q1.

|  |
| --- |
| Write a C program to perform Matrix Multiplication |

#include <stdio.h>

// function to get matrix elements entered by the user

void getMatrixElements(int matrix[][10], int row, int column) {

printf("\nEnter elements: \n");

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

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

printf("Enter a%d%d: ", i + 1, j + 1);

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

}

}

}

// function to multiply two matrices

void multiplyMatrices(int first[][10],

int second[][10],

int result[][10],

int r1, int c1, int r2, int c2) {

// Initializing elements of matrix mult to 0.

for (int i = 0; i < r1; ++i) {

for (int j = 0; j < c2; ++j) {

result[i][j] = 0;

}

}

// Multiplying first and second matrices and storing it in result

for (int i = 0; i < r1; ++i) {

for (int j = 0; j < c2; ++j) {

for (int k = 0; k < c1; ++k) {

result[i][j] += first[i][k] \* second[k][j];

}

}

}

}

// function to display the matrix

void display(int result[][10], int row, int column) {

printf("\nOutput Matrix:\n");

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

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

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

if (j == column - 1)

printf("\n");

}

}

}

int main() {

int first[10][10], second[10][10], result[10][10], r1, c1, r2, c2;

printf("Enter rows and column for the first matrix: ");

scanf("%d %d", &r1, &c1);

printf("Enter rows and column for the second matrix: ");

scanf("%d %d", &r2, &c2);

// Taking input until

// 1st matrix columns is not equal to 2nd matrix row

while (c1 != r2) {

printf("Error! Enter rows and columns again.\n");

printf("Enter rows and columns for the first matrix: ");

scanf("%d%d", &r1, &c1);

printf("Enter rows and columns for the second matrix: ");

scanf("%d%d", &r2, &c2);

}

// get elements of the first matrix

getMatrixElements(first, r1, c1);

// get elements of the second matrix

getMatrixElements(second, r2, c2);

// multiply two matrices.

multiplyMatrices(first, second, result, r1, c1, r2, c2);

// display the result

display(result, r1, c2);

return 0;

}

|  |
| --- |
| Q2 Write a C program to find Odd or Even number from a given set of numbers |

q2

#include <stdio.h>

int main() {

int num;

printf("Enter an integer: ");

scanf("%d", &num);

// true if num is perfectly divisible by 2

if(num % 2 == 0)

printf("%d is even.", num);

else

printf("%d is odd.", num);

return 0;

}

|  |
| --- |
|  |
| Q3.Write a C program to find Factorial of a given number without using Recursion |

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** i,fact=1,number;
5. printf("Enter a number: ");
6. scanf("%d",&number);
7. **for**(i=1;i<=number;i++){
8. fact=fact\*i;
9. }
10. printf("Factorial of %d is: %d",number,fact);
11. **return** 0;
12. }

|  |
| --- |
|  |
| Q4 Write a C program to find Factorial of a given number with using Recursion |
|  |

1. #include<stdio.h>
3. **long** factorial(**int** n)
4. {
5. **if** (n == 0)
6. **return** 1;
7. **else**
8. **return**(n \* factorial(n-1));
9. }
11. **void** main()
12. {
13. **int** number;
14. **long** fact;
15. printf("Enter a number: ");
16. scanf("%d", &number);
18. fact = factorial(number);
19. printf("Factorial of %d is %ld\n", number, fact);
20. **return** 0;
21. }

q5 Write a C program to find Fibonacci series without using Recursion

1. #include<stdio.h>
2. **int** main()
3. {
4. **int** n1=0,n2=1,n3,i,number;
5. printf("Enter the number of elements:");
6. scanf("%d",&number);
7. printf("\n%d %d",n1,n2);//printing 0 and 1
8. **for**(i=2;i<number;++i)//loop starts from 2 because 0 and 1 are already printed
9. {
10. n3=n1+n2;
11. printf(" %d",n3);
12. n1=n2;
13. n2=n3;
14. }
15. **return** 0;
16. }

q6 Write a C program to find Fibonacci series with using Recursion

1. #include<stdio.h>
2. **void** printFibonacci(**int** n){
3. **static** **int** n1=0,n2=1,n3;
4. **if**(n>0){
5. n3 = n1 + n2;
6. n1 = n2;
7. n2 = n3;
8. printf("%d ",n3);
9. printFibonacci(n-1);
10. }
11. }
12. **int** main(){
13. **int** n;
14. printf("Enter the number of elements: ");
15. scanf("%d",&n);
16. printf("Fibonacci Series: ");
17. printf("%d %d ",0,1);
18. printFibonacci(n-2);//n-2 because 2 numbers are already printed
19. **return** 0;
20. }

q7 Write a C program to implement Array operations such as Insert, Delete and Display

#include <stdio.h>

int main()

{

int n;

scanf(“%d”,&n);

int arr[n];

int i;

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

scanf(“%d”,&arr[i]);

}

int pos;

scanf(“%d”,&pos);

int ele;

scanf(“%d”,&ele);

if(pos > n)

printf(“Invalid Input”);

else

{

for (i = n – 1; i >= pos – 1; i–)

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

arr[pos-1] = ele;

printf(“Array after insertion is:\n”);

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

printf(“%d\n”, arr[i]);

}

return 0;

}

q8Write a C program to search a number using Linear Search method

#include <stdio.h>

int main()

{

int array[100], search, c, number;

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

scanf("%d",&number);

printf("Enter %d numbers\n", number);

for ( c = 0 ; c < number ; c++ )

scanf("%d",&array[c]);

printf("Enter the number to search\n");

scanf("%d",&search);

for ( c = 0 ; c < number ; c++ )

{

if ( array[c] == search ) /\* if required element found \*/

{

printf("%d is present at location %d.\n", search, c+1);

break;

}

}

if ( c == number )

printf("%d is not present in array.\n", search);

return 0;

}

q9Write a C program to search a number using Binary Search method

#include<stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

printf("Enter number of elements\n");

scanf("%d",&n);

printf("Enter %d integers\n", n);

for ( c = 0 ; c < n ; c++ )

scanf("%d",&array[c]);

printf("Enter value to find\n");

scanf("%d",&search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while( first <= last )

{

if ( array[middle] < search )

first = middle + 1;

else if ( array[middle] == search )

{

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

if ( first > last )

printf("Not found! %d is not present in the list.\n", search);

return 0;

}

q10 Write a C program to implement Linked list operations

#include <stdio.h>

#include <stdlib.h>

// Create a node

struct Node {

int data;

struct Node\* next;

};

// Insert at the beginning

void insertAtBeginning(struct Node\*\* head\_ref, int new\_data) {

// Allocate memory to a node

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

// insert the data

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

// Move head to new node

(\*head\_ref) = new\_node;

}

// Insert a node after a node

void insertAfter(struct Node\* prev\_node, int new\_data) {

if (prev\_node == NULL) {

printf("the given previous node cannot be NULL");

return;

}

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = prev\_node->next;

prev\_node->next = new\_node;

}

// Insert the the end

void insertAtEnd(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

struct Node\* last = \*head\_ref; /\* used in step 5\*/

new\_node->data = new\_data;

new\_node->next = NULL;

if (\*head\_ref == NULL) {

\*head\_ref = new\_node;

return;

}

while (last->next != NULL) last = last->next;

last->next = new\_node;

return;

}

// Delete a node

void deleteNode(struct Node\*\* head\_ref, int key) {

struct Node \*temp = \*head\_ref, \*prev;

if (temp != NULL && temp->data == key) {

\*head\_ref = temp->next;

free(temp);

return;

}

// Find the key to be deleted

while (temp != NULL && temp->data != key) {

prev = temp;

temp = temp->next;

}

// If the key is not present

if (temp == NULL) return;

// Remove the node

prev->next = temp->next;

free(temp);

}

// Search a node

int searchNode(struct Node\*\* head\_ref, int key) {

struct Node\* current = \*head\_ref;

while (current != NULL) {

if (current->data == key) return 1;

current = current->next;

}

return 0;

}

// Sort the linked list

void sortLinkedList(struct Node\*\* head\_ref) {

struct Node \*current = \*head\_ref, \*index = NULL;

int temp;

if (head\_ref == NULL) {

return;

} else {

while (current != NULL) {

// index points to the node next to current

index = current->next;

while (index != NULL) {

if (current->data > index->data) {

temp = current->data;

current->data = index->data;

index->data = temp;

}

index = index->next;

}

current = current->next;

}

}

}

// Print the linked list

void printList(struct Node\* node) {

while (node != NULL) {

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

node = node->next;

}

}

// Driver program

int main() {

struct Node\* head = NULL;

insertAtEnd(&head, 1);

insertAtBeginning(&head, 2);

insertAtBeginning(&head, 3);

insertAtEnd(&head, 4);

insertAfter(head->next, 5);

printf("Linked list: ");

printList(head);

printf("\nAfter deleting an element: ");

deleteNode(&head, 3);

printList(head);

int item\_to\_find = 3;

if (searchNode(&head, item\_to\_find)) {

printf("\n%d is found", item\_to\_find);

} else {

printf("\n%d is not found", item\_to\_find);

}

sortLinkedList(&head);

printf("\nSorted List: ");

printList(head);

}

q11 Write a C program to implement Stack operations such as PUSH,

POP and PEEK

#include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

top=-1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d",&n);

printf("\n\t STACK OPERATIONS USING ARRAY");

printf("\n\t--------------------------------");

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

return 0;

}

void push()

{

if(top>=n-1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i--)

printf("\n%d",stack[i]);

printf("\n Press Next Choice");

}

else

{

printf("\n The STACK is empty");

}

}

q12

|  |
| --- |
| Write a C program to implement the application of Stack (Notations) |

#include <stdio.h>

int MAXSIZE = 8;

int stack[8];

int top = -1;

/\* Check if the stack is empty \*/

int isempty(){

if(top == -1)

return 1;

else

return 0;

}

/\* Check if the stack is full \*/

int isfull(){

if(top == MAXSIZE)

return 1;

else

return 0;

}

/\* Function to return the topmost element in the stack \*/

int peek(){

return stack[top];

}

/\* Function to delete from the stack \*/

int pop(){

int data;

if(!isempty()) {

data = stack[top];

top = top - 1;

return data;

} else {

printf("Could not retrieve data, Stack is empty.\n");

}

}

/\* Function to insert into the stack \*/

int push(int data){

if(!isfull()) {

top = top + 1;

stack[top] = data;

} else {

printf("Could not insert data, Stack is full.\n");

}

}

/\* Main function \*/

int main(){

push(44);

push(10);

push(62);

push(123);

push(15);

printf("Element at top of the stack: %d\n" ,peek());

printf("Elements: \n");

// print stack data

while(!isempty()) {

int data = pop();

printf("%d\n",data);

}

printf("Stack full: %s\n" , isfull()?"true":"false");

printf("Stack empty: %s\n" , isempty()?"true":"false");

return 0;

}

q13

|  |
| --- |
| Write a C program to implement Queue operations such as ENQUEUE, DEQUEUE and Display |

1. #include <stdio.h>
3. #define MAX 50
5. void insert();
6. void delete();
7. void display();
8. int queue\_array[MAX];
9. int rear = - 1;
10. int front = - 1;
11. main()
12. {
13. int choice;
14. while (1)
15. {
16. printf("1.Insert element to queue **\n**");
17. printf("2.Delete element from queue **\n**");
18. printf("3.Display all elements of queue **\n**");
19. printf("4.Quit **\n**");
20. printf("Enter your choice : ");
21. scanf("%d", &choice);
22. switch (choice)
23. {
24. case 1:
25. insert();
26. **break**;
27. case 2:
28. delete();
29. **break**;
30. case 3:
31. display();
32. **break**;
33. case 4:
34. exit(1);
35. default:
36. printf("Wrong choice **\n**");
37. } */\* End of switch \*/*
38. } */\* End of while \*/*
39. } */\* End of main() \*/*
41. void insert()
42. {
43. int add\_item;
44. if (rear == MAX - 1)
45. printf("Queue Overflow **\n**");
46. else
47. {
48. if (front == - 1)
49. */\*If queue is initially empty \*/*
50. front = 0;
51. printf("Inset the element in queue : ");
52. scanf("%d", &add\_item);
53. rear = rear + 1;
54. queue\_array[rear] = add\_item;
55. }
56. } */\* End of insert() \*/*
58. void delete()
59. {
60. if (front == - 1 || front > rear)
61. {
62. printf("Queue Underflow **\n**");
63. return ;
64. }
65. else
66. {
67. printf("Element deleted from queue is : %d**\n**", queue\_array[front]);
68. front = front + 1;
69. }
70. } */\* End of delete() \*/*
72. void display()
73. {
74. int i;
75. if (front == - 1)
76. printf("Queue is empty **\n**");
77. else
78. {
79. printf("Queue is : **\n**");
80. for (i = front; i <= rear; i++)
81. printf("%d ", queue\_array[i]);
82. printf("**\n**");
83. }

q14

|  |
| --- |
| Write a C program to implement the Tree Traversals (Inorder, Preorder, Postorder) |
|  |

#include <stdio.h>

#include <stdlib.h>

struct node {

int item;

struct node\* left;

struct node\* right;

};

// Inorder traversal

void inorderTraversal(struct node\* root) {

if (root == NULL) return;

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

// preorderTraversal traversal

void preorderTraversal(struct node\* root) {

if (root == NULL) return;

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

// postorderTraversal traversal

void postorderTraversal(struct node\* root) {

if (root == NULL) return;

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

// Create a new Node

struct node\* createNode(value) {

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

newNode->item = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Insert on the left of the node

struct node\* insertLeft(struct node\* root, int value) {

root->left = createNode(value);

return root->left;

}

// Insert on the right of the node

struct node\* insertRight(struct node\* root, int value) {

root->right = createNode(value);

return root->right;

}

int main() {

struct node\* root = createNode(1);

insertLeft(root, 12);

insertRight(root, 9);

insertLeft(root->left, 5);

insertRight(root->left, 6);

printf("Inorder traversal \n");

inorderTraversal(root);

printf("\nPreorder traversal \n");

preorderTraversal(root);

printf("\nPostorder traversal \n");

postorderTraversal(root);

}

**[Pre](https://www.programiz.com/dsa/trees" \o "Tree Data Structure)**

q15 Write a C program to implement hashing using Linear Probing method

1. #include<stdio.h>
2. #include<stdlib.h>
4. */\* to store a data (consisting of key and value) in hash table array \*/*
5. struct item
6. {
7. int key;
8. int value;
9. };
11. */\* each hash table item has a flag (status) and data (consisting of key and value) \*/*
12. struct hashtable\_item
13. {
15. int flag;
16. */\**
17. *\* flag = 0 : data does not exist*
18. *\* flag = 1 : data exists*
19. *\* flag = 2 : data existed at least once*
20. *\*/*
22. struct item \*data;
24. };
26. struct hashtable\_item \*array;
27. int size = 0;
28. int max = 10;
30. */\* initializing hash table array \*/*
31. void init\_array()
32. {
33. int i;
34. for (i = 0; i < max; i++)
35. {
36. array[i].flag = 0;
37. array[i].data = NULL;
38. }
39. }
41. */\* to every key, it will generate a corresponding index \*/*
42. int hashcode(int key)
43. {
44. return (key % max);
45. }
47. */\* to insert an element in the hash table \*/*
48. void insert(int key, int value)
49. {
50. int index = hashcode(key);
51. int i = index;
53. */\* creating new item to insert in the hash table array \*/*
54. struct item \*new\_item = (struct item\*) malloc(sizeof(struct item));
55. new\_item->key = key;
56. new\_item->value = value;
58. */\* probing through the array until we reach an empty space \*/*
59. while (array[i].flag == 1)
60. {
62. if (array[i].data->key == key)
63. {
65. */\* case where already existing key matches the given key \*/*
66. printf("**\n** Key already exists, hence updating its value **\n**");
67. array[i].data->value = value;
68. return;
70. }
72. i = (i + 1) % max;
73. if (i == index)
74. {
75. printf("**\n** Hash table is full, cannot insert any more item **\n**");
76. return;
77. }
79. }
81. array[i].flag = 1;
82. array[i].data = new\_item;
83. size++;
84. printf("**\n** Key (%d) has been inserted **\n**", key);
86. }

89. */\* to remove an element from the hash table \*/*
90. void remove\_element(int key)
91. {
92. int index = hashcode(key);
93. int i = index;
95. */\* probing through array until we reach an empty space where not even once an element had been present \*/*
96. while (array[i].flag != 0)
97. {
99. if (array[i].flag == 1 && array[i].data->key == key )
100. {
102. *// case when data key matches the given key*
103. array[i].flag = 2;
104. array[i].data = NULL;
105. size--;
106. printf("**\n** Key (%d) has been removed **\n**", key);
107. return;
109. }
110. i = (i + 1) % max;
111. if (i == index)
112. {
113. **break**;
114. }
116. }
118. printf("**\n** This key does not exist **\n**");
120. }
122. */\* to display all the elements of hash table \*/*
123. void display()
124. {
125. int i;
126. for (i = 0; i < max; i++)
127. {
128. struct item \*current = (struct item\*) array[i].data;
130. if (current == NULL)
131. {
132. printf("**\n** Array[%d] has no elements **\n**", i);
133. }
134. else
135. {
136. printf("**\n** Array[%d] has elements -: **\n** %d (key) and %d(value) ", i, current->key, current->value);
137. }
138. }
140. }
142. int size\_of\_hashtable()
143. {
144. return size;
145. }
147. void main()
148. {
149. int choice, key, value, n, c;
150. clrscr();
152. array = (struct hashtable\_item\*) malloc(max \* sizeof(struct hashtable\_item\*));
153. init\_array();
155. do {
156. printf("Implementation of Hash Table in C with Linear Probing **\n\n**");
157. printf("MENU-: **\n**1.Inserting item in the Hashtable"
158. "**\n**2.Removing item from the Hashtable"
159. "**\n**3.Check the size of Hashtable"
160. "**\n**4.Display Hashtable"
161. "**\n\n** Please enter your choice-:");
163. scanf("%d", &choice);
165. switch(choice)
166. {
168. case 1:
170. printf("Inserting element in Hashtable**\n**");
171. printf("Enter key and value-:**\t**");
172. scanf("%d %d", &key, &value);
173. insert(key, value);
175. **break**;
177. case 2:
179. printf("Deleting in Hashtable **\n** Enter the key to delete-:");
180. scanf("%d", &key);
181. remove\_element(key);
183. **break**;
185. case 3:
187. n = size\_of\_hashtable();
188. printf("Size of Hashtable is-:%d**\n**", n);
190. **break**;
192. case 4:
194. display();
196. **break**;
198. default:
200. printf("Wrong Input**\n**");
202. }
204. printf("**\n** Do you want to continue-:(press 1 for yes)**\t**");
205. scanf("%d", &c);
207. }while(c == 1);
209. getch();
211. }