**Assignment 4**

**Question 1**

Write a function “insert\_any()” for inserting a node at any given position of the linked list. Assume position starts at 0.

**Sol.**

//Function to insert a node at any position.

void insert\_any()

{

int data\_value, key;

printf("\nEnter data of the node: ");

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

printf("\nEnter data of the node after which new node is to be inserted: ");scanf("%d", &key);

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

ptr = header;while(ptr->link != NULL && ptr->data != key)

{

ptr = ptr->link;

}

if(ptr->data == key)

{

temp->data = data\_value;

temp->link = ptr->link;

ptr->link = temp;

}

else

{

printf("\nValue %d not found\n",key);

}

}

**Question 2**

Write a function “delete\_beg()” for deleting a node from the beginning of the linked list.

**Sol.**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <malloc.h>

struct node { int data; struct node \* next; };

struct node \* start = NULL;

struct node \* create\_ll(struct node \* );

struct node \* display(struct node \* );

struct node \* insert\_beg(struct node \* );

struct node \* insert\_end(struct node \* );

struct node \* delete\_beg(struct node \* );

struct node \* delete\_end(struct node \* );

struct node \* delete\_list(struct node \* );

void main()

{

int option;

do {

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

printf("\n 1: Create a list");

printf("\n 2: Display the list");

printf("\n 3: Add a node at the beginning");

printf("\n 4: Add a node at the end");

printf("\n 5: Delete a node from the beginning");

printf("\n 6: Delete a node from the end");

printf("\n 7: Delete the entire list");

printf("\n 8: EXIT");

printf("\n\n Enter your option : ");

scanf("%d", &option);

switch (option)

{

case 1:

start = create\_ll(start);

printf("\n LINKED LIST CREATED");

break;

case 2:

start = display(start);

break;

case 3:

start = insert\_beg(start);

break;

case 4:

start = insert\_end(start);

break;

case 5:

start = delete\_beg(start);

break;

case 6:

start = delete\_end(start);

break;

case 7:

start = delete\_list(start);

printf("\n LINKED LIST DELETED");

break;

}

}

while (option != 8);

getch();

}

struct node \* create\_ll(struct node \* start) {

struct node \* new\_node, \* ptr;

int num;

printf("\n Enter -1 to end");

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

scanf("%d", & num);

while (num != -1)

{

new\_node = (struct node \* )

malloc(sizeof(struct node));

new\_node -> data = num;

if (start == NULL)

{

new\_node -> next = NULL;

start = new\_node;

}

else

{

ptr = start;

while (ptr -> next != NULL)

ptr = ptr -> next;

ptr -> next = new\_node;

new\_node -> next = NULL;

}

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

scanf("%d", & num);

}

return start;

}

struct node \* display(struct node \* start)

{

struct node \* ptr; ptr = start;

while (ptr != NULL)

{

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

ptr = ptr -> next;

}

return start;

}

struct node \* insert\_beg(struct node \* start)

{

struct node \* new\_node;

int num;

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

scanf("%d", & num);

new\_node = (struct node \*)

malloc(sizeof(struct node));

new\_node -> data = num;

new\_node -> next = start;

start = new\_node;

return start;

}

struct node \* insert\_end(struct node \* start)

{

struct node \* ptr, \* new\_node;

int num;

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

scanf("%d", &num);

new\_node = (struct node \* )

malloc(sizeof(struct node));

new\_node -> data = num;

new\_node -> next = NULL;

ptr = start;

while (ptr -> next != NULL)

ptr = ptr -> next;

ptr -> next = new\_node;

return start;

}

struct node \* delete\_beg(struct node \* start)

{

struct node \* ptr;

ptr = start;

start = start -> next;

free(ptr);

return start;

}

struct node \* delete\_end(struct node \* start)

{

struct node \* ptr, \* preptr;

ptr = start;

while (ptr -> next != NULL)

{

preptr = ptr;

ptr = ptr -> next;

}

preptr -> next = NULL;

free(ptr);

return start;

}

struct node \* delete\_list(struct node \* start)

{

struct node \* ptr;

if (start != NULL)

{

ptr = start;

while (ptr != NULL)

{

printf("\n %d is to be deleted next", ptr -> data);

start = delete\_beg(ptr);

ptr = start;

}

}

return start;

}

**Question 3**

Write a function “delete\_end()” for deleting a node from the end of the linked list.

**Sol.**

#include<stdio.h>

#include<cstdlib>

struct node

{

int data;

struct node \*link;

};

struct node \*header, \*ptr, \*ptr1, \*temp;

void delete\_end();

int main()

{

int choice;

int cont = 1;

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

header->data = NULL;

header->link = NULL;

delete\_end();

printf("\nContents of the linked list are: \n");

ptr = header;

while(ptr->link != NULL)

{

ptr = ptr->link;

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

}

return 0;);

}

void delete\_end()

{

if(header->link == NULL)

{

printf("\nEmpty Linked List. Deletion not possible.\n");

}

else

{

ptr = header;

while(ptr->link != NULL)

{

ptr1 = ptr;

ptr = ptr->link;

}

ptr1->link = ptr->link;

free(ptr);

printf("\nNode deleted from the end.\n"); }

}

**Question 4**

In the Binary Search algorithm, it is suggested to calculate the mid as beg + (end - beg) / 2 instead of (beg + end) / 2. Why is it so?

**Sol.**

There's no guarantee that beg+end is representable; but in the first case the intermediate values, as well as the expected result, are no larger than end, so there is no danger of overflow.

The first form can also be used for affine types like pointers and other random-access iterators, which can be subtracted to give a distance, but not added together.

**Question 5**

Write the algorithm/function for Ternary Search.

**Sol.**

#include <stdio.h>

int ternarySearch(int array[], int left, int right, int x)

if (right >= left)

{

int intvl = (right - left) / 3;

int leftmid = left + intvl;

int rightmid = leftmid + intvl;

if (array[leftmid] == x)

return leftmid;

if (array[rightmid] == x)

return rightmid;

if (x < array[leftmid])

{

return ternarySearch(array, left, leftmid, x);

}

else if (x > array[leftmid] && x < array[rightmid])

{

return ternarySearch(array, leftmid, rightmid, x);

}

else {

return ternarySearch(array, rightmid, right, x); }

}

return -1;

}

int main(void)

{

int array[] = {1, 2, 3, 5};

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

int find = 3;

printf("Position of %d is %d\n", find, ternarySearch(array, 0, size-1, find)); return 0;

}