Gabriel Chrisoberryl Guevara Kaawoan
1203230041
IF 03-03

1. Source Code :

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

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

int main() {
    // Deklarasi node-node
    struct Node l1, l2, l3, l4, l5, l6, l7, l8, l9;
    struct Node *link, *l3ptr;

    // Inisialisasi node-node dengan menggunakan potongan kode soal
    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";
```

```
    19.link = NULL;
    19.alphabet = "R";

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

    // Starting point
    l3ptr = &l7;

    // Akses data menggunakan printf
    printf("%s", l3.link->link->link->alphabet);// Menampilkan huruf I
       printf("%s",  l3.link->link->link->link->alphabet);//  Menampilkan
huruf N
            printf("%s",   l3.link->link->link->link->link->alphabet);//
Menampilkan huruf F
       printf("%s",  l3.link->link->link->link->link->link->alphabet);//
Menampilkan huruf O
    printf("%s", l3.link->link->alphabet);// Menampilkan huruf R
                                                          printf("%s",
l3.link->link->link->link->link->link->link->alphabet);//    Menampilkan
huruf M
    printf("%s", l3.alphabet);// Menampilkan huruf A
    printf("%s", l3.link->alphabet);// Menampilkan huruf T
    printf("%s", l3.link->link->link->alphabet);// Menampilkan huruf I
                                                          printf("%s",
l3.link->link->link->link->link->link->link->link->alphabet);//
Menampilkan huruf K
    printf("%s", l3.alphabet);// Menampilkan huruf A


    return 0;

}
```

Penjelasan Source Code :

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

Baris ini adalah bagian dari preprocessor directive.

- **#include <stdio.h>** : Mendukung fungsi input-output standar seperti **printf()**.
- **#include <stdlib.h>** : Mendukung alokasi memori dinamis seperti **malloc()**.

```
struct Node {
    char* alphabet;
    struct Node* link;
};
```

Di sini, kita mendefinisikan sebuah struktur bernama **Node**, yang memiliki dua anggota:

- **alphabet**, yang merupakan pointer ke karakter. Ini akan menyimpan karakter yang diasosiasikan dengan node tersebut.
- **link**, yang adalah pointer ke struktur Node itu sendiri. Ini akan digunakan untuk menautkan node-node dalam linked list.

```
int main() {
    // Deklarasi node-node
    struct Node l1, l2, l3, l4, l5, l6, l7, l8, l9;
    struct Node *link, *l3ptr;
```

- Variabel **l1** hingga **l9** adalah instance dari struktur **Node**, yang mewakili node-node dalam linked list.
- **link** adalah pointer yang akan digunakan untuk menavigasi melalui linked list.
- **l3ptr** adalah pointer yang digunakan untuk menunjukkan ke node tertentu dalam linked list.

```
    // Inisialisasi node-node dengan menggunakan potongan kode soal
    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";
```

- Setiap node diinisialisasi dengan menetapkan **link** ke **NULL** karena belum dihubungkan ke node lain.
- **alphabet** diatur sesuai dengan karakter yang diberikan.

```
    // Mengatur koneksi antar node sesuai dengan urutan yang diinginkan
    l7.link = &l1;// Menyambungkan ke l1
    l1.link = &l8;// Menyambungkan ke l1
    l8.link = &l2;// Menyambungkan ke l1
    l2.link = &l5;// Menyambungkan ke l1
    l5.link = &l3;// Menyambungkan ke l1
    l3.link = &l6;// Menyambungkan ke l1
    l6.link = &l9;
    l9.link = &l4;
    l4.link = &l7;
```

- Setiap node dihubungkan dengan node berikutnya dengan menetapkan alamat node tersebut ke anggota **link**. Dengan demikian, kita membuat suatu linked list.

```
    // Starting point
    l3ptr = &l7;
```

- **l3ptr** menunjukkan ke node yang akan menjadi titik awal navigasi dalam linked list.

```
// Akses data menggunakan printf
    printf("%s", l3.link->link->link->alphabet);// Menampilkan huruf I
      printf("%s",  l3.link->link->link->link->alphabet);//  Menampilkan
huruf N
           printf("%s",   l3.link->link->link->link->link->alphabet);//
Menampilkan huruf F
       printf("%s",  l3.link->link->link->link->link->link->alphabet);//
Menampilkan huruf O
    printf("%s", l3.link->link->alphabet);// Menampilkan huruf R
                                                          printf("%s",
l3.link->link->link->link->link->link->link->alphabet);//    Menampilkan
huruf M
    printf("%s", l3.alphabet);// Menampilkan huruf A
    printf("%s", l3.link->alphabet);// Menampilkan huruf T
    printf("%s", l3.link->link->link->alphabet);// Menampilkan huruf I
```

```
                                                        printf("%s",
l3.link->link->link->link->link->link->link->link->alphabet);//
Menampilkan huruf K
    printf("%s", l3.alphabet);// Menampilkan huruf A


    return 0;

}
```

- Melalui **l3ptr**, kita dapat mengakses data dalam linked list dengan menggunakan operator panah (**->**) untuk mengakses anggota **link** dari setiap node.
- Dalam setiap **printf**, kita mengakses data dari node-node tertentu dalam linked list, sehingga mencetak urutan karakter yang telah ditentukan.
- Dengan demikian, setelah menjalankan program ini, akan dicetak serangkaian huruf sesuai dengan urutan akses yang ditentukan dalam **printf**.

Hasil Run Program :

```
PS C:\Users\varry> cd "c:\Users\varry\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
INFORMATIKA
PS C:\Users\varry>
```

2. Source Code :

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'twoStacks' function below.
```

```
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER maxSum
 *  2. INTEGER_ARRAY a
 *  3. INTEGER_ARRAY b
 */

int twoStacks(int maxSum, int a_count, int* a, int b_count, int*
b) {
    int a_index = 0, b_index = 0;
    int count = 0, sum = 0;
        // Hitung berapa banyak elemen dari stack a yang dapat
diambil
    while (a_index < a_count && sum + a[a_index] <= maxSum) {
        sum += a[a_index];
        a_index++;
        count++;
    }
    int maxCount = count;
    // Coba menambahkan elemen dari stack b
    while (b_index < b_count && a_index >= 0) {
        sum += b[b_index];
        b_index++;
           // Kurangi elemen dari stack a sampai totalnya kurang
dari atau sama dengan maxSum
        while (sum > maxSum && a_index > 0) {
            a_index--;
            sum -= a[a_index];
        }
            // Periksa apakah jumlah elemen saat ini lebih besar
dari yang sudah diperoleh
        if (sum <= maxSum && count < a_index + b_index) {
            count = a_index + b_index;
        }
    }
    return count;
}

int main()
{
    // FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");
```

```
    int g = parse_int(ltrim(rtrim(readline())));

    for (int g_itr = 0; g_itr < g; g_itr++) {
                                char**  first_multiple_input  =
split_string(rtrim(readline()));

        int n = parse_int(*(first_multiple_input + 0));

        int m = parse_int(*(first_multiple_input + 1));

        int maxSum = parse_int(*(first_multiple_input + 2));

        char** a_temp = split_string(rtrim(readline()));

        int* a = malloc(n * sizeof(int));

        for (int i = 0; i < n; i++) {
            int a_item = parse_int(*(a_temp + i));

            *(a + i) = a_item;
        }

        char** b_temp = split_string(rtrim(readline()));

        int* b = malloc(m * sizeof(int));

        for (int i = 0; i < m; i++) {
            int b_item = parse_int(*(b_temp + i));

            *(b + i) = b_item;
        }

        int result = twoStacks(maxSum, n, a, m, b);

        // fprintf(fptr, "%d\n", result);
        printf("%d\n", result);

        free(a);
        free(b);
    }

    // fclose(fptr);
```

```
            return 0;
        }

        char* readline() {
            size_t alloc_length = 1024;
            size_t data_length = 0;

            char* data = malloc(alloc_length);

            while (true) {
                char* cursor = data + data_length;
                  char* line = fgets(cursor, alloc_length - data_length,
stdin);

                if (!line) {
                    break;
                }

                data_length += strlen(cursor);

                 if (data_length < alloc_length - 1 || data[data_length -
1] == '\n') {
                    break;
                }

                alloc_length <<= 1;

                data = realloc(data, alloc_length);

                if (!data) {
                    data = NULL;

                    break;
                }
            }

            if (data[data_length - 1] == '\n') {
                data[data_length - 1] = '\0';

                data = realloc(data, data_length);

                if (!data) {
                    data = NULL;
```

```
            }
        } else {
            data = realloc(data, data_length + 1);

            if (!data) {
                data = NULL;
            } else {
                data[data_length] = '\0';
            }
        }

        return data;
    }

    char* ltrim(char* str) {
        if (!str) {
            return NULL;
        }

        if (!*str) {
            return str;
        }

        while (*str != '\0' && isspace(*str)) {
            str++;
        }

        return str;
    }

    char* rtrim(char* str) {
        if (!str) {
            return NULL;
        }

        if (!*str) {
            return str;
        }

        char* end = str + strlen(str) - 1;

        while (end >= str && isspace(*end)) {
            end--;
```

```
        }

        *(end + 1) = '\0';

        return str;
    }

    char** split_string(char* str) {
        char** splits = NULL;
        char* token = strtok(str, " ");

        int spaces = 0;

        while (token) {
            splits = realloc(splits, sizeof(char*) * ++spaces);

            if (!splits) {
                return splits;
            }

            splits[spaces - 1] = token;

            token = strtok(NULL, " ");
        }

        return splits;
    }

    int parse_int(char* str) {
        char* endptr;
        int value = strtol(str, &endptr, 10);

        if (endptr == str || *endptr != '\0') {
            exit(EXIT_FAILURE);
        }

        return value;
    }
```

Penjelasan Source Code :

1. **Inisialisasi Tumpukan (Stacks) :**
   - Inisialisasi tumpukan pertama (Stack A) dengan elemen :
     4 5 2 1 1
   - Inisialisasi tumpukan kedua (Stack B) dengan elemen:
     3 1 1 2

2. **Pengambilan Elemen dari Stack A :**
   - Lakukan pengambilan elemen dari Stack A sampai jumlahnya melebihi maxSum atau hingga Stack A kosong :
     - ❖ Ambil elemen 4. Total sum menjadi 4.
     - ❖ Ambil elemen 5. Total sum menjadi 9.
     - ❖ Ambil elemen 2. Total sum menjadi 11.

3. **Pengambilan Elemen dari Stack B :**
   - Setelah mengambil semua elemen dari Stack A yang memenuhi kriteria, lakukan pengambilan elemen dari Stack B satu per satu sambil mempertimbangkan total sum :
     - ❖ Ambil elemen 3. Total sum menjadi 14.
     - ❖ Ambil elemen 1. Total sum menjadi 15.
     - ❖ Ambil elemen 1. Total sum menjadi 16.
     - ❖ Ambil elemen 2. Total sum menjadi 18.

4. **Pemeriksaan Kembali Total Sum :**
   - Total sum sekarang telah melebihi maxSum. Oleh karena itu, perlu dilakukan pengurangan elemen dari Stack A untuk memastikan bahwa total sum tidak melebihi maxSum.

5. **Pengurangan Elemen dari Stack A :**
   - Lakukan pengurangan elemen dari Stack A satu per satu hingga total sum tidak melebihi maxSum :
     - ❖ Hapus elemen 2 dari Stack A. Total sum menjadi 16.
     - ❖ Total sum masih melebihi maxSum, lanjutkan pengurangan:
     - ❖ Hapus elemen 1 dari Stack A. Total sum menjadi 15.
     - ❖ Sekarang total sum tidak melebihi maxSum, maka pengurangan dihentikan.

6. **Hasil Akhir :**
   - Setelah mengambil dan mengurangi elemen dari kedua Stack, hasil akhirnya adalah:
     Stack A : 4 5
     Stack B : 3 1 1

     Total elemen yang diambil dari kedua Stack adalah 5.

Dalam proses ini, strategi yang digunakan adalah mengambil elemen dari Stack A sebanyak mungkin sampai total sum melebihi maxSum, kemudian mempertimbangkan elemen dari Stack B sambil memastikan total sum tidak melebihi maxSum, dan terakhir melakukan pengurangan dari Stack A jika diperlukan untuk memastikan total sum tidak melebihi maxSum.

Hasil Run Program :