CS102 - Lab 7 - 22/02/2024

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 7
//PROGRAM 1
//Date : 22/02/2024

/*
1.Write a C program to implement the stack using the singly linked list
data structure.
*/


#include <stdio.h>
#include <stdlib.h>

typedef struct node {
  int data;
  struct node *next;
} node;

void push(node **top, int item) {
  node *new_node = (node*) malloc(sizeof(node));
  if (new_node == NULL) {
      printf("Memory allocation error\n");
      return;
  }
  new_node->data = item;
  new_node->next = *top;
  *top = new_node;
  printf("Element %d pushed to the stack\n", item);
}

void pop(node **top) {
  if (*top == NULL) {
      printf("Stack underflow\n");
      return;
  }
```

```
  node *temp = *top;
  *top = (*top)->next;
  int popped_element = temp->data;
  free(temp);
  printf("Popped element: %d\n", popped_element);
}

void peek(node *top) {
  if (top == NULL) {
      printf("Stack is empty\n");
      return;
  }
  printf("Top element: %d\n", top->data);
}

void display(node *top) {
  if (top == NULL) {
      printf("Stack is empty\n");
      return;
  }
  printf("Stack elements: ");
  while (top != NULL) {
      printf("%d  ", top->data);
      top = top->next;
  }
  printf("\n");
}

int main() {
  node *top = NULL;
  int choice, item;

  while (1) {
      printf("\nStack using Linked List\n");
      printf("1. Push  ");
      printf("2. Pop  ");
      printf("3. Peek  ");
      printf("4. Display  ");
      printf("5. Exit\n");
      printf("Enter your choice: ");
```

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

        switch (choice) {
            case 1:
                printf("Enter the element to push: ");
                scanf("%d", &item);
                push(&top, item);
                break;
            case 2:
                pop(&top);
                break;
            case 3:
                peek(top);
                break;
            case 4:
                display(top);
                break;
            case 5:
                printf("Exiting program\n");
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT 1:

```
Stack using Linked List
1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 1
Enter the element to push: 3
Element 3 pushed to the stack

Stack using Linked List
1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 1
Enter the element to push: 4
Element 4 pushed to the stack

Stack using Linked List
1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 1
Enter the element to push: 5
Element 5 pushed to the stack

Stack using Linked List
1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 2
Popped element: 5

Stack using Linked List
1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 3
Top element: 4

Stack using Linked List
1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 4
Stack elements: 4  3

Stack using Linked List
1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice:
```

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 7
//PROGRAM 2
//Date : 22/02/2024


/*
2.Write a C program to implement the queue using the singly
linked list data structure.
*/

#include <stdio.h>
#include <stdlib.h>

typedef struct node {
   int data;
   struct node *next;
} node;


typedef struct queue{
   node *front;
   node *rear;
} queue;


void create_Q(queue *q) {
   q->front = q->rear = NULL;
}


void enqueue(queue *q, int item) {
```

```
    node *new_node = (node*) malloc(sizeof(node));
    if (new_node == NULL) {
        printf("Memory allocation error\n");
        return;
    }
    new_node->data = item;
    new_node->next = NULL;


    if (q->rear == NULL) {
        q->front = q->rear = new_node;
    } else {
        q->rear->next = new_node;
        q->rear = new_node;
    }
    printf("Enqueued element: %d\n", item);
}


void dequeue(queue *q) {
    if (q->front == NULL) {

        printf("Queue is empty\n");
        return;
    }

    node *temp = q->front;
    int del_ele = temp->data;

    if (q->front == q->rear) {
        q->front = q->rear = NULL;
    } else {
        q->front = q->front->next;
    }
```

```
    free(temp);
    printf("Dequeued element: %d\n", del_ele);
}


void display(queue q) {
    if (q.front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    node *p = q.front;
    while (p != NULL) {
        printf("%d  ", p->data);
        p = p->next;
    }
    printf("\n");
}


int main() {
    queue q;
    create_Q(&q);
    int choice, item;
    while (1) {
        printf("\nQueue using Linked List\n");
        printf("1. Enqueue  ");
        printf("2. Dequeue  ");
        printf("3. Display  ");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter the element to enqueue: ");
                scanf("%d", &item);
                enqueue(&q, item);
                break;
            case 2:
                dequeue(&q);
                break;
            case 3:
                display(q);
                break;
            case 4:
                printf("Exiting program\n");
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }


    return 0;
}
```

OUTPUT 2:

```
Queue using Linked List
1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 1
Enter the element to enqueue: 6
Enqueued element: 6

Queue using Linked List
1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 1
Enter the element to enqueue: 8
Enqueued element: 8

Queue using Linked List
1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 1
Enter the element to enqueue: 9
Enqueued element: 9

Queue using Linked List
1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 2
Dequeued element: 6

Queue using Linked List
1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 3
Queue elements: 8  9

Queue using Linked List
1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice:
```

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 7
//PROGRAM 3
//Date: 22/02/2024

/*
3.Write a C program to convert a given infix expression to
its postfix equivalent using the stack.
Implement the stack using the singly linked list data
structure.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX 100

typedef struct Node {
    char data;
    struct Node* next;
} Node;

Node* top = NULL;

void push(char item) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = item;
    newNode->next = top;
    top = newNode;
}
```

```
char pop() {
    if (top == NULL) {
        printf("Stack Underflow.");
        exit(1);
    } else {
        char item = top->data;
        Node* temp = top;
        top = top->next;
        free(temp);
        return item;
    }
}

int is_operator(char symbol) {
    return (symbol == '^' || symbol == '*' || symbol == '/'
|| symbol == '+' || symbol == '-');
}

int precedence(char symbol) {
    if (symbol == '^') {
        return 3;
    } else if (symbol == '*' || symbol == '/') {
        return 2;
    } else if (symbol == '+' || symbol == '-') {
        return 1;
    } else {
        return 0;
    }
}

void infixToPostfix(char infix_exp[], char postfix_exp[]) {
    int i, j;
    char item;
```

```
    char x;
    push('(');
    strcat(infix_exp, ")");
    i = 0;
    j = 0;
    item = infix_exp[i];

    while (item != '\0') {
        if (item == '(') {
            push(item);
        } else if (isdigit(item) || isalpha(item)) {
            postfix_exp[j] = item;
            j++;
        } else if (is_operator(item) == 1) {
            x = pop();
            while (is_operator(x) == 1 && precedence(x) >=
precedence(item)) {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
            push(x);
            push(item);
        } else if (item == ')') {
            x = pop();
            while (x != '(') {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
        } else {
            printf("\nInvalid infix Expression.\n");
            exit(1);
        }
```

```
        i++;
        item = infix_exp[i];
    }
    postfix_exp[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];
    int choice;

    do {
        printf("\nMenu\n");
        printf("1. Enter Infix expression and convert to
Postfix\n");
        printf("2. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter Infix expression : ");
                scanf(" %[^\n]", infix);
                infixToPostfix(infix, postfix);
                printf("Postfix Expression: %s\n", postfix);
                break;
            case 2:
                printf("Exiting program\n");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 2);

    return 0;
}
```

OUTPUT3:

```
Menu
1. Enter Infix expression and convert to Postfix
2. Exit
Enter your choice: 1
Enter Infix expression : a*b+c*(d-e)
Postfix Expression: ab*cde-*+

Menu
1. Enter Infix expression and convert to Postfix
2. Exit
Enter your choice: 1
Enter Infix expression : a-b*(c/d-e/f)
Postfix Expression: abcd/ef/-*-

Menu
1. Enter Infix expression and convert to Postfix
2. Exit
Enter your choice: 2
Exiting program
```

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 7
//PROGRAM 4
//Date : 22/02/2024

/*
4.Write a C program to create a doubly linked list of integers and perform
the following operations:
• Insert a node at the beginning
• Insert a node at the end
• Insert a node after a given position
• Delete a node at the beginning
• Delete a node at the end
• Delete a node at a given position
• Display the elements of the list in forward and reverse order.
*/


#include <stdio.h>
#include <stdlib.h>

typedef struct node {
   int data;
   struct node *prev;
   struct node *next;
} node;

void traverse(node *head) {
   int count_f = 1;
   int count_r = 1;
   node *p_f = head;
   node *p_r = head;

   printf("\nDoubly Linked List (Forward):\n");
   while (p_f != NULL) {
       printf("  Node_%d: %d\n", count_f, p_f->data);
       count_f++;
```

```
        p_f = p_f->next;
    }

    printf("\nDoubly Linked List (Reverse):\n");
    while (p_r != NULL && p_r->next != NULL) {
        p_r = p_r->next;
    }

    while (p_r != NULL) {
        printf("  Node_%d: %d\n", count_r, p_r->data);
        count_r++;
        p_r = p_r->prev;
    }
}

node* create_node(int item) {
    node *new_node = (node*)malloc(sizeof(node));
    if (new_node == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    new_node->data = item;
    new_node->prev = NULL;
    new_node->next = NULL;
    return new_node;
}

void insert_begin(node **head, int item) {
    node *new_node = create_node(item);
    new_node->next = *head;

    if (*head != NULL) {
        (*head)->prev = new_node;
    }

    *head = new_node;
}

void insert_end(node **head, int item) {
    node *new_node = create_node(item);
```

```
    if (*head == NULL) {
        *head = new_node;
        return;
    }

    node *p = *head;
    while (p->next != NULL) {
        p = p->next;
    }

    p->next = new_node;
    new_node->prev = p;
}

void insert_any_position(node **head, int item, int pos) {
    if (pos < 1) {
        printf("Invalid position\n");
        return;
    }

    node *new_node = create_node(item);
     if (pos == 1) {
        new_node->next = *head;

        if (*head != NULL) {
            (*head)->prev = new_node;
        }

        *head = new_node;
        return;
    }

    node *p = *head;
    for (int i = 1; i < pos - 1 && p != NULL; i++) {
        p = p->next;
    }

    if (p == NULL) {
        printf("Position out of range\n");
```

```
        free(new_node);
        return;
    }

    new_node->next = p->next;
    new_node->prev = p;

    if (p->next != NULL) {
        p->next->prev = new_node;
    }

    p->next = new_node;
}

void delete_begin(node **head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    node *temp = *head;
    *head = (*head)->next;

    if (*head != NULL) {
        (*head)->prev = NULL;
    }

    free(temp);
}

void delete_end(node **head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    node *p = *head, *q = NULL;
    while (p->next != NULL) {
        q = p;
        p = p->next;
```

```
    }

    if (q != NULL) {
        q->next = NULL;
    } else {
        *head = NULL;
    }

    free(p);
}

void delete_any_pos(node **head, int pos) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    node *p = *head, *q = NULL;
    if (pos == 1) {
        *head = p->next;

        if (*head != NULL) {
            (*head)->prev = NULL;
        }

        free(p);
        return;
    }

    for (int i = 1; i < pos && p != NULL; i++) {
        q = p;
        p = p->next;
    }

    if (p == NULL) {
        printf("Position out of range\n");
        return;
    }

    q->next = p->next;
```

```
    if (p->next != NULL) {
        p->next->prev = q;
    }

    free(p);
}

int display_menu() {
    int choice;

    printf("\nDoubly Linked List:\n");
    printf("1. Insert at beginning   ");
    printf("2. Insert at end ");
    printf("3. Insert at any position\n");
    printf("4. Delete from beginning ");
    printf("5. Delete from end ");
    printf("6. Delete from any position\n");
    printf("7. Traverse ");
    printf("8. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    return choice;
}

int main() {
    node *head = NULL;
    int choice, item, pos;

    while(1) {
        choice = display_menu();

        switch(choice) {
            case 1:
                printf("Enter the item to be inserted at the beginning: ");
                scanf("%d", &item);
                insert_begin(&head, item);
                break;
            case 2:
```

```
                printf("Enter the item to be inserted at the end: ");
                scanf("%d", &item);
                insert_end(&head, item);
                break;
            case 3:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                printf("Enter the position at which to insert: ");
                scanf("%d", &pos);
                insert_any_position(&head, item, pos);
                break;
            case 4:
                delete_begin(&head);
                break;
            case 5:
                delete_end(&head);
                break;
            case 6:
                printf("Enter the position at which to delete: ");
                scanf("%d", &pos);
                delete_any_pos(&head, pos);
                break;
            case 7:
                traverse(head);
                break;
            case 8:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT 4:

```
Doubly Linked List:
1. Insert at beginning   2. Insert at end 3. Insert at any position
4. Delete from beginning 5. Delete from end 6. Delete from any position
7. Traverse 8. Exit
Enter your choice: 1
Enter the item to be inserted at the beginning: 3

Doubly Linked List:
1. Insert at beginning   2. Insert at end 3. Insert at any position
4. Delete from beginning 5. Delete from end 6. Delete from any position
7. Traverse 8. Exit
Enter your choice: 1
Enter the item to be inserted at the beginning: 4

Doubly Linked List:
1. Insert at beginning   2. Insert at end 3. Insert at any position
4. Delete from beginning 5. Delete from end 6. Delete from any position
7. Traverse 8. Exit
Enter your choice: 7

Doubly Linked List (Forward):
  Node_1: 4
  Node_2: 3

Doubly Linked List (Reverse):
  Node_1: 3
  Node_2: 4

Doubly Linked List:
1. Insert at beginning   2. Insert at end 3. Insert at any position
4. Delete from beginning 5. Delete from end 6. Delete from any position
7. Traverse 8. Exit
Enter your choice: 4

Doubly Linked List:
1. Insert at beginning   2. Insert at end 3. Insert at any position
4. Delete from beginning 5. Delete from end 6. Delete from any position
7. Traverse 8. Exit
Enter your choice: 7

Doubly Linked List (Forward):
  Node_1: 3

Doubly Linked List (Reverse):
  Node_1: 3
```