VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

PMCA501P - Data Structures and Algorithms

Srijan Dutta | 23MCA0131 |

Digital Assignment 1

1️⃣ Given an array A of n integers. You have to make a queue and stack of the given
integers. Queue should contain only prime numbers and stack should contain only
composite numbers. Display queue and stack contents
Let the array A contains 5 integers: 7, 21, 18, 3, 12 then the content of queue and stack
will be :
Queue : 7 , 3
Stack : 12 , 18 , 21

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

int isPrime(int n) {
    int i, c = 0;
```

```
    for (i = 2; i <= n / 2; i++) {
        if (n % i == 0)
            c++;
    }
    if (c > 1 || n == 1)
        return 0;
    else
        return 1;
}

int main() {
    int n = 1, i = 0, j = 0, k;         // Declaring necessary variables
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    k = n - 1;
    int *arr = (int *)malloc(sizeof(int) * n);
    int *stack = (int *)malloc(sizeof(int) * n);
    int *queue = (int *)malloc(sizeof(int) * n);

    // Entering elements
    printf("Enter the array elements: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
        if (isPrime(arr[i]) == 1) {
            queue[j] = arr[i];
            j++;
        } else {
            stack[k] = arr[i];
            k--;
        }
    }
    // Display
    printf("Queue: ");
    for (i = 0; i < j; i++) {
        printf("%d ", queue[i]);
    }
    printf("\nStack: ");
    for (i = n - 1; i > k; i--) {
        printf("%d ", stack[i]);
    }
    // Freeing the memory
    free(arr);
    free(stack);
    free(queue);

    return 0;
}
```

1️⃣ Output

```
PS D:\Program\College\MCA-VIT-Vellore\Data Structures and Algorithms\Assignments\Assesment 1> ./p1.exe
Enter the size of the array: 5
Enter the array elements: 7 8 12 13 21
Queue: 7 13
Stack: 8 12 21
PS D:\Program\College\MCA-VIT-Vellore\Data Structures and Algorithms\Assignments\Assesment 1> ./p1.exe
Enter the size of the array: 10
Enter the array elements: 12 15 17 23 29 32 40 43 48 53
Queue: 17 23 29 43 53
Stack: 12 15 32 40 48
PS D:\Program\College\MCA-VIT-Vellore\Data Structures and Algorithms\Assignments\Assesment 1>
```

2️⃣ Using push and pop operations of stack create a queue and display the contents of
queue. NOTE: Stack's property is LIFO and Queue's property is FIFO.
Input : Stack contents say 1,2,3,4,5
Output: Queue contents: 1,2,3,4,5

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

typedef struct stack {
    int data;
    struct stack* next;
} Stack;

Stack* createNode(int data) {
    Stack* newNode = (Stack*)malloc(sizeof(Stack));
    if (!newNode) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
```

```
    return newNode;
}

void push(Stack** top, int data) {
    Stack* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
}

int pop(Stack** top) {
    if (*top == NULL) {
        fprintf(stderr, "Stack underflow\n");
        exit(1);
    }
    Stack* temp = *top;
    int poppedData = temp->data;
    *top = (*top)->next;
    free(temp);
    return poppedData;
}

void enqueue(Stack** queue, int data) {
    push(queue, data);
}

int dequeue(Stack** inputStack, Stack** outputStack) {
    if (*outputStack == NULL) {
        while (*inputStack != NULL) {
            push(outputStack, pop(inputStack));
        }
    }
    return pop(outputStack);
}

void displayQueue(Stack* inputStack, Stack* outputStack) {
    printf("Queue contents: ");
    while (inputStack != NULL) {
        push(&outputStack, pop(&inputStack));
    }
    while (outputStack != NULL) {
        printf("%d", pop(&outputStack));

        if (outputStack != NULL) {
            printf(", ");
        }
    }

    printf("\n");
}


int main() {
    Stack* inputStack = NULL;
    Stack* outputStack = NULL;

    int n = 0, i = 0, data;
```

```
    while(n !=3){
        printf("-------MENU--------\n1. Enqueue\n2.Display\n3.Exit\n--------------");
        printf("\nEnter your choice:");
        scanf("%d", &n);
        if(n == 1){
            printf("Enter the number you want to add in Queue : \t");
            scanf("%d", &data);
            enqueue(&inputStack,data);
        }
        if(n == 2){
            displayQueue(inputStack, outputStack);
        }
    }

    displayQueue(inputStack, outputStack);

    return 0;
}
```

2️⃣ Output

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1> ./p2.exe
-------MENU--------
1. Enqueue
2.Display
3.Exit
--------------
Enter your choice:1
Enter the number you want to add in Queue :     5
-------MENU--------
1. Enqueue
2.Display
3.Exit
--------------
Enter your choice:1
Enter the number you want to add in Queue :     25
-------MENU--------
1. Enqueue
2.Display
3.Exit
--------------
Enter your choice:1
Enter the number you want to add in Queue :     12
-------MENU--------
1. Enqueue
2.Display
3.Exit
--------------
Enter your choice:1
Enter the number you want to add in Queue :     3
-------MENU--------
1. Enqueue
2.Display
3.Exit
--------------
Enter your choice:2
Queue contents: 5, 25, 12, 3
-------MENU--------
```

3️⃣ Given two sequences pushed and popped with distinct values, write a program to
return true if and only if this could have been the result of a sequence of push and pop operations on an initially empty stack.

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

bool validateStackSequences(int* pushed, int pushedSize, int* popped, int poppedSize) {
    int i = 0, j = 0;
    int stack[pushedSize];
    int top = -1;

    while (i < pushedSize) {
        if (top != -1 && stack[top] == popped[j]) {
            top--;
            j++;
        } else {
            stack[++top] = pushed[i];
            i++;
        }
    }

    while (top != -1 && stack[top] == popped[j]) {
        top--;
        j++;
    }

    return (i == pushedSize) && (j == poppedSize);
}

int main() {
    int pushed[] = {5, 10, 15, 20, 25};
    int popped1[] = {20, 25, 15, 10, 5};
    int popped2[] = {20, 15, 25, 5, 10};

    int pushedSize = sizeof(pushed) / sizeof(pushed[0]);
    int poppedSize = sizeof(popped1) / sizeof(popped1[0]);

    bool result1 = validateStackSequences(pushed, pushedSize, popped1, poppedSize);
    bool result2 = validateStackSequences(pushed, pushedSize, popped2, poppedSize);

    printf("Output for Example 1: %d\n", result1);
    printf("Output for Example 2: %d\n", result2);

    return 0;
}
```

3️⃣ Output

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1> ./p3.exe
Output for Example 1: 1
Output for Example 2: 0
PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1>
```

4️⃣ Given a doubly linked list, find the middle of the linked list. For example, if the given linked list is 1->2->3->4->5 then the output should be 3. If there are even nodes, then there would be two middle nodes so print the second middle element. For example, if the given linked list is 1->2->3->4->5->6 then the output should be 4.

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

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

void findMiddle(struct Node *head){
    struct Node *turtle = head;
    struct Node *rabbit = head;

    while(rabbit != NULL && rabbit -> next != NULL){
        turtle= turtle->next ;
        rabbit = rabbit->next->next;
    }
    printf("The middle of the list is : %d.", turtle -> data);
}

struct Node* createnode(int data){
    struct Node *newnode = (struct Node*)malloc(sizeof(struct Node));
    newnode -> data = data;
```

```
    newnode -> prev = NULL;
    newnode -> next = NULL;
    return newnode;
}

void insert(struct Node **head, int data){
    struct Node *newnode = createnode(data);
    if(*head == NULL){
        *head = newnode;
    }
    else{
        struct Node *current = *head;
        while(current -> next != NULL){
            current = current -> next;
        }
        current -> next = newnode;
        newnode -> prev = current;
    }
}

void display(struct Node *head){
    while(head!=NULL) {
        printf("%d ", head->data );
        head  = head->next;
    }
}
int main(){
    int data;
    printf("Enter the array elements (Press 0 to exit): ");
    scanf("%d", &data);
    struct Node *head = createnode(data);
    while(data != 0){
        scanf("%d",&data );
        if(data != 0)
            insert(&head, data);

    }
    printf("List: ");
    display(head);
    printf("\n");
    findMiddle(head);
    return 0;
}
```

4️⃣ Output

```
PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1> ./p4.exe
Enter the array elements (Press 0 to exit): 1 2 3 4 5 6 0
List: 1 2 3 4 5 6
The middle of the list is : 4.
PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1> ./p4.exe
Enter the array elements (Press 0 to exit): 22 56 5 7 89 223 15 23 0
List: 22 56 5 7 89 223 15 23
The middle of the list is : 89.
PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1> ./p4.exe
Enter the array elements (Press 0 to exit): 1 5 4 88 99 3 2 3 4 5 75 44 33 0
List: 1 5 4 88 99 3 2 3 4 5 75 44 33
The middle of the list is : 2.
PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1>
```

5️⃣ Let’s simulate a game ‘Hot potato’. It is the game where children stand in a circle and
pass some objects to their neighbors. At a certain point in the game, the action is
stopped and the child who has the object is removed from the circle. Play continues
until only one child is left. Let’s simulate the scenario by removing a child when the
count becomes 7.
The program will input a list of names and a constant, call it “num,” to be used for
counting. It will return the name of the last person remaining after repetitive counting
by num.
To simulate the circle, we will use a queue. Assume that the child at the front of the
queue says 1 and joins back the queue (dequeued and enqueued immediately). The
child at the front will say 2 and joins back the queue. This repeats until a child says 7
(dequeued permanently). The game again starts from count 1. This process will
continue until only one name remains (the size of the queue is 1).

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

char r_name[50];

struct Node {
    char name[50];
    struct Node* next;
};

struct Queue {
    struct Node* front;
    struct Node* rear;
};

struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
```

```
    if (queue == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        exit(1);
    }
    queue->front = NULL;
    queue->rear = NULL;
    return queue;
}

struct Node* createNode(char* name) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    if (temp == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        exit(1);
    }
    strcpy(temp->name, name);
    temp->next = NULL;
    return temp;
}

void enqueue(struct Queue* queue, char* name) {
    struct Node* newNode = createNode(name);
    if (queue->rear == NULL) {
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        (queue->rear)->next = newNode;
        queue->rear = newNode;
    }
}

char* dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        fprintf(stderr, "Queue is empty\n");
        exit(1);
    }
    struct Node* temp = queue->front;
    queue->front = (queue->front)->next;
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    strcpy(r_name, temp -> name);
    free(temp);
    return r_name;
}

int main() {
    int n;
    printf("Enter the number of children: ");
    scanf("%d", &n);

    struct Queue* queue = createQueue();

    for (int i = 0; i < n; i++) {
        char name[50];
        printf("\nEnter the name of child %d: ", i + 1);
```

```
        scanf("%s", &name);
        enqueue(queue, name);
    }

    int count = 1;
    while (queue->front != queue->rear) {
        char* removedName = dequeue(queue);
        printf("Child %s says %d\n", removedName, count);

        if (count == 7) {
            printf("Child %s is removed from the game.\n", removedName);
            count = 1;
        } else {
            enqueue(queue, removedName);
            count++;
        }
    }
    printf("The winner is: %s\n", queue->front->name);
    free(queue);
    return 0;
}
```

5️⃣ Output

```
PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1> ./p5.exe
Enter the number of children: 3

Enter the name of child 1: Srijan

Enter the name of child 2: Raju

Enter the name of child 3: Shaan
Child Srijan says 1
Child Raju says 2
Child Shaan says 3
Child Srijan says 4
Child Raju says 5
Child Shaan says 6
Child Srijan says 7
Child Srijan is removed from the game.
Child Raju says 1
Child Shaan says 2
Child Raju says 3
Child Shaan says 4
Child Raju says 5
Child Shaan says 6
Child Raju says 7
Child Raju is removed from the game.
The winner is: Shaan
PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1> ./p5.exe
Enter the number of children: 2

Enter the name of child 1: Shaan

Enter the name of child 2: Srijan
Child Shaan says 1
Child Srijan says 2
Child Shaan says 3
Child Srijan says 4
Child Shaan says 5
Child Srijan says 6
Child Shaan says 7
Child Shaan is removed from the game.
The winner is: Srijan
PS D:\Program\College\MCA-VIT-Vellore\Sem 1\Data Structures and Algorithms\Assignments\Assesment 1>
```