Aryaman Sodha

Competitive coding

IEECS

LEVEL 0:

CODE:

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

    int data;

    struct Node\* next;

    struct Node\* prev;

} Node;

typedef struct {

    Node\* head;

} DoublyLinkedList;

DoublyLinkedList\* create\_list() {

    DoublyLinkedList\* dll = (DoublyLinkedList\*)malloc(sizeof(DoublyLinkedList));

    dll->head = NULL;

    return dll;

}

void insert\_at\_head(DoublyLinkedList\* dll, int data) {

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

    new\_node->data = data;

    new\_node->next = dll->head;

    new\_node->prev = NULL;

    if (dll->head)

        dll->head->prev = new\_node;

    dll->head = new\_node;

}

void insert\_at\_tail(DoublyLinkedList\* dll, int data) {

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

    new\_node->data = data;

    new\_node->next = NULL;

    if (!dll->head) {

        new\_node->prev = NULL;

        dll->head = new\_node;

        return;

    }

    Node\* temp = dll->head;

    while (temp->next)

        temp = temp->next;

    temp->next = new\_node;

    new\_node->prev = temp;

}

void traverse\_forward(DoublyLinkedList\* dll) {

    Node\* temp = dll->head;

    while (temp) {

        printf("%d ", temp->data);

        temp = temp->next;

    }

    printf("\n");

}

void traverse\_backward(DoublyLinkedList\* dll) {

    Node\* temp = dll->head;

    if (!temp)

        return;

    while (temp->next)

        temp = temp->next;

    while (temp) {

        printf("%d ", temp->data);

        temp = temp->prev;

    }

    printf("\n");

}

int main() {

    DoublyLinkedList\* dll = create\_list();

    int choice, data;

    while (1) {

        printf("\n1. Insert at Head\n2. Insert at Tail\n3. Traverse Forward\n4. Traverse Backward\n5. Exit\nEnter choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to insert at head: ");

                scanf("%d", &data);

                insert\_at\_head(dll, data);

                break;

            case 2:

                printf("Enter value to insert at tail: ");

                scanf("%d", &data);

                insert\_at\_tail(dll, data);

                break;

            case 3:

                printf("Forward Traversal: ");

                traverse\_forward(dll);

                break;

            case 4:

                printf("Backward Traversal: ");

                traverse\_backward(dll);

                break;

            case 5:

                exit(0);

            default:

                printf("Invalid choice!\n");

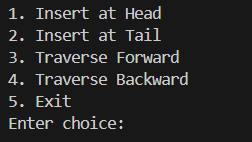
        }

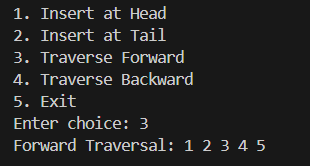
    }

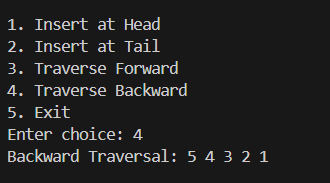
    return 0;

}

SAMPLE I/P and O/P :







Time complexities of various operations :

|  |  |
| --- | --- |
| Insert at Head | * O(1) |

|  |  |
| --- | --- |
| Insert at Tail | * O(n) |

|  |  |
| --- | --- |
| Traverse Forward | * O(n) |

|  |  |
| --- | --- |
| Traverse Backward | * O(n) |

Space complexities :

|  |  |
| --- | --- |
| Insert at Head | * O(1) |

|  |  |
| --- | --- |
| Insert at Tail | * O(1) |

|  |  |
| --- | --- |
| Traverse Forward | * O(1) |

|  |  |
| --- | --- |
| Traverse Backward | * O(1) |

LEVEL = 1 :

CODE :

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

typedef struct Node {

    int data;

    int min;

    int max;

    struct Node\* next;

} Node;

typedef struct {

    Node\* top;

} Stack;

void initStack(Stack\* s) {

    s->top = NULL;

}

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

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

    if (!newNode) {

        printf("Stack Overflow\n");

        return;

    }

    newNode->data = x;

    newNode->min = (s->top == NULL) ? x : (x < s->top->min ? x : s->top->min);

    newNode->max = (s->top == NULL) ? x : (x > s->top->max ? x : s->top->max);

    newNode->next = s->top;

    s->top = newNode;

}

void pop(Stack\* s) {

    if (s->top == NULL) {

        printf("Stack Underflow\n");

        return;

    }

    Node\* temp = s->top;

    s->top = s->top->next;

    free(temp);

}

int top(Stack\* s) {

    return (s->top == NULL) ? INT\_MIN : s->top->data;

}

int getMin(Stack\* s) {

    return (s->top == NULL) ? INT\_MIN : s->top->min;

}

int getMax(Stack\* s) {

    return (s->top == NULL) ? INT\_MIN : s->top->max;

}

int main() {

    Stack s;

    initStack(&s);

    int choice, value;

    while (1) {

        printf("\n1. Push\n2. Pop\n3. Top\n4. Get Min\n5. Get Max\n6. Exit\nEnter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to push: ");

                scanf("%d", &value);

                push(&s, value);

                break;

            case 2:

                pop(&s);

                break;

            case 3:

                printf("Top element: %d\n", top(&s));

                break;

            case 4:

                printf("Minimum element: %d\n", getMin(&s));

                break;

            case 5:

                printf("Maximum element: %d\n", getMax(&s));

                break;

            case 6:

                exit(0);

            default:

                printf("Invalid choice!\n");

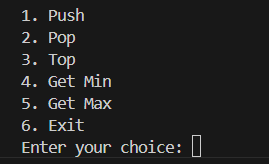
        }

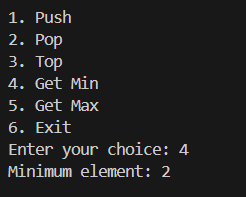
    }

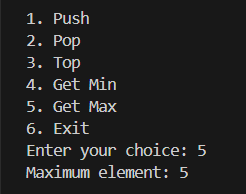
    return 0;

}

SAMPLE I/P and O/P :







Time complexity:

|  |  |  |
| --- | --- | --- |
| push(x) | * **O(1)** |  |

|  |  |
| --- | --- |
| pop() | * **O(1)** |

|  |  |
| --- | --- |
| top() | * **O(1)** |

|  |  |
| --- | --- |
| getMin() | * **O(1)** |

|  |  |
| --- | --- |
| getMax() | * **O(1)** |

Overall space complexity:

* **O(n)**