Iftala Zahri Sukmana	Exercise 5 Queue	Kamis, 9 Maret 2023
5025221002		Struktur Data
Teknik Informatika		FX. Arunanto
Institut Teknologi Sepuluh Nopember		Paraf:

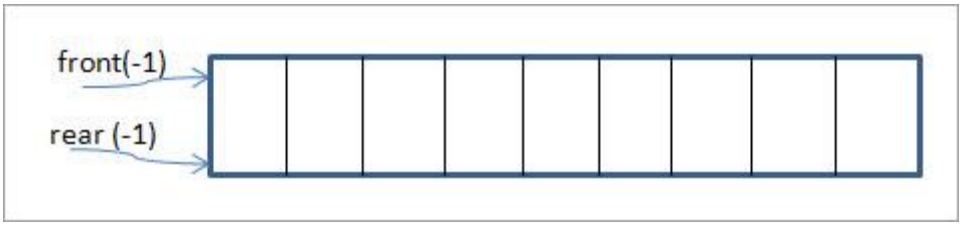
Soal

1. Write declarations/functions to implement a queue of double values.
2. A priority queue is one in which items are added to the queue based on a priority number. Jobs with higher-priority numbers are closer to the head of the queue than those with lower priority numbers. a job is added to the queue in front of all jobs of lower priority but after all jobs of greater or equal priority.
3. Write declarations and functions to implement a priority queue. each item in the queue has a job number (integer) and a priority number. Implement, at least, the following functions: (i) initialize an empty queue, (ii) add a job in its appropriate place in the queue, (iii) delete and dispose of the job at the head of the queue, and (iv) given a job number, remove that job from the queue.
4. An input line contains a word consisting of lowercase letters only. explain how a stack can be used to determine whether the word is a palindrome.
5. Show how to implement a queue using two stacks.
6. Show how to implement a stack using two queues.
7. A stack, S1, contains some numbers in arbitrary order. using another stack, S2, for temporary storage, show how to sort the numbers in S1 such that the smallest is at the top of S1 and the largest is at the bottom.

Jawaban

1. Queue adalah struktur data yang mirip dengan Stack. Perbedaan yang mencolok dibandingkan dengan Stack adalah menggunakan sistem FIFO (First In, First Out), dimana item atau elemen pertama yang ditambahkan ke dalam Queue akan menjadi item pertama yang dikeluarkan dari Antrian. Apabila menggunakan analogi, Queue dapat diibaratkan sebagai antrian pembeli untuk membeli makanan, dimana yang pertama mengantre akan menjadi yang pertama mendapatkan makanan.

Dalam istilah software, Queue adalah sebuah aturan atau koleksi elemen yang diatur secara linear atau barisan, seperti yang digambarkan sebagai berikut:



Queue memiliki dua ujung, yaitu "front" dan "rear" dari queue. Apabila queue dalam keadaan kosong, maka kedua pointer akan diatur menjadi -1 (dan item pertama pada queue akan memiliki indeks 0). "rear" pointer adalah posisi dimana elemen dimasukkan ke dalam queue, operasi ini "enqueue". Sedangkan "front" pointer adalah tempat dimana elemen dihilangkan dari queue, operasi ini disebut "dequeue".

Apabila sebuah queue[SIZE] memiliki nilai "rear" pointer sebesar SIZE-1, maka dapat dikatakan queue itu penuh. Apabila nilai dari "front" adalah NULL, maka queue itu kosong.

Operasi dasar pada Queue

Operasi-operasi yang dapat dilakukan pada Queue adalah sebagai berikut:

a. Enqueue

   Enqueue melakukan penambahan item atau elemen pada queue. Setiap penambahan item itu terjadi pada belakang queue. Enqueue memiliki algoritma sebagai berikut:

   - Melakukan pengecekan pada queue apabila penuh
   - Apabila queue penuh, tampilkan informasi "Queue Penuh" dan hentikan operasi.
   - Apabila queue tidak penuh, lakukan increment pada "rear"
   - Tambahkan elemen pada lokasi yang ditunjuk oleh "rear"
   - Return success

b. Dequeue

   Dequeue melakukan penghapusan item atau elemen pada queue. Setiap penghapusan item itu terjadi pada depan queue. Dequeue memiliki algoritma sebagai berikut:

   - Melakukan pengecekan pada queue apabila kosong

- Apabila queue kosong, tampilkan informasi "Queue Kosong" dan hentikan operasi.
- Apabila queue tidak kosong, akses elemen yang ditunjuk oleh "front"
- Lakukan increment pada "front" ke elemen yang ditunjuk
- Return success

c. isEmpty

isEmpty akan melakukan pengecekan queue apabila queue dalam keadaan kosong

d. isFull

isFull akan melakukan pengecekan queue apabila queue dalam keadaan penuh

e. peek

peek akan mengambil nilai yang ada di depan queue namun tidak melakukan modifikasi pada nilai tersebut

2. Double

```c
#include<stdio.h>

#define MAX_SIZE 100

typedef struct {
    double data[MAX_SIZE];
    int front;
    int rear;
    int size;
} Queue;

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

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

int is_full(Queue *q) {
    return q->size == MAX_SIZE;
}

void enqueue(Queue *q, double value) {
    if (is_full(q)) {
```

```
        printf("Queue is full\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->data[q->rear] = value;
    q->size++;
}

double dequeue(Queue *q) {
    if (is_empty(q)) {
        printf("Queue is empty\n");
        return 0;
    }
    double value = q->data[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->size--;
    return value;
}

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

    enqueue(&q, 3.14);
    enqueue(&q, 2.71);

    double value = dequeue(&q);
    printf("%f\n", value);

    return 0;
}
```

3. Priority queue adalah struktur data dimana setiap item memiliki asosiasi dan nilai prioritas. Item yang memiliki prioritas lebih tinggi akan didahulukan. Apabila dua item memiliki nilai prioritas yang sama, maka data dikeluarkan dalam urutan mereka dimasukkan

   Operasi utama pada priority queue adalah

   a. Enqueue(item,prioritas)

      Menambahkan item pada queue bersama dengan nilai prioritasnya

   b. Dequeue()

      Menghapus item dengan prioritas tertinggi dari queue

   c. Peek()

Mengembalikan nilai item dengan prioritas tertinggi tanpa menghilangkannya dari queue

d. Is_empty()

Melakukan pengecekan apakah sebuah queue apakah isinya kosong

Bentuk implementasi dari priority qeueu pada C dapat menggunakan linked list. Setiap node dari linked list terdiri dari item dan nilai prioritasnya. Nilai head dari list adalah item dengan prioritas tertinggi.

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

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

typedef struct {
    Node* head;
} PriorityQueue;

void init(PriorityQueue* pq) {
    pq->head = NULL;
}

int is_empty(PriorityQueue* pq) {
    return pq->head == NULL;
}

void enqueue(PriorityQueue* pq, int data, int priority) {
    Node* new_node = (Node*) malloc(sizeof(Node));
    new_node->data = data;
    new_node->priority = priority;
    new_node->next = NULL;

    if (is_empty(pq) || priority > pq->head->priority) {
        new_node->next = pq->head;
        pq->head = new_node;
    } else {
        Node* curr = pq->head;
        while (curr->next != NULL && priority <= curr->next-
>priority) {
            curr = curr->next;
        }
        new_node->next = curr->next;
```

```
        curr->next = new_node;
    }
}

int dequeue(PriorityQueue* pq) {
    if (is_empty(pq)) {
        printf("Priority queue is empty\n");
        return -1;
    }
    int data = pq->head->data;
    Node* temp = pq->head;
    pq->head = pq->head->next;
    free(temp);
    return data;
}

int peek(PriorityQueue* pq) {
    if (is_empty(pq)) {
        printf("Priority queue is empty\n");
        return -1;
    }
    return pq->head->data;
}

int main() {
    PriorityQueue pq;
    init(&pq);

    enqueue(&pq, 5, 2);
    enqueue(&pq, 3, 1);
    enqueue(&pq, 10, 3);

    printf("%d\n", dequeue(&pq)); // prints 10
    printf("%d\n", dequeue(&pq)); // prints 5
    printf("%d\n", dequeue(&pq)); // prints 3

    return 0;
}
```

4. functions

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

typedef struct Node {
    int job_num;
    int priority;
```

```
    struct Node* next;
} Node;

typedef struct {
    Node* head;
} PriorityQueue;

void init(PriorityQueue* pq) {
    pq->head = NULL;
}

int is_empty(PriorityQueue* pq) {
    return pq->head == NULL;
}

void enqueue(PriorityQueue* pq, int job_num, int priority) {
    Node* new_node = (Node*) malloc(sizeof(Node));
    new_node->job_num = job_num;
    new_node->priority = priority;
    new_node->next = NULL;

    if (is_empty(pq) || priority > pq->head->priority) {
        new_node->next = pq->head;
        pq->head = new_node;
    } else {
        Node* curr = pq->head;
        while (curr->next != NULL && priority <= curr->next-
>priority) {
            curr = curr->next;
        }
        new_node->next = curr->next;
        curr->next = new_node;
    }
}

int dequeue(PriorityQueue* pq) {
    if (is_empty(pq)) {
        printf("Priority queue is empty\n");
        return -1;
    }
    int job_num = pq->head->job_num;
    Node* temp = pq->head;
    pq->head = pq->head->next;
    free(temp);
    return job_num;
}

void remove_job(PriorityQueue* pq, int job_num) {
```

```
    Node* curr = pq->head;
    Node* prev = NULL;
    while (curr != NULL && curr->job_num != job_num) {
        prev = curr;
        curr = curr->next;
    }
    if (curr == NULL) {
        printf("Job not found in queue\n");
        return;
    }
    if (prev == NULL) {
        pq->head = curr->next;
    } else {
        prev->next = curr->next;
    }
    free(curr);
}
```

5. Palindrom merupakan sebuah kata yang sama ketika dibaca baik dari depan maupun dari belakang kata. Untuk menentukan apakah sebuah kata adalah palindrom, maka dapat menggunakan stack, yang memiliki algoritma sebagai berikut:
    a. Buatlah stack kosong
    b. Masukkan setiap karakter ke stack
    c. Keluarkan setiap karakter dari stack dan tambahkan ke string baru
    d. Bandingkan dengan input awal

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

#define MAX_LEN 100

struct Stack {
    char data[MAX_LEN];
    int top;
};

void push(struct Stack* stack, char c) {
    if (stack->top == MAX_LEN - 1) {
        printf("Stack Overflow\n");
        return;
    }
```

```
    stack->data[++stack->top] = c;
}

char pop(struct Stack* stack) {
    if (stack->top == -1) {
        printf("Stack Underflow\n");
        exit(1);
    }
    return stack->data[stack->top--];
}

bool isPalindrome(char* word) {
    int len = strlen(word);
    struct Stack stack;
    stack.top = -1;

    for (int i = 0; i < len; i++) {
        push(&stack, word[i]);
    }

    char newWord[MAX_LEN];
    int i = 0;
    while (stack.top != -1) {
        newWord[i++] = pop(&stack);
    }
    newWord[i] = '\0';

    if (strcmp(word, newWord) == 0) {
        return true;
    }
    return false;
}

int main() {
    char word[MAX_LEN];
    printf("Enter a word: ");
    scanf("%s", word);
    if (isPalindrome(word)) {
        printf("%s is a palindrome.\n", word);
    } else {
        printf("%s is not a palindrome.\n", word);
    }
    return 0;
}
```

6. Untuk implementasi queue menggunakan dua stack, bisa menggunakan cara beirkut
    a. Buatlah dua stack A dan B

b. Ketika melakukan enqueue, masukkan ke dalam stack A
c. Ketika melakukan dequeue, cek terlebih dahulu apakah list B itu kosong, lalu keluarkan seluruh data di stack A dan masukkan ke stack B secara terbalik. Apabila stack B tidak kosong, maka lakukan penghapusan item di stack B.

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

#define MAX_SIZE 100

struct stack {
    int arr[MAX_SIZE];
    int top;
};

void push(struct stack *s, int data) {
    if (s->top >= MAX_SIZE - 1) {
        printf("Stack overflow\n");
        return;
    }
    s->arr[++(s->top)] = data;
}

int pop(struct stack *s) {
    if (s->top == -1) {
        printf("Stack underflow\n");
        return -1;
    }
    return s->arr[(s->top)--];
}

struct queue {
    struct stack s1;
    struct stack s2;
};

void enqueue(struct queue *q, int data) {
    push(&(q->s1), data);
}

int dequeue(struct queue *q) {
    if (q->s1.top == -1 && q->s2.top == -1) {
        printf("Queue underflow\n");
        return -1;
    }
    if (q->s2.top == -1) {
        while (q->s1.top != -1) {
```

```
            push(&(q->s2), pop(&(q->s1)));
        }
    }
    return pop(&(q->s2));
}

int main() {
    struct queue q;
    q.s1.top = -1;
    q.s2.top = -1;

    enqueue(&q, 1);
    enqueue(&q, 2);
    enqueue(&q, 3);
    printf("%d ", dequeue(&q));
    printf("%d ", dequeue(&q));
    enqueue(&q, 4);
    printf("%d ", dequeue(&q));
    printf("%d ", dequeue(&q));

    return 0;
}
```

7. Dalam membuat stack menggunakan dua queue, pendekatan bisa dilakukan dengan cara berikut:
    a. Buatlah stack dengan dua buah pointer “front” dan “back”, definisikan juga fungsi push dan pop
    b. Lakukan enqueue data ke queue
    c. Fungsi pop cukup dengan mengecek apakah queue itu kosong dan kembalikan nilai -1. Kalua tidak, maka lakukan dequeue hingga menyisakan satu buah elemen yang terakhir dan enqueue ke queue lainnya, yang menyebabkan urutannya menjadi terbalik. Setelah itu, lakukan dequeue dari elemen terakhir lalu melakukan enqueue kembali ke queue normal pada data di queue lainnya.

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

#define MAX_SIZE 100

struct queue {
    int arr[MAX_SIZE];
    int front, rear;
};

void enqueue(struct queue *q, int data) {
```

```c
    if (q->rear == MAX_SIZE - 1) {
        printf("Queue overflow\n");
        return;
    }
    q->arr[++(q->rear)] = data;
}

int dequeue(struct queue *q) {
    if (q->front > q->rear) {
        printf("Queue underflow\n");
        return -1;
    }
    return q->arr[(q->front)++];
}

struct stack {
    struct queue q1;
    struct queue q2;
};

void push(struct stack *s, int data) {
    if (s->q1.rear == MAX_SIZE - 1) {
        printf("Stack overflow\n");
        return;
    }
    enqueue(&(s->q1), data);
}

int pop(struct stack *s) {
    if (s->q1.front > s->q1.rear) {
        printf("Stack underflow\n");
        return -1;
    }
    while (s->q1.front != s->q1.rear) {
        enqueue(&(s->q2), dequeue(&(s->q1)));
    }
    int data = dequeue(&(s->q1));
    while (s->q2.front <= s->q2.rear) {
        enqueue(&(s->q1), dequeue(&(s->q2)));
    }
    return data;
}

int main() {
    struct stack s;
    s.q1.front = s.q1.rear = -1;
    s.q2.front = s.q2.rear = -1;
```

```
    push(&s, 1);
    push(&s, 2);
    push(&s, 3);
    printf("%d ", pop(&s));
    printf("%d ", pop(&s));
    push(&s, 4);
    printf("%d ", pop(&s));
    printf("%d ", pop(&s));

    return 0;
}
```

8. Berikut adalah algortma dari permintaan:
    a. Buat stack temporary bernama S2
    b. Apabila S1 tidak kosong, hapus elemen dari S1 dan madukkan ke S2
    c. Apabila S2 tidak kosong, dan top() dari list lebih besar dari temp, maka hapus top() dari S2 ke S1
    d. Tambahkan temp ke S2

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

#define MAX_SIZE 100

struct stack {
    int arr[MAX_SIZE];
    int top;
};

void push(struct stack *s, int data) {
    if (s->top == MAX_SIZE - 1) {
        printf("Stack overflow\n");
        return;
    }
    s->arr[++(s->top)] = data;
}

int pop(struct stack *s) {
    if (s->top == -1) {
        printf("Stack underflow\n");
        return -1;
    }
    return s->arr[(s->top)--];
}
```

```c
void sort_stack(struct stack *s) {
    struct stack s2;
    s2.top = -1;

    while (s->top != -1) {
        int temp = pop(s);
        while (s2.top != -1 && s2.arr[s2.top] > temp) {
            push(s, pop(&s2));
        }
        push(&s2, temp);
    }

    // Copy the sorted elements back to the original stack
    while (s2.top != -1) {
        push(s, pop(&s2));
    }
}

int main() {
    struct stack s;
    s.top = -1;

    push(&s, 3);
    push(&s, 1);
    push(&s, 4);
    push(&s, 2);

    sort_stack(&s);

    while (s.top != -1) {
        printf("%d ", pop(&s));
    }

    return 0;
}
```

- We define a stack structure with an array and a top pointer for tracking the top element of the stack. We also define a push and a pop function for manipulating the stack.
- The sort_stack function sorts the input stack in ascending order using a temporary stack S2. It repeatedly pops an element from S1, compares it with the top element of S2, and either pushes it onto S2 or moves elements from S2 back to S1 until the correct position for the element is found. After all elements have been processed, the sorted stack is in S2.

- In the main function, we create a stack and push some elements onto it. We then call the sort_stack function to sort the stack in ascending order and print the sorted elements. The output should be "1 2 3 4"