

LAPORAN TUGAS
Linkin' _(park) & Game of Two Stacks
(SOAL No. 1 & 2)



Telkom
University

Laporan ini disusun dalam rangka
Penugasan Pemrograman Sistem dan Komputer

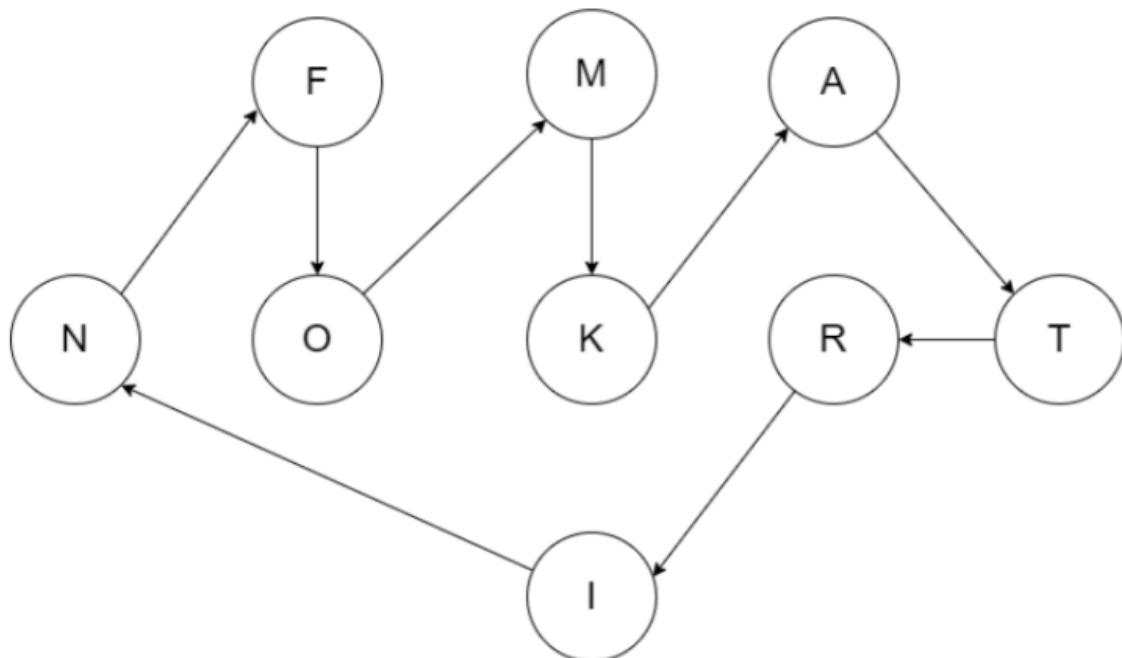
INFORMATIKA
IF 03-01

DANI ADINUGROHO
(1203230119)

FAKULTAS INFORMATIKA (FIF)
TELKOM UNIVERSITY SURABAYA

MARET 2024

SOAL 1 (Asisten Sherlock Holmes)



Pada setiap batu tersebut saling terhubung dengan tanda panah yang dimulai dari batu dengan huruf "N" dan seterusnya mengikuti arah panah. Ternyata, jika kita mengikuti arah panah dari batu dengan huruf "N", dan menggabungkan huruf-huruf pada batu-batu tersebut, kita akan membentuk sebuah kata yang sangat penting: "INFORMATIKA". Tapi tunggu, Sherlock Holmes memberimu sebuah catatan dalam bentuk kode yang akan membantumu menyelesaikan teka-teki ini. Kode tersebut menggunakan konsep **Self Referential Structures** untuk merepresentasikan hubungan antar batu-batu.

Dalam soal diatas, kita disuruh Sherlock untuk melakukan investigasi dimana ada berbagai batu dengan tiap huruf. Kita akan membentuk sebuah kata "INFORMATIKA" tetapi Sherlock Holmes memberi sebuah catatan yang bertulis "**Self Referential Structures**". Maksudnya, kita membuat sebuah *struct* untuk melakukan sebuah penghubungan pada suatu pointer.

Kententuan:

1. **Wajib** menggunakan potongan kode yang ada pada tabel diatas untuk inisialisasi (tidak ada penambahan / pengurangan ketika inisialisasi).
2. Pastikan ketika membuat linking / koneksi **harus sesuai** dengan gambar diatas (urutan dimulai dari huruf N dan mengikuti arah panah).
3. Lakukan semua akses data dengan menggunakan **I3 sebagai starting point**.

```

#include <stdio.h>

struct AsistenSherlockHolmes
{
    struct AsistenSherlockHolmes *link;
    char *alphabet;
};

int main(){
    struct AsistenSherlockHolmes 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";
}

```

(Gambar 1 Pembuatan Variabel Link)

1. Fungsi Struct AsistenSherlockHolmes, fungsi ini adalah gambaran dari single-linked list. Dengan menghubungkan batu-ke-batu lain. Nah struct AsistenSherlockHolmes *link, **AsistenSherlockHolmes** sendiri ini yang disebut self-referential structure dengan link nanti di main().
2. Isi main(), ada struct AsistenSherlockHolmes dkk variabel-nya, dilanjut dengan Peng-inisialisasi variable dan value. L(n).link ini berkerja sebagai linking ke variabel(n) lain. Sedangkan alphabet ini akan dilakukan pemanggilan pada tujuan link.

```
/* Melakukan penghubungan menggunakan pointer link terhadap Variable lain */
l7.link = &l1;
l1.link = &l8;
l8.link = &l2;
l2.link = &l5;
l5.link = &l3;
l3.link = &l6;
l6.link = &l9;
l9.link = &l4;
l4.link = &l7;
```

(Gambar 2 Penghubungan link Variabel)

1. Tiap l(n) yang terhubung, sesuai dengan petunjuk Sherlock Holmes sampai l(n) terakhir.

```
// Printout dengan menghubungkan awalan l3 menjadi "INFORMATIKA"
printf("%s", l3.link->link->link->alphabet);
printf("%s", l3.link->link->link->link->alphabet);
printf("%s", l3.link->link->link->link->link->link->alphabet);
printf("%s", l3.link->link->alphabet);
printf("%s", l3.link->link->link->link->link->link->link->alphabet);
printf("%s", l3.alphabet);
printf("%s", l3.link->link->link->link->link->link->link->link->link->alphabet);
printf("%s", l3.link->link->link->alphabet);
printf("%s", l3.link->link->link->link->link->link->link->alphabet);
printf("%s", l3.alphabet);

return 0;
```

(Gambar 3 Cara printout)

Nah, salah satu ketentuan yang diberikan tadi adalah kita memulai link dari l3, dengan caranya **l3.link** dan seterusnya, lalu diakhiri dengan **alphabet**. Jadi kesimpulan saat l3 ini dihubungkan dengan link ke-n kalinya, nilai dari l3 akan berubah seperti

Gambar 2. Inilah poin dari Pointer **Self-Referential Single Link**.

```
INFORMATIKA
PS C:\Users\nurul\projects\Semester 2\Tugas 8>
```

(Hasil dari sesuai petunjuk Sherlock Holmes)

SOAL 2 (Game Of Two Stacks)

2. Selesaikan soal pada link di bawah ini!

<https://www.hackerrank.com/challenges/game-of-two-stacks/problem>

Tambahkan **visualisasi** alur penyelesaiannya (per langkah) dalam bentuk stack pada laporan apabila input diubah menjadi sebagai berikut.

1
5 4 11
4 5 2 1 1
3 1 1 2



Jadi inti dari soal ini adalah kita melakukan perubahan pada. . .

$a = [1, 2, 3, 4, 5]$

$b = [6, 7, 8, 9]$

The maximum number of values Nick can remove is 4. There are two sets of choices with this result.

1. Remove 1, 2, 3, 4 from a with a sum of 10.
2. Remove 1, 2, 3 from a and 6 from b with a sum of 12.

Dimana ada dua Tumpukan, A dan B dengan nilai masing-masing. Lalu dengan penjumlahan **Sum**, ini sebagai berapa angka yang dapat diambil oleh Nicker. (Sebenarnya tidak harus 1, 2 , 3, 4 secara urut. Tapi dengan menambahkan $A(n = \text{top}) + B(n = \text{top}) + A... + B...$). Dengan menggunakan konsep **Stack**, maka ada beberapa gambaran mengenai stack.

Stack:

- Konsep LIFO(Last In First Out) atau FILO(First in Last Out).
- #define MAX_SIZE, sebagai Maks. Ukuran Stack.
- Push(), memasukkan suatu value ke stack->num[stack->top].
- Pop(), mengeluarkan nilai mulai dari tumpukan paling atas.
- Top()/Peek(), menunjukkan nilai teratas.

```

#include <stdio.h>

#define MAX_SIZE 10000 // Maksimum Ukuran Stack

typedef struct Stack{
    int num[MAX_SIZE];
    int top;
} Stack;

void push(Stack *stack, int value) {
    stack->top++;
    stack->num[stack->top] = value;
}

int pop(Stack *stack) {
    return stack->num[stack->top--];
}

int top(Stack *stack) {
    return stack->num[stack->top];
}

```

(Gambar 1 Tentang Stack)

```

int main() {
    Stack a, b;
    int n, m, g, maxSum;

    printf("Masukkan jumlah game: ");
    scanf("%d", &g);

    for (int i = 0; i < g; i++) {
        printf("Masukkan jumlah elemen stack a: ");
        scanf("%d", &n);
        printf("Masukkan jumlah elemen stack b: ");
        scanf("%d", &m);
        printf("Masukkan batas jumlah maksimum: ");
        scanf("%d", &maxSum);

        // Membaca elemen-elemen stack a
        for (int j = 0; j < n; j++) {
            scanf("%d", &a.num[j]);
        }
        // Membaca elemen-elemen stack b
        for (int j = 0; j < m; j++) {
            scanf("%d", &b.num[j]);
        }

        int result = twoStacks(maxSum, a, n, b, m);
        // Memanggil fungsi twoStacks dan mencetak hasilnya
        printf("Hasil game %d: %d\n", i + 1, result);
    }

    return 0;
}

```

(Gambar 2 Isi main)

1. Dalam perulangan **FOR** i, sebelumnya kita telah memasukkan sebuah banyaknya permainan yang akan kita mainkan. $g = \text{game}$.
2. Dan **FOR** j, disitu terdapat dua user-input a dan b, tetapi kita akan menggunakan pointer dari stack dengan variable **num**. Sebenarnya bisa dengan tanpa stack, **int a[n] dan b[m]**, maksudnya kita menginisialisasi panjang mereka dengan n&m.
3. **Int result** dengan value sebuah fungsi **twoStacks(maxSum, a, n, b, m)**, dimana value yang akan hasilkan berupa. . return **Bilang bulat**.

```
// Fungsi untuk menghitung jumlah maksimum elemen yang bisa diambil dari dua tumpukan
int twoStacks(int maxSum, Stack a, int a_count, Stack b, int b_count){
    int sum = 0, count = 0;
    Stack stackA, stackB;
    a.top = b.top = -1; // Inisialisasi nilai top untuk stack a dan b

    // Mengisi stack A dari input
    for (int i = a_count - 1; i >= 0; i--) {
        push(&stackA, a.num[i]);
    }
    // Mengisi stack B dari input
    for (int i = b_count - 1; i >= 0; i--) {
        push(&stackB, b.num[i]);
    }

    // Menentukan panjang tumpukan yang akan diproses
    int length = a_count >= b_count ? b_count : a_count;

    for (int i = 0; i < length; i++) {
        // Jika penambahan elemen stack A masih memenuhi batas maxSum
        if (sum + top(&stackA) <= maxSum) {
            sum += pop(&stackA); // Menambahkan nilai dari stack A
            printf("%d\n", sum);
            count++;
        }

        // Jika penambahan elemen stack B masih memenuhi batas maxSum
        if (sum + top(&stackB) <= maxSum) {
            sum += pop(&stackB); // Menambahkan nilai dari stack B
            printf("%d\n", sum);
            count++;
        }
    }
    return count; //return nilai count dari banyaknya penjumlahan
}
```

(Gambar 3 Fungsi twoStacks)

1. Int sum sebagai penjumlahan yang akan dilakukan dalam perulangan for (`int i = 0; i < length; i++)`, begitu juga count sebagai penanda bahwa terjadinya penambahan.
2. **a.top = b.top = -1**, sebagai inisialisasi Top A dan B, saat masuk ke **push(&stackA, a.num[i])** dan **push(&stackB, b.num[i])**, dan pada push A dan B memiliki panjang indeks(a_count & b_count) tersendiri.
3. Return count, mengembalikan nilai count ke main() Variabel **result**.

```
Masukkan jumlah game: 1
Masukkan jumlah elemen stack a: 5
Masukkan jumlah elemen stack b: 4
Masukkan batas jumlah maksimum: 12
1 2 3 4 5
6 7 8 9
1
7
9
12
Hasil game 1: 4
```

(Contoh dari soal Hackerrank)

```
Masukkan jumlah game: 1
Masukkan jumlah elemen stack a: 5
Masukkan jumlah elemen stack b: 4
Masukkan batas jumlah maksimum: 11
4 5 2 1 1
3 1 1 2
4
7
8
9
11
Hasil game 1: 5
```

(Hasil dari soal diatas)

Referensi

geeksforgeeks. *Self Referential Structures*. Diambil kembali dari GeeksforGeeks:
<https://www.geeksforgeeks.org/self-referential-structures/>