1. Write a c program for linked list.

A screenshot of a computer

Description automatically generated

1. Write a c program for double linked list.

A screenshot of a computer

Description automatically generated

1. Write a c program circular linked list.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Write a c program for array implementation of stack.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Write a c program that implements a stack using linked list.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

6)Array implementation of stack.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

void insert(int arr[], int \*size, int value);

void delete(int arr[], int \*size, int value);

int search(int arr[], int size, int value);

void display(int arr[], int size);

int main() {

int arr[MAX\_SIZE], size = 0;

int operations[][3] = {

{1, 10, 0}, // Insert 10

{1, 20, 0}, // Insert 20

{1, 30, 0}, // Insert 30

{3, 20, 0}, // Search 20

{2, 20, 0}, // Delete 20

{4, 0, 0}, // Display list

{3, 30, 0}, // Search 30

{4, 0, 0} // Display list

};

int num\_operations = sizeof(operations) / sizeof(operations[0]);

for (int i = 0; i < num\_operations; i++) {

int choice = operations[i][0];

int value = operations[i][1];

switch (choice) {

case 1: // Insert

if (size < MAX\_SIZE) {

insert(arr, &size, value);

printf("Inserted %d.\n", value);

} else {

printf("List is full.\n");

}

break;

case 2: // Delete

delete(arr, &size, value);

break;

case 3: // Search

{

int index = search(arr, size, value);

if (index != -1)

printf("Found %d at index %d.\n", value, index);

else

printf("%d not found.\n", value);

}

break;

case 4: // Display

display(arr, size);

break;

default:

printf("Invalid choice.\n");

}

}

return 0;

}

void insert(int arr[], int \*size, int value) {

arr[(\*size)++] = value;

}

void delete(int arr[], int \*size, int value) {

int index = search(arr, \*size, value);

if (index != -1) {

for (int i = index; i < \*size - 1; i++)

arr[i] = arr[i + 1];

(\*size)--;

printf("Deleted %d.\n", value);

} else {

printf("%d not found.\n", value);

}

}

int search(int arr[], int size, int value) {

for (int i = 0; i < size; i++)

if (arr[i] == value)

return i;

return -1;

}

void display(int arr[], int size) {

if (size == 0) {

printf("List is empty.\n");

return;

}

printf("List: ");

for (int i = 0; i < size; i++)

printf("%d ", arr[i]);

printf("\n");

}

A screenshot of a computer program

Description automatically generated\

7)implement of circular doubly linked list using c.

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

struct Node\* prev;

} Node;

void insert(Node\*\* head, int value);

void delete(Node\*\* head, int value);

Node\* search(Node\* head, int value);

void display(Node\* head);

int main() {

Node\* head = NULL;

int operations[][3] = {

{1, 10, 0}, // Insert 10

{1, 20, 0}, // Insert 20

{1, 30, 0}, // Insert 30

{3, 20, 0}, // Search 20

{2, 20, 0}, // Delete 20

{4, 0, 0}, // Display list

{3, 30, 0}, // Search 30

{4, 0, 0} // Display list

};

int num\_operations = sizeof(operations) / sizeof(operations[0]);

for (int i = 0; i < num\_operations; i++) {

int choice = operations[i][0];

int value = operations[i][1];

switch (choice) {

case 1: // Insert

insert(&head, value);

printf("Inserted %d.\n", value);

break;

case 2: // Delete

delete(&head, value);

break;

case 3: // Search

if (search(head, value))

printf("Found %d.\n", value);

else

printf("%d not found.\n", value);

break;

case 4: // Display

display(head);

break;

default:

printf("Invalid choice.\n");

}

}

return 0;

}

void insert(Node\*\* head, int value) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = value;

newNode->next = newNode->prev = newNode;

if (\*head == NULL) {

\*head = newNode;

} else {

Node\* last = (\*head)->prev;

newNode->next = \*head;

newNode->prev = last;

last->next = newNode;

(\*head)->prev = newNode;

}

}

void delete(Node\*\* head, int value) {

if (\*head == NULL) {

printf("List is empty.\n");

return;

}

Node\* temp = \*head;

do {

if (temp->data == value) {

if (temp->next == temp) {

free(temp);

\*head = NULL;

} else {

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

if (\*head == temp) \*head = temp->next;

free(temp);

}

printf("Deleted %d.\n", value);

return;

}

temp = temp->next;

} while (temp != \*head);

printf("%d not found.\n", value);

}

Node\* search(Node\* head, int value) {

if (head == NULL) return NULL;

Node\* temp = head;

do {

if (temp->data == value) return temp;

temp = temp->next;

} while (temp != head);

return NULL;

}

void display(Node\* head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node\* temp = head;

printf("List: ");

do {

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

temp = temp->next;

} while (temp != head);

printf("\n");

}

A screenshot of a computer program

Description automatically generated

8)implement polynomial addition using linked list.

#include <stdio.h>

#include <stdlib.h>

// Node structure for polynomial terms

typedef struct Node {

int coef; // Coefficient

int exp; // Exponent

struct Node\* next;

} Node;

// Function prototypes

Node\* createNode(int coef, int exp);

Node\* insertNode(Node\* head, int coef, int exp);

Node\* addPolynomials(Node\* poly1, Node\* poly2);

void displayPolynomial(Node\* head);

int main() {

// Hardcoded polynomials

Node\* poly1 = NULL;

Node\* poly2 = NULL;

// Polynomial 1: 5x^2 + 4x + 2

poly1 = insertNode(poly1, 5, 2);

poly1 = insertNode(poly1, 4, 1);

poly1 = insertNode(poly1, 2, 0);

// Polynomial 2: 3x^3 + 6x^2 + 1

poly2 = insertNode(poly2, 3, 3);

poly2 = insertNode(poly2, 6, 2);

poly2 = insertNode(poly2, 1, 0);

// Displaying polynomials

printf("Polynomial 1: ");

displayPolynomial(poly1);

printf("Polynomial 2: ");

displayPolynomial(poly2);

// Adding polynomials

Node\* sum = addPolynomials(poly1, poly2);

// Displaying the result

printf("Sum: ");

displayPolynomial(sum);

return 0;

}

// Function to create a new node

Node\* createNode(int coef, int exp) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->coef = coef;

newNode->exp = exp;

newNode->next = NULL;

return newNode;

}

// Function to insert a node in a sorted manner

Node\* insertNode(Node\* head, int coef, int exp) {

Node\* newNode = createNode(coef, exp);

if (head == NULL || head->exp < exp) {

newNode->next = head;

head = newNode;

} else {

Node\* current = head;

while (current->next != NULL && current->next->exp >= exp) {

current = current->next;

}

if (current->exp == exp) {

current->coef += coef;

free(newNode);

} else {

newNode->next = current->next;

current->next = newNode;

}

}

return head;

}

// Function to add two polynomials

Node\* addPolynomials(Node\* poly1, Node\* poly2) {

Node\* result = NULL;

while (poly1 != NULL && poly2 != NULL) {

if (poly1->exp > poly2->exp) {

result = insertNode(result, poly1->coef, poly1->exp);

poly1 = poly1->next;

} else if (poly1->exp < poly2->exp) {

result = insertNode(result, poly2->coef, poly2->exp);

poly2 = poly2->next;

} else {

result = insertNode(result, poly1->coef + poly2->coef, poly1->exp);

poly1 = poly1->next;

poly2 = poly2->next;

}

}

while (poly1 != NULL) {

result = insertNode(result, poly1->coef, poly1->exp);

poly1 = poly1->next;

}

while (poly2 != NULL) {

result = insertNode(result, poly2->coef, poly2->exp);

poly2 = poly2->next;

}

return result;

}

// Function to display a polynomial

void displayPolynomial(Node\* head) {

if (head == NULL) {

printf("0\n");

return;

}

Node\* temp = head;

while (temp != NULL) {

if (temp != head && temp->coef > 0) {

printf(" + ");

}

if (temp->exp == 0) {

printf("%d", temp->coef);

} else if (temp->exp == 1) {

printf("%dx", temp->coef);

} else {

printf("%dx^%d", temp->coef, temp->exp);

}

temp = temp->next;

}

printf("\n");

}

A screenshot of a computer program

Description automatically generated

9)c program to convert infix exp to postfix exp.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

// Define the maximum size for the stack

#define MAX 100

// Stack structure

typedef struct {

int top;

char items[MAX];

} Stack;

// Function prototypes

void initStack(Stack\* s);

int isEmpty(Stack\* s);

void push(Stack\* s, char c);

char pop(Stack\* s);

int precedence(char op);

void infixToPostfix(const char\* infix, char\* postfix);

int main() {

char infix[] = "A+B\*(C^D-E)"; // Hardcoded infix expression

char postfix[MAX];

infixToPostfix(infix, postfix);

printf("Infix expression: %s\n", infix);

printf("Postfix expression: %s\n", postfix);

return 0;

}

// Initialize stack

void initStack(Stack\* s) {

s->top = -1;

}

// Check if stack is empty

int isEmpty(Stack\* s) {

return s->top == -1;

}

// Push an item onto the stack

void push(Stack\* s, char c) {

if (s->top == MAX - 1) {

printf("Stack overflow\n");

exit(1);

}

s->items[++(s->top)] = c;

}

// Pop an item from the stack

char pop(Stack\* s) {

if (isEmpty(s)) {

printf("Stack underflow\n");

exit(1);

}

return s->items[(s->top)--];

}

// Get the precedence of operators

int precedence(char op) {

switch (op) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return 0;

}

}

// Convert infix expression to postfix

void infixToPostfix(const char\* infix, char\* postfix) {

Stack s;

initStack(&s);

int k = 0;

for (int i = 0; infix[i] != '\0'; i++) {

char ch = infix[i];

if (isalnum(ch)) {

postfix[k++] = ch;

} else if (ch == '(') {

push(&s, ch);

} else if (ch == ')') {

while (!isEmpty(&s) && s.items[s.top] != '(') {

postfix[k++] = pop(&s);

}

pop(&s); // Remove '('

} else { // Operator

while (!isEmpty(&s) && precedence(s.items[s.top]) >= precedence(ch)) {

postfix[k++] = pop(&s);

}

push(&s, ch);

}

}

while (!isEmpty(&s)) {

postfix[k++] = pop(&s);

}

postfix[k] = '\0'; // Null terminate the postfix expression

}

A screenshot of a computer program

Description automatically generated

10)c program to evaluate a postfix expression.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX 100

// Stack structure for operands

typedef struct {

int top;

int items[MAX];

} Stack;

// Function prototypes

void initStack(Stack\* s);

int isEmpty(Stack\* s);

void push(Stack\* s, int value);

int pop(Stack\* s);

int evaluatePostfix(const char\* postfix);

int main() {

// Example postfix expression

char postfix[] = "231\*+9-"; // You can change this to test other expressions

// Evaluate the postfix expression

int result = evaluatePostfix(postfix);

printf("Postfix expression: %s\n", postfix);

printf("Evaluation result: %d\n", result);

return 0;

}

// Initialize stack

void initStack(Stack\* s) {

s->top = -1;

}

// Check if stack is empty

int isEmpty(Stack\* s) {

return s->top == -1;

}

// Push an item onto the stack

void push(Stack\* s, int value) {

s->items[++(s->top)] = value;

}

// Pop an item from the stack

int pop(Stack\* s) {

return s->items[(s->top)--];

}

// Evaluate a postfix expression

int evaluatePostfix(const char\* postfix) {

Stack s;

initStack(&s);

for (int i = 0; postfix[i] != '\0'; i++) {

char ch = postfix[i];

if (isdigit(ch)) {

push(&s, ch - '0'); // Convert char to int

} else {

int val2 = pop(&s);

int val1 = pop(&s);

int result;

switch (ch) {

case '+': result = val1 + val2; break;

case '-': result = val1 - val2; break;

case '\*': result = val1 \* val2; break;

case '/': result = val1 / val2; break;

default: printf("Invalid operator\n"); exit(1);

}

push(&s, result);

}

}

return pop(&s); // Final result

}

A screenshot of a computer program

Description automatically generated