

Idil Haq Al Farisi

1203230070

Informatika

1

```
C ujicoba.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct Node {
5      char *alphabet;
6      struct Node *link;
7  } Node;
8
9  int main() {
10     Node l1, l2, l3, l4, l5, l6, l7, l8, l9, l10, l11;
11
12     // Inisialisasi
13     l1.alphabet = "I";
14     l2.alphabet = "N";
15     l3.alphabet = "F";
16     l4.alphabet = "O";
17     l5.alphabet = "R";
18     l6.alphabet = "M";
19     l7.alphabet = "A";
20     l8.alphabet = "T";
21     l9.alphabet = "I";
22     l10.alphabet = "K";
23     l11.alphabet = "A";
24
25     // Menghubungkan node sesuai dengan instruksi yang diberikan
26     l1.link = &l2;
27     l2.link = &l3;
28     l3.link = &l4;
29     l4.link = &l5;
30     l5.link = &l6;
31     l6.link = &l7;
32     l7.link = &l8;
33     l8.link = &l9;
34     l9.link = &l10;
35     l10.link = &l11;
36     l11.link = NULL;
37
38     // Mengakses data menggunakan l1 sebagai titik awal
39     Node* temp = &l1;
40     while(temp != NULL) {
41         printf("%s", temp->alphabet);
42         temp = temp->link;
43     }
44
45     return 0;
46
47 }
```

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

Berfungsi untuk mengimpor library untuk fungsi fungsi standar

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

Bagian ini berfungsi untuk mendeklarasikan struct Node yang isinya yaitu alphabet dan link. Alphabet sendiri adalah karakter (char) dan link adalah merupakan pointer ke node yang lain. Typedef berfungsi untuk membuat “alias” untuk struct node, sehingga nanti bisa menggunakan node sebagai tipe data.

```
int main() {
```

Baris ini merupakan bagian utama program yang pertama kali dijalankan.

```
Node l1, l2, l3, l4, l5, l6, l7, l8, l9;
```

Baris ini mendeklarasikan variabel dari tipe data Node dari struct node sebelumnya. Setiap variabel di dalamnya akan dipakai sebagai elemen dalam linked list

```
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';
```

Baris ini berfungsi untuk menginisialisasi setiap Node dengan karakter tertentu dan mengatur/men-set link semuanya ke NULL. Ini berfungsi untuk mengisi bagian alphabet dari setiap node-node dan memastikan link node nya tidak mem-point (menunjuk) ke mana pun (bernilai NULL).

```
l7.link = &l1; //N -> F
l1.link = &l8; //F -> O
l8.link = &l2; //O -> M
l2.link = &l5; //M -> K
l5.link = &l3; //K -> A
l3.link = &l6; //A -> T
l6.link = &l9; //T -> R
l9.link = &l4; //R -> I
l4.link = &l7; //I -> N
```

Bagian Ini Menghubungkan setiap Node yang lainnya, untuk membentuk sebuah linked list. Ini dibuat dengan mengatur bagian link dari setiap node mem-point (menunjuk) ke node lainnya. Urutannya sendiri nantinya akan menghasilkan kata INFORMATIKA.

```
char word[] = {
    l3.link->link->link->alphabet,
    l3.link->link->link->link->alphabet,
    l3.link->link->link->link->link->alphabet,
    l3.link->link->link->link->link->link->alphabet,
    l3.link->link->alphabet,
    l3.link->link->link->link->link->link->link->alphabet,
    l3.link->link->link->link->link->link->link->link->link->link->alphabet,
    l3.link->link->link->link->link->link->link->link->link->link->link->
>alphabet,
    l3.link->link->link->alphabet,
    l3.link->link->link->link->link->link->link->link->alphabet,
    l3.link->link->link->link->link->link->link->link->link->alphabet
};
```

Baris ini membuat dan mendeklarasikan array karakter (char) word yang berisi karakter dari setiap node dalam linked list dan diakses melalui link dari node l3. Urutannya membentuk kata INFORMATIKA.

```
printf("%s", word);
```

Baris ini berfungsi untuk mencetak kata INFORMATIKA.

```
return 0;
```

Baris ini berfungsi untuk mengakhiri program dan mengembalikan nilai 0 yang mana menunjukkan bahwa program telah sukses dijalankan.

Output



```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS C:\Users\USER\Desktop\ALQ> cd "c:\Users\USER\Desktop\ALQ\" ; if ($?) { gcc oth1.c -o oth1 } ; if ($?) { .\oth1 }
INFORMATIKA
PS C:\Users\USER\Desktop\ALQ>
```

```

#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*);
int twoStacks(int maxSum, int a_count, int* a, int b_count, int* b) {
    int sum = 0, count = 0, temp = 0, i = 0, j = 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(fp_ptr, "%d\n", result);
}
fclose(fp_ptr);
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;
}

```

```

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

```

Bagian ini berfungsi untuk mendefinisikan dan mendeklarasikan function (fungsi) yang akan dijalankan program, fungsi yang digunakan untuk membaca input, menghilangkan/memotong spasi di awal dan akhir string, membagi string menjadi beberapa string dan mengubah string menjadi integer.

```

int twoStacks(int maxSum, int a_count, int* a, int b_count, int* b) {
    int sum = 0, count = 0, temp = 0, i = 0, j = 0;
    while(i < a_count && sum + a[i] <= maxSum) { //picking from stack a
        sum += a[i];
        i++;
    }
    count = i; //count of picked elements
    while(j < b_count && i >= 0) { //picking from stack b
        sum += b[j];
        j++;
        while(sum > maxSum && i > 0) { //if sum exceeds maxSum, remove
elements from stack a
            i--;
            sum -= a[i];
        }
        if(sum <= maxSum && i + j > count) {
            count = i + j;
        }
    }
    return count;
}

```

Baris ini mendeklarasikan function (fungsi) twoStacks. Fungsi ini menerima batas jumlah maksimum (maxSum), jumlah elemen dalam dua stack (a_count dan b_count), dan dua stack itu sendiri (a dan b). Fungsi ini mengembalikan jumlah elemen maksimum yang dapat diambil dari dua stack tersebut tanpa melebihi maxSum.

```
int main()  
{
```

Baris ini merupakan bagian utama program yang pertama kali dijalankan. Fungsi ini membaca input, memanggil fungsi twoStacks dengan input yang sesuai, dan mencetak output.


```

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;
}

```

Bagian ini berfungsi untuk mendefinisikan fungsi-fungsi yang telah didefinisikan sebelumnya. Fungsi-fungsi ini digunakan untuk membaca input, memotong spasi di awal dan akhir string, membagi string menjadi beberapa string, dan mengubah string menjadi integer.

Kode ini membaca input berupa dua stack dan batas jumlah maksimum, mencari jumlah elemen maksimum yang dapat diambil dari dua stack tersebut tanpa melebihi batas tersebut, dan mencetak hasilnya. Kode ini juga menunjukkan penggunaan struct data stack dan search, serta penggunaan fungsi untuk membaca dan memproses input.

Jika Input :

```

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

```

1. dimulai dengan stack a dan b kosong. Maksimum jumlah yang bisa atau diizinkan adalah 11.

```

a: []
b: []
sum: 0
count: 0

```

-
2. Tambahkan elemen dari stack a ke dalam sum selama sum tidak melebihi 11.

```
a: [4, 5]
b: []
sum: 9
count: 2
```

-
-
3. Tambahkan elemen dari stack b ke dalam sum. Jika sum melebihi 11, hapus elemen dari awal a.

```
a: [4, 5]
b: [3]
sum: 12
count: 3
```

Karena sum melebihi 11, hapus elemen dari awal a dan kurangi sum dengan elemen tersebut.

```
a: [5]
b: [3]
sum: 8
count: 2
```

-
-
-
4. Ulangi langkah 3 sampai semua elemen di b telah ditambahkan.

```
a: [5]
b: [3, 1]
sum: 9
count: 3
a: [5]
b: [3, 1, 1]
sum: 10
count: 4
a: []
b: [3, 1, 1, 2]
sum: 12
count: 5
```

Karena sum melebihi 11, hapus elemen dari awal a dan kurangi sum dengan elemen tersebut. Karena a sudah kosong, maka tidak bisa menghapus elemen lagi.

```
a: []
b: [3, 1, 1, 2]
sum: 12
count: 5
```

-
-
-
-
5. count adalah jumlah elemen maksimum yang bisa kita ambil dari kedua stack tanpa melebihi maxSum. Dalam input ini, output count adalah 5.

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

Compilation Successful :)

Click the Submit Code button to run your code against all the test cases.

Input (stdin)

[Download](#)

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

Your Output (stdout)

```
1 5
```

Congratulations

You solved this challenge. Would you like to challenge your friends?



[Next Challenge](#)

✓ Test case 0

✓ Test case 1

✓ Test case 2

✓ Test case 3

✓ Test case 4

✓ Test case 5

✓ Test case 6

Compiler Message

Success

Input (stdin)

[Download](#)

```
1 1
2 5 4 10
3 4 2 4 6 1
4 2 1 8 5
```

Expected Output


[Download](#)


```
1 4
```


Congratulations


You solved this challenge. Would you like to challenge your friends? [f](#) [t](#) [in](#)


[Next Challenge](#)


✓ Test case 7 


✓ Test case 8 

✓ Test case 9 

✓ Test case 10 

✓ Test case 11 

✓ Test case 12 

✓ Test case 13 

Compiler Message

Success

Input (stdin)

[Download](#)

```
1 1
2 5 4 10
3 4 2 4 6 1
4 2 1 8 5
```

Expected Output

[Download](#)

```
1 4
```