

Nama : Naufal Fahmi Ahnaf

Kelas : IF-03-01

NIM : 1203230034

1.(A)Source Code & Penjelasan Setiap Line/ Barisnya

```
1  #include <stdio.h>
2
3
4  // Mendefinisikan struct Batu untuk menyimpan alphabet
5  struct Batu {
6      char alphabet; // Menyimpan huruf pada batu
7      struct Batu *link; // Pointer ke Batu berikutnya dalam urutan
8  };
9
10 int main() {
11     // Inisialisasi batu sesuai dengan petunjuk
12     struct Batu l1, l2, l3, l4, l5, l6, l7, l8, l9;
13
14     l1.link = NULL;
15     l1.alphabet = 'F';
16
17     l2.link = NULL;
18     l2.alphabet = 'M';
19
20     l3.link = NULL;
21     l3.alphabet = 'A';
22
23     l4.link = NULL;
24     l4.alphabet = 'I';
25
26     l5.link = NULL;
27     l5.alphabet = 'K';
28
29     l6.link = NULL;
30     l6.alphabet = 'T';
31
32     l7.link = NULL;
33     l7.alphabet = 'N';
34
35     l8.link = NULL;
36     l8.alphabet = 'O';
37
38     l9.link = NULL;
39     l9.alphabet = 'R';
40
41
42     // Mengatur koneksi antar batu sesuai dengan urutan yang diberikan
43     l7.link = &l1;
44     l1.link = &l8;
45     l8.link = &l2;
46     l2.link = &l5;
47     l5.link = &l3;
48     l3.link = &l6;
49     l6.link = &l9;
50     l9.link = &l4;
51     l4.link = &l7;
52
53
54     // Mengakses huruf pada batu menggunakan l3 sebagai titik awal
55     printf("%c", l3.link->link->link->alphabet); // Output: "I"
56     printf("%c", l3.link->link->link->link->alphabet); // Output: "N"
57     printf("%c", l3.link->link->link->link->link->alphabet); // Output: "F"
58     printf("%c", l3.link->link->link->link->link->link->alphabet); // Output: "O"
59     printf("%c", l3.link->link->alphabet); // Output: "R"
60     printf("%c", l3.link->link->link->link->link->link->link->alphabet); // Output: "M"
61     printf("%c", l3.alphabet); // Output: "A"
62     printf("%c", l3.link->alphabet); // Output: "T"
63     printf("%c", l3.link->link->link->alphabet); // Output: "I"
64     printf("%c", l3.link->link->link->link->link->link->link->link->alphabet); // Output: "K"
65     printf("%c", l3.alphabet); // Output: "A"
66
67     return 0;
68 }
69
```

Tugas OTH Algoritma dan Struktur Data

(B) Outputnya :

```
PS C:\Users\TOSHIBA\Downloads\ASD Praktikum\Materi\Week6> cd "c:\Users\TOSHIBA\Downloads\ASD Praktikum\Materi\Week6" ; if ($?) { gcc Tgs6praktikum.c -o Tgs6praktikum } ; if ($?)  
{ .\Tgs6praktikum }  
INFORMATIKA  
PS C:\Users\TOSHIBA\Downloads\ASD Praktikum\Materi\Week6>
```

2.(A) Source Code & Penjelasan per line / perbarisnya dan visualisasinya

Tugas OTH Algoritma dan Struktur Data

```
1 #include <assert.h> //Header file untuk menambahkan fungsi assertion.
2 #include <stdbool.h> // Header file untuk menggunakan tipe data boolean (true dan false).
3 #include <stddef.h> //Header file untuk mendefinisikan beberapa tipe dan konstanta dasar.
4 #include <stdint.h> //Header file untuk mendefinisikan tipe data dengan lebar bit tertentu.
5 #include <stdio.h> //Header file untuk fungsi input-output standar.
6 #include <stdlib.h> // Header file untuk fungsi-fungsi umum, seperti alokasi memori dinamis.
7
8 // Struktur data untuk merepresentasikan stack
9 typedef struct {
10     int top; // Indeks atas stack
11     int capacity; // Kapasitas maksimum stack
12     int* array; // Array untuk menyimpan elemen-elemen stack
13 } Stack;
14
15 // Fungsi untuk membuat stack baru dengan kapasitas tertentu
16 Stack* createStack(int capacity) {
17     // Mengalokasikan memori untuk stack
18     Stack* stack = (Stack*)malloc(sizeof(Stack));
19     stack->capacity = capacity;
20     stack->top = -1; // Stack awalnya kosong
21     stack->array = (int*)malloc(stack->capacity * sizeof(int));
22     return stack;
23 }
24
25 // Fungsi untuk memeriksa apakah stack kosong
26 bool isEmpty(Stack* stack) {
27     return stack->top == -1;
28 }
29
30 // Fungsi untuk memeriksa apakah stack penuh
31 bool isFull(Stack* stack) {
32     return stack->top == stack->capacity - 1;
33 }
34
35 // Fungsi untuk menambahkan elemen ke dalam stack
36 void push(Stack* stack, int item) {
37     if (isFull(stack)) return; // Jika stack penuh, tidak melakukan apa-apa
38     stack->array[++stack->top] = item; // Menambahkan elemen ke dalam stack
39 }
40
41 // Fungsi untuk menghapus dan mengembalikan elemen teratas dari stack
42 int pop(Stack* stack) {
43     if (isEmpty(stack)) return -1; // Jika stack kosong, mengembalikan nilai -1
44     return stack->array[stack->top--]; // Menghapus dan mengembalikan elemen teratas dari stack
45 }
46
47 // Fungsi untuk melihat elemen teratas dari stack tanpa menghapusnya
48 int peek(Stack* stack) {
49     if (isEmpty(stack)) return -1; // Jika stack kosong, mengembalikan nilai -1
50     return stack->array[stack->top]; // Mengembalikan elemen teratas dari stack
51 }
52
53 // Fungsi untuk menghapus stack dan membebaskan memori yang digunakan
54 void deleteStack(Stack* stack) {
55     free(stack->array); // Membebaskan memori array
56     free(stack); // Membebaskan memori stack
57 }
58
59 // Fungsi untuk menyelesaikan masalah dua stack dengan batasan jumlah maksimum
60 int twoStacks(int maxSum, int a_count, int* a, int b_count, int* b) {
61     int sum = 0, countA = 0, countB = 0, moves = 0; // Inisialisasi variabel-variabel
62
63     // Membuat stack untuk kedua tumpukan
64     Stack* stackA = createStack(a_count);
65     Stack* stackB = createStack(b_count);
66
67     // Memasukkan elemen-elemen dari tumpukan A ke dalam stackA hingga jumlahnya tidak melebihi maxSum
68     while (countA < a_count && sum + a[countA] <= maxSum) {
69         sum += a[countA];
70         push(stackA, a[countA]);
71         countA++;
72     }
73
74     // Memasukkan elemen-elemen dari tumpukan B ke dalam stackB jika jumlah mereka bersamaan dengan stackA tidak melebihi maxSum
75     while (countB < b_count && !isFull(stackA)) {
76         sum += b[countB];
77         push(stackB, b[countB]);
78         countB++;
79         while (sum > maxSum && !isEmpty(stackA)) {
80             sum -= pop(stackA);
81             countA--;
82         }
83         if (sum <= maxSum && countA + countB > moves) moves = countA + countB;
84     }
85
86     // Menghapus elemen dari stackB dan menambahkan ke stackA untuk memaksimalkan jumlah elemen pada keduanya
87     while (!isEmpty(stackB)) {
88         sum += pop(stackB);
89         countB--; //Mengurangi jumlah elemen dalam stackB karena satu elemen telah dihapus.
90         while (sum > maxSum && !isEmpty(stackA)) {
91             sum -= pop(stackA);
92             countA--; //Mengurangi jumlah elemen dalam stackA karena satu elemen telah dihapus.
93         }
94         if (sum <= maxSum && countA + countB > moves) moves = countA + countB;
95     }
96
97     // Menghapus kedua stack untuk membebaskan memori yang digunakan
98     deleteStack(stackA);
99     deleteStack(stackB);
100
101     return moves; // Mengembalikan jumlah maksimum langkah yang mungkin
102 }
103
104 // Fungsi utama program
105 int main() {
106     int g;
107     scanf("%d", &g); // Meminta input jumlah kasus
108
109     for (int g_itr = 0; g_itr < g; g_itr++) { // Loop untuk setiap kasus
110         int n, m, maxSum;
111         scanf("%d %d %d", &n, &m, &maxSum); // Meminta input jumlah elemen dan batasan jumlah maksimum
112
113         // Mengalokasikan memori dan mendapatkan input untuk tumpukan A
114         int* a = malloc(n * sizeof(int));
115         for (int i = 0; i < n; i++) {
116             scanf("%d", &a[i]);
117         }
118
119         // Mengalokasikan memori dan mendapatkan input untuk tumpukan B
120         int* b = malloc(m * sizeof(int));
121         for (int i = 0; i < m; i++) {
122             scanf("%d", &b[i]);
123         }
124
125         // Memanggil fungsi twoStacks untuk menyelesaikan masalah dan mencetak hasilnya
126         int result = twoStacks(maxSum, n, a, m, b);
127         printf("%d\n", result);
128
129         // Membebaskan memori yang digunakan untuk array a dan b
130         free(a);
131         free(b);
132     }
133
134     return 0;
135 }
136
```

(B) Ouputnya :

```
PS C:\Users\TOSHIBA\Downloads\ASD Praktikum\Materi\Week6> cd "c:\Users\TOSHIBA\Downloads\ASD Praktikum\Materi\Week6"
s2praw7 }
1
5 4 11
4 5 2 1 1
3 1 1 2
5
PS C:\Users\TOSHIBA\Downloads\ASD Praktikum\Materi\Week6>
```

Setiap langkah dalam penyelesaian masalah dua stack dapat divisualisasikan sebagai berikut:

1. Inisialisasi Stack:

- Stack A dan Stack B dibuat kosong.
- Kapasitas maksimum kedua stack disesuaikan dengan input.

2. Memasukkan Elemen Stack A:

- Elemen-elemen dari tumpukan A dimasukkan ke dalam Stack A hingga jumlahnya tidak melebihi maxSum.

3. Memasukkan Elemen Stack B:

- Elemen-elemen dari tumpukan B dimasukkan ke dalam Stack B jika jumlah mereka bersamaan dengan Stack A tidak melebihi maxSum.
- Jika penambahan elemen dari Stack B menyebabkan jumlah melebihi maxSum, maka elemen-elemen dari Stack A akan dihapus satu per satu hingga jumlah kembali kurang dari atau sama dengan maxSum.

4. Memaksimalkan Jumlah Elemen:

- Elemen-elemen dari Stack B dihapus dan ditambahkan ke Stack A secara berurutan untuk memaksimalkan jumlah elemen di kedua stack.
- Selama proses ini, jumlah elemen di kedua stack diperiksa untuk memastikan bahwa jumlahnya tidak melebihi maxSum.
- Langkah-langkah ini diulangi hingga Stack B kosong.

5. Menghitung Langkah Maksimum:

- Setiap kali jumlah elemen di kedua stack memenuhi batasan maxSum dan jumlah total elemen melebihi nilai sebelumnya, langkah-langkah yang diambil disimpan sebagai langkah maksimum.

6. Menghapus dan Membebaskan Memori:

- Setelah menyelesaikan masalah, stack-stack yang telah dibuat dihapus dan memori yang digunakan dibebaskan.

Dengan demikian, algoritma ini secara iteratif mengoptimalkan penempatan elemen di kedua stack untuk mencapai jumlah maksimum dengan memperhatikan batasan yang diberikan.