**CODE NO. 2**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

These three header files are being included in the code. The stdio.h provides the input/output, the stdlib.h provdes the memory allocation and the conio.h provides the console input and output.

int choice;

// structure definition

struct node {

struct node \*prev;

int x;

struct node \*next;

};

Int choice is used to store the user’s menu choice. This also has a structure named node it represents a node in a doubly linked list. The prev is used to point the previous node in the list

Int x is used as an integer variable to store the data of the node. The next pointer is used to point the next node in the list

struct node \*head, \*curr, \*tail; // structure declaration

// function prototyping

void create();

void display();

void disnormal();

void disreverse();

void sort();

void ascend();

void descend();

void insert();

void dele();

void search();

These functions perform various operations on the doubly linked list program. Such as creating, displaying in a normal order or disreverse order, sorting, inserting, deleting and searching.

int main () {

int close = 0; // initialize ng value for close para masatisfy yung condition below

while(close != 1) { // magrarun continously unless close = 1

menu:

system("cls");

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

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

printf("\t| MAIN MENU |\n");

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

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

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

printf("\t| 1. Create a Doubly Linked List |\n");

printf("\t| 2. Display a Doubly Linked List |\n");

printf("\t| 3. Sort a Doubly Linked List |\n");

printf("\t| 4. Insert a Node |\n");

printf("\t| 5. Delete a Node |\n");

printf("\t| 6. Search a number in the list |\n");

printf("\t| 7. Exit |\n");

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

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

printf("\t [Enter choice]: ");

scanf("%d", &choice);

system("cls");

switch (choice) {

case 1:

create();

break;

case 2:

display();

break;

case 3:

sort();

break;

case 4:

insert();

break;

case 5:

dele();

break;

case 6:

search();

break;

case 7:

printf("\n\t[ Thank you for using our program! ]\n\n");

close = 1; // the program will exit

break;

default: printf("\n\n\t[ Invalid number! Choose 1-7 only. ]");

printf(" \n\n\tPress any key to continue...\n");

getch();

goto menu;

}

}

printf("\n\t=====< End of program >=====");

getch();

return 0;

}

This main function is used to execute the program. It displays the menu based on the choice of the user. It calls the different function to perform specific operations on the program. The program will keep running unless the user chooses to exit the program.

void create() {

head = curr = tail = NULL; // initialize the pointer to null

curr = (struct node \*)malloc(sizeof(struct node)); // allocate the node

printf("\n\t=================[ OUTPUT ]================\n\n");

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

printf("\tEnter a value for x [ 0 to terminate ]: ");

scanf("%d", &curr->x);

while(curr->x != 0) {

if(head == NULL) {

head = curr;

head->prev = NULL;

head->next = NULL;

tail = curr;

} else {

tail->next = curr;

curr->prev = tail;

curr->next = NULL;

tail = curr;

}

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

printf("\tEnter a value for x [ 0 to terminate ]: ");

scanf("%d", &curr->x);

}

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

printf("\n\t=====< End of program >=====");

printf("\n\tPress enter to return to main menu...");

getch();

}

void disnormal() {

if (head == NULL) {

return;

}

else {

printf("\n");

curr = head;

while (curr != NULL) {

printf("\t[ %d ] ", curr->x);

curr = curr->next;

}

}

}

void disreverse() {

if (head == NULL) {

return;

} else {

printf("\n");

curr = tail;

while (curr != NULL) {

printf("\t[ %d ] ", curr->x);

curr = curr->prev;

}

}

}

void display() {

int c;

int exit = 0;

while(exit == 0) {

system("cls");

printf("\n\t=========================[ OUTPUT ]==========================\n");

choice:

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

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

printf("\t| How do you want your linked list data to be display? |\n");

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

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

printf("\t| [ 1 ] Normal Order |\n");

printf("\t| [ 2 ] Reverse Order |\n");

printf("\t| [ 3 ] Exit |\n");

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

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

printf("\n\t[ Enter your choice ]: ");

scanf("%d", &c);

system("cls");

switch (c) {

case 1:

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

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

printf("\n\t| Linked List Data in Normal Order: |");

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

if (head == NULL) {

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

printf("\t| [ Empty! ] |\n");

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

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

printf("\n\t<Create a doubly linked list first>");

printf("\n\tPress enter to continue...");

} else {

disnormal();

printf("\n\n\tPress enter to continue...");

getch();

}

break;

case 2:

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

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

printf("\n\t| Linked List Data in Reverse Order: |");

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

if (head == NULL) {

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

printf("\t| [ Empty! ] |\n");

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

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

printf("\n\t<Create a doubly linked list first>");

printf("\n\tPress enter to continue...");

} else {

disreverse();

printf("\n\n\tPress enter to continue...");

getch();

}

break;

case 3:

printf("\n\t=====< End of program >=====");

printf("\n\tPress enter to return to main menu...");

exit = 1;

break;

default:

printf("\n\n\t[ Invalid Input! ]\n");

printf("\n\tPlease enter 1-3 only! Press any key to continue...");

getch();

system("cls");

goto choice;

}

getch();

}

}

void ascend() {

int temp;

struct node \*curr;

struct node \*tail;

//Check whether list is empty

if(head == NULL) {

return;

}

else {

//Current will point to head

curr = head;

while(curr->next != NULL) {

//tail will point to node next to current

tail = curr->next;

while(tail != NULL) {

//If current's data is greater than tail's data, swap the data of current and tail

if(curr->x > tail->x) {

temp = curr->x;

curr->x = tail->x;

tail->x = temp;

}

tail = tail->next;

}

curr = curr->next;

}

}

disnormal();

}

void descend() {

int temp;

struct node \*curr;

struct node \*tail;

//Check whether list is empty

if(head == NULL) {

return;

}

else {

//Current will point to head

curr = head;

while(curr->next != NULL) {

// tail will point to node next to current

tail = curr->next;

while(tail != NULL) {

//If current's data is less than tail's data, swap the data of current and tail

if(curr->x < tail->x) {

temp = curr->x;

curr->x = tail->x;

tail->x = temp;

}

tail = tail->next;

}

curr = curr->next;

}

}

disnormal();

}

This display function is responsible for displaying the doubly linked list. It allows the user to choose what they want to display whether in normal or disreverse order.

void sort() {

int c;

int exit = 0;

while(exit == 0) {

system("cls");

printf("\n\t=========================[ OUTPUT ]==========================\n");

choice:

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

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

printf("\t| How do you want your linked list data to be sorted? |\n");

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

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

printf("\t| [ 1 ] Ascending Order |\n");

printf("\t| [ 2 ] Descending Order |\n");

printf("\t| [ 3 ] Exit |\n");

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

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

printf("\n\t[ Enter choice ]: ");

scanf("%d", &c);

system("cls");

switch (c) {

case 1:

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

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

printf("\n\t| Linked List Data in Ascending Order: |");

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

if (head == NULL) {

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

printf("\t| [ Empty! ] |\n");

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

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

printf("\n\t--< Create a doubly linked list first >--");

}

else {

ascend();

printf("\n\n\tPress enter to continue...");

getch();

}

break;

case 2:

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

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

printf("\n\t| Linked List Data in Descending Order: |");

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

if (head == NULL) {

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

printf("\t| [ Empty! ] |\n");

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

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

printf("\n\t--< Create a doubly linked list first >--");

}

else {

descend();

printf("\n\n\tPress enter to continue...");

getch();

}

break;

case 3:

printf("\n\t=====< End of program >=====");

printf("\n\n\tPress enter to return to main menu...");

exit = 1;

break;

default:

printf("\n\n\t[ Invalid Input! ]\n");

printf("\n\tPlease enter 1-3 only! Press any key to continue...");

getch();

system("cls");

goto choice;

}

getch();

}

}

The doubly linked list must be sorted using the sort function. The user can select whether they want to sort the list in ascending or descending order using the choices provided. It uses the ascend or descend feature to order the list in accordance with the user's selection.

void insert() {

char deci; // decision

input:

system("cls");

printf("\n\t=================[ OUTPUT ]================\n");

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

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

printf("\n\t| Linked List Data: |");

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

ascend();

// check if the list is empty

if(head == NULL) {

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

printf("\t| [ Empty! ] |\n");

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

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

printf("\n\t--< Create a doubly linked list first >--");

printf("\n\tPress any key to continue...");

getch();

return;

}

// create a new node and assign the value

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

printf("\n\n\tEnter a value for new node: ");

scanf("%d", &newnode->x);

newnode->prev = NULL;

newnode->next = NULL;

if(newnode->x < head->x) { //check if the new node should be inserted at the beginning

newnode->next = head;

head->prev = newnode;

head = newnode;

}

else if(newnode->x > tail->x) { //check if the new node should be inserted at the end

newnode->prev = tail;

tail->next = newnode;

tail = newnode;

}

else { // insert the new node in the middle of the list

struct node \*temp = head; // temp will point to head

while(temp->next != NULL && temp->next->x < newnode->x) { // temp pointer will move to the next node if true

temp = temp->next;

}

newnode->prev = temp; // newnode->prev will point to temp

newnode->next = temp->next; //newnode->next will point to the node next to temp

if(temp->next != NULL) {

temp->next->prev = newnode; // the node next to temp will point to new node

}

temp->next = newnode; // the new node will become the node next to temp

}

//printing of new linked list data after insertion

again:

system("cls");

printf("\n\t============[ After Insertion ]============\n");

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

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

printf("\n\t| New Linked List Data: |");

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

disnormal();

printf("\n\n\tDo you want to insert another node? [ Y/N ]: ");

scanf(" %c", &deci);

switch (deci) {

case 'Y':

case 'y':

goto input;

break;

case 'N':

case 'n':

printf("\n\n\tPress enter to return to main menu...");

break;

default:

printf("\n\n\t[ Invalid Input ]\n\n\tPlease enter Y or N. Press enter to continue");

getch();

goto again;

}

getch();

}

A new node is inserted into the doubly linked list using the insert function. It asks the user to provide a value for the new node and, based on that value, determines whether the node should be added at the start, middle, or end of the list. In order to properly link the new node, it updates the appropriate pointers.

void dele() {

int numdel; // number to be deleted

char d; // decision

int exit = 0;

system("cls");

printf("\n\t=================[ OUTPUT ]================\n\n");

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

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

printf("\n\t| Linked List Data: |");

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

ascend();

if (head == NULL) {

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

printf("\t| [Empty!] |\n");

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

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

printf("\n\t--< Create a doubly linked list first >--");

printf("\n\tPress any key to continue...");

getch();

return;

}

do {

retry:

printf("\n\n\tEnter the number you want to delete: ");

scanf("%d", &numdel);

struct node \*temp = head;

struct node \*curr = NULL;

if (temp != NULL && temp->x == numdel) {

head = temp->next;

free(temp);

} else {

while (temp != NULL && temp->x != numdel) {

curr = temp;

temp = temp->next;

}

if (temp == NULL) {

printf("\n\t[ The number you enter is not in the list! ]");

goto choose;

}

curr->next = temp->next;

free(temp);

}

ulit:

system("cls");

printf("\n\t=============[ After Deletion ]============\n");

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

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

printf("\n\t| New Linked List Data: |");

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

disnormal();

if (head == NULL) {

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

printf("\t| [ Empty! ] |\n");

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

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

printf("\n\tPress any key to continue...");

getch();

return;

}

choose:

printf("\n\n\tDo you want to try again? [ Y/N ]: ");

scanf(" %c", &d);

switch (d) {

case 'Y':

case 'y':

goto retry;

break;

case 'N':

case 'n':

printf("\n\n\tPress enter to return to main menu...");

exit = 1;

break;

default:

printf("\n\n\t[ Invalid Input ]\n\n\tPlease enter Y or N. Press enter to continue");

getch();

goto ulit;

}

} while(exit != 1);

getch();

}

A node in the doubly linked list is removed using the dele function. It asks the user to provide a value that should be destroyed, looks for the node that has that value, and then deletes it. It deals with situations in which the node that needs to be deleted is at the start, middle, or end of the list.

void search() {

struct node \*ptr; // a pointer that travel accross the nodes

int val; //value to be search

int posi; // position

char decide;

int found;

int exit;

int occur;

if(head == NULL) {

system("cls");

printf("\n\t=================[ OUTPUT ]================\n");

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

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

printf("\n\t| Linked List Data: |");

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

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

printf("\t| [ Empty! ] |\n");

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

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

printf("\n\t--< Create a doubly linked list first >--");

printf("\n\tPress any key to continue...");

getch();

return;

}

do {

retry:

system("cls");

posi = occur = found = exit = 0;

printf("\n\n\tEnter a value to be searched: ");

scanf("%d", &val);

ptr = head;

while(ptr != NULL) {

posi++;

if(ptr->x == val) {

printf("\n\t[%d is in the list! It is in the node number: %d]", val, posi);

found = 1;

occur++;

}

ptr = ptr->next;

}

printf("\n\n\t[There is/are %d occurrences of the number %d.]\n", occur, val);

if(found == 0) {

printf("\n\n\t[ %d is not in the list!]", val);

}

printf("\n\t============[ FOR CHECKING ]===============\n");

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

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

printf("\n\t| Linked List Data: |");

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

disnormal();

isapa:

printf("\n\n\tDo you want to search another value? [ Y/N ]: ");

scanf(" %c", &decide);

switch (decide) {

case 'Y':

case 'y':

goto retry;

break;

case 'N':

case 'n':

printf("\n\n\tPress enter to return to main menu...");

exit = 1;

break;

default:

printf("\n\n\t[ Invalid Input ]\n\n\tPlease enter Y or N. Press enter to continue...");

getch();

goto isapa;

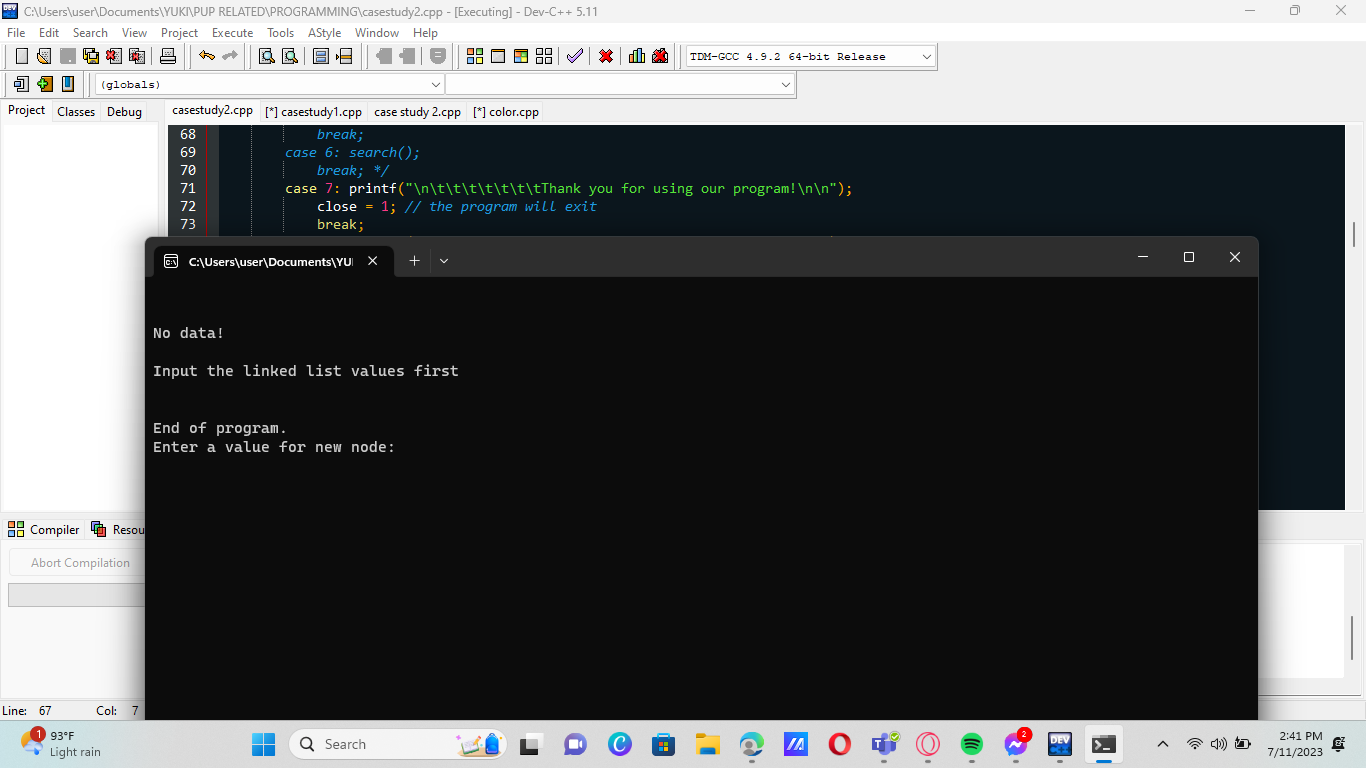
}

} while (exit != 1);

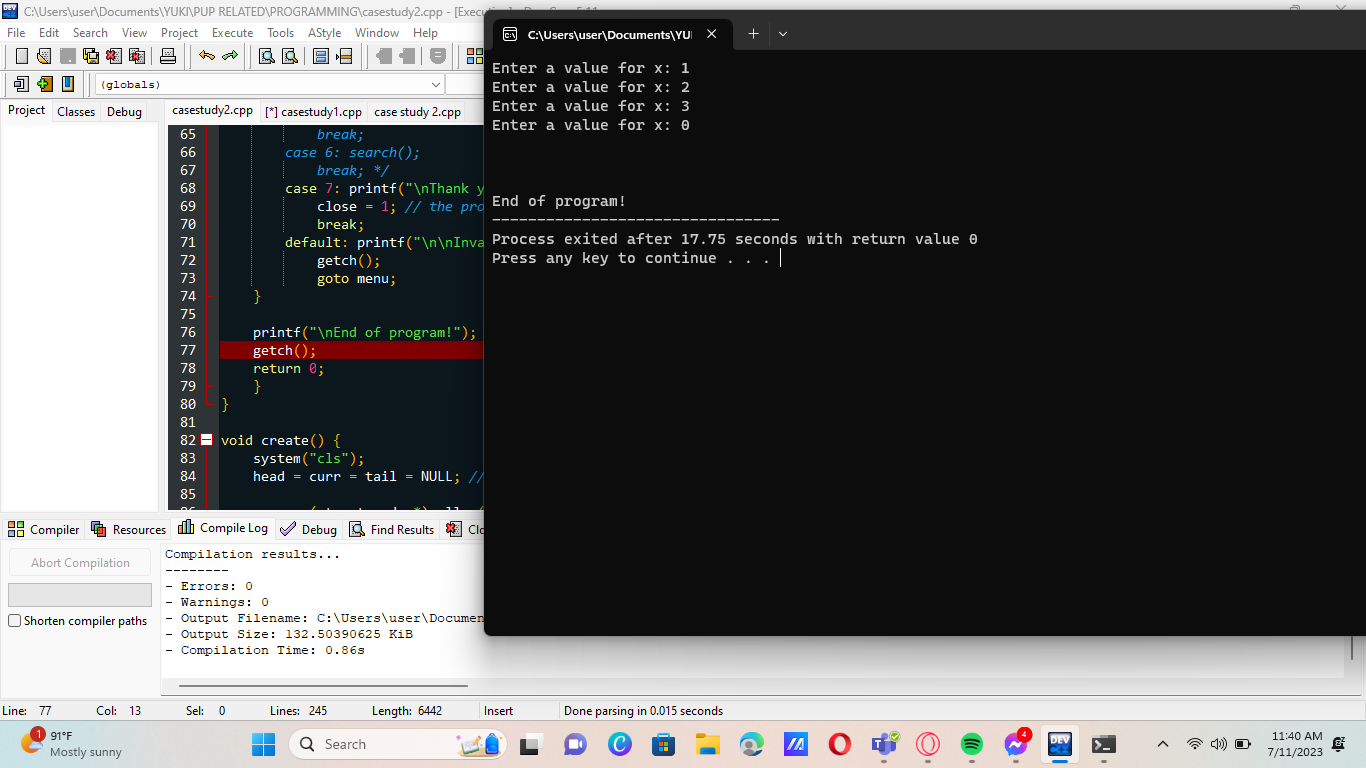
getch();

}

The doubly linked list can be searched for a value using the search function. It asks the user to enter a value to search for, iterates through the list to find every instance of the value, and then it displays where each instance is located.

[ERRORS ENCOUNTERED]

The program is spontaneously running and not going back to main.



The program terminates instantly.