp == temp2->exp) {

insertNode(&result, temp1->coeff - temp2->coeff, temp1->exp);

temp1 = temp1->next;

temp2 = temp2->next;

} else if (temp1->exp > temp2->exp) {

insertNode(&result, temp1->coeff, temp1->exp);

temp1 = temp1->next;

} else {

insertNode(&result, -temp2->coeff, temp2->exp);

temp2 = temp2->next;

}

}

while (temp1 != NULL) {

insertNode(&result, temp1->coeff, temp1->exp);

temp1 = temp1->next;

}

while (temp2 != NULL) {

insertNode(&result, -temp2->coeff, temp2->exp);

temp2 = temp2->next;

}

return result;

}

// Function to multiply two polynomials

PolyNode\* multiplyPolynomials(PolyNode\* poly1, PolyNode\* poly2) {

PolyNode\* result = NULL;

PolyNode\* temp1 = poly1;

PolyNode\* temp2;

71

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

while (temp1 != NULL) {

temp2 = poly2;

while (temp2 != NULL) {

int coeff = temp1->coeff \* temp2->coeff;

int exp = temp1->exp + temp2->exp;

PolyNode\* resTemp = result;

PolyNode\* prev = NULL;

while (resTemp != NULL && resTemp->exp > exp) {

prev = resTemp;

resTemp = resTemp->next;

}

if (resTemp != NULL && resTemp->exp == exp) {

resTemp->coeff += coeff;

} else {

PolyNode\* newNode = createNode(coeff, exp);

if (prev == NULL) {

newNode->next = result;

result = newNode;

} else {

newNode->next = prev->next;

prev->next = newNode;

}

}

temp2 = temp2->next;

}

temp1 = temp1->next;

}

return result;

}

int main() {

PolyNode\* poly1 = NULL;

PolyNode\* poly2 = NULL;

// Example polynomials

insertNode(&poly1, 5, 0);

insertNode(&poly1, 4, 1);

insertNode(&poly1, 3, 2);

insertNode(&poly2, 1, 1);

insertNode(&poly2, 3, 2);

insertNode(&poly2, 2, 3);

printf("Polynomial 1: ");

displayPoly(poly1);

printf("Polynomial 2: ");

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

72

displayPoly(poly2);

PolyNode\* sum = addPolynomials(poly1, poly2);

printf("Polynomial Addition: ");

displayPoly(sum);

PolyNode\* difference = subtractPolynomials(poly1, poly2);

printf("Polynomial Subtraction: ");

displayPoly(difference);

PolyNode\* product = multiplyPolynomials(poly1, poly2);

printf("Polynomial Multiplication: ");

displayPoly(product);

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

73

**OUTPUT:**

Polynomial 1: 3x^2 + 4x^1 + 5x^0

Polynomial 2: 2x^3 + 3x^2 + 1x^1

Polynomial Addition: 2x^3 + 6x^2 + 5x^1 + 5x^0

Polynomial Subtraction: -2x^3 + 0x^2 + 3x^1 + 5x^0

Polynomial Multiplication: 6x^5 + 9x^4 + 3x^3 +

### Result:

74

**The program is executed, it displays the given polynomials and performs addition, subtraction, and multiplication**

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

|  |  |
| --- | --- |
| **Ex. No. : 8.a** | **Date :** |
| **Tree Traversal** | |
| **Write a C program to implement a Binary tree and perform the following tree traversal operation.**   1. **Inorder Traversal** 2. **Preorder Traversal** 3. **Postorder Traversal**   **Aim:**  To implement a binary tree in C and perform Inorder, Preorder, and Postorder tree traversal operations.  **Algorithm:**  1. Define a structure for the tree node with integer data and left and right child pointers.  2. Implement functions to create a new node and insert nodes into the tree.  Tree Traversal Algorithms  Inorder Traversal  1. Traverse the left subtree.  2. Visit the root node.  3. Traverse the right subtree.  Preorder Traversal  1. Visit the root node.  2. Traverse the left subtree.  3. Traverse the right subtree.  Postorder Traversal  1. Traverse the left subtree.  2. Traverse the right subtree.  3. Visit the root node. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

75

**Program:**

#include <stdio.h>

#include <stdlib.h>

// Define the structure of a tree node

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Function for Inorder Traversal

void inorderTraversal(struct Node\* node) {

if (node == NULL) return;

inorderTraversal(node->left);

printf("%d ", node->data);

inorderTraversal(node->right);

}

// Function for Preorder Traversal

void preorderTraversal(struct Node\* node) {

if (node == NULL) return;

printf("%d ", node->data);

preorderTraversal(node->left);

preorderTraversal(node->right);

}

// Function for Postorder Traversal

void postorderTraversal(struct Node\* node) {

if (node == NULL) return;

postorderTraversal(node->left);

postorderTraversal(node->right);

printf("%d ", node->data);

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

76

**Program:**

// Main function to demonstrate the tree traversals

int main() {

// Creating a sample binary tree

struct Node\* root = createNode(1);

root->left = createNode(2);

root->right = createNode(3);

root->left->left = createNode(4);

root->left->right = createNode(5);

root->right->left = createNode(6);

root->right->right = createNode(7);

// Performing Inorder Traversal

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

// Performing Preorder Traversal

printf("Preorder Traversal: ");

preorderTraversal(root);

printf("\n");

// Performing Postorder Traversal

printf("Postorder Traversal: ");

postorderTraversal(root);

printf("\n");

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

77

**OUTPUT:**

Inorder Traversal: 4 2 5 1 6 3 7

Preorder Traversal: 1 2 4 5 3 6 7

Postorder Traversal: 4 5 2 6 7 3 1

### Result:

### The C program successfully implements a binary tree and performs the following tree traversal operations:

### Inorder Traversal: Visits the nodes in the order: Left, Root, Right.

### Preorder Traversal: Visits the nodes in the order: Root, Left, Right.

### Postorder Traversal: Visits the nodes in the order: Left, Right, Root.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

78

**Ex. No. : 8.b Date :**

**Implementation of Binary Search tree**

Write a C program to implement a Binary Search Tree and perform the following operations.

1. Insert
2. Delete
3. Search
4. Display

**Aim:**

To implement a Binary Search Tree (BST) in C and perform the following operations:

Insert a node

Delete a node

Search for a node

Display the tree using Inorder Traversal

**Algorithm:**

Binary Search Tree Implementation

1. Define a structure for the tree node with integer data and left and right child pointers.

2. Implement functions to insert, delete, search, and display nodes in the tree.

Insert Operation

1. Start at the root.

2. If the tree is empty, create a new node as the root.

3. Recursively find the correct position in the left or right subtree.

4. Insert the node at the found position.

Delete Operation

1. Search for the node to be deleted.

2. If the node has no children, simply delete it.

3. If the node has one child, replace the node with its child.

4. If the node has two children, find the inorder successor (smallest node in the right subtree), replace the node with the successor, and delete the successor.

Search Operation

1. Start at the root.

2. If the data matches the root, return the node.

3. Recursively search in the left or right subtree based on the comparison of data.

Display Operation (Inorder Traversal)

1. Traverse the left subtree.

2. Visit the root node.

3. Traverse the right subtree.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

79

**Program:**

#include <stdio.h>

#include <stdlib.h>

// Define the structure of a tree node

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Function to insert a node in the BST

struct Node\* insertNode(struct Node\* root, int data) {

if (root == NULL) return createNode(data);

if (data < root->data)

root->left = insertNode(root->left, data);

else if (data > root->data)

root->right = insertNode(root->right, data);

return root;

}

// Function to find the minimum value node in the BST

struct Node\* findMin(struct Node\* root) {

while (root->left != NULL) root = root->left;

return root;

}

// Function to delete a node from the BST

struct Node\* deleteNode(struct Node\* root, int data) {

if (root == NULL) return root;

if (data < root->data)

root->left = deleteNode(root->left, data);

else if (data > root->data)

root->right = deleteNode(root->right, data);

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

80

else {

if (root->left == NULL) {

struct Node\* temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

struct Node\* temp = root->left;

free(root);

return temp;

}

struct Node\* temp = findMin(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

// Function to search for a node in the BST

struct Node\* searchNode(struct Node\* root, int data) {

if (root == NULL || root->data == data)

return root;

if (data < root->data)

return searchNode(root->left, data);

else

return searchNode(root->right, data);

}

// Function for Inorder Traversal

void inorderTraversal(struct Node\* root) {

if (root == NULL) return;

inorderTraversal(root->left);

printf("%d ", root->data);

inorderTraversal(root->right);

}

// Main function to demonstrate the BST operations

int main() {

struct Node\* root = NULL;

int choice, data;

while (1) {

printf("\nBinary Search Tree Operations:\n");

printf("1. Insert\n2. Delete\n3. Search\n4. Display (Inorder Traversal)\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

81

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &data);

root = insertNode(root, data);

break;

case 2:

printf("Enter the value to delete: ");

scanf("%d", &data);

root = deleteNode(root, data);

break;

case 3:

printf("Enter the value to search: ");

scanf("%d", &data);

struct Node\* result = searchNode(root, data);

if (result != NULL)

printf("Value %d found in the tree.\n", data);

else

printf("Value %d not found in the tree.\n", data);

break;

case 4:

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 5:

exit(0);

default:

printf("Invalid choice! Please try again.\n");

}

}

return 0;

}

**OUTPUT:**

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 1

Enter the value to insert: 50

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 1

Enter the value to insert: 30

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 1

Enter the value to insert: 70

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 1

Enter the value to insert: 20

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 1

Enter the value to insert: 40

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 1

Enter the value to insert: 60

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 1

Enter the value to insert: 80

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 4

Inorder Traversal: 20 30 40 50 60 70 80

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

82

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 3

Enter the value to search: 60

Value 60 found in the tree.

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 3

Enter the value to search: 100

Value 100 not found in the tree.

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 2

Enter the value to delete: 70

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 4

Inorder Traversal: 20 30 40 50 60 80

Binary Search Tree Operations:

1. Insert

2. Delete

3. Search

4. Display (Inorder Traversal)

5. Exit

Enter your choice: 5

# Result:

# The C program successfully implements a Binary Search Tree and performs the following operations:Insert: Adds a new node to the BST.Delete: Removes a node from the BST.Search: Searches for a node in the BST.Display: Displays the nodes of the BST using Inorder Traversal.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

83

|  |  |
| --- | --- |
| **Ex. No. : 8.c** | **Date :** |
|  | **Implementation of AVL Tree** |
| Write a function in C program to insert a new node with a given value into an AVL tree. Ensure that the tree remains balanced after insertion by performing rotations if necessary. Repeat the above operation to delete a node from AVL tree.  **Aim:**  To implement functions in C for inserting and deleting nodes in an AVL tree while ensuring that the tree remains balanced after each operation by performing necessary rotations.  **Algorithm:**  Insertion in an AVL Tree  1. Insert the node as in a normal binary search tree (BST).  2. Update the height of the ancestor nodes.  3. Check the balance factor of each node:  • If the balance factor is greater than 1 or less than -1, the tree is unbalanced and requires rotation.  4. Perform rotations to balance the tree:  • Left Left Case  • Right Right Case  • Left Right Case  • Right Left Case  Deletion from an AVL Tree  1. Delete the node as in a normal BST.  2. Update the height of the ancestor nodes.  3. Check the balance factor of each node:  • If the balance factor is greater than 1 or less than -1, the tree is unbalanced and requires rotation.  4. Perform rotations to balance the tree:  • Left Left Case  • Right Right Case  • Left Right Case  • Right Left Case | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

84

**Program:**

#include <stdio.h>

#include <stdlib.h>

// Node structure

typedef struct Node {

int key;

struct Node \*left;

struct Node \*right;

int height;

} Node;

// Function to get the height of the tree

int height(Node \*N) {

if (N == NULL)

return 0;

return N->height;

}

// Function to get maximum of two integers

int max(int a, int b) {

return (a > b) ? a : b;

}

// Function to create a new node

Node\* newNode(int key) {

Node\* node = (Node\*)malloc(sizeof(Node));

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1; // new node is initially added at leaf

return(node);

}

// Right rotate subtree rooted with y

Node \*rightRotate(Node \*y) {

Node \*x = y->left;

Node \*T2 = x->right;

// Perform rotation

x->right = y;

y->left = T2;

// Update heights

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

// Return new root

return x;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

85

// Left rotate subtree rooted with x

Node \*leftRotate(Node \*x) {

Node \*y = x->right;

Node \*T2 = y->left;

// Perform rotation

y->left = x;

x->right = T2;

// Update heights

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

// Return new root

return y;

}

// Get balance factor of node N

int getBalance(Node \*N) {

if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

// Insert a node

Node\* insert(Node\* node, int key) {

if (node == NULL)

return(newNode(key));

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

else // Equal keys are not allowed in BST

return node;

// Update height of this ancestor node

node->height = 1 + max(height(node->left), height(node->right));

// Get the balance factor of this ancestor node

int balance = getBalance(node);

// If this node becomes unbalanced, then

// there are 4 cases

// Left Left Case

if (balance > 1 && key < node->left->key)

return rightRotate(node);

86

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

// Right Right Case

if (balance < -1 && key > node->right->key)

return leftRotate(node);

// Left Right Case

if (balance > 1 && key > node->left->key) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

// Right Left Case

if (balance < -1 && key < node->right->key) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

// return the (unchanged) node pointer

return node;

}

// Function to find the node with the minimum key value

Node \*minValueNode(Node\* node) {

Node\* current = node;

// loop down to find the leftmost leaf

while (current->left != NULL)

current = current->left;

return current;

}

// Delete a node

Node\* deleteNode(Node\* root, int key) {

// STEP 1: PERFORM STANDARD BST DELETE

if (root == NULL)

return root;

// If the key to be deleted is smaller than the root's key,

// then it lies in left subtree

if (key < root->key)

root->left = deleteNode(root->left, key);

// If the key to be deleted is greater than the root's key,

// then it lies in right subtree

else if (key > root->key)

root->right = deleteNode(root->right, key);

// if key is same as root's key, then This is the node

// to be deleted

else {

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

87

// node with only one child or no child

if ((root->left == NULL) || (root->right == NULL)) {

Node \*temp = root->left ? root->left : root->right;

// No child case

if (temp == NULL) {

temp = root;

root = NULL;

} else // One child case

\*root = \*temp; // Copy the contents of

// the non-empty child

free(temp);

} else {

// node with two children: Get the inorder

// successor (smallest in the right subtree)

Node\* temp = minValueNode(root->right);

// Copy the inorder successor's data to this node

root->key = temp->key;

// Delete the inorder successor

root->right = deleteNode(root->right, temp->key);

}

}

// If the tree had only one node then return

if (root == NULL)

return root;

// STEP 2: UPDATE HEIGHT OF THE CURRENT NODE

root->height = 1 + max(height(root->left), height(root->right));

// STEP 3: GET THE BALANCE FACTOR OF THIS NODE (to check whether

// this node became unbalanced)

int balance = getBalance(root);

// If this node becomes unbalanced, then there are 4 cases

// Left Left Case

if (balance > 1 && getBalance(root->left) >= 0)

return rightRotate(root);

// Left Right Case

if (balance > 1 && getBalance(root->left) < 0) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

// Right Right Case

if (balance < -1 && getBalance(root->right) <= 0)

return leftRotate(root);

// Right Left Case

if (balance < -1 && getBalance(root->right) > 0) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

// Utility function to print the tree in order

void inOrder(Node \*root) {

if (root != NULL) {

inOrder(root->left);

printf("%d ", root->key);

inOrder(root->right);

}

}

int main() {

Node \*root = NULL;

// Insert nodes

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 40);

root = insert(root, 50);

root = insert(root, 25);

// Print the tree

printf("Inorder traversal of the constructed AVL tree is \n");

inOrder(root);

// Delete node

root = deleteNode(root, 40);

// Print the tree

printf("\nInorder traversal after deletion of 40 \n");

inOrder(root);

return 0;

}

**OUTPUT:**

Inorder traversal of the constructed AVL tree is

10 20 25 30 40 50

Inorder traversal after deletion of 40

10 20 25 30 50

Result: The program successfully inserts and deletes nodes in the AVL tree, ensuring the tree remains balanced after each operation. The in-order traversal of the tree shows the sorted order of elements, indicating the correctness of the AVL tree implementation.

88

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Ex. No. :9.a Date :**

### Graph Representation

**Create a data structure to represent a graph. You can choose either adjacency matrix or adjacency list representation.**

**Aim:**

To implement a graph data structure using an adjacency list representation in C.

**Algorithm:**

Graph Representation Using Adjacency List

1. Define the structure for the adjacency list node: Each node represents an edge in the graph.

2. Define the structure for the adjacency list: An array of adjacency list nodes, where each index represents a vertex in the graph.

3. Define the structure for the graph: Includes the number of vertices and an array of adjacency lists.

4. Implement functions for:

• Adding an edge to the graph.

• Printing the graph to show its adjacency list representation**.**

**Program:**

#include <stdio.h>

#include <stdlib.h>

// Structure to represent an adjacency list node

typedef struct AdjListNode {

int dest;

struct AdjListNode\* next;

} AdjListNode;

// Structure to represent an adjacency list

typedef struct AdjList {

struct AdjListNode \*head;

} AdjList**;**

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

89

// Structure to represent a graph

typedef struct Graph {

int V;

struct AdjList\* array;

} Graph;

// Function to create a new adjacency list node

AdjListNode\* newAdjListNode(int dest) {

AdjListNode\* newNode = (AdjListNode\*)malloc(sizeof(AdjListNode));

newNode->dest = dest;

newNode->next = NULL;

return newNode;

}

// Function to create a graph with V vertices

Graph\* createGraph(int V) {

Graph\* graph = (Graph\*)malloc(sizeof(Graph));

graph->V = V;

// Create an array of adjacency lists. Size of array will be V

graph->array = (AdjList\*)malloc(V \* sizeof(AdjList));

// Initialize each adjacency list as empty by making head as NULL

for (int i = 0; i < V; ++i)

graph->array[i].head = NULL;

return graph;

}

// Function to add an edge to an undirected graph

void addEdge(Graph\* graph, int src, int dest) {

// Add an edge from src to dest. A new node is added to the adjacency

// list of src. The node is added at the beginning

AdjListNode\* newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

// Since graph is undirected, add an edge from dest to src also

newNode = newAdjListNode(src);

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

90

91

**Program:**

// Function to print the adjacency list representation of graph

void printGraph(Graph\* graph) {

for (int v = 0; v < graph->V; ++v) {

AdjListNode\* pCrawl = graph->array[v].head;

printf("\n Adjacency list of vertex %d\n head ", v);

while (pCrawl) {

printf("-> %d", pCrawl->dest);

pCrawl = pCrawl->next;

}

printf("\n");

}

}

int main() {

// Create the graph given in the above figure

int V = 5;

Graph\* graph = createGraph(V);

addEdge(graph, 0, 1);

addEdge(graph, 0, 4);

addEdge(graph, 1, 2);

addEdge(graph, 1, 3);

addEdge(graph, 1, 4);

addEdge(graph, 2, 3);

addEdge(graph, 3, 4);

// Print the adjacency list representation of the above graph

printGraph(graph);

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**OUTPUT:**

Adjacency list of vertex 0

head -> 4-> 1

Adjacency list of vertex 1

head -> 4-> 3-> 2-> 0

Adjacency list of vertex 2

head -> 3-> 1

Adjacency list of vertex 3

head -> 4-> 2-> 1

Adjacency list of vertex 4

head -> 3-> 1-> 0

**Result:**

The program successfully creates and prints the adjacency list representation of an undirected graph.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

92

|  |  |
| --- | --- |
| **Ex. No. : 9.b** | **Date :** |
|  | **BFS Implementation** |
| **Write a function to perform Breadth-First Search on the graph. Start the search from a specified source vertex. Print the visited vertices in BFS order.**  **Aim:**  To implement a function in C to perform Breadth-First Search (BFS) on a graph represented using an adjacency list. The BFS function will start from a specified source vertex and print the visited vertices in BFS order.  **Algorithm:**  Breadth-First Search (BFS)  1. Initialize:  • Create a boolean array visited to keep track of visited vertices.  • Create a queue to manage the vertices to be explored.  2. Start BFS from the source vertex:  • Mark the source vertex as visited and enqueue it.  3. Process the queue:  • While the queue is not empty:  • Dequeue a vertex from the queue and print it.  • Get all adjacent vertices of the dequeued vertex. If an adjacent vertex has not been visited, mark it as visited and enqueue it.  **Program:**  #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  // Structure to represent an adjacency list node  typedef struct AdjListNode {  int dest;  struct AdjListNode\* next;  } AdjListNode;  // Structure to represent an adjacency list  typedef struct AdjList {  struct AdjListNode \*head;  } AdjList;  // Structure to represent a graph  typedef struct Graph {  int V;  struct AdjList\* array;  } Graph;  // Function to create a new adjacency list node  AdjListNode\* newAdjListNode(int dest) {  AdjListNode\* newNode = (AdjListNode\*)malloc(sizeof(AdjListNode));  newNode->dest = dest;  newNode->next = NULL;  return newNode;  } | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

93

// Function to create a graph with V vertices

Graph\* createGraph(int V) {

Graph\* graph = (Graph\*)malloc(sizeof(Graph));

graph->V = V;

graph->array = (AdjList\*)malloc(V \* sizeof(AdjList));

for (int i = 0; i < V; ++i)

graph->array[i].head = NULL;

return graph;

}

// Function to add an edge to an undirected graph

void addEdge(Graph\* graph, int src, int dest) {

AdjListNode\* newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

newNode = newAdjListNode(src);

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

// Function to perform BFS starting from a given source vertex

void BFS(Graph\* graph, int startVertex) {

// Create a visited array and mark all vertices as not visited

bool \*visited = (bool\*)malloc(graph->V \* sizeof(bool));

for (int i = 0; i < graph->V; i++)

visited[i] = false;

// Create a queue for BFS

int \*queue = (int\*)malloc(graph->V \* sizeof(int));

int front = 0;

int rear = 0;

// Mark the current node as visited and enqueue it

visited[startVertex] = true;

queue[rear++] = startVertex;

while (front < rear) {

// Dequeue a vertex from queue and print it

int currentVertex = queue[front++];

printf("%d ", currentVertex);

// Get all adjacent vertices of the dequeued vertex

AdjListNode\* adjList = graph->array[currentVertex].head;

while (adjList != NULL) {

int adjVertex = adjList->dest;

if (!visited[adjVertex]) {

visited[adjVertex] = true;

queue[rear++] = adjVertex;

}

adjList = adjList->next;

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

94

}

}

free(visited);

free(queue);

}

// Function to print the adjacency list representation of the graph

void printGraph(Graph\* graph) {

for (int v = 0; v < graph->V; ++v) {

AdjListNode\* pCrawl = graph->array[v].head;

printf("\n Adjacency list of vertex %d\n head ", v);

while (pCrawl) {

printf("-> %d", pCrawl->dest);

pCrawl = pCrawl->next;

}

printf("\n");

}

}

int main() {

int V = 5;

Graph\* graph = createGraph(V);

addEdge(graph, 0, 1);

addEdge(graph, 0, 4);

addEdge(graph, 1, 2);

addEdge(graph, 1, 3);

addEdge(graph, 1, 4);

addEdge(graph, 2, 3);

addEdge(graph, 3, 4);

printf("Graph adjacency list:\n");

printGraph(graph);

printf("\nBreadth First Traversal starting from vertex 0:\n");

BFS(graph, 0);

return 0;

}

**Result:**

The program successfully creates a graph using an adjacency list representation and performs Breadth-First Search (BFS) starting from a specified vertex, printing the visited vertices in BFS order.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

95

|  |  |
| --- | --- |
| **Ex. No. :9.c** | **Date :** |
| **DFS Implementation** | |
| Write a function to perform Depth-First Search on the graph. Start the search from a specified source vertex. Print the visited vertices in DFS order.  **Aim:**  To implement a function in C to perform Depth-First Search (DFS) on a graph represented using an adjacency list. The DFS function will start from a specified source vertex and print the visited vertices in DFS order.  **Algorithm:**  Depth-First Search (DFS)  1. Initialize:  • Create a boolean array visited to keep track of visited vertices.  2. Start DFS from the source vertex:  • Mark the source vertex as visited and print it.  • Recursively visit all adjacent vertices of the current vertex that have not been visited**.**  **Program:**  #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  // Structure to represent an adjacency list node  typedef struct AdjListNode {  int dest;  struct AdjListNode\* next;  } AdjListNode;  // Structure to represent an adjacency list  typedef struct AdjList {  struct AdjListNode \*head;  } AdjList; | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

96

// Structure to represent a graph

typedef struct Graph {

int V;

struct AdjList\* array;

} Graph;

// Function to create a new adjacency list node

AdjListNode\* newAdjListNode(int dest) {

AdjListNode\* newNode = (AdjListNode\*)malloc(sizeof(AdjListNode));

newNode->dest = dest;

newNode->next = NULL;

return newNode;

}

// Function to create a graph with V vertices

Graph\* createGraph(int V) {

Graph\* graph = (Graph\*)malloc(sizeof(Graph));

graph->V = V;

graph->array = (AdjList\*)malloc(V \* sizeof(AdjList));

for (int i = 0; i < V; ++i)

graph->array[i].head = NULL;

return graph;

}

// Function to add an edge to an undirected graph

void addEdge(Graph\* graph, int src, int dest) {

AdjListNode\* newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

newNode = newAdjListNode(src);

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

// Function to perform DFS from a given vertex

void DFSUtil(Graph\* graph, int v, bool visited[]) {

// Mark the current node as visited and print it

visited[v] = true;

printf("%d ", v);

// Recur for all the vertices adjacent to this vertex

AdjListNode\* adjList = graph->array[v].head;

while (adjList != NULL) {

int adjVertex = adjList->dest;

if (!visited[adjVertex])

DFSUtil(graph, adjVertex, visited);

adjList = adjList->next;

}

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

97

// Function to perform DFS starting from a given vertex

void DFS(Graph\* graph, int startVertex) {

// Initialize visited array

bool \*visited = (bool\*)malloc(graph->V \* sizeof(bool));

for (int i = 0; i < graph->V; i++)

visited[i] = false;

// Call the recursive helper function to perform DFS

DFSUtil(graph, startVertex, visited);

free(visited);

}

// Function to print the adjacency list representation of the graph

void printGraph(Graph\* graph) {

for (int v = 0; v < graph->V; ++v) {

AdjListNode\* pCrawl = graph->array[v].head;

**printf("\n Adjacency list of vertex %d\n head ", v);**

**while (pCrawl) {**

**printf("-> %d", pCrawl->dest);**

**pCrawl = pCrawl->next;**

**}**

**printf("\n");**

**}**

**}**

**/**

**int main() {**

**int V = 5;**

**Graph\* graph = createGraph(V);**

**addEdge(graph, 0, 1);**

**addEdge(graph, 0, 4);**

**addEdge(graph, 1, 2);**

**addEdge(graph, 1, 3);**

**addEdge(graph, 1, 4);**

**addEdge(graph, 2, 3);**

**addEdge(graph, 3, 4);**

**printf("Graph adjacency list:\n");**

**printGraph(graph);**

**printf("\nDepth First Traversal starting from vertex 0:\n");**

**DFS(graph, 0);**

**return 0;**

**}**

98

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**OUTPUT**

**Graph adjacency list:**

**Adjacency list of vertex 0**

**head -> 4-> 1**

**Adjacency list of vertex 1**

**head -> 4-> 3-> 2-> 0**

**Adjacency list of vertex 2**

**head -> 3-> 1**

**Adjacency list of vertex 3**

**head -> 4-> 2-> 1**

**Adjacency list of vertex 4**

**head -> 3-> 1-> 0**

**Depth First Traversal starting from vertex 0:**

**0 1 2 3 4**

**Result:**

**The program successfully creates a graph using an adjacency list representation and performs Depth-First Search (DFS) starting from a specified vertex, printing the visited vertices in DFS order.**

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

99

|  |  |
| --- | --- |
| **Ex. No. :10.a** | **Date :** |
| **Implement a function for linear search** | |
| **Implement a function for linear search that takes an array of integers and a key to search for. Print the index of the key if found, or a message indicating that the key is not present in the array.**  **Aim:**  To implement a function in C for performing a linear search on an array of integers. The function will take an array and a key to search for, and it will print the index of the key if found, or a message indicating that the key is not present in the array.  **Algorithm:**  Linear Search  1. Input:  • An array of integers.  • The key to search for.  2. Process:  • Iterate through each element in the array.  • Compare each element with the key.  • If a match is found, print the index and terminate the search.  • If no match is found by the end of the array, print a message indicating that the key is not present.  3. Output:  • The index of the key if found.  • A message indicating the key is not present if no match is found. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

100

**Program:**

**#include <stdio.h>**

// Function to perform linear search

void linearSearch(int arr[], int size, int key) {

for (int i = 0; i < size; i++) {

if (arr[i] == key) {

printf("Key %d found at index %d\n", key, i);

return;

}

}

printf("Key %d not found in the array\n", key);

}

int main() {

int arr[] = {2, 4, 0, 1, 9, 5, 6};

int size = sizeof(arr) / sizeof(arr[0]);

int key = 5;

printf("Array: ");

for (int i = 0; i < size; i++) {

printf("%d ", arr[i]);

}

printf("\n");

linearSearch(arr, size, key);

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

101

**OUTPUT:**

Array: 2 4 0 1 9 5 6

Key 5 found at index 5

Array: 2 4 0 1 9 5 6

Key 7 not found in the array

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

102

**RESULT:**

The program successfully implements a linear search function and demonstrates its functionality by searching for a key in a given array of integers.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

103

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

104

|  |  |
| --- | --- |
| **Ex. No. :10.b** | **Date :** |
| **Implement a function for binary search** | |
| Implement a function for binary search that takes a sorted array of integers and a key to search for. Print the index of the key if found, or a message indicating that the key is not present in the array.  **Aim:**  To implement a function in C for performing a binary search on a sorted array of integers. The function will take a sorted array and a key to search for, and it will print the index of the key if found, or a message indicating that the key is not present in the array.  **Algorithm:**    Binary Search  1. Input:  • A sorted array of integers.  • The key to search for.  2. Process:  • Initialize two pointers: left to the start of the array and right to the end of the array.  • While left is less than or equal to right:  • Calculate the middle index mid.  • If the element at mid is equal to the key, print the index mid and terminate the search.  • If the element at mid is greater than the key, move the right pointer to mid - 1.  • If the element at mid is less than the key, move the left pointer to mid + 1.  • If the key is not found, print a message indicating that the key is not present.  3. Output:  • The index of the key if found.  • A message indicating the key is not present if no match is found. | |

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

105

**Program:**

#include <stdio.h>

// Function to perform binary search

void binarySearch(int arr[], int size, int key) {

int left = 0;

int right = size - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

// Check if key is present at mid

if (arr[mid] == key) {

printf("Key %d found at index %d\n", key, mid);

return;

}

// If key greater, ignore left half

if (arr[mid] < key) {

left = mid + 1;

}

// If key is smaller, ignore right half

else {

right = mid - 1;

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

106

**Program:**

**}**

**}**

// If we reach here, the element was not present

printf("Key %d not found in the array\n", key);

}

int main() {

int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int size = sizeof(arr) / sizeof(arr[0]);

int key = 5;

printf("Sorted array: ");

for (int i = 0; i < size; i++) {

printf("%d ", arr[i]);

}

printf("\n");

binarySearch(arr, size, key);

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

107

**OUTPUT:**

108

**Sorted array: 1 2 3 4 5 6 7 8 9**

**Key 5 found at index 4**

**Sorted array: 1 2 3 4 5 6 7 8 9**

**Key 10 not found in the array**

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Result:**

The program successfully implements a binary search function and demonstrates its functionality by searching for a key in a given sorted array of integers.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

109

**Ex. No. : 11 Date :**

### Priority Interview

**Write a C p r o g r a m t o t a k e n n u m b e r s a n d s o r t t h e n u m b e r s i n a s c e n d i n g o r d e r . T r y t o i m p l e m e n t t h e s a m e u s i n g f o l l o w i n g s o r t i n g t e c h n i q u e s .**

**1. Quick Sort**

**2. Merge Sort**

**3. Bubble Sort**

**Aim:**

To implement a C program that takes n numbers from the user and sorts them in ascending order using three different sorting techniques: Quick Sort, Merge Sort, and Bubble Sort.

**Algorithm:**

Quick Sort

1. Partition:

• Choose a pivot element from the array.

• Rearrange the elements such that elements less than the pivot are on the left, and elements greater than the pivot are on the right.

2. Recursive Sort:

• Recursively apply the partition process to the sub-arrays formed by partitioning.

Merge Sort

1. Divide:

• Divide the array into two halves.

2. Conquer:

• Recursively sort the two halves.

3. Combine:

• Merge the two sorted halves to produce the sorted array.

Bubble Sort

1. Iterate:

• Repeatedly step through the array.

2. Compare and Swap:

• Compare adjacent elements and swap them if they are in the wrong order.

3. Repeat:

• Continue the process until no more swaps are needed.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

110

**Program:**

#include <stdio.h>

#include <stdlib.h>

// Function prototypes

void quickSort(int arr[], int low, int high);

int partition(int arr[], int low, int high);

void mergeSort(int arr[], int left, int right);

void merge(int arr[], int left, int mid, int right);

void bubbleSort(int arr[], int n);

void printArray(int arr[], int size);

// Main function

int main() {

int n, choice;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements: ");

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

printf("Choose sorting technique:\n");

printf("1. Quick Sort\n2. Merge Sort\n3. Bubble Sort\n");

scanf("%d", &choice);

switch(choice) {

case 1:

quickSort(arr, 0, n - 1);

printf("Sorted array using Quick Sort: ");

break;

case 2:

mergeSort(arr, 0, n - 1);

printf("Sorted array using Merge Sort: ");

break;

case 3:

bubbleSort(arr, n);

printf("Sorted array using Bubble Sort: ");

break;

default:

printf("Invalid choice!\n");

return 1;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

111

printArray(arr, n);

return 0;

}

// Quick Sort

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return (i + 1);

}

// Merge Sort

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[left + i];

for (int j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

112

113

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// Bubble Sort

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (arr[j] > arr[j+1]) {

int temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

// Function to print the array

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**OUTPUT:**

**Enter the number of elements: 5**

**Enter the elements: 5 2 9 1 5**

**Choose sorting technique:**

**1. Quick Sort**

**2. Merge Sort**

**3. Bubble Sort**

**1**

**Sorted array using Quick Sort: 1 2 5 5 9**

**Enter the number of elements: 5**

**Enter the elements: 5 2 9 1 5**

**Choose sorting technique:**

**1. Quick Sort**

**2. Merge Sort**

**3. Bubble Sort**

**2**

**Sorted array using Merge Sort: 1 2 5 5 9**

**Result:**

The program successfully sorts the array using the chosen sorting technique and prints the sorted array.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

114

**Ex. No. :12 Date :**

**I m p l e m e n t a t i o n of H a s h i n g T e c h n i q u e s**

**Write a C p r o g r a m t o c r e a t e a h a s h t a b l e a n d p e r f o r m c o l l i s i o n r e s o l u t i o n u s i n g t h e f o l l o w i n g t e c h n i q u e s .**

* 1. **O p e n a d d r e s s i n g**
  2. **C l o s e d A d d r e s s i n g**
  3. **R e h a s h i n g**

**Aim:**

To implement a function in C for performing a binary search on a sorted array of integers. The function will take a sorted array and a key to search for, and it will print the index of the key if found, or a message indicating that the key is not present in the array.

**Algorithm:**

To create a C program that implements a hash table and performs collision resolution using the following techniques:

1. Open Addressing

2. Closed Addressing (Chaining)

3. Rehashing

Algorithm

Open Addressing

1. Hash Function:

• Use a hash function to determine the initial index.

2. Probe Sequence:

• Use linear probing, quadratic probing, or double hashing to resolve collisions.

Closed Addressing (Chaining)

1. Hash Function:

• Use a hash function to determine the initial index.

2. Linked List:

• Use a linked list at each index to handle collisions.

Rehashing

1. Initial Hash Function:

• Use a hash function to determine the initial index.

2. Rehash:

• When the load factor exceeds a threshold, create a larger table and rehash all existing elements.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

115

**Program:**

#include <stdio.h>

#include <stdlib.h>

#define TABLE\_SIZE 10

#define LOAD\_FACTOR\_THRESHOLD 0.7

// Open Addressing - Linear Probing

void insertLinearProbing(int table[], int size, int key) {

int hash = key % size;

int originalHash = hash;

while (table[hash] != -1) {

hash = (hash + 1) % size;

if (hash == originalHash) {

printf("Table is full!\n");

return;

}

}

table[hash] = key;

}

int searchLinearProbing(int table[], int size, int key) {

int hash = key % size;

int originalHash = hash;

while (table[hash] != -1) {

if (table[hash] == key)

return hash;

hash = (hash + 1) % size;

if (hash == originalHash)

break;

}

return -1;

}

// Closed Addressing (Chaining)

typedef struct Node {

int key;

struct Node\* next;

} Node;

void insertChaining(Node\* table[], int size, int key) {

int hash = key % size;

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->key = key;

newNode->next = table[hash];

table[hash] = newNode;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

116

int searchChaining(Node\* table[], int size, int key) {

int hash = key % size;

Node\* temp = table[hash];

while (temp) {

if (temp->key == key)

return hash;

temp = temp->next;

}

return -1;

}

// Rehashing

typedef struct {

int\* table;

int size;

int count;

} HashTable;

HashTable\* createHashTable(int size) {

HashTable\* ht = (HashTable\*)malloc(sizeof(HashTable));

ht->table = (int\*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++) {

ht->table[i] = -1;

}

ht->size = size;

ht->count = 0;

return ht;

}

void rehash(HashTable\* ht) {

int oldSize = ht->size;

int\* oldTable = ht->table;

ht->size \*= 2;

ht->table = (int\*)malloc(ht->size \* sizeof(int));

for (int i = 0; i < ht->size; i++) {

ht->table[i] = -1;

}

ht->count = 0;

for (int i = 0; i < oldSize; i++) {

if (oldTable[i] != -1) {

insertLinearProbing(ht->table, ht->size, oldTable[i]);

ht->count++;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

117

}

free(oldTable);

}

void insertRehashing(HashTable\* ht, int key) {

if ((float)ht->count / ht->size > LOAD\_FACTOR\_THRESHOLD) {

rehash(ht);

}

insertLinearProbing(ht->table, ht->size, key);

ht->count++;

}

int searchRehashing(HashTable\* ht, int key) {

return searchLinearProbing(ht->table, ht->size, key);

}

// Print functions

void printTable(int table[], int size) {

for (int i = 0; i < size; i++) {

printf("%d ", table[i]);

}

printf("\n");

}

void printChainingTable(Node\* table[], int size) {

for (int i = 0; i < size; i++) {

printf("[%d] -> ", i);

Node\* temp = table[i];

while (temp) {

printf("%d -> ", temp->key);

temp = temp->next;

}

printf("NULL\n");

}

}

// Main function

int main() {

int choice, key, result;

int table[TABLE\_SIZE];

for (int i = 0; i < TABLE\_SIZE; i++) {

table[i] = -1;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

118

Node\* chainingTable[TABLE\_SIZE] = { NULL };

HashTable\* rehashingTable = createHashTable(TABLE\_SIZE);

printf("Choose collision resolution technique:\n");

printf("1. Open Addressing (Linear Probing)\n");

printf("2. Closed Addressing (Chaining)\n");

printf("3. Rehashing\n");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Open Addressing (Linear Probing):\n");

insertLinearProbing(table, TABLE\_SIZE, 10);

insertLinearProbing(table, TABLE\_SIZE, 20);

insertLinearProbing(table, TABLE\_SIZE, 30);

printTable(table, TABLE\_SIZE);

result = searchLinearProbing(table, TABLE\_SIZE, 20);

printf("Key 20 found at index %d\n", result);

break;

case 2:

printf("Closed Addressing (Chaining):\n");

insertChaining(chainingTable, TABLE\_SIZE, 10);

insertChaining(chainingTable, TABLE\_SIZE, 20);

insertChaining(chainingTable, TABLE\_SIZE, 30);

printChainingTable(chainingTable, TABLE\_SIZE);

result = searchChaining(chainingTable, TABLE\_SIZE, 20);

printf("Key 20 found at index %d\n", result);

break;

case 3:

printf("Rehashing:\n");

insertRehashing(rehashingTable, 10);

insertRehashing(rehashingTable, 20);

insertRehashing(rehashingTable, 30);

printTable(rehashingTable->table, rehashingTable->size);

result = searchRehashing(rehashingTable, 20);

printf("Key 20 found at index %d\n", result);

break;

default:

printf("Invalid choice!\n");

break;

}

return 0;

}

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

119

**OUTPUT:**

Choose collision resolution technique:

1. Open Addressing (Linear Probing)

2. Closed Addressing (Chaining)

3. Rehashing

1

Open Addressing (Linear Probing):

10 -1 -1 -1 -1 -1 -1 -1 -1 -1

20 -1 -1 -1 -1 -1 -1 -1 -1 -1

30 -1 -1 -1 -1 -1 -1 -1 -1 -1

Key 20 found at index 1

Choose collision resolution technique:

1. Open Addressing (Linear Probing)

2. Closed Addressing (Chaining)

3. Rehashing

2

Closed Addressing (Chaining):

[0] -> NULL

[1] -> NULL

[2] -> NULL

[3] -> NULL

[4] -> NULL

[5] -> NULL

[6] -> NULL

[7] -> NULL

[8] -> NULL

[9] -> 30 -> 20 -> 10 -> NULL

Key 20 found at index 9

Choose collision resolution technique:

1. Open Addressing (Linear Probing)

2. Closed Addressing (Chaining)

3. Rehashing

3

Rehashing:

10 -1 -1 -1 -1 -1 -1 -1 -1 -1

20 -1 -1 -1 -1 -1 -1 -1 -1 -1

30 -1 -1 -1 -1 -1 -1 -1 -1 -1

Key 20 found at index 1

### Result:

### The program successfully implements a hash table with three different collision resolution techniques and demonstrates their functionality.

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

120



**Mini Projects**

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

121

|  |  |
| --- | --- |
| **Ex. No. :13** | **Date :** |
| **Mini Projects** |  |
| Mini Projects   * Snakes Game * Sudoku * Travel Planner * Cash Flow Minimiser Text Editor Cut, Copy, Paste   **AIM:**  To create a C program that implements a hash table and performs collision resolution using the following techniques:  1. Open Addressing  2. Closed Addressing (Chaining)  3. Rehashing  Algorithm  Open Addressing  1. Hash Function:  • Use a hash function to determine the initial index.  2. Probe Sequence:  • Use linear probing, quadratic probing, or double hashing to resolve collisions.  Closed Addressing (Chaining)  1. Hash Function:  • Use a hash function to determine the initial index.  2. Linked List:  • Use a linked list at each index to handle collisions.  Rehashing  1. Initial Hash Function:  • Use a hash function to determine the initial index.  2. Rehash:  • When the load factor exceeds a threshold, create a larger table and rehash all existing element. |  |

122

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**Program:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#define TABLE\_SIZE 10**

**#define LOAD\_FACTOR\_THRESHOLD 0.7**

**typedef struct {**

**int key;**

**int value;**

**} HashItem;**

**// Hash Function**

**unsigned int hashFunction(int key) {**

**return key % TABLE\_SIZE;**

**}**

**// Open Addressing**

**typedef struct {**

**HashItem\* table[TABLE\_SIZE];**

**} OpenAddressingHashTable;**

**void initOpenAddressingTable(OpenAddressingHashTable\* hashTable) {**

**for (int i = 0; i < TABLE\_SIZE; i++) {**

**hashTable->table[i] = NULL;**

**}**

**}**

**void insertOpenAddressing(OpenAddressingHashTable\* hashTable, int key, int value) {**

**unsigned int hashIndex = hashFunction(key);**

**while (hashTable->table[hashIndex] != NULL) {**

**hashIndex = (hashIndex + 1) % TABLE\_SIZE;**

**}**

**hashTable->table[hashIndex] = (HashItem\*)malloc(sizeof(HashItem));**

**hashTable->table[hashIndex]->key = key;**

**hashTable->table[hashIndex]->value = value;**

**}**

123

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**HashItem\* searchOpenAddressing(OpenAddressingHashTable\* hashTable, int key) {**

**unsigned int hashIndex = hashFunction(key);**

**while (hashTable->table[hashIndex] != NULL) {**

**if (hashTable->table[hashIndex]->key == key) {**

**return hashTable->table[hashIndex];**

**}**

**hashIndex = (hashIndex + 1) % TABLE\_SIZE;**

**}**

**return NULL;**

**}**

**// Closed Addressing (Chaining)**

**typedef struct ListNode {**

**int key;**

**int value;**

**struct ListNode\* next;**

**} ListNode;**

**typedef struct {**

**ListNode\* table[TABLE\_SIZE];**

**} ChainingHashTable;**

**void initChainingTable(ChainingHashTable\* hashTable) {**

**for (int i = 0; i < TABLE\_SIZE; i++) {**

**hashTable->table[i] = NULL;**

**}**

**}**

**void insertChaining(ChainingHashTable\* hashTable, int key, int value) {**

**unsigned int hashIndex = hashFunction(key);**

**ListNode\* newNode = (ListNode\*)malloc(sizeof(ListNode));**

**newNode->key = key;**

**newNode->value = value;**

**newNode->next = hashTable->table[hashIndex];**

**hashTable->table[hashIndex] = newNode;**

**}**

124

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

**ListNode\* searchChaining(ChainingHashTable\* hashTable, int key) {**

**unsigned int hashIndex = hashFunction(key);**

**ListNode\* currentNode = hashTable->table[hashIndex];**

**while (currentNode != NULL) {**

**if (currentNode->key == key) {**

**return currentNode;**

**}**

**currentNode = currentNode->next;**

**}**

**return NULL;**

**}**

**// Rehashing**

**typedef struct {**

**HashItem\*\* table;**

**int size;**

**int count;**

**} RehashingHashTable;**

**RehashingHashTable\* createRehashingTable(int size) {**

**RehashingHashTable\* hashTable = (RehashingHashTable\*)malloc(sizeof(RehashingHashTable));**

**hashTable->size = size;**

**hashTable->count = 0;**

**hashTable->table = (HashItem\*\*)malloc(sizeof(HashItem\*) \* size);**

**for (int i = 0; i < size; i++) {**

**hashTable->table[i] = NULL;**

**}**

**return hashTable;**

**}**

**unsigned int rehashingHashFunction(int key, int size) {**

**return key % size;**

**}**

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

125

**void rehash(RehashingHashTable\* hashTable) {**

**int oldSize = hashTable->size;**

**HashItem\*\* oldTable = hashTable->table;**

**hashTable->size \*= 2;**

**hashTable->count = 0;**

**hashTable->table = (HashItem\*\*)malloc(sizeof(HashItem\*) \* hashTable->size);**

**for (int i = 0; i < hashTable->size; i++) {**

**hashTable->table[i] = NULL;**

**}**

**for (int i = 0; i < oldSize; i++) {**

**if (oldTable[i] != NULL) {**

**insertRehashing(hashTable, oldTable[i]->key, oldTable[i]->value);**

**free(oldTable[i]);**

**}**

**}**

**free(oldTable);**

**}**

**void insertRehashing(RehashingHashTable\* hashTable, int key, int value) {**

**if ((float)hashTable->count / hashTable->size >= LOAD\_FACTOR\_THRESHOLD) {**

**rehash(hashTable);**

**}**

**unsigned int hashIndex = rehashingHashFunction(key, hashTable->size);**

**while (hashTable->table[hashIndex] != NULL) {**

**hashIndex = (hashIndex + 1) % hashTable->size;**

**}**

**hashTable->table[hashIndex] = (HashItem\*)malloc(sizeof(HashItem));**

**hashTable->table[hashIndex]->key = key;**

**hashTable->table[hashIndex]->value = value;**

**hashTable->count++;**

**}**

**HashItem\* searchRehashing(RehashingHashTable\* hashTable, int key) {**

**unsigned int hashIndex = rehashingHashFunction(key, hashTable->size);**

**while (hashTable->table[hashIndex] != NULL) {**

**if (hashTable->table[hashIndex]->key == key) {**

**return hashTable->table[hashIndex];**

**}**

**hashIndex = (hashIndex + 1) % hashTable->size;**

**}**

**return NULL;**

**}**

**// Main Function**

**int main() {**

**// Open Addressing**

**OpenAddressingHashTable oaTable;**

**initOpenAddressingTable(&oaTable);**

**insertOpenAddressing(&oaTable, 1, 10);**

**insertOpenAddressing(&oaTable, 11, 20);**

**HashItem\* oaItem = searchOpenAddressing(&oaTable, 11);**

**if (oaItem) {**

**printf("Open Addressing: Key: %d, Value: %d\n", oaItem->key, oaItem->value);**

**} else {**

**printf("Open Addressing: Key not found\n");**

**}**

**// Chaining**

**ChainingHashTable cTable;**

**initChainingTable(&cTable);**

**insertChaining(&cTable, 1, 10);**

**insertChaining(&cTable, 11, 20);**

**ListNode\* cItem = searchChaining(&cTable, 11);**

**if (cItem) {**

**printf("Chaining: Key: %d, Value: %d\n", cItem->key, cItem->value);**

**} else {**

**printf("Chaining: Key not found\n");**

**}**

Department of Computer Science and Business Systems, Rajalakshmi Engineering College

123

126

**// Rehashing**

**RehashingHashTable\* rTable = createRehashingTable(TABLE\_SIZE);**

**insertRehashing(rTable, 1, 10);**

**insertRehashing(rTable, 11, 20);**

**HashItem\* rItem = searchRehashing(rTable, 11);**

**if (rItem) {**

**printf("Rehashing: Key: %d, Value: %d\n", rItem->key, rItem->value);**

**} else {**

**printf("Rehashing: Key not found\n");**

**}**

**return 0;**

**}**

**Output :**

(1, 10)

(11, 20)

(21, 30)

Key not found

**Github: https://github.com/Subhu-1512**