**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 01\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#define MAX\_DAYS 7**

**struct Plan {**

**char day[10];**

**int date;**

**char desc[400];**

**};**

**int main() {**

**struct Plan\* p = calloc(MAX\_DAYS, sizeof(struct Plan));**

**if (!p) {**

**printf("Failed to allocate memory.\n");**

**return 1;**

**}**

**for (int i = 0; i < MAX\_DAYS; i++) {**

**printf("Enter name of the day: ");**

**scanf("%s", p[i].day);**

**printf("Enter date of the day: ");**

**scanf("%d", &(p[i].date));**

**printf("Enter description of the activity: ");**

**scanf("%s", p[i].desc);**

**p[i].desc[strcspn(p[i].desc, "\n")] = 0;**

**scanf("%[^\n]", &p[i].desc);**

**}**

**printf("\*\*\*\* Activity description for %d days \*\*\*\*\n", MAX\_DAYS);**

**for (int i = 0; i < MAX\_DAYS; i++) {**

**printf("\nName of the day: %s", p[i].day);**

**printf("\nDate of the day: %d", p[i].date);**

**printf("\nActivity description: %s", p[i].desc);**

**}**

**free(p);**

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 02\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**int strLength(const char \*str)**

**{**

**int len = 0;**

**while (str[len] != '\0')**

**len++;**

**return len;**

**}**

**int findPattern(const char \*str, const char \*pattern)**

**{**

**int i = 0, j = 0;**

**for (i = 0; str[i] != '\0'; i++)**

**{**

**for (j = 0; pattern[j] != '\0' && str[i + j] == pattern[j]; j++)**

**;**

**if (pattern[j] == '\0')**

**return i;**

**}**

**return -1;**

**}**

**void replaceString(char \*str, const char \*pattern, const char \*replace)**

**{**

**int patLen = strLength(pattern);**

**int repLen = strLength(replace);**

**int strLen = strLength(str);**

**int index = findPattern(str, pattern);**

**while (index != -1)**

**{**

**for (int i = 0; i < repLen; i++)**

**str[index + i] = replace[i];**

**strLen = strLength(str);**

**index = findPattern(str, pattern);**

**}**

**if (index == -1)**

**printf("Pattern not found in the main string.\n");**

**}**

**int main()**

**{**

**char mainString[100], pattern[50], replace[50];**

**printf("Enter the main string: \n");**

**scanf(" %s", mainString);**

**printf("Enter the pattern string:\n ");**

**scanf(" %s", pattern);**

**printf("Enter the replace string: \n");**

**scanf(" %s", replace);**

**replaceString(mainString, pattern, replace);**

**printf("Modified String: %s\n", mainString);**

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 03\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdbool.h>**

**#define MAX 100**

**struct Stack {**

**int data[MAX];**

**int top;**

**};**

**void initialize(struct Stack \*stack) {**

**stack->top = -1;**

**}**

**void push(struct Stack \*stack, int value) {**

**if (stack->top == MAX - 1) {**

**printf("Stack Overflow: Cannot push element onto a full stack.\n");**

**} else {**

**stack->data[++stack->top] = value;**

**printf("Pushed %d onto the stack.\n", value);**

**}**

**}**

**int pop(struct Stack \*stack) {**

**if (stack->top == -1) {**

**printf("Stack Underflow: Cannot pop element from an empty stack.\n");**

**return -1;**

**}**

**return stack->data[stack->top--];**

**}**

**bool isPalindrome(const char \*str) {**

**struct Stack stack;**

**initialize(&stack);**

**const char \*start = str;**

**const char \*end = str;**

**while (\*end != '\0') {**

**end++;**

**}**

**end--;**

**while (start <= end) {**

**push(&stack, \*start++);**

**if (start <= end)**

**push(&stack, \*end--);**

**}**

**while (stack.top >= 0) {**

**if (\*str++ != stack.data[stack.top--])**

**return false;**

**}**

**return true;**

**}**

**void displayStack(const struct Stack \*stack) {**

**printf("Stack Contents:\n");**

**for (int i = 0; i <= stack->top; i++)**

**printf("%d\n", stack->data[i]);**

**}**

**int main() {**

**struct Stack stack;**

**initialize(&stack);**

**int choice, element;**

**char str[MAX];**

**while (1) {**

**printf("\nStack Operations Menu:\n");**

**printf("1. Push Element\n");**

**printf("2. Pop Element\n");**

**printf("3. Check Palindrome\n");**

**printf("4. Display Stack\n");**

**printf("5. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**printf("Enter an element to push onto the stack: ");**

**scanf("%d", &element);**

**push(&stack, element);**

**break;**

**case 2:**

**element = pop(&stack);**

**if (element != -1)**

**printf("Popped %d from the stack.\n", element);**

**break;**

**case 3:**

**printf("Enter a string to check if it's a palindrome: ");**

**scanf(" %[^\n]", str);**

**if (isPalindrome(str))**

**printf("The string is a palindrome.\n");**

**else**

**printf("The string is not a palindrome.\n");**

**break;**

**case 4:**

**displayStack(&stack);**

**break;**

**case 5:**

**return 0;**

**default:**

**printf("Invalid choice. Please select a valid option.\n");**

**}**

**}**

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 04\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#define MAX\_STACK\_SIZE 100**

**struct Stack**

**{**

**int top;**

**char items[MAX\_STACK\_SIZE];**

**};**

**void initialize(struct Stack \*stack)**

**{**

**stack->top = -1;**

**}**

**int isEmpty(struct Stack \*stack)**

**{**

**return stack->top == -1;**

**}**

**void push(struct Stack \*stack, char item)**

**{**

**if (stack->top == MAX\_STACK\_SIZE - 1)**

**{**

**printf("Stack overflow!\n");**

**exit(1);**

**}**

**stack->items[++stack->top] = item;**

**}**

**char pop(struct Stack \*stack)**

**{**

**if (isEmpty(stack))**

**{**

**printf("Stack underflow!\n");**

**exit(1);**

**}**

**return stack->items[stack->top--];**

**}**

**int isOperator(char c)**

**{**

**return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '%' || c == '^');**

**}**

**int getPrecedence(char c)**

**{**

**if (c == '^')**

**return 3;**

**if (c == '\*' || c == '/' || c == '%')**

**return 2;**

**if (c == '+' || c == '-')**

**return 1;**

**return 0;**

**}**

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

**{**

**struct Stack stack;**

**initialize(&stack);**

**int i, j;**

**i = j = 0;**

**while (infix[i] != '\0')**

**{**

**if (isalnum(infix[i]))**

**{**

**postfix[j++] = infix[i++];**

**}**

**else if (infix[i] == '(')**

**{**

**push(&stack, infix[i++]);**

**}**

**else if (infix[i] == ')')**

**{**

**while (!isEmpty(&stack) && stack.items[stack.top] != '(')**

**{**

**postfix[j++] = pop(&stack);**

**}**

**if (!isEmpty(&stack) && stack.items[stack.top] == '(')**

**{**

**pop(&stack);**

**}**

**i++;**

**}**

**else if (isOperator(infix[i]))**

**{**

**while (!isEmpty(&stack) && getPrecedence(infix[i]) <=**

**getPrecedence(stack.items[stack.top]))**

**{**

**postfix[j++] = pop(&stack);**

**}**

**push(&stack, infix[i++]);**

**}**

**}**

**while (!isEmpty(&stack))**

**{**

**postfix[j++] = pop(&stack);**

**}**

**postfix[j] = '\0';**

**}**

**int main()**

**{**

**char infix[100];**

**char postfix[100];**

**printf("Enter an infix expression: ");**

**scanf(" %s", infix);**

**infixToPostfix(infix, postfix);**

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

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 05\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#define MAX\_STACK\_SIZE 100**

**struct Stack { int top; int items[MAX\_STACK\_SIZE]; };**

**void initialize(struct Stack \*stack){ stack->top=-1; }**

**int isEmpty(struct Stack \*stack){ return stack->top==-1; }**

**void push(struct Stack \*stack, int item){if(stack->top==MAX\_STACK\_SIZE-1){printf("Stack overflow!\n");exit(1);}stack->items[++stack->top]=item;}**

**int pop(struct Stack \*stack){if(isEmpty(stack)){printf("Stack underflow!\n");exit(1);}return stack->items[stack->top--];}**

**int evaluatePostfix(char \*expression)**

**{**

**struct Stack stack;**

**initialize(&stack);**

**int len = strlen(expression);**

**for (int i = 0; i < len; i++)**

**if (isdigit(expression[i]))**

**push(&stack, expression[i] - '0');**

**else**

**{**

**int operand2 = pop(&stack);**

**int operand1 = pop(&stack);**

**switch (expression[i])**

**{**

**case '+': push(&stack, operand1 + operand2); break;**

**case '-': push(&stack, operand1 - operand2); break;**

**case '\*': push(&stack, operand1 \* operand2); break;**

**case '/': push(&stack, operand1 / operand2); break;**

**case '%': push(&stack, operand1 % operand2); break;**

**}**

**}**

**return stack.items[stack.top];**

**}**

**int main()**

**{**

**char expression[100];**

**printf("Enter a postfix expression: ");**

**scanf(" %[^\n]", expression);**

**int result = evaluatePostfix(expression);**

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

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*5b\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**void hanoi(int n, char S, char T, char D)**

**{**

**if (!n) return;**

**hanoi(n - 1, S, D, T);**

**printf("Move disc %d from %c to %c\n", n, S, D);**

**hanoi(n - 1, T, S, D);**

**}**

**int main()**

**{**

**int n;**

**printf("Enter number of discs: ");**

**scanf("%d", &n);**

**hanoi(n, 'A', 'B', 'C');**

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 06\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdlib.h>**

**#define Max 3**

**void insert(char\*, int\*, int\*);**

**void del(char\*, int\*, int\*);**

**void display(char[], int, int);**

**int main()**

**{**

**char q[Max];**

**int r = -1, f = 0, cnt = 0;**

**while (1)**

**{**

**printf("1: Insert\n2: Delete\n3: Display\n4: Exit\n");**

**printf("Enter choice\n");**

**scanf("%d", &cnt);**

**switch (cnt)**

**{**

**case 1:**

**insert(q, &r, &cnt);**

**break;**

**case 2:**

**del(q, &f, &cnt);**

**break;**

**case 3:**

**display(q, f, cnt);**

**break;**

**default:**

**exit(0);**

**}**

**}**

**}**

**void insert(char\* q, int\* r, int\* cnt)**

**{**

**if (\*cnt == Max)**

**{**

**printf("C Q overflow\n");**

**return;**

**}**

**\*r = (\*r + 1) % Max;**

**printf("enter the ele: ");**

**scanf(" %c", &q[\*r]);**

**\*cnt += 1;**

**}**

**void del(char\* q, int\* f, int\* cnt)**

**{**

**if (\*cnt == 0)**

**{**

**printf("C Queue is empty\n");**

**return;**

**}**

**printf("Element deleted from circular queue is %c\n", q[(\*f)]);**

**(\*f) = (\*f + 1) % Max;**

**\*cnt -= 1;**

**}**

**void display(char q[], int f, int cnt)**

**{**

**if (cnt == 0)**

**{**

**printf("Circular Queue is empty\n");**

**return;**

**}**

**printf("Circular Queue contents are:\n");**

**for (int i = f, c = 0; c < cnt; i = (i + 1) % Max, c++)**

**printf("%d : %c\n", i, q[i]);**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 07\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdlib.h>**

**typedef struct node {**

**char USN[10], name[20], branch[10];**

**int sem;**

**long int ph;**

**struct node \*link;**

**} nd;**

**nd \*insert\_front(nd \*head, nd \*new\_node) {**

**new\_node->link = head;**

**return new\_node;**

**}**

**nd \*insert\_rear(nd \*head, nd \*new\_node) {**

**if (head == NULL) {**

**return new\_node;**

**}**

**nd \*current = head;**

**while (current->link != NULL) {**

**current = current->link;**

**}**

**current->link = new\_node;**

**return head;**

**}**

**nd \*delete\_front(nd \*head) {**

**if (head == NULL) {**

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

**return head;**

**}**

**nd \*temp = head;**

**head = head->link;**

**printf("Deleted USN: %s\n", temp->USN);**

**free(temp);**

**return head;**

**}**

**nd \*delete\_rear(nd \*head) {**

**if (head == NULL) {**

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

**return head;**

**}**

**if (head->link == NULL) {**

**free(head);**

**return NULL;**

**}**

**nd \*current = head;**

**while (current->link->link != NULL) {**

**current = current->link;**

**}**

**nd \*temp = current->link;**

**current->link = NULL;**

**printf("Deleted USN: %s\n", temp->USN);**

**free(temp);**

**return head;**

**}**

**void print\_list(nd \*head) {**

**if (head == NULL) {**

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

**return;**

**}**

**nd \*current = head;**

**while (current != NULL) {**

**printf("USN: %s, Name: %s, Branch: %s, Sem: %d, Phone: %ld\n", current->USN, current->name, current->branch, current->sem, current->ph);**

**current = current->link;**

**}**

**}**

**nd \*create(nd \*head) {**

**int n;**

**printf("Enter value for n\n");**

**scanf("%d", &n);**

**for (int i = 0; i < n; i++) {**

**nd \*new\_node = (nd \*) malloc(sizeof(nd));**

**printf("Enter USN, Name, Branch, Sem and Phone of the student:\n");**

**scanf("%s%s%s%d%ld", (new\_node->USN), (new\_node->name), (new\_node->branch), &(new\_node->sem), &(new\_node->ph));**

**head = insert\_rear(head, new\_node);**

**}**

**return head;**

**}**

**int main() {**

**nd \*first = NULL;**

**int ch;**

**for (;;) {**

**printf("1. Create N students\n2. Status of SLL\n3. Insert front\n4. Insert rear\n5. Delete front\n6. Delete rear\n7. Display\n8. Exit\n");**

**scanf("%d", &ch);**

**switch (ch) {**

**case 1:**

**first = create(first);**

**break;**

**case 2:**

**;**

**break;**

**case 3:**

**;**

**break;**

**case 4:**

**first = insert\_rear(first, (nd \*) malloc(sizeof(nd)));**

**break;**

**case 5:**

**first = delete\_front(first);**

**break;**

**case 6:**

**first = delete\_rear(first);**

**break;**

**case 7:**

**print\_list(first);**

**break;**

**case 8:**

**exit(0);**

**}**

**}**

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 08\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**struct Employee{**

**char SSN[15];**

**char Name[50];**

**char Dept[50];**

**char Designation[50];**

**float Sal;**

**char PhNo[15];**

**struct Employee \*prev;**

**struct Employee \*next;**

**};**

**struct Employee \*createEmployee()**

**{**

**struct Employee \*newEmployee = (struct Employee \*)malloc(sizeof(struct Employee));**

**if (newEmployee == NULL)**

**{**

**printf("Memory allocation failed.\n");**

**exit(1);**

**}**

**printf("Enter Employee Details:\n");**

**printf("SSN: ");**

**scanf(" %[^\n]", newEmployee->SSN);**

**printf("Name: ");**

**scanf(" %[^\n]", newEmployee->Name);**

**printf("Department: ");**

**scanf(" %[^\n]", newEmployee->Dept);**

**printf("Designation: ");**

**scanf(" %[^\n]", newEmployee->Designation);**

**printf("Salary: ");**

**scanf("%f", &newEmployee->Sal);**

**printf("Phone Number: ");**

**scanf(" %[^\n]", newEmployee->PhNo);**

**newEmployee->prev = NULL;**

**newEmployee->next = NULL;**

**return newEmployee;**

**}**

**void displayEmployee(struct Employee \*employee)**

**{**

**printf("SSN: %s\n", employee->SSN);**

**printf("Name: %s\n", employee->Name);**

**printf("Department: %s\n", employee->Dept);**

**printf("Designation: %s\n", employee->Designation);**

**printf("Salary: %.2f\n", employee->Sal);**

**printf("Phone Number: %s\n", employee->PhNo);**

**}**

**struct Employee \*createDLL(struct Employee \*head, int N)**

**{**

**for (int i = 0; i < N; i++)**

**{**

**struct Employee \*newEmployee = createEmployee();**

**if (head == NULL)**

**{**

**head = newEmployee;**

**}**

**else**

**{**

**struct Employee \*current = head;**

**while (current->next != NULL)**

**{**

**current = current->next;**

**}**

**current->next = newEmployee;**

**newEmployee->prev = current;**

**}**

**printf("Employee added to the end of the list.\n");**

**}**

**return head;**

**}**

**void displayDLL(struct Employee \*head)**

**{**

**struct Employee \*current = head;**

**int count = 0;**

**if (current == NULL)**

**{**

**printf("Doubly Linked List is empty.\n");**

**}**

**else**

**{**

**printf("Doubly Linked List Contents:\n");**

**while (current != NULL)**

**{**

**displayEmployee(current);**

**current = current->next;**

**count++;**

**}**

**printf("Total number of nodes: %d\n", count);**

**}**

**}**

**struct Employee \*insertEnd(struct Employee \*head)**

**{**

**struct Employee \*newEmployee = createEmployee();**

**if (head == NULL)**

**{**

**head = newEmployee;**

**}**

**else**

**{**

**struct Employee \*current = head;**

**while (current->next != NULL)**

**{**

**current = current->next;**

**}**

**current->next = newEmployee;**

**newEmployee->prev = current;**

**}**

**printf("Employee added to the end of the list.\n");**

**return head;**

**}**

**struct Employee \*deleteFront(struct Employee \*head)**

**{**

**if (head == NULL)**

**{**

**printf("Doubly Linked List is empty. Nothing to delete.\n");**

**}**

**else**

**{**

**struct Employee \*temp = head;**

**head = head->next;**

**if (head != NULL)**

**{**

**head->prev = NULL;**

**}**

**free(temp);**

**printf("Employee deleted from the front of the list.\n");**

**}**

**return head;**

**}**

**struct Employee \*enqueueFront(struct Employee \*head)**

**{**

**head = insertEnd(head);**

**return head;**

**}**

**struct Employee \*dequeueEnd(struct Employee \*head)**

**{**

**head = deleteFront(head);**

**return head;**

**}**

**int main()**

**{**

**struct Employee \*head = NULL;**

**int choice, N;**

**while (1)**

**{**

**printf("\nDoubly Linked List Operations Menu :\n");**

**printf("1. Create DLL of N Employees Data by End Insertion\n");**

**printf("2. Display DLL and Count Nodes\n");**

**printf("3. Insert Employee at End\n");**

**printf("4. Delete Employee from Front\n");**

**printf("5. Demonstrate DLL as Double - Ended Queue\n");**

**printf("6. Exit\n");**

**printf("Enter your choice : ");**

**scanf("% d", &choice);**

**switch (choice)**

**{**

**case 1:**

**printf("Enter the number of employees(N) : ");**

**scanf("% d", &N);**

**head = createDLL(head, N);**

**break;**

**case 2:**

**displayDLL(head);**

**break;**

**case 3:**

**head = insertEnd(head);**

**break;**

**case 4:**

**head = deleteFront(head);**

**break;**

**case 5:**

**printf("Demonstrating DLL as a Double - Ended Queue :\n");**

**printf("1. Enqueue at Front\n");**

**printf("2. Dequeue at End\n");**

**printf("3. Exit\n");**

**printf("Enter your choice : ");**

**scanf("% d", &choice);**

**switch (choice)**

**{**

**case 1:**

**head = enqueueFront(head);**

**break;**

**case 2:**

**head = dequeueEnd(head);**

**break;**

**case 3:**

**break;**

**default:**

**printf("Invalid choice.Please select a valid option.\n");**

**}**

**break;**

**case 6:**

**exit(0);**

**default:**

**printf("Invalid choice.Please select a valid option.\n");**

**}**

**}**

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 09\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <malloc.h>**

**typedef struct node**

**{**

**int cf, px, py, pz, flag;**

**struct node \*link;**

**} NODE;**

**NODE \*getnode()**

**{**

**NODE \*x = (NODE \*)malloc(sizeof(NODE));**

**if (x == NULL)**

**{**

**printf("Insufficient memory\n");**

**exit(0);**

**}**

**return x;**

**}**

**void display(NODE \*head)**

**{**

**if (head->link == head)**

**{**

**printf("Polynomial does not exist\n");**

**return;**

**}**

**NODE \*temp = head->link;**

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

**while (temp != head)**

**{**

**printf("%dx^%d y^%d z^%d", temp->cf, temp->px, temp->py, temp->pz);**

**if (temp->link != head)**

**printf(" + ");**

**temp = temp->link;**

**}**

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

**}**

**NODE \*insert\_rear(int cf, int x, int y, int z, NODE \*head)**

**{**

**NODE \*temp = getnode();**

**temp->cf = cf;**

**temp->px = x;**

**temp->py = y;**

**temp->pz = z;**

**temp->flag = 0;**

**NODE \*cur = head->link;**

**while (cur->link != head)**

**{**

**cur = cur->link;**

**}**

**cur->link = temp;**

**temp->link = head;**

**return head;**

**}**

**NODE \*read\_poly(NODE \*head)**

**{**

**int cf, px, py, pz;**

**int ch = 1;**

**while (ch)**

**{**

**printf("\nEnter coeff: ");**

**scanf("%d", &cf);**

**printf("\nEnter x, y, z powers(0-indicate NO term): ");**

**scanf("%d%d%d", &px, &py, &pz);**

**head = insert\_rear(cf, px, py, pz, head);**

**printf("\nIf you wish to continue press 1 otherwise 0: ");**

**scanf("%d", &ch);**

**}**

**return head;**

**}**

**NODE \*add\_poly(NODE \*h1, NODE \*h2, NODE \*h3)**

**{**

**NODE \*p1 = h1->link;**

**NODE \*p2 = h2->link;**

**while (p1 != h1)**

**{**

**while (p2 != h2)**

**{**

**if (p1->px == p2->px && p1->py == p2->py && p1->pz == p2->pz)**

**{**

**p1->cf += p2->cf;**

**break;**

**}**

**p2 = p2->link;**

**}**

**h3 = insert\_rear(p1->cf, p1->px, p1->py, p1->pz, h3);**

**p1 = p1->link;**

**}**

**p2 = h2->link;**

**while (p2 != h2)**

**{**

**if (p2->flag == 0)**

**h3 = insert\_rear(p2->cf, p2->px, p2->py, p2->pz, h3);**

**p2 = p2->link;**

**}**

**return h3;**

**}**

**void evaluate(NODE \*he)**

**{**

**NODE \*head = he;**

**int x, y, z;**

**float result = 0;**

**printf("\nEnter x, y, z, terms to evaluate: ");**

**scanf("%d%d%d", &x, &y, &z);**

**he = he->link;**

**while (he != head)**

**{**

**result += he->cf \* pow(x, he->px) \* pow(y, he->py) \* pow(z, he->pz);**

**he = he->link;**

**}**

**printf("\nPolynomial result is: %f", result);**

**}**

**void main()**

**{**

**NODE \*h1, \*h2, \*h3, \*he;**

**int ch;**

**while (1)**

**{**

**printf("\n\n1.Evaluate polynomial\n2.Add two polynomials\n3.Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &ch);**

**switch (ch)**

**{**

**case 1:**

**he = getnode();**

**he->link = he;**

**printf("\nEnter polynomial to evaluate:\n");**

**he = read\_poly(he);**

**display(he);**

**evaluate(he);**

**free(he);**

**break;**

**case 2:**

**h1 = getnode();**

**h2 = getnode();**

**h3 = getnode();**

**h1->link = h1;**

**h2->link = h2;**

**h3->link = h3;**

**printf("\nEnter the first polynomial:");**

**h1 = read\_poly(h1);**

**printf("\nEnter the second polynomial:");**

**h2 = read\_poly(h2);**

**h3 = add\_poly(h1, h2, h3);**

**printf("\nFirst polynomial is: ");**

**display(h1);**

**printf("\nSecond polynomial is: ");**

**display(h2);**

**printf("\nThe sum of 2 polynomials is: ");**

**display(h3);**

**break;**

**case 3:**

**exit(0);**

**break;**

**default:**

**printf("\nInvalid entry");**

**break;**

**}**

**}**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 10\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node**

**{**

**int data;**

**struct Node \*left;**

**struct Node \*right;**

**};**

**struct Node \*createNode(int value)**

**{**

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

**if (newNode == NULL)**

**{**

**printf("Memory allocation failed.\n");**

**exit(1);**

**}**

**newNode->data = value;**

**newNode->left = NULL;**

**newNode->right = NULL;**

**return newNode;**

**}**

**struct Node \*insertNode(struct Node \*root, int value)**

**{**

**if (root == NULL)**

**{**

**return createNode(value);**

**}**

**if (value < root->data)**

**{**

**root->left = insertNode(root->left, value);**

**}**

**else if (value > root->data)**

**{**

**root->right = insertNode(root->right, value);**

**}**

**return root;**

**}**

**void inorderTraversal(struct Node \*root)**

**{**

**if (root != NULL)**

**{**

**inorderTraversal(root->left);**

**printf("%d ", root->data);**

**inorderTraversal(root->right);**

**}**

**}**

**void preorderTraversal(struct Node \*root)**

**{**

**if (root != NULL)**

**{**

**printf("%d ", root->data);**

**preorderTraversal(root->left);**

**preorderTraversal(root->right);**

**}**

**}**

**void postorderTraversal(struct Node \*root)**

**{**

**if (root != NULL)**

**{**

**postorderTraversal(root->left);**

**postorderTraversal(root->right);**

**printf("%d ", root->data);**

**}**

**}**

**int searchBST(struct Node \*root, int key)**

**{**

**if (root == NULL)**

**{**

**return 0;**

**}**

**if (root->data == key)**

**{**

**return 1;**

**}**

**if (key < root->data)**

**{**

**return searchBST(root->left, key);**

**}**

**else**

**{**

**return searchBST(root->right, key);**

**}**

**}**

**int main()**

**{**

**struct Node \*root = NULL;**

**int choice, key;**

**int values[] = {6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2};**

**int numValues = sizeof(values) / sizeof(values[0]);**

**for (int i = 0; i < numValues; i++)**

**{**

**root = insertNode(root, values[i]);**

**}**

**while (1)**

**{**

**printf("\nBinary Search Tree (BST) Operations Menu:\n");**

**printf("1. Inorder Traversal\n");**

**printf("2. Preorder Traversal\n");**

**printf("3. Postorder Traversal\n");**

**printf("4. Search for a Key\n");**

**printf("5. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice)**

**{**

**case 1:**

**printf("Inorder Traversal: ");**

**inorderTraversal(root);**

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

**break;**

**case 2:**

**printf("Preorder Traversal: ");**

**preorderTraversal(root);**

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

**break;**

**case 3:**

**printf("Postorder Traversal: ");**

**postorderTraversal(root);**

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

**break;**

**case 4:**

**printf("Enter the key to search: ");**

**scanf("%d", &key);**

**if (searchBST(root, key))**

**{**

**printf("Key found in the BST.\n");**

**}**

**else**

**{**

**printf("Key not found in the BST.\n");**

**}**

**break;**

**case 5:**

**return 0;**

**default:**

**printf("Invalid choice. Please select a valid option.\n");**

**}**

**}**

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 11\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_CITIES 20**

**struct Graph**

**{**

**int vertices;**

**int adjMatrix[MAX\_CITIES][MAX\_CITIES];**

**};**

**struct Graph createGraph(int N)**

**{**

**struct Graph graph;**

**graph.vertices = N;**

**for (int i = 0; i < N; i++)**

**{**

**for (int j = 0; j < N; j++)**

**{**

**graph.adjMatrix[i][j] = 0;**

**}**

**}**

**printf("Enter connections between cities (0 for no connection, 1 for connection):\n");**

**for (int i = 0; i < N; i++)**

**{**

**for (int j = 0; j < N; j++)**

**{**

**if (i != j)**

**{**

**printf("Is there a connection between city %d and city %d? (1/0): ", i + 1, j + 1);**

**scanf("%d", &graph.adjMatrix[i][j]);**

**}**

**}**

**}**

**return graph;**

**}**

**void DFS(struct Graph graph, int start, int visited[])**

**{**

**visited[start] = 1;**

**printf("City %d is reachable.\n", start + 1);**

**for (int i = 0; i < graph.vertices; i++)**

**{**

**if (graph.adjMatrix[start][i] == 1 && !visited[i])**

**{**

**DFS(graph, i, visited);**

**}**

**}**

**}**

**void printDFS(struct Graph graph, int start)**

**{**

**int visited[MAX\_CITIES] = {0};**

**printf("Nodes reachable from City %d using DFS:\n", start + 1);**

**DFS(graph, start, visited);**

**}**

**void BFS(struct Graph graph, int start, int visited[])**

**{**

**int queue[MAX\_CITIES];**

**int front = -1, rear = -1;**

**visited[start] = 1;**

**queue[++rear] = start;**

**while (front <= rear)**

**{**

**int current = queue[front++];**

**printf("City %d is reachable.\n", current + 1);**

**for (int i = 0; i < graph.vertices; i++)**

**{**

**if (graph.adjMatrix[current][i] == 1 && !visited[i])**

**{**

**visited[i] = 1;**

**queue[++rear] = i;**

**}**

**}**

**}**

**}**

**void printBFS(struct Graph graph, int start)**

**{**

**int visited[MAX\_CITIES] = {0};**

**printf("Nodes reachable from City %d using BFS:\n", start + 1);**

**BFS(graph, start, visited);**

**}**

**int main()**

**{**

**int N, startCity;**

**printf("Enter the number of cities: ");**

**scanf("%d", &N);**

**struct Graph cityGraph = createGraph(N);**

**printf("Enter the starting city to explore: ");**

**scanf("%d", &startCity);**

**if (startCity > 0 && startCity <= N)**

**{**

**printDFS(cityGraph, startCity - 1);**

**printBFS(cityGraph, startCity - 1);**

**}**

**else**

**{**

**printf("Invalid starting city. Please enter a valid city number.\n");**

**}**

**return 0;**

**}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DSA 12\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_MEMORY\_LOCATIONS 100**

**#define MAX\_KEYS 100**

**struct EmployeeRecord**

**{**

**int key;**

**};**

**struct MemoryLocation**

**{**

**int key;**

**struct EmployeeRecord employee;**

**};**

**void initializeHashTable(struct MemoryLocation hashTable[], int m)**

**{**

**for (int i = 0; i < m; i++)**

**{**

**hashTable[i].key = -1;**

**}**

**}**

**void insertEmployee(struct MemoryLocation hashTable[], int m, struct EmployeeRecord employee)**

**{**

**int key = employee.key;**

**int index = key % m;**

**while (hashTable[index].key != -1)**

**{**

**index = (index + 1) % m;**

**}**

**hashTable[index].key = key;**

**hashTable[index].employee = employee;**

**}**

**void displayHashTable(struct MemoryLocation hashTable[], int m)**

**{**

**printf("\nHash Table Contents:\n");**

**for (int i = 0; i < m; i++)**

**{**

**if (hashTable[i].key != -1)**

**{**

**printf("Location %d: Key: %d\n", i, hashTable[i].key);**

**}**

**}**

**}**

**int main()**

**{**

**int m;**

**int n;**

**struct EmployeeRecord employeeRecords[MAX\_KEYS];**

**struct MemoryLocation hashTable[MAX\_MEMORY\_LOCATIONS];**

**printf("Enter the number of memory locations (m) in the hash table: ");**

**scanf("%d", &m);**

**printf("Enter the number of employee records (n): ");**

**scanf("%d", &n);**

**if (m <= 0 || n <= 0)**

**{**

**printf("Invalid input. Please enter valid values for m and n.\n");**

**return 1;**

**}**

**initializeHashTable(hashTable, m);**

**printf("Enter the %d employee records (each record includes a 4-digit key):\n", n);**

**for (int i = 0; i < n; i++)**

**{**

**scanf("%d", &employeeRecords[i].key);**

**insertEmployee(hashTable, m, employeeRecords[i]);**

**}**

**displayHashTable(hashTable, m);**

**return 0;**

**}**