PRACTICAL TEST 2024 – BCS/16/21/005/TZ

Github Username: Abdillah-Ali

Github repository Link: **https://github.com/Abdillah-Ali/Practical_Test_2024.git**

1. SECTION A: Stack

a) Write a program to reverse a list of given numbers in stack

CODE

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

struct Stack {
    int data[MAX];
    int top;
};

void initStack(struct Stack *s) {
    s->top = -1;
}

int isFull(struct Stack *s) {
    return s->top == MAX - 1;
}

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

```
void push(struct Stack *s, int value) {
    if (!isFull(s)) {
        s->data[++(s->top)] = value;
    }
}

int pop(struct Stack *s) {
    if (!isEmpty(s)) {
        return s->data[(s->top)--];
    }
    return -1;
}

void reverseList(int *list, int n) {
    struct Stack s;
    initStack(&s);
    for (int i = 0; i < n; i++) {
        push(&s, list[i]);
    }
    for (int i = 0; i < n; i++) {
        list[i] = pop(&s);
    }
}

int main() {
    int list[] = {45, 67, 89, 123, 456};
    int n = sizeof(list) / sizeof(list[0]);
    printf("Original list: ");
    for (int i = 0; i < n; i++) {
```

```
        printf("%d ", list[i]);
    }
    printf("\n");
    reverseList(list, n);
    printf("Reversed list: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", list[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT

```
Original list: 45 67 89 123 456
Reversed list: 456 123 89 67 45
```

b) Write a program to check nesting of parentheses using a stack.

CODE

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

#define MAX 100

typedef struct Stack {
    char data[MAX];
    int top;
} Stack;

void initStack(Stack *s) {
    s->top = -1;
```

```
}

int isFull(Stack *s) {
    return s->top == MAX - 1;
}

int isEmpty(Stack *s) {
    return s->top == -1;
}

void push(Stack *s, char ch) {
    if (!isFull(s)) {
        s->data[++(s->top)] = ch;
    }
}

char pop(Stack *s) {
    if (!isEmpty(s)) {
        return s->data[(s->top)--];
    }
    return '\0';
}

int checkParentheses(const char *expr) {
    Stack s;
    initStack(&s);

    for (int i = 0; expr[i] != '\0'; i++) {
        if (expr[i] == '(' || expr[i] == '{' || expr[i] == '[') {
            push(&s, expr[i]);
```

```
        } else if (expr[i] == ')' || expr[i] == '}' || expr[i] == ']') {
            if (isEmpty(&s)) return 0;

            char top = pop(&s);
            if ((expr[i] == ')' && top != '(') ||
                (expr[i] == '}' && top != '{') ||
                (expr[i] == ']' && top != '[')) {
                return 0;
            }
        }
    }

    return isEmpty(&s);
}

int main() {
    char expr[100];
    printf("Enter an expression: ");
    scanf("%s", expr);

    if (checkParentheses(expr)) {
        printf("The parentheses are balanced.\n");
    } else {
        printf("The parentheses are not balanced.\n");
    }

    return 0;
}
```

OUTPUT

```
Enter an expression: (45 + 67) * {123 / [456 - 78]}
The parentheses are not balanced.
```

c) Write a program to implement a stack that stores names and age of students in the class.

CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100

typedef struct Student {
    char name[50];
    int age;
} Student;

typedef struct Stack {
    Student data[MAX];
    int top;
} Stack;

void initStack(Stack *s) {
    s->top = -1;
}

int isFull(Stack *s) {
    return s->top == MAX - 1;
}

int isEmpty(Stack *s) {
    return s->top == -1;
```

```
}

void push(Stack *s, Student student) {
    if (!isFull(s)) {
        s->data[++(s->top)] = student;
    }
}

Student pop(Stack *s) {
    if (!isEmpty(s)) {
        return s->data[(s->top)--];
    }
    Student empty = {"", -1};
    return empty;
}

int main() {
    Stack s;
    initStack(&s);

    Student students[] = {{"Alice", 13}, {"Bob", 17}, {"Charlie", 22}};
    int n = sizeof(students) / sizeof(students[0]);

    for (int i = 0; i < n; i++) {
        push(&s, students[i]);
    }

    printf("Students popped from stack:\n");
    while (!isEmpty(&s)) {
        Student student = pop(&s);
```

```
        printf("Name: %s, Age: %d\n", student.name, student.age);
    }

    return 0;
}
```

OUTPUT

```
Students popped from stack:
Name: Charlie, Age: 22
Name: Bob, Age: 17
Name: Alice, Age: 13
```

d) Write a program to input two stacks and compare their contents

CODE

```
 #include <stdio.h>
#include <stdlib.h>
#define MAX 100

typedef struct Stack {
    int data[MAX];
    int top;
} Stack;

void initStack(Stack *s) {
    s->top = -1;
}

int isFull(Stack *s) {
```

```
    return s->top == MAX - 1;
}

int isEmpty(Stack *s) {
    return s->top == -1;
}

void push(Stack *s, int value) {
    if (!isFull(s)) {
        s->data[++(s->top)] = value;
    }
}

int pop(Stack *s) {
    if (!isEmpty(s)) {
        return s->data[(s->top)--];
    }
    return -1;
}

int compareStacks(Stack *s1, Stack *s2) {
    if (s1->top != s2->top) {
        return 0;
    }

    for (int i = 0; i <= s1->top; i++) {
        if (s1->data[i] != s2->data[i]) {
            return 0;
        }
    }
```

```
    return 1;
}

int main() {
    Stack s1, s2;
    initStack(&s1);
    initStack(&s2);

    int n1, n2, value;

    printf("Enter the number of elements in stack 1: ");
    scanf("%d", &n1);
    printf("Enter the elements of stack 1:\n");
    for (int i = 0; i < n1; i++) {
        scanf("%d", &value);
        push(&s1, value);
    }

    printf("Enter the number of elements in stack 2: ");
    scanf("%d", &n2);
    printf("Enter the elements of stack 2:\n");
    for (int i = 0; i < n2; i++) {
        scanf("%d", &value);
        push(&s2, value);
    }

    if (compareStacks(&s1, &s2)) {
        printf("The stacks are identical.\n");
    } else {
```

```
        printf("The stacks are different.\n");
    }

    return 0;
}
```

OUTPUT

```
Enter the number of elements in stack 1: 2
Enter the elements of stack 1:
2
2
Enter the number of elements in stack 2: 2
Enter the elements of stack 2:
2
3
The stacks are different.
```

2. SECTION B: Queue

a) Given a Queue Q,write a method that will find the max element in the queue. You may only use queue operations such as eneque, dequeue, size etc.. No other data structure can be used other than queues. Queue must remain intact after finding the max.

CODE

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

#define MAX 100

typedef struct Queue {
    int data[MAX];
    int front, rear, size;
} Queue;
```

```
void initQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

int isFull(Queue *q) {
    return q->size == MAX;
}

int isEmpty(Queue *q) {
    return q->size == 0;
}

void enqueue(Queue *q, int value) {
    if (!isFull(q)) {
        q->rear = (q->rear + 1) % MAX;
        q->data[q->rear] = value;
        q->size++;
    }
}

int dequeue(Queue *q) {
    if (!isEmpty(q)) {
        int value = q->data[q->front];
        q->front = (q->front + 1) % MAX;
        q->size--;
        return value;
    }
    return -1;
```

```
}

int findMax(Queue *q) {
    int max = -1;
    int n = q->size;

    for (int i = 0; i < n; i++) {
        int value = dequeue(q);
        if (value > max) max = value;
        enqueue(q, value);  // Re-enqueue to preserve the queue
    }

    return max;
}

int main() {
    Queue q;
    initQueue(&q);

    enqueue(&q, 17);
    enqueue(&q, 2);
    enqueue(&q, 5);
    enqueue(&q, 3);

    printf("Max element in the queue: %d\n", findMax(&q));

    return 0;
}
```

OUTPUT

```
Max element in the queue: 17
```

b) Write a program to implement a dequeue with the help of a linked list.

CODE

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

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

typedef struct Deque {
    Node *front;
    Node *rear;
} Deque;

void initDeque(Deque *dq) {
    dq->front = dq->rear = NULL;
}

int isEmpty(Deque *dq) {
    return dq->front == NULL;
}

void enqueueFront(Deque *dq, int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = dq->front;
```

```
    newNode->prev = NULL;

    if (isEmpty(dq)) {
        dq->rear = newNode;
    } else {
        dq->front->prev = newNode;
    }

    dq->front = newNode;
}

void enqueueRear(Deque *dq, int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
    newNode->prev = dq->rear;

    if (isEmpty(dq)) {
        dq->front = newNode;
    } else {
        dq->rear->next = newNode;
    }

    dq->rear = newNode;
}

int dequeueFront(Deque *dq) {
    if (isEmpty(dq)) return -1;

    Node *temp = dq->front;
```

```
    int value = temp->data;

    dq->front = dq->front->next;
    if (dq->front) {
        dq->front->prev = NULL;
    } else {
        dq->rear = NULL;
    }

    free(temp);
    return value;
}

int dequeueRear(Deque *dq) {
    if (isEmpty(dq)) return -1;

    Node *temp = dq->rear;
    int value = temp->data;

    dq->rear = dq->rear->prev;
    if (dq->rear) {
        dq->rear->next = NULL;
    } else {
        dq->front = NULL;
    }

    free(temp);
    return value;
}
```

```
int main() {
    Deque dq;
    initDeque(&dq);

    enqueueRear(&dq, 10);
    enqueueRear(&dq, 20);
    enqueueFront(&dq, 5);

    printf("Dequeue front: %d\n", dequeueFront(&dq));
    printf("Dequeue rear: %d\n", dequeueRear(&dq));

    return 0;
}
```

OUTPUT

```
Dequeue front: 5
Dequeue rear: 20
```

3. SECTION C: Linked List

a) Write a program that removes all nodes that have duplicate information.

CODE

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

typedef struct Node {
    int data;
    struct Node *next;
} Node;
```

```
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void removeDuplicates(Node** head) {
    Node *current = *head, *prev = NULL, *temp;
    while (current != NULL) {
        Node *runner = current->next;
        Node *prevRunner = current;
        while (runner != NULL) {
            if (runner->data == current->data) {
                temp = runner;
                prevRunner->next = runner->next;
                runner = runner->next;
                free(temp);
            } else {
                prevRunner = runner;
                runner = runner->next;
            }
        }
        prev = current;
        current = current->next;
    }
}

void printList(Node *head) {
    while (head != NULL) {
```

```
        printf("%d , ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    Node *head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(2);
    head->next->next->next = createNode(3);
    head->next->next->next->next = createNode(3);

    printf("Original list:\n");
    printList(head);

    removeDuplicates(&head);

    printf("List after removing duplicates:\n");
    printList(head);

    return 0;
}
```

OUTPUT

```
Original list:
1 , 2 , 2 , 3 , 3 , NULL
List after removing duplicates:
1 , 2 , 3 , NULL
```

b) Writeaprogramtomergetwolinked lists.

CODE

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

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

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

Node* mergeLists(Node* l1, Node* l2) {
    if (!l1) return l2;
    if (!l2) return l1;

    if (l1->data < l2->data) {
        l1->next = mergeLists(l1->next, l2);
        return l1;
    } else {
        l2->next = mergeLists(l1, l2->next);
        return l2;
    }
}
```

```
void printList(Node *head) {
    while (head != NULL) {
        printf("%d , ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    Node *l1 = createNode(1);
    l1->next = createNode(3);
    l1->next->next = createNode(5);

    Node *l2 = createNode(2);
    l2->next = createNode(4);
    l2->next->next = createNode(6);

    printf("List 1:\n");
    printList(l1);
    printf("List 2:\n");
    printList(l2);

    Node *merged = mergeLists(l1, l2);

    printf("Merged list:\n");
    printList(merged);

    return 0;
}
```

OUTPUT

```
List 1:
1 , 3 , 5 , NULL
List 2:
2 , 4 , 6 , NULL
Merged list:
1 , 2 , 3 , 4 , 5 , 6 , NULL
```

c) Given a linked list sorted in increasing order, write a function that removes duplicate nodes from it by traversing the list only once. For example, the list {1, 2, 2, 2, 3, 4, 4, 5} should be converted into the list {1, 2, 3, 4, 5}

CODE

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

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

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
```

```
    return newNode;
}

void removeDuplicates(Node* head) {
    Node* current = head;
    while (current && current->next) {
        if (current->data == current->next->data) {
            Node* temp = current->next;
            current->next = current->next->next;
            free(temp);
        } else {
            current = current->next;
        }
    }
}

void printList(Node *head) {
    while (head != NULL) {
        printf("%d , ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    Node *head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(2);
    head->next->next->next = createNode(3);
    head->next->next->next->next = createNode(4);
```

```
    head->next->next->next->next->next = createNode(4);
    head->next->next->next->next->next->next = createNode(5);

    printf("Original list:\n");
    printList(head);

    removeDuplicates(head);

    printf("List after removing duplicates:\n");
    printList(head);

    return 0;
}
```

OUTPUT

```
Original list:
1 , 2 , 2 , 3 , 4 , 4 , 5 , NULL
List after removing duplicates:
1 , 2 , 3 , 4 , 5 , NULL
```