Nama : mohammad rafi priyadi

Kelas : IF 03 02

Nim : 1203230060

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

struct Stone {
    char abjad;
    struct Stone *next;
};

void addStone(struct Stone **head, char abjad) {
    struct Stone *newStone = (struct Stone *)malloc(sizeof(struct Stone));
    newStone->abjad = abjad;
    newStone->next = 0;

    if (*head == 0) {
        *head = newStone;
        return;
    }

    struct Stone *restStone = *head;
    while (restStone->next !=0) {
        restStone = restStone->next;
    }
    restStone->next = newStone;
}

void printStones(struct Stone *TOP) {
    struct Stone *current = TOP;
    while (current != 0) {
        printf("%c", current->abjad);
        current = current->next;
    }
}

int main() {
    struct Stone *TOP = 0;
    // Menambahkan ke linked list
    addStone(&TOP, 'I');addStone(&TOP, 'N');  addStone(&TOP, 'F');
     addStone(&TOP, 'O');addStone(&TOP, 'R'); addStone(&TOP, 'M');
  addStone(&TOP, 'A'); addStone(&TOP, 'T');  addStone(&TOP, 'I');
    addStone(&TOP, 'K'); addStone(&TOP, 'A')
    // Mencetak huruf
    printStones(TOP);
```

```
    struct Stone *current = TOP;
    struct Stone *next;
    while (current != 0) {
        next = current->next;
        (current);
        current = next;
    }

    return 0;
}
```

Baris 4-7:

Mendefinisikan struktur data Stone yang memiliki dua elemen:

abjad: Karakter untuk menyimpan huruf

next: Pointer untuk menunjuk ke batu berikutnya dalam linked list

Baris 9-29:

Mendefinisikan fungsi addStone untuk menambahkan batu baru ke linked list:

Menerima pointer ke pointer head dan karakter abjad sebagai input

Mengalokasikan memori untuk batu baru

Memeriksa apakah alokasi memori berhasil

Mengisi nilai abjad dan next batu baru

Menambahkan batu baru ke linked list:

Jika linked list kosong, batu baru menjadi kepala linked list

Jika linked list tidak kosong, temukan batu terakhir dan tambahkan batu baru di akhir

Baris 31-39:

Mendefinisikan fungsi printStones untuk mencetak karakter dari linked list:

Menerima pointer TOP ke kepala linked list sebagai input

Iterasi melalui linked list dan mencetak karakter dari setiap batu

```
INFORMATIKA
```

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
```

```c
#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 *);


int twoStacks(int maxSum, int a_count, int *a, int b_count, int *b)
{
    int i = 0, j = 0, sum = 0, count = 0;
    while (i < a_count && sum + a[i] <= maxSum)
    {
        sum += a[i];
        i++;
    }
    count = i;
    while (j < b_count && i >= 0)
    {
        sum += b[j];
        j++;
        while (sum > maxSum && i > 0)
        {
            i--;
            sum -= a[i];
        }
        if (sum <= maxSum && i + j > count)
        {
            count = i + j;
        }
    }
    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);
    }

    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 = '\0';

            break;
        }
    }

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

        data = realloc(data, data_length);

        if (!data)
        {
            data = '\0';
        }
    }
    else
    {
        data = realloc(data, data_length + 1);

        if (!data)
        {
            data = '\0';
        }
        else
        {
```

```
            data[data_length] = '\0';
        }
    }

    return data;
}

char *ltrim(char *str)
{
    if (!str)
    {
        return '\0';
    }

    if (!*str)
    {
        return str;
    }

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

    return str;
}

char *rtrim(char *str)
{
    if (!str)
    {
        return '\0';
    }

    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;
}
```

readline()

Fungsi: Membaca baris input dari pengguna.

Deskripsi:

Mengalokasikan memori untuk menampung input pengguna.

Membaca baris input menggunakan fgets.

Menyesuaikan ukuran memori yang dialokasikan jika diperlukan.

Mengembalikan pointer ke baris input yang dibaca.

2. ltrim()

Fungsi: Menghilangkan spasi di awal string.

Deskripsi:

Mengembalikan pointer ke string asli jika sudah tanpa spasi di awal.

Melompati karakter spasi di awal string hingga menemukan karakter non-spasi.

Mengembalikan pointer ke karakter non-spasi tersebut.

3. rtrim()

Fungsi: Menghilangkan spasi di akhir string.

Deskripsi:

Mengembalikan pointer ke string asli jika sudah tanpa spasi di akhir.

Menemukan karakter terakhir string.

Melompati karakter spasi di akhir string mundur dari belakang.

Menambahkan karakter null terminator (\0) di posisi setelah karakter non-spasi terakhir.

Mengembalikan pointer ke string yang sudah di-trim.

4. split_string()

Fungsi: Membagi string menjadi array of string berdasarkan pembatas.

Deskripsi:

Mengalokasikan memori untuk array of string.

Menggunakan strtok untuk memecah string berdasarkan pembatas.

Menambahkan setiap token (bagian yang dipisahkan) ke dalam array of string.

Mengembalikan pointer ke array of string yang dihasilkan.

5. parse_int()

Fungsi: Mengubah string representasi integer menjadi integer sebenarnya.

Deskripsi:

Menggunakan strtol untuk mengonversi string menjadi integer.

Memeriksa apakah konversi berhasil.

Mengembalikan nilai integer jika konversi berhasil, atau keluar dari program jika gagal.

Input (stdin)

```
1
5 4 10
4 2 4 6 1
2 1 8 5
```

Your Output (stdout)

```
4
```

Expected Output

```
4
```