**NAME:** AGNIV PRAMANICK **SECTION:** A **USN:** 1NT21IS017 **DATE:** 27/02/23

**Q. Write a program to implement singly linked list using c program.**

**THEORY**

1. A linked list is a group of randomly stored items in memory, collectively referred to as nodes.
2. A node has two fields: the data that is stored there and a pointer that points to the node in memory that comes after it.
3. Pointer to the null is contained in the list's final node.

A single linked list is a grouping of an ordered set of elements. Depending on the requirements of the programme, the number of pieces may change. Data and link components make up a node in a single linked list. While the link portion of the node contains the address of its immediate successor, the data portion of the node keeps the actual information that will be represented by the node.

A single-linked list or a one-way chain can only be traversed in one direction. In other words, since each node only carries a pointer to the next item, we cannot traverse the list backwards.

///////////////////////////////////////////////////////////////////////////////

**ALGORITHM**

**Insert at beginning**

Step 1: IF PTR = NULL then, Write OVERFLOW Go to Step 7. [END OF IF]

Step 2: SET NEW\_NODE = PTR

Step 3: SET PTR = PTR -> NEXT

Step 4: SET NEW\_NODE -> DATA = VAL

Step 5: SET NEW\_NODE -> NEXT = HEAD

Step 6: SET HEAD = NEW\_NODE

Step 7: EXIT

**Insert at end**

Step 1: IF AVAIL = NULL Write OVERFLOW. Go to Step 10. [END OF IF]

Step 2: SET NEW\_NODE = AVAIL

Step 3: SET AVAIL = AVAIL -> NEXT

Step 4: SET NEW\_NODE -> DATA = VAL

Step 5: SET NEW\_NODE -> NEXT = NULL

Step 6: SET TEMP = HEAD

Step 7: Repeat Step 8 while TEMP -> NEXT! = NULL

Step 8: SET TEMP = TEMP -> NEXT [END OF LOOP]

Step 9: SET TEMP -> NEXT = NEW\_NODE

Step 10: EXIT

**Insert random**

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 12

[END OF IF]

Step 2: SET NEW\_NODE = AVAIL

Step 3: SET AVAIL = AVAIL -> NEXT

Step 4: SET NEW\_NODE -> DATA = VAL

Step 5: SET TEMP = HEAD

Step 6: SET PRETEMP = TEMP

Step 7: Repeat Steps 8 and 9 while TEMP -> DATA! = NUM

Step 8: SET TEMP = TEMP

Step 9: SET TEMP = TEMP -> NEXT

[END OF LOOP]

Step 10: TEMP -> NEXT = NEW\_NODE

Step 11: SET NEW\_NODE -> NEXT = TEMP

Step 12: EXIT

**Delete at beginning**

Step 1: IF HEAD = NULL

Write UNDERFLOW

Go to Step 5

[END OF IF]

Step 2: SET PTR = HEAD

Step 3: SET HEAD = HEAD -> NEXT

Step 4: FREE PTR

Step 5: EXIT

**Delete at end**

Step 1: IF HEAD = NULL

Write UNDERFLOW

Go to Step 8

[END OF IF]

Step 2: SET PTR = HEAD

Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != NULL

Step 4: SET PREPTR = PTR

Step 5: SET PTR = PTR -> NEXT

[END OF LOOP]

Step 6: SET PREPTR -> NEXT = NULL

Step 7: FREE PTR

Step 8: EXIT

**Delete random**

Step 1: IF HEAD = NULL

Write UNDERFLOW

Go to Step 10

[END OF IF]

Step 2: SET PTR = HEAD

Step 3: SET PREPTR = PTR

Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM

Step 5: SET PREPTR = PTR

Step 6: SET PTR = PTR -> NEXT

[END OF LOOP]

Step 7: SET TEMP = PTR

Step 8: SET PREPTR -> NEXT = PTR -> NEXT

Step 9: FREE TEMP

Step 10: EXIT

**SEARCH**

Step 1: [INITIALIZE] SET PTR = HEAD

Step 2: Repeat Steps 3 and 4 while PTR != NULL

Step 3: If ITEM = PTR -> DATA

SET POS = PTR

Go To Step 5

ELSE

SET PTR = PTR -> NEXT

[END OF IF]

[END OF LOOP]

Step 4: SET POS = NULL

Step 5: EXIT

///////////////////////////////////////////////////////////////////////////////////// **CODE**

#include<stdio.h> //library functions inserted

#include<stdlib.h>

struct node //basic node structure given

{

int data;

struct node \*next;

};

struct node \*head;

// implementing user defined functions for all possible combinations

void beginsert ();

void lastinsert ();

void randominsert();

void begin\_delete();

void last\_delete();

void random\_delete();

void display();

void search();

// main function starts

void main ()

{

int choice =0;

while(choice != 9) //making a menu based on choice in a while loop

{

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

printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n5.Delete from last\n6.Delete node after specified location\n7.Search for an element\n8.Show\n9.Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice); //choice stored

switch(choice)

{

case 1:

beginsert();

break;

case 2:

lastinsert();

break;

case 3:

randominsert();

break;

case 4:

begin\_delete();

break;

case 5:

last\_delete();

break;

case 6:

random\_delete();

break;

case 7:

search();

break;

case 8:

display();

break;

case 9:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

}

void beginsert() //to insert an element

{

struct node \*ptr;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW"); //if ptr is null only

}

else

{

printf("\nEnter value\n");

scanf("%d",&item); //item value stored

ptr->data = item;

ptr->next = head;

head = ptr;

printf("\nNode inserted");

}

}

void lastinsert() //to insert but from the last

{

struct node \*ptr,\*temp;

int item;

ptr = (struct node\*)malloc(sizeof(struct node)); //ptr memory allocated

if(ptr == NULL)

{

printf("\nOVERFLOW"); //overflow statement given

}

else

{

printf("\nEnter value?\n");

scanf("%d",&item); //item stored

ptr->data = item;

if(head == NULL)

{

ptr -> next = NULL;

head = ptr;

printf("\nNode inserted");

}

else

{

temp = head; //if head is not null

while (temp -> next != NULL)

{

temp = temp -> next;

}

temp->next = ptr;

ptr->next = NULL;

printf("\nNode inserted");

}

}

}

void randominsert() //randomly inserting element

{

int i,loc,item;

struct node \*ptr, \*temp;

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

if(ptr == NULL)

{

printf("\nOVERFLOW"); //overflow statement given

}

else

{

printf("\nEnter element value");

scanf("%d",&item);

ptr->data = item;

printf("\nEnter the location after which you want to insert "); //asking for the location

scanf("\n%d",&loc); //value stored

temp=head;

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

{

temp = temp->next;

if(temp == NULL) //case of temp to be null

{

printf("\ncan't insert\n");

return;

}

}

ptr ->next = temp ->next;

temp ->next = ptr;

printf("\nNode inserted");

}

}

void begin\_delete() //to delete an element

{

struct node \*ptr;

if(head == NULL)

{

printf("\nList is empty\n");

}

else

{

ptr = head;

head = ptr->next;

free(ptr); //to free ptr

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

}

}

void last\_delete() //to delete from last

{

struct node \*ptr,\*ptr1;

if(head == NULL)

{

printf("\nlist is empty"); //list empty

}

else if(head -> next == NULL)

{

head = NULL;

free(head);

printf("\nOnly node of the list deleted ...\n"); //if head is null

}

else

{

ptr = head; //make ptr equal to head

while(ptr->next != NULL)

{

ptr1 = ptr;

ptr = ptr ->next;

}

ptr1->next = NULL;

free(ptr);

printf("\nDeleted Node from the last ...\n");

}

}

void random\_delete() //to randomly delete the element

{

struct node \*ptr,\*ptr1;

int loc,i;

printf("\n Enter the location of the node after which you want to perform deletion \n");

scanf("%d",&loc); //store the loc value

ptr=head;

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

{

ptr1 = ptr;

ptr = ptr->next;

if(ptr == NULL) //if ptr is null pointer

{

printf("\nCan't delete");

return;

}

}

ptr1 ->next = ptr ->next;

free(ptr);

printf("\nDeleted node %d ",loc+1); //node deleted

}

void search() // to search an element

{

struct node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item); //item to be searched stored

while (ptr!=NULL)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("Item not found\n"); //as flag value is 1

}

}

}

void display() //to display elements

{

struct node \*ptr;

ptr = head;

if(ptr == NULL) //no print if ptr is null

{

printf("Nothing to print");

}

else

{

printf("\nprinting values . . . . .\n"); //values printed

while (ptr!=NULL)

{

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

ptr = ptr -> next;

}

}

} //end

//////////////////////////////////////////////////////////////////////////////////////////////////////////

**OUTPUT**























