**INDEX**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Serial  Number | Title of the program | Date | | Signature | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
| 10. |  |  | |  | |
| 11. |  |  | |  | |
| 12. |  |  | |  | |
| 13. |  |  | |  | |
| 14. |  |  | |  | |
| 15. |  |  | |  | |
| 16. |  | |  | |  |

**Experiment – 1**

**Aim –**

Write a program to implement all operations on 1-D array

**Expiation –**

#include <stdio.h>

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

printf("Array elements: ");

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

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

}

printf("\n");

}

int insertion(int arr[],int size,int element,int position){

if(position<0 || position>size){

printf("INVALID POSITION FOR INSERTION \n");

return size;

}

for (int i=size-1;i>=position;i--){

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

}

arr[position]=element;

return size+1;

}

int deletion(int arr[],int size,int position){

if(position<0 || position>=size){

printf("INVALID POSITION FOR DELETION \n");

return size;

}

for(int i=position;i<size-1;i++){

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

}

return size-1;

}

int search(int arr[],int size,int element){

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

if(arr[i]==element){

return i;

}

}

return -1;

}

void update(int arr[],int size,int position,int newValue){

if(position>=0 && position<size){

arr[position]=newValue;

}

else{

printf("INVALID POSITION FOR UPDATING \n");

}

}

int main(){

int arr[100];int size=0;

size=insertion(arr,size,100,0);

size=insertion(arr,size,200,1);

size=insertion(arr,size,300,2);

printf("After Insertion:\n");

display(arr,size);

size=deletion(arr,size,1);

printf("After Deletion: \n");

display(arr,size);

int searchIndex=search(arr,size,300);

if(searchIndex!=-1){

printf("Element 300 found at index %d\n",searchIndex);

}

else{

printf("Element 300 not found\n");

}

update(arr,size,1,400);

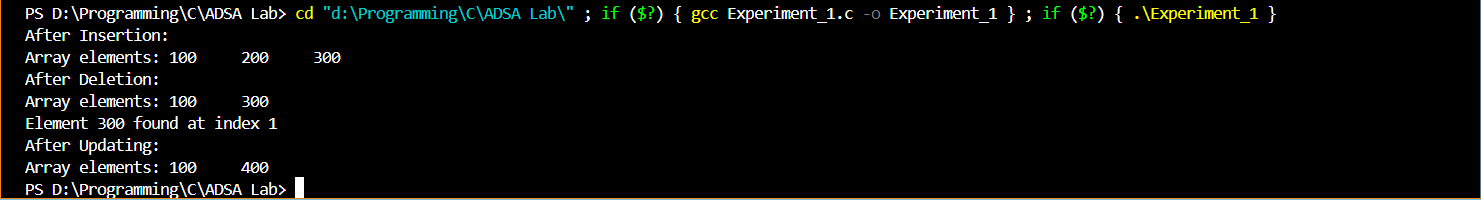
printf("After Updating: \n");

display(arr,size);

return 0;

}

**Output –**

****

**Experiment-2**

**Aim –**

Write a program to implement all operations on simple linked list.

**Expiation –**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node\* next;

};

void insertBegin(struct Node\*\*head,int data){

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

newNode->data=data;

newNode->next=\*head;

\*head=newNode;

}

void insertEnd(struct Node\*\*head,int data){

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

newNode->data=data;

newNode->next=NULL;

if(\*head==NULL){

\*head=newNode;

return;

}

struct Node\*current=\*head;

while(current->next!=NULL){

current=current->next;

}

current->next=newNode;

}

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

if(\*head==NULL){

printf("List is empty\n");

return;

}

if((\*head)->data==data){

struct Node\*temp=\*head;

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

free(temp);

return;

}

struct Node\*current=\*head;

while(current->next!=NULL && current->next->data!=data){

current=current->next;

}

if(current->next==NULL){

printf("Value not found in list\n");

return;

}

struct Node\*temp=current->next;

current->next = current->next->next;

free(temp);

}

void display(struct Node\*head){

struct Node\*current=head;

while(current!=NULL){

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

current=current->next;

}

printf("NULL \n");

}

int main(){

struct Node\*head=NULL;

insertEnd(&head,100);

insertEnd(&head,200);

insertBegin(&head,50);

insertEnd(&head,300);

printf("Linked List after insertion \n");

display(head);

deleteNode(&head,200);

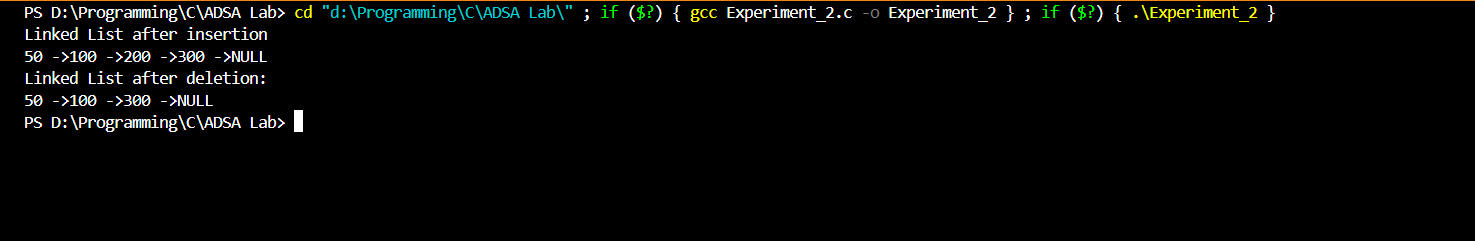
printf("Linked List after deletion: \n");

display(head);

return 0;

}

**Output-**

****

**Experiment-3**

**Aim –**

Write a program to implement all operations on a circular linked list.

**Expiation –**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node\*next;

};

struct Node\*insertBegin(struct Node\*head,int data){

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

newNode->data=data;

if(head==NULL){

newNode->next=newNode;

}

else{

struct Node\*current=head;

while(current->next!=head){

current=current->next;

}

current->next=newNode;

newNode->next=head;

}

return newNode;

}

void display(struct Node\*head){

if(head==NULL){

printf("List is empty \n");

return;

}

struct Node\*current=head;

do{

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

current=current->next;

}

while(current!=head);

printf("...\n");

}

int main(){

struct Node\*head=NULL;

head=insertBegin(head,100);

head=insertBegin(head,200);

head=insertBegin(head,300);

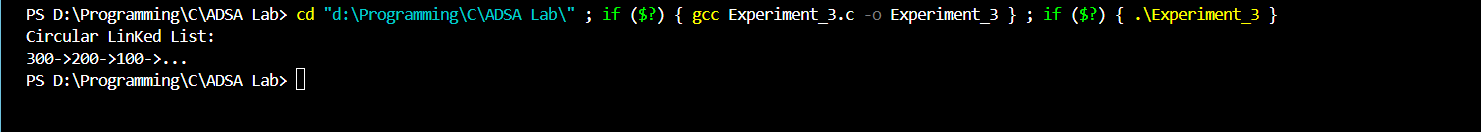
printf("Circular LinKed List: \n");

display(head);

return 0;

}

**Output-**

****

**Experiment-4**

**Aim –**

Write a program to implement all operations on a doubly linked list.

**Expiation –**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node\*prev;

struct Node\*next;

};

void insertEnd(struct Node\*\*head,int data){

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

newNode->data=data;

newNode->next=NULL;

if(\*head==NULL){

newNode->prev=NULL;

\*head=newNode;

return;

}

struct Node\*current=\*head;

while(current->next!=NULL){

current=current->next;

}

current->next=newNode;

newNode->prev=current;

}

void display(struct Node\*head){

printf("Forward: ");

struct Node\*current=head;

while(current!=NULL){

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

current=current->next;

}

printf("NULL \n");

printf("Backward: ");

current=head;

while(current->next!=NULL){

current=current->next;

}

while(current!=NULL){

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

current=current->prev;

}

printf("NULL \n");

}

int main(){

struct Node\*head=NULL;

insertEnd(&head,900);

insertEnd(&head,800);

insertEnd(&head,700);

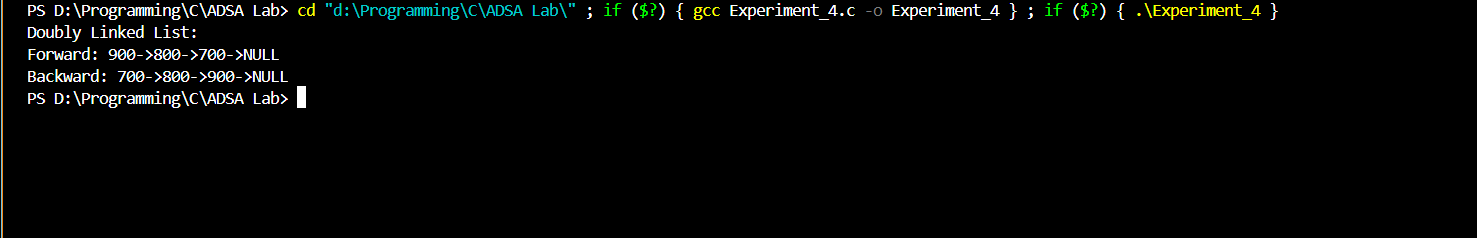
printf("Doubly Linked List: \n");

display(head);

return 0;

}

**Output –**

****

**Experiment-5**

**Aim –**

Write a program to implement all operations on a doubly circular linked list.

**Expiation –**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node\*prev;

struct Node\*next;

};

struct Node\*insertEnd(struct Node\*head,int data){

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

newNode->data=data;

if(head==NULL){

newNode->prev=newNode;

newNode->next=newNode;

return newNode;

}

struct Node\*last=head->prev;

newNode->next=head;

newNode->prev=last;

head->prev=newNode;

last->next=newNode;

return head;

}

void display(struct Node\*head) {

if(head==NULL){

printf("List is empty \n");

return;

}

struct Node\*current=head;

printf("Forward: ");

do{

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

current=current->next;

}

while(current!=head);

printf("...\n");

current=head->prev;

printf("Backward: ");

do{

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

current=current->prev;

}

while(current!=head->prev);

printf(".... \n");

}

int main(){

struct Node\*head=NULL;

head=insertEnd(head,900);

head=insertEnd(head,800);

head=insertEnd(head,700);

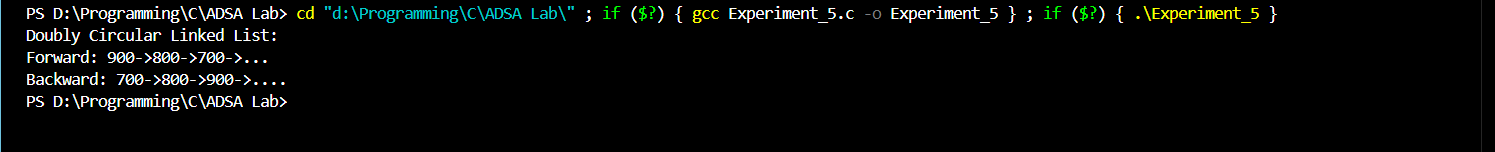
printf("Doubly Circular Linked List: \n");

display(head);

return 0;

}

**Output -**

****

**Experiment-6**

**Aim –**

Write a program to implement all operations on stack using array.

**Expiation –**

#include<stdio.h>

#include<stdbool.h>

#define MAX\_SIZE 100

struct Stack{

int arr[MAX\_SIZE];

int top;

};

void initializeStack(struct Stack\*stack){

stack->top=-1;

}

bool isEmpty(struct Stack\*stack){

return stack->top==-1;

}

bool isFull(struct Stack\*stack){

return stack->top==MAX\_SIZE-1;

}

void push(struct Stack\*stack,int value){

if(isFull(stack)){

printf("Stack overflow,cannot push %d\n",value);

return;

}

stack->top++;

stack->arr[stack->top]=value;

}

int pop(struct Stack\*stack){

if(isEmpty(stack)){

printf("Stack underflow,cannot pop\n");

return -1;

}

int value=stack->arr[stack->top];

stack->top--;

return value;

}

int peek(struct Stack\*stack){

if(isEmpty(stack)){

printf("Stack is empty,no top element\n");

return -1; }

return stack->arr[stack->top]; }

int main(){

struct Stack stack;

initializeStack(&stack);

push(&stack,150);

push(&stack,250);

push(&stack,300);

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

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

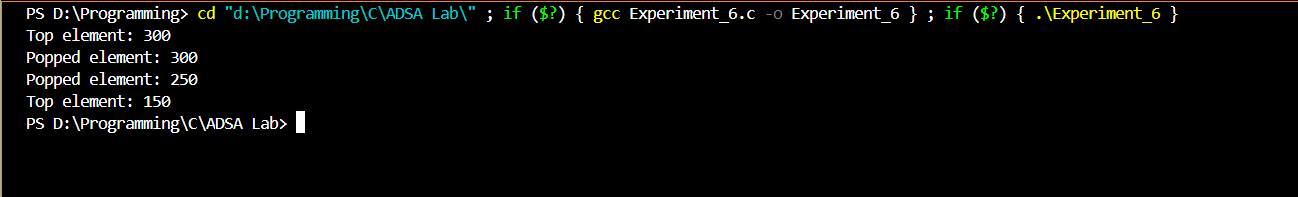
printf("Popped element: %d\n",pop(&stack));

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

return 0;

}

**Output –**

****

**Experiment – 7**

**Aim –**

Write a program to implement all operations on stack using linked list.

**Expiation –**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct Node{

int data;

struct Node\*next;

};

struct Stack{

struct Node\*top;

};

void initializeStack(struct Stack\*stack){

stack->top=NULL;

}

bool isEmpty(struct Stack\*stack){

return stack->top==NULL;

}

void push(struct Stack\*stack,int value){

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

newNode->data=value;

newNode->next=stack->top;

stack->top=newNode;

}

int pop(struct Stack\*stack){

if(isEmpty(stack)){

printf("Stack underflow,cannot pop\n");

return -1;

}

struct Node\* temp=stack->top;

int value=temp->data;

stack->top=temp->next;

free(temp);

return value;

}

int peek(struct Stack\*stack){

if(isEmpty(stack)){

printf("Stack is empty,no top element\n");

return -1;

}

return stack->top->data;

}

void display(struct Stack\*stack){

struct Node\*current=stack->top;

printf("Stack elements: ");

while(current!=NULL){

printf("%d \t",current->data);

current=current->next;

}

printf("\n");

}

int main(){

struct Stack stack;

initializeStack(&stack);

push(&stack,100);

push(&stack,250);

push(&stack,300);

display(&stack);

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

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

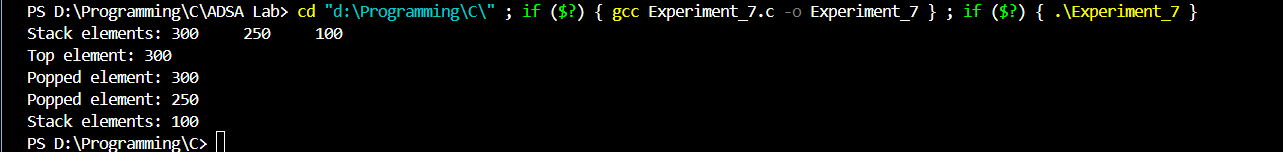
printf("Popped element: %d\n",pop(&stack));

display(&stack);

return 0;

}

**Output –**



**Experiment – 8**

**Aim –**

Write a program to implement all operations on queue using array.

**Expiation –**

#include <stdio.h>

#include <stdbool.h>

#define MAX 100

struct Queue {

int items[MAX];

int front;

int rear;

};

void initializeQueue(struct Queue\*queue) {

queue->front=-1;

queue->rear=-1;

}

bool isEmpty(struct Queue\*queue) {

return queue->front == -1;

}

bool isFull(struct Queue\*queue) {

return queue->rear==MAX-1;

}

void enqueue(struct Queue\*queue, int value) {

if (isFull(queue)) {

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

return;

}

if (isEmpty(queue)) {

queue->front = 0;

}

queue->rear++;

queue->items[queue->rear] = value;

printf("%d enqueued to queue\n", value);

}

int dequeue(struct Queue\*queue) {

if (isEmpty(queue)) {

printf("Queue is empty!\n");

return -1;

}

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

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

queue->front = -1;

queue->rear = -1;

}

else {

queue->front++;

}

return item;

}

void display(struct Queue\*queue) {

if (isEmpty(queue)) {

printf("Queue is empty!\n");

return;

}

printf("Queue elements: ");

for (int i = queue->front; i <= queue->rear; i++) {

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

}

printf("\n");

}

int main() {

struct Queue queue;

initializeQueue(&queue);

enqueue(&queue, 100);

enqueue(&queue, 200);

enqueue(&queue, 300);

display(&queue);

printf("%d dequeued from queue\n", dequeue(&queue));

display(&queue);

enqueue(&queue, 400);

enqueue(&queue, 500);

display(&queue);

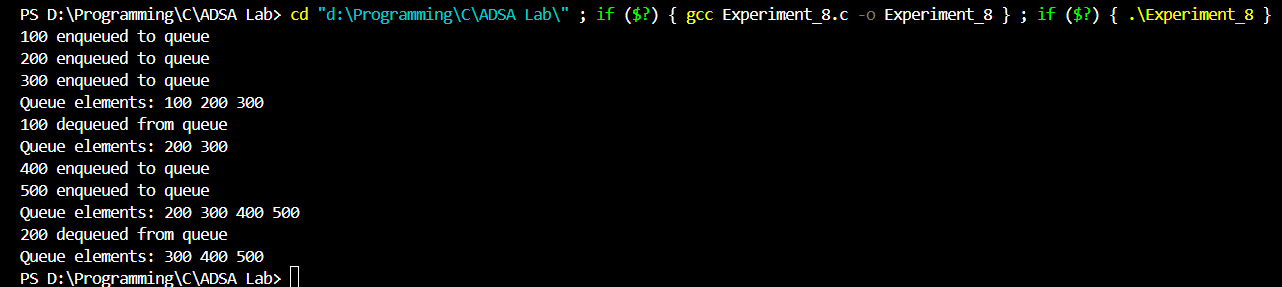
printf("%d dequeued from queue\n", dequeue(&queue));

display(&queue);

return 0;

}

**Output –**



**Experiment – 9**

**Aim –**

Write a program to implement all operations on queue using Linked List.

**Expiation –**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct Node {

int data;

struct Node\* next;

};

struct Queue {

struct Node\* front;

struct Node\* rear; };

struct Queue\* createQueue() {

struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

queue->front = NULL;

queue->rear = NULL;

return queue;

}

bool isEmpty(struct Queue\* queue) {

return queue->front == NULL; }

void enqueue(struct Queue\* queue, int value) {

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

newNode->data = value;

newNode->next = NULL;

if (isEmpty(queue)) {

queue->front = newNode;

}

else {

queue->rear->next = newNode;

}

queue->rear = newNode;

printf("%d enqueued to queue\n", value);

}

int dequeue(struct Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue is empty!\n");

return -1;

}

struct Node\* temp = queue->front;

int item = temp->data;

queue->front = queue->front->next;

if (queue->front == NULL) {

queue->rear = NULL;

}

free(temp);

return item;

}

void display(struct Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue is empty!\n");

return;

}

struct Node\* current = queue->front;

printf("Queue elements: ");

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

int main() {

struct Queue\* queue = createQueue();

enqueue(queue, 100);

enqueue(queue, 200);

enqueue(queue, 300);

display(queue);

printf("%d dequeued from queue\n", dequeue(queue));

display(queue);

enqueue(queue, 400);

enqueue(queue, 500);

display(queue);

printf("%d dequeued from queue\n", dequeue(queue));

display(queue);

while (!isEmpty(queue)) {

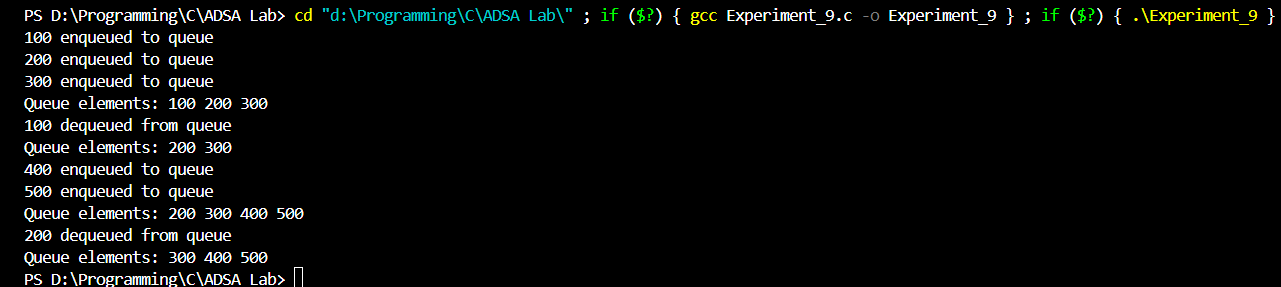
dequeue(queue);

}

free(queue);

return 0; }

**Output –**



**Experiment – 10**

**Aim –**

Write a program to implement hashing techniques.

**Expiation –**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TABLE\_SIZE 100

struct KeyValue {

char key[50];

char value[50];

struct KeyValue\* next;

};

struct HashTable {

struct KeyValue\* table[TABLE\_SIZE];

} HashTable;

unsigned int hash(const char\* key) {

unsigned long int hashval = 0;

for (int i = 0; key[i] != '\0'; i++) {

hashval = (hashval << 5) + key[i];

}

return hashval % TABLE\_SIZE;

}

struct HashTable\* createHashTable() {

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

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

ht->table[i] = NULL;

}

return ht;

}

void insert(struct HashTable\* ht, const char\* key, const char\* value) {

unsigned int index = hash(key);

struct KeyValue\* newEntry = (struct KeyValue\*)malloc(sizeof(struct KeyValue));

strcpy(newEntry->key, key);

strcpy(newEntry->value, value);

newEntry->next = NULL;

if (ht->table[index] == NULL) {

ht->table[index] = newEntry;

}

else {

struct KeyValue\* current = ht->table[index];

while (current->next != NULL) {

if (strcmp(current->key, key) == 0) {

strcpy(current->value, value);

free(newEntry);

return;

}

current = current->next;

}

current->next = newEntry;

}

}

const char\* search(struct HashTable\* ht, const char\* key) {

unsigned int index = hash(key);

struct KeyValue\* current = ht->table[index];

while (current != NULL) {

if (strcmp(current->key, key) == 0) {

return current->value;

}

current = current->next;

}

return NULL;

}

void delete(struct HashTable\* ht, const char\* key) {

unsigned int index = hash(key);

struct KeyValue\* current = ht->table[index];

struct KeyValue\* prev = NULL;

while (current != NULL) {

if (strcmp(current->key, key) == 0) {

if (prev == NULL) {

// Removing the first entry in the chain

ht->table[index] = current->next;

} else {

// Bypass the current entry

prev->next = current->next;

}

free(current);

return;

}

prev = current;

current = current->next;

}

printf("Key not found: %s\n", key);

}

void freeHashTable(struct HashTable\* ht) {

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

struct KeyValue\* current = ht->table[i];

while (current != NULL) {

struct KeyValue\* toDelete = current;

current = current->next;

free(toDelete);

}

}

free(ht);

}

int main() {

struct HashTable\* ht = createHashTable();

insert(ht, "name", "John Doe");

insert(ht, "age", "30");

insert(ht, "city", "New York");

printf("Searching for 'name': %s\n", search(ht, "name"));

printf("Searching for 'age': %s\n", search(ht, "age"));

printf("Searching for 'city': %s\n", search(ht, "city"));

insert(ht, "age", "31");

printf("Updated age: %s\n", search(ht, "age"));

delete(ht, "city");

printf("Searching for 'city' after deletion: %s\n", search(ht, "city"));

freeHashTable(ht);

return 0;

}

**Output –**

