LEMBAR KERJA PRAKTIKUM 4 STRUKTUR DATA

STACK & QUEUE

A. Tujuan Praktikum

- Mahasiswa dapat mengimplementasikan Stack.
- Mahasiswa dapat mengimplementasikan Queue.

B. Implementasikan Program di bawah ini

PROGRAM 1. Implementasi Stack

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

#define STACKSIZE 100
#define TRUE 1
#define FALSE 0

typedef struct {
   int item[STACKSIZE];
   int top;
} Stack;

/* Function Prototypes */
void init(Stack *s);
int is_empty(Stack *s);
int is_full(Stack *s);
int push(Stack *s, int x);
int pop(Stack *s, int *value);
int stack_top(Stack *s, int *value);

int main(void)
{
   Stack s;
   int choice, value;
   char ch;

   init(&s);

   do {
      printf("\n===== STACK MENU =====\n");
      printf("1. Push\n");
      printf("2. Pop\n");
      printf("3. Stack Top\n");
      printf("4. Exit\n");
```

```
        printf("======================\n");
        printf("Enter your choice: ");

        if (scanf("%d", &choice) != 1) {
            printf("Invalid input!\n");
            while (getchar() != '\n'); // clear buffer
            continue;
        }

        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                if (push(&s, value))
                    printf("Value pushed successfully.\n");
                else
                    printf("Stack Overflow!\n");
                break;

            case 2:
                if (pop(&s, &value))
                    printf("Popped value: %d\n", value);
                else
                    printf("Stack Underflow!\n");
                break;

            case 3:
                if (stack_top(&s, &value))
                    printf("Top value: %d\n", value);
                else
                    printf("Stack is empty.\n");
                break;

            case 4:
                printf("Exiting...\n");
                break;

            default:
                printf("Invalid choice!\n");
        }

        while (getchar() != '\n'); // clear buffer

    } while (choice != 4);

    return 0;
}
```

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

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

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

int push(Stack *s, int x)
{
  if (is_full(s))
    return FALSE;

  s->item[++(s->top)] = x;
  return TRUE;
}

int pop(Stack *s, int *value)
{
  if (is_empty(s))
    return FALSE;

  *value = s->item[(s->top)--];
  return TRUE;
}

int stack_top(Stack *s, int *value)
{
  if (is_empty(s))
    return FALSE;

  *value = s->item[s->top];
  return TRUE;
}
```

PROGRAM 2. Implementasi Queue

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

#define MAXQUEUE 5
```

```
#define TRUE 1
#define FALSE 0

typedef struct {
   int item[MAXQUEUE];
   int front;
   int rear;
} Queue;

/* Prototypes */
void init(Queue *q);
int is_empty(Queue *q);
int is_full(Queue *q);
int enqueue(Queue *q, int x);
int dequeue(Queue *q, int *value);

int main(void)
{
   Queue q;
   int choice, value;

   init(&q);

   do {
      printf("\n===== QUEUE MENU =====\n");
      printf("1. Enqueue\n");
      printf("2. Dequeue\n");
      printf("3. Exit\n");
      printf("======================\n");
      printf("Enter your choice: ");

      if (scanf("%d", &choice) != 1) {
         printf("Invalid input!\n");
         while (getchar() != '\n');
         continue;
      }

      switch (choice) {
         case 1:
            printf("Enter value: ");
            scanf("%d", &value);
            if (enqueue(&q, value))
               printf("Value inserted successfully.\n");
            else
               printf("Queue Overflow!\n");
            break;

         case 2:
```

```
            if (dequeue(&q, &value))
               printf("Removed value: %d\n", value);
            else
               printf("Queue Underflow!\n");
            break;

         case 3:
            printf("Exiting...\n");
            break;

         default:
            printf("Invalid choice!\n");
      }

      while (getchar() != '\n');

   } while (choice != 3);

   return 0;
}

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

int is_empty(Queue *q)
{
   return (q->front > q->rear);
}

int is_full(Queue *q)
{
   return (q->rear == MAXQUEUE - 1);
}

int enqueue(Queue *q, int x)
{
   if (is_full(q))
      return FALSE;

   q->item[++(q->rear)] = x;
   return TRUE;
}

int dequeue(Queue *q, int *value)
{
```

```
  if (is_empty(q))
    return FALSE;

  *value = q->item[(q->front)++];
  return TRUE;
}
```

C. Berdasarkan dua program di atas, lakukan proses tracing,dan watch variables untuk melihat alur kerja dari program tersebut. Setelah memahami alur program, diskusikan hal-hal berikut ini dengan asisten dan teman-teman Anda (hasil diskusi tidak perlu dituliskan).

1. Lakukan operasi-operasi sebagai berikut secara berurutan pada Stack : push(4), push(8), pop(), push(6), push(19), pop(), pop(), push(8). Apa saja nilai yang di pop dari Stack ? Bagaimana kondisi Stack (isi Stack) setelah dilakukan operasi-operasi tersebut.
2. Ubah nilai STACKSIZE menjadi 3, kemudian jalankan program stack dan lakukan operasi berikut: push(10), push(20), push(30), push(40). Apa yang terjadi ketika push(40) dijalankan? Berapa nilai top tepat sebelum operasi terakhir dilakukan? Apa isi array saat kapasitas maksimum tercapai? Berdasarkan hasil eksperimen yang Anda lakukan, simpulkan bagaimana stack menentukan kondisi penuh.
3. Perhatikan fungsi empty pada queue. Mengapa kondisi pengecekan untuk kondisi empty adalah pq->rear-pq->front+1==0 ? Apakah ada alternatif lainnya yang lebih baik?
4. Lakukan operasi-operasi sebagai berikut secara berurutan pada Queue : enqueue(4), enqueue(8), enqueue(6), enqueue(19), enqueue(8), dequeue(), dequeue(), dequeue(). Berapa nilai rear dan front-nya setelah semua operasi tersebut?. Jika dilakukan operasi ini enqueue(11), bagaimana kondisi Queue?
5. Gunakan nilai MAXQUEUE = 5. Jalankan operasi berikut secara berurutan:enqueue(1), enqueue(2), enqueue(3), enqueue(4), dequeue(), dequeue(), enqueue(5), enqueue(6). Selama menjalankan program, gunakan watch variables untuk mengamati perubahan nilai front dan rear. Berapa nilai front dan rear setelah setiap operasi? Apakah semua elemen array terisi secara berurutan? Apakah terdapat ruang kosong yang tidak bisa digunakan kembali? Bagaimana mekanisme pergerakan front dan rear bekerja pada queue linear.
6. Gunakan data yang sama pada kedua program: 4, 8, 6, 2. Pada stack: lakukan push semua elemen, lalu pop semuanya. Pada queue: lakukan enqueue semua elemen, lalu dequeue semuanya. Catat urutan elemen yang keluar dari stack. Catat urutan elemen yang keluar dari queue. Bandingkan hasilnya. Berdasarkan hasil dari kedua program, simpulkan perbedaan fundamental LIFO dan FIFO dalam implementasi array.

D. Video dan Hasil Diskusi (dikumpulkan selambatnya satu minggu setelah praktikum ini)

Setelah Anda mempelajari program di atas, kerjakan tugas di bawah ini dengan kelompok Anda, lalu buat video diskusinya dan tuliskan hasil diskusinya (PDF), masukkan ke gdrive, lalu kumpulkan link gdrive pada form berikut: https://forms.gle/JCWa8XJy9n7Vnc7r7

BAGIAN 1 – MODIFIKASI PROGRAM

1. Stack → Linked List

Ubah implementasi Stack berbasis array menjadi Stack berbasis Linked List.
Ketentuan:

- Gunakan struct node (data dan pointer next).
- Top direpresentasikan sebagai pointer.
- Implementasikan kembali operasi push dan pop.
- Program tetap dapat dijalankan seperti sebelumnya.

2. Queue → Linked List

Ubah implementasi Queue berbasis array menjadi Queue berbasis Linked List.
Ketentuan:

- Gunakan struct node (data dan pointer next).
- Gunakan pointer front dan rear.
- Implementasikan kembali enqueue dan dequeue.
- Program tetap dapat dijalankan seperti sebelumnya.

BAGIAN 2 – DISKUSI DAN ANALISIS

Setelah program berhasil dimodifikasi dan diuji, diskusikan pertanyaan berikut bersama kelompok Anda.

1. Perbedaan Implementasi

Setelah Anda mencoba kedua versi (array dan linked list):

- Apa perbedaan utama yang Anda rasakan saat mengubah kodenya?
- Bagian mana yang paling berbeda: penggunaan indeks atau pointer?
- Menurut Anda, mana yang lebih mudah dipahami? Mengapa?

2. Tentang Kapasitas

- Pada versi array, apa yang terjadi jika kapasitas penuh?
- Apakah hal tersebut terjadi pada versi linked list?
- Apakah linked list benar-benar tidak terbatas? Jelaskan dengan sederhana.

3. Tentang Operasi Dasar

Bandingkan hasil pengujian Anda:

- Apakah operasi push dan pop pada stack tetap berjalan dengan cara yang sama?
- Apakah enqueue dan dequeue pada queue tetap mengikuti prinsip FIFO?
- Apakah waktu eksekusinya terasa berbeda?

4. Tentang Penggunaan Memori

- Bagaimana perbedaan cara penyimpanan data antara array dan linked list?
- Mengapa pada linked list kita perlu menggunakan malloc dan free?
- Apa yang bisa terjadi jika lupa melakukan free?

5. Kesimpulan Kelompok

Berdasarkan pengalaman Anda memodifikasi program:

- Kapan sebaiknya menggunakan array?
- Kapan sebaiknya menggunakan linked list?

- Apakah Stack dan Queue bergantung pada jenis penyimpanan tertentu, atau hanya pada aturan akses datanya?

Jawab dengan bahasa Anda sendiri berdasarkan pengalaman mengerjakan tugas ini.