//Write a menu driven code to implement Singly Linked List

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

struct node

{

int data;

struct node \*next;

};

struct node \*start = NULL;

struct node \*createSLL(struct node \*start);

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

struct node \*InsertAtBeginning(struct node \*start);

struct node \*InsertAtEnd(struct node \*start);

struct node \*InsertBefore(struct node \*start);

struct node \*DeleteBeginning(struct node \*start);

struct node \*DeleteEnd(struct node \*start);

struct node \*DeleteNode(struct node \*start);

struct node \*ForwardTraversal(struct node \*start);

struct node \*BackwardTraversal(struct node \*start);

struct node \*Sorting(struct node \*start);

struct node \*Count(struct node \*start);

struct node \*Search(struct node \*start);

int main()

{

int choice;

start = createSLL(start);

printf("\nSINGLY LINKED LIST CREATED\n");

start = display(start);

do {

printf("\n\n\*\*\*\*List of Operations\*\*\*\*");

printf("\n 1: Insert at beginning");

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

printf("\n 3: Insert at before a node");

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

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

printf("\n 6: Delete node before a specified location");

printf("\n 7: Forward Traversal");

printf("\n 8: Backward Traversal");

printf("\n 9: Sorting");

printf("\n 10: Count number of nodes");

printf("\n 11: Search an element");

printf("\n 12: EXIT");

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

scanf("%d", &choice);

switch (choice) {

case 1:

start = InsertAtBeginning(start);

printf("\n");

start = display(start);

break;

case 2:

start = InsertAtEnd(start);

printf("\n");

start = display(start);

break;

case 3:

start = InsertBefore(start);

printf("\n");

start = display(start);

break;

case 4:

start = DeleteBeginning(start);

printf("\n");

start = display(start);

break;

case 5:

start = DeleteEnd(start);

printf("\n");

start = display(start);

break;

case 6:

start = DeleteNode(start);

printf("\n");

start = display(start);

break;

case 7:

start = ForwardTraversal(start);

printf("\n");

break;

case 8:

start = BackwardTraversal(start);

printf("\n");

start = display(start);

break;

case 9:

start = Sorting(start);

printf("\n");

start = display(start);

break;

case 10:

start = Count(start);

printf("\n");

break;

case 11:

start = Search(start);

printf("\n");

break;

case 12:

printf("\n\tEXIT POINT");

break;

}

} while (choice != 12);

return 0;

}

struct node \*createSLL(struct node \*start)

{

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

int val;

printf("\nEnter a value(enter -1 to end): ");

scanf("%d", &val);

while (val != -1) {

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

new\_node->data = val;

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("Enter a value: ");

scanf("%d", &val);

}

return start;

}

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

{

struct node \*ptr;

ptr = start;

if (ptr == NULL) {

printf("\tEmpty List!");

}

else {

while (ptr != NULL) {

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

ptr = ptr->next;

}

}

return start;

}

struct node \*InsertAtBeginning(struct node \*start)

{

struct node \*new\_node;

int val;

printf("Enter a value: ");

scanf("%d", &val);

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

new\_node->data = val;

new\_node->next = start;

start = new\_node;

return start;

}

struct node \*InsertAtEnd(struct node \*start)

{

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

int val;

printf("Enter a value: ");

scanf("%d", &val);

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

new\_node->data = val;

new\_node->next = NULL;

ptr = start;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=new\_node;

return start;

}

struct node \*InsertBefore(struct node \*start)

{

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

int val, num;

printf("Enter a value: ");

scanf("%d", &val);

printf("Enter the number before which the data has to be inserted: ");

scanf("%d", &num);

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

new\_node->data = val;

ptr = start;

while (ptr->data != num) {

preptr = ptr;

ptr = ptr->next;

}

preptr -> next = new\_node;

new\_node -> next = ptr;

return start;

}

struct node \*DeleteBeginning(struct node \*start)

{

struct node \*ptr;

ptr = start;

start = start->next;

free(ptr);

return start;

}

struct node \*DeleteEnd(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 \*DeleteNode(struct node \*start)

{

struct node \*preptr, \*ptr;

int val;

printf("Enter the value before which the data has to be deleted: ");

scanf("%d", &val);

ptr = start;

if(ptr->data == val-1) {

start = DeleteBeginning(start);

return start;

}

else {

while(ptr->data != val-1) {

preptr = ptr;

ptr = ptr->next;

}

preptr->next = ptr->next;

free(ptr);

return start;

}

}

struct node \*ForwardTraversal(struct node \*start)

{

struct node \*ptr;

ptr = start;

if (ptr == NULL) {

printf("\tEmpty List!");

}

else {

printf("\n");

while (ptr != NULL) {

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

ptr = ptr->next;

}

}

return start;

}

struct node \*BackwardTraversal(struct node \*start)

{

struct node\* prev = NULL;

struct node\* current = start;

struct node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

start = prev;

}

struct node \*Sorting(struct node \*start)

{

struct node \*ptr1, \*ptr2;

int temp;

ptr1 = start;

while (ptr1->next != NULL) {

ptr2 = ptr1->next;

while (ptr2 != NULL) {

if (ptr1->data > ptr2->data) {

temp = ptr1->data;

ptr1->data = ptr2->data;

ptr2->data = temp;

}

ptr2 = ptr2->next;

}

ptr1 = ptr1->next;

}

return start;

}

struct node \*Count(struct node \*start)

{

int i;

i=0;

while(start!=NULL) {

i=i+1;

start=start->next;

}

printf("Number of nodes in the list: %d", i);

}

struct node \*Search(struct node \*start)

{

struct node\* current;

int val;

printf("Enter a value that is to be searched: ");

scanf("%d", &val);

if(start == NULL) printf("\tEmpty List!");

else {

current = start;

while (current != NULL) {

if (current -> data == val) printf("\tElement found");

break;

}

current = current->next;

}

if(current == NULL) {

printf("\tElement not found");

}

}













