#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

These are the required header files for input/output, dynamic memory allocation, and console input/output operations.

int choice;

Declares an integer variable choice to store the user's menu choice.

// structure definition

struct node {

struct node \*prev;

int x;

struct node \*next;

};

Defines a structure called node that represents a node in the doubly linked list. It has three members: prev (pointer to the previous node), x (an integer value), and next (pointer to the next node).

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

Declares three pointers to node structures: head (points to the first node), curr (used for traversal), and tail (points to the last node).

// function prototyping

void create();

void display();

void disnormal();

void disreverse();

void sort();

void ascend();

void descend();

void insert();

void dele();

void search();

Function prototypes for various functions used in the program. These functions are defined later in the code.

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\033[35m==============================================\033[0m\n");

printf("\t\033[35m| |\033[0m\n");

printf("\t\033[35m| \033[37mMAIN MENU \033[35m|\n");

printf("\t\033[35m| |\033[0m\n");

printf("\t\033[35m|=============================================\033[0m\n");

printf("\t\033[35m| |\033[0m\n");

printf("\t\033[35m| \033[37m1. Create a Doubly Linked List \033[35m|\n");

printf("\t\033[35m| \033[37m2. Display a Doubly Linked List \033[35m|\n");

printf("\t\033[35m| \033[37m3. Sort a Doubly Linked List \033[35m|\n");

printf("\t\033[35m| \033[37m4. Insert a Node \033[35m|\n");

printf("\t\033[35m| \033[37m5. Delete a Node \033[35m|\n");

printf("\t\033[35m| \033[37m6. Search a number in the list \033[35m|\n");

printf("\t\033[35m| \033[37m7. Exit \033[35m|\n");

printf("\t\033[35m| |\033[0m\n");

printf("\t\033[35m==============================================\033[0m\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[ \033[31mInvalid number! Choose 1-7 only. \033[0m]");

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

getch();

goto menu;

}

}

The main function of the program. It contains a while loop that runs until the close variable becomes 1. This loop is responsible for displaying the menu and executing the corresponding functionality based on the user's choice.

printf("\n\t=========< \033[31mEnd of program \033[0m>=========");

getch();

return 0;

}

void create() {

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

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

printf("\n\t\033[36m====================================================\033[0m");

printf("\n\t\033[36m| \033[37mOUTPUT \033[36m|\033[0m");

printf("\n\t\033[36m----------------------------------------------------\033[0m\n\n");

printf("\tEnter a value for x \033[33m [ 0 to terminate ]\033[0m : ");

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 \033[33m [ 0 to terminate ]\033[0m : ");

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

}

printf("\n\t\033[36m----------------------------------------------------\033[0m\n");

printf("\n\t=================<\033[31m End of program \033[0m>=================\n");

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;

}

}

}

The create function is responsible for creating a doubly linked list by taking input from the user. It initializes the head, curr, and tail pointers to NULL.

void display() {

int c;

int exit = 0;

while(exit == 0) {

system("cls");

printf("\n\t\033[32m=============================================================\033[0m");

printf("\n\t\033[32m| \033[37mOUTPUT \033[32m|\033[0m");

choice:

printf("\n\t\033[32m|===========================================================|\n");

printf("\t\033[32m| \033[37mHow do you want your linked list data to be display? \033[32m|\033[0m\n");

printf("\t\033[32m|-----------------------------------------------------------\033[32m|\033[0m\n");

printf("\t\033[32m| \033[32m|\033[0m\n");

printf("\t\033[32m|\033[0m [ 1 ] Normal Order \033[32m|\033[0m\n");

printf("\t\033[32m|\033[0m [ 2 ] Reverse Order \033[32m|\033[0m\n");

printf("\t\033[32m|\033[0m [ 3 ] Exit \033[32m|\033[0m\n");

printf("\t\033[32m| \033[32m|\033[0m\n");

printf("\t\033[32m=============================================================\033[0m\n");

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

scanf("%d", &c);

system("cls");

switch (c) {

case 1:

printf("\n\t\033[32m==============================================");

printf("\n\t\033[32m| \033[0mLinked List Data in Normal Order: \033[32m|");

printf("\n\t\033[32m==============================================\033[0m\n");

if (head == NULL) {

printf("\t\033[32m| |\n");

printf("\t\033[32m| \033[31m[ Empty! ] \033[32m|\n");

printf("\t\033[32m| |\n");

printf("\t\033[32m==============================================\033[0m\n");

printf("\n\t=====< \033[31mCreate a doubly linked list first\033[0m >=====\n");

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

} else {

disnormal();

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

getch();

}

break;

case 2:

printf("\n\t\033[32m==============================================");

printf("\n\t\033[32m| \033[0mLinked List Data in Reverse Order: \033[32m|");

printf("\n\t\033[32m==============================================\033[0m\n");

if (head == NULL) {

printf("\t\033[32m| |\n");

printf("\t\033[32m| \033[31m[ Empty! ] \033[32m|\n");

printf("\t\033[32m| |\n");

printf("\t\033[32m==============================================\033[0m\n");

printf("\n\t=====< \033[31mCreate a doubly linked list first\033[0m >=====\n");

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

} else {

disreverse();

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

getch();

}

break;

case 3:

printf("\n\t========< \033[31mEnd of program \033[37m>========\033[0m\n\n");

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

exit = 1;

break;

default:

printf("\n\n\t[\033[31m Invalid Input! \033[37m]\033[0m\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();

}

void sort() {

int c;

int exit = 0;

while(exit == 0) {

system("cls");

printf("\n\t\033[36m=============================================================\033[0m");

printf("\n\t\033[36m| \033[37mOUTPUT \033[36m|\033[0m");

choice:

printf("\n\t\033[36m|===========================================================|\033[0m\n");

printf("\t\033[36m| \033[37mHow do you want your linked list data to be sorted? \033[36m|\n");

printf("\t\033[36m|-----------------------------------------------------------\033[36m|\n");

printf("\t\033[36m| \033[36m|\n");

printf("\t\033[36m| \033[37m[ 1 ] Ascending Order \033[36m|\n");

printf("\t\033[36m| \033[37m[ 2 ] Descending Order \033[36m|\n");

printf("\t\033[36m| \033[37m[ 3 ] Exit \033[36m|\n");

printf("\t\033[36m| \033[36m|\n");

printf("\033[36m\t=============================================================\033[0m\n");

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

scanf("%d", &c);

system("cls");

switch (c) {

case 1:

printf("\n\t\033[35m===================================================");

printf("\n\t\033[35m| \033[37mLinked List Data in Ascending Order: \033[35m|");

printf("\n\t\033[35m|-------------------------------------------------|\033[0m\n");

if (head == NULL) {

printf("\t\033[35m| |\n");

printf("\t\033[35m| \033[31m[ Empty! ] \033[35m|\033[0m\n");

printf("\t\033[35m| |\n");

printf("\t\033[35m===================================================\033[0m\n");

printf("\n\t===< \033[31mCreate a doubly linked list first\033[0m >====\n");

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

}

else {

ascend();

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

getch();

}

break;

case 2:

printf("\n\t\033[35m===================================================");

printf("\n\t\033[35m| \033[37mLinked List Data in Descending Order: \033[35m|");

printf("\n\t\033[35m|-------------------------------------------------|\033[0m\n");

if (head == NULL) {

printf("\t\033[35m| |\n");

printf("\t\033[35m| \033[31m[ Empty! ] \033[35m|\033[0m\n");

printf("\t\033[35m| |\n");

printf("\t\033[35m===================================================\033[0m\n");

printf("\n\t===< \033[31mCreate a doubly linked list first\033[0m >====\n");

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

}

else {

descend();

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

getch();

}

break;

case 3:

printf("\n\t========< \033[31mEnd of program \033[37m>========\033[0m\n\n");

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

exit = 1;

break;

default:

printf("\n\n\t\033[0m[\033[31m Invalid Input! \033[37m]\033[0m\n");

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

getch();

system("cls");

goto choice;

}

getch();

}

}

The sort function is used to sort the doubly linked list in ascending or descending order.

void insert() {

char deci; // decision

input:

system("cls");

printf("\n\t\033[32m=========================================\033[0m");

printf("\n\t\033[32m| \033[37mOUTPUT \033[32m|\033[0m");

printf("\n\t\033[32m|=======================================|\033[0m\n");

printf("\t\033[32m| \033[37mLinked List Data: \033[32m|\n");

printf("\t\033[32m|=======================================|\033[0m\n");

ascend();

// check if the list is empty

if(head == NULL) {

printf("\t\033[32m| |\n");

printf("\t\033[32m| \033[31m[ Empty! ] \033[32m|\n");

printf("\t\033[32m| |\n");

printf("\t\033[32m=========================================\033[0m\n");

printf("\n\t=====< \033[31mCreate a doubly linked list first\033[0m >=====\n");

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\t\033[33mEnter a value for new node: \033[0m");

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\033[35m============\033[37m[ After Insertion ]\033[35m============\n");

printf("\n\t\033[35m===========================================");

printf("\n\t\033[35m| \033[37mNew Linked List Data: \033[35m|");

printf("\n\t\033[35m-------------------------------------------\033[0m\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\033[32m[\033[31m Invalid Input \033[32m]\n\n\t\033[37mPlease enter Y or N. Press enter to continue\033[0m");

getch();

goto again;

}

getch();

}

The insert function is used to insert a new node into the doubly linked list.

void dele() {

int numdel; // number to be deleted

char d; // decision

int exit = 0;

system("cls");

printf("\n\t\033[32m=========================================\033[0m");

printf("\n\t\033[32m| \033[37mOUTPUT \033[32m|\033[0m");

printf("\n\t\033[32m|=======================================|\033[0m\n");

printf("\t\033[32m| \033[37mLinked List Data: \033[32m|\n");

printf("\t\033[32m|=======================================|\033[0m\n");

ascend();

if (head == NULL) {

printf("\t\033[32m| |\n");

printf("\t\033[32m| \033[31m[ Empty! ] \033[32m|\n");

printf("\t\033[32m| |\n");

printf("\t\033[32m=========================================\033[0m\n");

printf("\n\t=====< \033[31mCreate a doubly linked list first\033[0m >=====\n");

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;

delete(temp);

} else {

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

curr = temp;

temp = temp->next;

}

if (temp == NULL) {

printf("\n\t[\033[31mThe number you enter is not in the list!\033[0m]");

goto choose;

}

curr->next = temp->next;

delete(temp);

}

ulit:

system("cls");

printf("\n\t\033[32m=============\033[37m[ After Deletion ]\033[32m============\n");

printf("\n\t\033[32m===========================================");

printf("\n\t\033[32m| \033[37mNew Linked List Data: \033[32m|");

printf("\n\t\033[32m-------------------------------------------\033[0m");

disnormal();

if (head == NULL) {

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

printf("\t\033[31m| [ Empty! ] \033[31m|\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[ \033[31mInvalid Input \033[0m]\n\n\tPlease enter Y or N. Press enter to continue");

getch();

goto ulit;

}

} while(exit != 1);

getch();

}

The dele function is used to delete a node from the doubly linked 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\033[34m=========================================\033[0m");

printf("\n\t\033[34m| \033[37mOUTPUT \033[34m|\033[0m");

printf("\n\t\033[34m|=======================================|\033[0m\n");

printf("\t\033[34m| \033[37mLinked List Data: \033[34m|\n");

printf("\t\033[34m|=======================================|\033[0m\n");

printf("\t\033[34m| |\n");

printf("\t\033[34m| \033[31m[ Empty! ] \033[34m|\n");

printf("\t\033[34m| |\n");

printf("\t\033[34m=========================================\033[0m\n");

printf("\n\t===< \033[31mCreate a doubly linked list first\033[0m >===\n");

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: \033[0m");

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[\033[32mThere is/are %d occurrences of the number %d.\033[0m]\n", occur, val);

if(found == 0) {

printf("\n\n\t[\033[31m %d is not in the list!\033[0m]\n", val);

}

printf("\n\t\033[34m============\033[37m[ FOR CHECKING ]\033[34m===============\033[0m\n");

printf("\n\t\033[34m===========================================");

printf("\n\t| \033[37mLinked List Data: \033[34m|");

printf("\n\t\033[34m-------------------------------------------\033[0m\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\033[31m[ Invalid Input! ]\033[0m\n\n\tPlease enter Y or N. Press enter to continue...");

getch();

goto isapa;

}

} while (exit != 1);

getch();

}

The search function is used to search for a value in the doubly linked list and display its position and occurrences.

The program also contains other helper functions such as ascend, descend, disnormal, and disreverse, which are used by the main functions to perform specific tasks.