

Nama : Hafizhah Nur Zahira

NIM : 1203230024

Kelas : IF 03-02

## Tugas OTH Struct Stack

### 1. Asisten Sherlock Holmes

#### ➤ Source Code

```
#include <stdio.h>

// Mendefinisikan struct Batu untuk menyimpan alphabet
struct Batu {
    char alphabet; // Menyimpan huruf pada batu
    struct Batu *link; // Pointer ke Batu berikutnya dalam urutan
};

int main() {
    // Inisialisasi batu sesuai dengan petunjuk
    struct Batu l1, l2, l3, l4, l5, l6, l7, l8, l9;

    l1.link = NULL;
    l1.alphabet = 'F';

    l2.link = NULL;
    l2.alphabet = 'M';

    l3.link = NULL;
    l3.alphabet = 'A';

    l4.link = NULL;
    l4.alphabet = 'I';

    l5.link = NULL;
    l5.alphabet = 'K';

    l6.link = NULL;
    l6.alphabet = 'T';

    l7.link = NULL;
    l7.alphabet = 'N';

    l8.link = NULL;
    l8.alphabet = 'O';

    l9.link = NULL;
    l9.alphabet = 'R';
```

```

// Mengatur koneksi antar batu sesuai dengan urutan yang diberikan
17.link = &l1;
11.link = &l8;
18.link = &l2;
12.link = &l5;
15.link = &l3;
13.link = &l6;
16.link = &l9;
19.link = &l4;
14.link = &l7;

// Mengakses huruf pada batu menggunakan l3 sebagai titik awal
printf("%c", 13.link->link->link->alphabet); // Output: "I"
printf("%c", 13.link->link->link->link->alphabet); // Output: "N"
printf("%c", 13.link->link->link->link->link->alphabet); // Output: "F"
printf("%c", 13.link->link->link->link->link->link->alphabet); // Output:
"O"
printf("%c", 13.link->link->alphabet); // Output: "R"
printf("%c", 13.link->link->link->link->link->link->link->alphabet); //
Output: "M"
printf("%c", 13.alphabet); // Output: "A"
printf("%c", 13.link->alphabet); // Output: "T"
printf("%c", 13.link->link->link->alphabet); // Output: "I"
printf("%c", 13.link->link->link->link->link->link->link->link->
>alphabet); // Output: "K"
printf("%c", 13.alphabet); // Output: "A"

return 0;
}

```

#### ➤ Penjelasan


Kode tersebut menggunakan self-referential structure. Pada kode ini, struct 'Batu' didefinisikan untuk menyimpan sebuah huruf dari elemen alphabet dan pointer link ke struct batu. Kemudian beberapa variabel struct 'Batu' seperti { l1, l2, l3, l4, l5, l6, l7, l8, l9 } diinialisasikan dengan huruf-huruf yang sudah ditentukan.

Selanjutnya, kita melakukan proses pengaturan koneksi antar batu agar sesuai dengan urutan yang diminta di pdf. Misal, l7.link diatur menunjuk ke l1 kemudian l1.link menunjuk ke l8 dan seterusnya sesuai dengan urutan yang diminta.

Terakhir, kita melakukan akses terhadap huruf-huruf pada batu menggunakan l3 yang telah ditentukan menjadi titik awal. Untuk menghubungkan huruf-huruf pada batu kita menggunakan pointer ke struct 'Batu' agar dapat menghubungkan huruf-huruf tersebut sesuai dengan hubungan yang telah kita atur.

Misal, printf("%c", l3.link -> link -> link -> link -> alphabet); akan mencetak huruf 'I' karena kita mengikuti alur pointer dari l3 ke l6 lalu ke l9 dan akhirnya ke l4 yang memiliki nilai huruf 'I'. Printf berguna untuk mencetak huruf-huruf sesuai dengan urutan yang diinginkan.

## ➤ Output



```
PS D:\Struktur Data\praktikum6> cd "d:\Struktur Data\praktikum6\" ; if ($?) { gcc oth1.c -o oth1 } ; if ($?) { .\oth1 }
d:\Struktur Data\praktikum6\> if ($?) { gcc oth1.c -o oth1 } ; if ($?) { .\oth1 }ta\praktikum6> cd "
INFORMATIKA
PS D:\Struktur Data\praktikum6>
```

## 2. Game of TwoStack (HackerRank)

### ➤ Source Code

```
#include <assert.h> //Header file untuk menambahkan fungsi assertion.
#include <stdbool.h> // Header file untuk menggunakan tipe data boolean (true
dan false).
#include <stddef.h> //Header file untuk mendefinisikan beberapa tipe dan
konstanta dasar.
#include <stdint.h> //Header file untuk mendefinisikan tipe data dengan lebar
bit tertentu.
#include <stdio.h> //Header file untuk fungsi input-output standar.
#include <stdlib.h> // Header file untuk fungsi-fungsi umum, seperti alokasi
memori dinamis.

// Struktur data untuk merepresentasikan stack
typedef struct {
    int top;           // Indeks atas stack
    int capacity;      // Kapasitas maksimum stack
    int* array;        // Array untuk menyimpan elemen-elemen stack
} Stack;

// Fungsi untuk membuat stack baru dengan kapasitas tertentu
Stack* createStack(int capacity) {
    // Mengalokasikan memori untuk stack
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->capacity = capacity;
    stack->top = -1;    // Stack awalnya kosong
    stack->array = (int*)malloc(stack->capacity * sizeof(int));
    return stack;
}

// Fungsi untuk memeriksa apakah stack kosong
bool isEmpty(Stack* stack) {
    return stack->top == -1;
}

// Fungsi untuk memeriksa apakah stack penuh
bool isFull(Stack* stack) {
    return stack->top == stack->capacity - 1;
}
```

```

// Fungsi untuk menambahkan elemen ke dalam stack
void push(Stack* stack, int item) {
    if (isFull(stack)) return; // Jika stack penuh, tidak melakukan apa-apa
    stack->array[++stack->top] = item; // Menambahkan elemen ke dalam stack
}

// Fungsi untuk menghapus dan mengembalikan elemen teratas dari stack
int pop(Stack* stack) {
    if (isEmpty(stack)) return -1; // Jika stack kosong, mengembalikan nilai -1
    return stack->array[stack->top--]; // Menghapus dan mengembalikan elemen teratas dari stack
}

// Fungsi untuk melihat elemen teratas dari stack tanpa menghapusnya
int peek(Stack* stack) {
    if (isEmpty(stack)) return -1; // Jika stack kosong, mengembalikan nilai -1
    return stack->array[stack->top]; // Mengembalikan elemen teratas dari stack
}

// Fungsi untuk menghapus stack dan membebaskan memori yang digunakan
void deleteStack(Stack* stack) {
    free(stack->array); // Membebaskan memori array
    free(stack); // Membebaskan memori stack
}

// Fungsi untuk menyelesaikan masalah dua stack dengan batasan jumlah maksimum
int twoStacks(int maxSum, int a_count, int* a, int b_count, int* b) {
    int sum = 0, countA = 0, countB = 0, moves = 0; // Inisialisasi variabel-variabel

    // Membuat stack untuk kedua tumpukan
    Stack* stackA = createStack(a_count);
    Stack* stackB = createStack(b_count);

    // Memasukkan elemen-elemen dari tumpukan A ke dalam stackA hingga jumlahnya tidak melebihi maxSum
    while (countA < a_count && sum + a[countA] <= maxSum) {
        sum += a[countA];
        push(stackA, a[countA]);
        countA++;
    }

    // Memasukkan elemen-elemen dari tumpukan B ke dalam stackB jika jumlah mereka bersamaan dengan stackA tidak melebihi maxSum

```

```

    while (countB < b_count && !isFull(stackA)) {
        sum += b[countB];
        push(stackB, b[countB]);
        countB++;
        while (sum > maxSum && !isEmpty(stackA)) {
            sum -= pop(stackA);
            countA--;
        }
        if (sum <= maxSum && countA + countB > moves) moves = countA +
countB;
    }

    // Menghapus elemen dari stackB dan menambahkan ke stackA untuk
memaksimalkan jumlah elemen pada keduanya
    while (!isEmpty(stackB)) {
        sum += pop(stackB);
        countB--; //Mengurangi jumlah elemen dalam stackB karena satu elemen
telah dihapus.
        while (sum > maxSum && !isEmpty(stackA)) {
            sum -= pop(stackA);
            countA--; //Mengurangi jumlah elemen dalam stackA karena satu
elemen telah dihapus.
        }
        if (sum <= maxSum && countA + countB > moves) moves = countA +
countB;
    }

    // Menghapus kedua stack untuk membebaskan memori yang digunakan
    deleteStack(stackA);
    deleteStack(stackB);

    return moves; // Mengembalikan jumlah maksimum langkah yang mungkin
}

// Fungsi utama program
int main() {
    int g;
    scanf("%d", &g); // Meminta input jumlah kasus

    for (int g_itr = 0; g_itr < g; g_itr++) { // Loop untuk setiap kasus
        int n, m, maxSum;
        scanf("%d %d %d", &n, &m, &maxSum); // Meminta input jumlah elemen
dan batasan jumlah maksimum

        // Mengalokasikan memori dan mendapatkan input untuk tumpukan A
        int* a = malloc(n * sizeof(int));
        for (int i = 0; i < n; i++) {
            scanf("%d", &a[i]);

```

```

    }

    // Mengalokasikan memori dan mendapatkan input untuk tumpukan B
    int* b = malloc(m * sizeof(int));
    for (int i = 0; i < m; i++) {
        scanf("%d", &b[i]);
    }

    // Memanggil fungsi twoStacks untuk menyelesaikan masalah dan
    mencetak hasilnya
    int result = twoStacks(maxSum, n, a, m, b);
    printf("%d\n", result);

    // Membebaskan memori yang digunakan untuk array a dan b
    free(a);
    free(b);
}

return 0;
}

```

#### ➤ Penjelasan

Program ini menggunakan struktur data stack untuk mengimplementasikan operasi-operasi dasar seperti push (untuk menambahkan elemen ke dalam stack), pop (untuk menghapus dan mengembalikan elemen teratas dari stack), peek (untuk melihat elemen teratas dari stack tanpa menghapus), isEmpty (untuk memeriksa apakah stack kosong), dan isFull (untuk memeriksa apakah stack penuh).

Struktur Data Stack ini mendefinisikan struktur data 'stack' yang memiliki 3 variabel anggota yaitu 'top' sebagai indeks atas stack, 'capacity' sebagai kapasitas maksimum stack, dan array untuk menyimpan elemen stack. Kemudian ada fungsi-fungsi stack yaitu :

- createStack
- isEmpty dan isFull
- push
- pop
- peek
- deleteStack (menghapus stack dan membebaskan memori yang digunakan)

Kemudian ada function twoStack, function ini digunakan untuk menyelesaikan masalah dua stack dengan batasan jumlah maksimum. Fungsi ini mengambil argumen berupa jumlah maksimum ('maxSum'), jumlah elemen dalam stack A ('a\_count'), elemen-elemen stack A ('a'), jumlah elemen dalam stack B ('b\_count'), dan elemen-elemen stack B ('b').

Pada function 'main' yang merupakan program utama akan melakukan input-output dan memanggil fungsi 'twoStack' untuk setiap kasus yang diberikan dan akan mencetak hasilnya. Selain itu, program juga menggunakan alokasi memori dinamis untuk mengalokasikan memori yang diperlukan untuk stack dan array yang digunakan

dalam program ini. Setelah digunakan memori tersebut akan dibebaskan menggunakan fungsi 'free' untuk mencegah kebocoran memori.

### ➤ Output



```
PS D:\Struktur Data\praktikum6> cd "d:\Struktur Data\praktikum6\" ; if ($?) { gcc oth2.c -o oth2 } ; if ($?) { .\oth2 }
1
5 4 10
4 2 4 6 1
2 1 8 5
4
PS D:\Struktur Data\praktikum6>
```