

SOAL LATIHAN ASD- STRUCT DAN STACK

Week 6

Nama : Nabila Amilatul Jannah

Kelas : IF 03-02

NIM : 1203230103

```
#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 sesuai dengan petunjuk
    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";

    // Mengatur koneksi antar node sesuai dengan urutan yang diberikan
    l7.link = &l1;
    l1.link = &l8;
    l8.link = &l2;
    l2.link = &l5;
```

```

15.link = &l3;
13.link = &l6;
16.link = &l9;
19.link = &l4;
14.link = &l7;

// Starting point
l3ptr = &l7;

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

return 0;
}

```

Penjelasan Kode

```

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

```

Kode di atas mendeklarasikan sebuah struct **Node** yang memiliki dua anggota: **alphabet** yang merupakan pointer ke **char**, dan **link** yang merupakan pointer ke struktur **Node** link

```

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

```

di fungsi main() ini beberapa variabel dari tipe struct Node dideklarasikan: l1, l2, l3, l4, l5, l6, l7, l8, dan l9. Ini adalah node-node yang akan digunakan. Variabel lain yang dideklarasikan adalah link dan l3ptr. link adalah pointer ke struktur Node yang akan digunakan untuk menyimpan alamat dari node lain. l3ptr adalah pointer ke struktur Node yang nantinya akan digunakan untuk mengakses atau menunjuk ke l3.

```
// Inisialisasi node-node sesuai dengan petunjuk
11.link = NULL;
11.alphabet = "F";
12.link = NULL;
12.alphabet = "M";
13.link = NULL;
13.alphabet = "A";
14.link = NULL;
14.alphabet = "I";
15.link = NULL;
15.alphabet = "K";
16.link = NULL;
16.alphabet = "T";
17.link = NULL;
17.alphabet = "N";
18.link = NULL;
18.alphabet = "O";
19.link = NULL;
19.alphabet = "R";
```

kode yang melakukan inisialisasi node-node yang telah dideklarasikan sebelumnya. Setiap node diinisialisasi dengan nilai ke 'alphabet' dan 'link'

Setiap node diberi 'link' yang menunjuk ke 'NULL' yang menandakan bahwa setiap node belum memiliki node terhubung setelahnya. Nilai string yang sesuai ditugaskan ke anggota 'alphabet' dari setiap node. Misalnya 'I1' diberi nilai F, 'I2' diberi nilai M dan seterusnya

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

Kode ini mengatur koneksi antara node-node yang telah diinisialisasi sebelumnya, sesuai dengan urutan yang diberikan. Misalnya,

17.link = &l1;: Menghubungkan node 17 dengan node 11. Artinya, node 17 akan memiliki koneksi yang menunjuk ke node 11.

11.link = &l8;: Menghubungkan node 11 dengan node 18. Artinya, node 11 akan memiliki koneksi yang menunjuk ke node 18.

Dan Seterusnya

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

I3ptr dipersiapkan sebagai titik awal untuk melakukan iterasi pada kode ini. I3ptr menjadi suatu Alamat memori yang menyimpan I7. Misalnya, kita bisa menggunakan I3ptr untuk melakukan iterasi lain yang melibatkan node I7 dan node yang terhubung lainnya.

```
// Akses data menggunakan printf dan I3 sebagai titik awal
printf("%s", l3.link->link->link->alphabet); // Output huruf I
printf("%s", l3.link->link->link->link->alphabet); // Output huruf N
printf("%s", l3.link->link->link->link->link->alphabet); // Output huruf F
printf("%s", l3.link->link->link->link->link->link->alphabet); // Output
huruf O
printf("%s", l3.link->link->alphabet); // Output huruf R
printf("%s", l3.link->link->link->link->link->link->link->alphabet); //
Output huruf M
printf("%s", l3.alphabet); // Output huruf A
printf("%s", l3.link->alphabet); // Output huruf T
printf("%s", l3.link->link->link->alphabet); // Output huruf I
printf("%s", l3.link->link->link->link->link->link->link->link->
>alphabet); // Output huruf K
printf("%s", l3.alphabet); // Output huruf A
```

Kode program tersebut mengakses data dari sebuah struktur yang mungkin disebut l3. Struktur tersebut memiliki properti link yang merupakan pointer ke struktur lainnya dengan properti alphabet yang mungkin berisi sebuah karakter.

printf("%s", l3.link->link->link->alphabet); - Program mencetak huruf I. Ini berarti program mengakses properti alphabet dari struktur yang di-referensi oleh pointer link sebanyak tiga kali.

printf("%s", l3.link->link->link->link->alphabet); - Program mencetak huruf N. Program mengakses properti alphabet dari struktur yang di-referensi oleh pointer link sebanyak empat kali.

printf("%s", l3.link->link->link->link->link->alphabet); - Program mencetak huruf F. Program mengakses properti alphabet dari struktur yang di-referensi oleh pointer link sebanyak lima kali.

Dan seterusnya

Input

```
File Edit Selection View Go Run ... TugasPointer
#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 sesuai dengan petunjuk
    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";
}
```

```
File Edit Selection View Go Run ... TugasPointer
int main() {
    l9.alphabet = "K";

    // Mengatur koneksi antar node sesuai dengan urutan yang diberikan
    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;

    // Starting point
    l3ptr = &l7;

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

    return 0;
}
```

Output

```
#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 sesuai dengan petunjuk
    l1.link = NULL;
    l1.alphabet = "F";
    l2.link = NULL;
```

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

2.

Sample Input 0

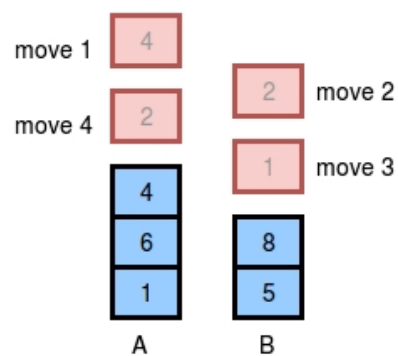
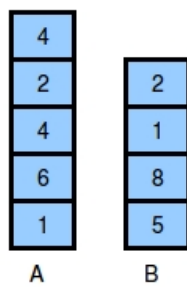
```
1
5 4 10
4 2 4 6 1
2 1 8 5
```

Sample Output 0

```
4
```

Explanation 0

The two stacks initially look like this:



```

#include <stdio.h>

int twoStacks(int maxSum, int a[], int n, int b[], int m) {
    int sum = 0, count = 0, i = 0, j = 0;

    // Hitung elemen stack pertama yang dapat dimasukkan
    while (i < n && (sum += a[i++]) <= maxSum) count++;

    // Tambahkan elemen stack kedua dan sesuaikan dengan batas maksimum
    while (j < m && i >= 0) {
        sum += b[j++];
        while (sum > maxSum && i > 0) sum -= a[--i];
        if (sum <= maxSum && i + j > count) count = i + j;
    }
    return count;
}

int main() {
    int g;
    scanf("%d", &g);
    while (g--) {
        int n, m, maxSum;
        scanf("%d%d%d", &n, &m, &maxSum);
        int a[n], b[m];

        // Masukkan elemen stack pertama dan kedua
        for (int i = 0; i < n; scanf("%d", &a[i++]));
        for (int i = 0; i < m; scanf("%d", &b[i++]));

        // Hitung dan tampilkan jumlah maksimum elemen yang dapat diambil dari
        // kedua stack
        printf("%d\n", twoStacks(maxSum, a, n, b, m));
    }
    return 0;
}

```

Penjelasan program

```

int twoStacks(int maxSum, int a[], int n, int b[], int m) {
    int sum = 0, count = 0, i = 0, j = 0;

```

Kode `int twoStacks(int maxSum, int a[], int n, int b[], int m) {...}` menginisialisasi fungsi `twoStacks` yang akan menghitung jumlah maksimum elemen yang diambil dari (`a[]` dan `b[]`) dengan batas jumlah total yang tidak boleh melebihi `maxSum`.

`int sum = 0`: Variabel `sum` digunakan untuk mengecek jumlah total elemen dari tumpukan yang diproses saat ini.

`int count = 0`: Variabel `count` akan menyimpan jumlah elemen dari tumpukan pertama yang dimasukkan ke dalam `sum` tanpa melebihi `maxSum`.

int i = 0: Variabel i digunakan sebagai indeks untuk tumpukan pertama a[].
int j = 0: Variabel j digunakan sebagai indeks untuk tumpukan kedua b[].

```
// Hitung elemen stack pertama yang dapat dimasukkan  
while (i < n && (sum += a[i++]) <= maxSum) count++;
```

i < n: Ini adalah kondisi pertama dari loop while. Ini memastikan bahwa kita tidak melampaui ukuran tumpukan pertama a[].

(sum += a[i++]) <= maxSum: Ini adalah kondisi kedua. Di dalam loop, setiap elemen dari tumpukan pertama a[] akan ditambahkan ke variabel sum, dan nilai i akan ditingkatkan setelah itu. Proses ini akan terus berlanjut selama jumlah total elemen yang ditambahkan (sum) masih kurang dari atau sama dengan maxSum.

count++: Ini adalah pernyataan yang akan dieksekusi jika kedua kondisi sebelumnya terpenuhi. Setiap kali sebuah elemen dari tumpukan pertama ditambahkan ke sum (sesuai dengan kondisi kedua), maka count akan ditingkatkan, yang menghitung jumlah elemen tumpukan pertama yang dapat dimasukkan ke dalam sum tanpa melebihi maxSum.

```
// Tambahkan elemen stack kedua dan sesuaikan dengan batas maksimum  
while (j < m && i >= 0) {  
    sum += b[j++];  
    while (sum > maxSum && i > 0) sum -= a[--i];  
    if (sum <= maxSum && i + j > count) count = i + j;  
}  
return count;  
}
```

1. while (j < m && i >= 0) { Ini adalah loop utama yang akan berjalan selama masih ada elemen dalam tumpukan kedua (b[]) dan indeks tumpukan pertama (a[]) masih valid (tidak kurang dari 0). Loop ini bertujuan untuk memproses tumpukan kedua dan memastikan bahwa jumlah total elemen yang diambil dari kedua tumpukan tidak melebihi maxSum.
2. sum += b[j++]; Di dalam loop utama, setiap elemen dari tumpukan kedua (b[]) akan ditambahkan ke dalam variabel sum, dan nilai indeks tumpukan kedua (j) akan ditingkatkan.
3. while (sum > maxSum && i > 0) sum -= a[--i]; Ini adalah loop bersarang yang akan dieksekusi jika jumlah total elemen (sum) yang diambil dari kedua tumpukan melebihi maxSum. Loop ini bertujuan untuk mengurangi jumlah elemen dari tumpukan pertama (a[]) sampai sum tidak lagi melebihi maxSum. Ini dilakukan dengan mengurangi elemen terakhir dari tumpukan pertama (a[]) dari sum sampai sum tidak lagi melebihi maxSum.
4. if (sum <= maxSum && i + j > count) count = i + j; Setelah memastikan bahwa jumlah elemen yang diambil tidak melebihi maxSum, program memeriksa apakah jumlah total elemen (i + j) yang diambil dari kedua tumpukan lebih besar dari count. Jika ya, maka count diperbarui dengan jumlah total elemen tersebut.

```
int main() {  
    int g;  
    scanf("%d", &g);  
    while (g--) {  
        int n, m, maxSum;  
        scanf("%d%d%d", &n, &m, &maxSum);  
        int a[n], b[m];
```



```
int g; scanf("%d", &g);
```

Deklarasi variabel `g` yang akan menyimpan jumlah kasus uji yang akan dimasukkan oleh user melalui fungsi `scanf()`.

```
while (g--) { ... }
```

Looping `while` yang akan berjalan sebanyak `g` kali, yang menandakan jumlah kasus uji yang akan diuji.

```
int n, m, maxSum; scanf("%d%d%d", &n, &m, &maxSum);
```

Meminta user untuk memasukkan nilai `n` (jumlah elemen tumpukan pertama `a[]`), `m` (jumlah elemen tumpukan kedua `b[]`), dan `maxSum` (batas jumlah total yang dapat diambil dari kedua tumpukan).

```
int a[n], b[m];
```

Mendeklarasikan array `a[]` dan `b[]` dengan ukuran yang sesuai dengan jumlah elemen `n` dan `m`.

```
for (int i = 0; i < n; scanf("%d", &a[i++]));  
for (int i = 0; i < m; scanf("%d", &b[i++]));
```

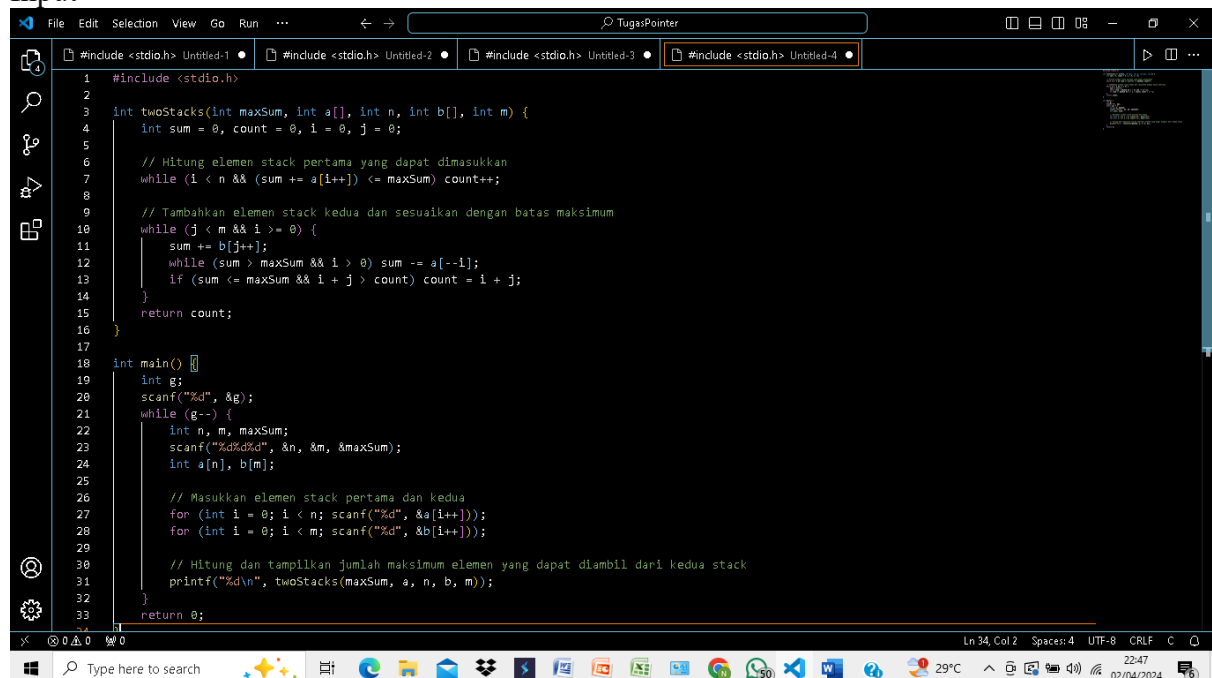
Melakukan loop `for` untuk memasukkan elemen-elemen tumpukan pertama `a[]` dan tumpukan kedua `b[]` menggunakan fungsi `scanf()`.

Ini dilakukan dengan menggunakan `scanf("%d", &a[i++])` dan `scanf("%d", &b[i++])`, yang membaca satu elemen integer dan menyimpannya di dalam array `a[]` dan `b[]` pada setiap iterasi.

```
printf("%d\n", twoStacks(maxSum, a, n, b, m));
```

Memanggil fungsi `twoStacks()` untuk menghitung jumlah maksimum elemen yang dapat diambil dari kedua tumpukan dan menampilkannya menggunakan `printf()`.

Input



```
#include <stdio.h>

int twoStacks(int maxSum, int a[], int n, int b[], int m) {
    int sum = 0, count = 0, i = 0, j = 0;

    // Hitung elemen stack pertama yang dapat dimasukkan
    while (i < n && (sum += a[i]) <= maxSum) count++;

    // Tambahkan elemen stack kedua dan sesuaikan dengan batas maksimum
    while (j < m && i >= 0) {
        sum += b[j++];
        while (sum > maxSum && i > 0) sum -= a[i--];
        if (sum <= maxSum && i + j > count) count = i + j;
    }

    return count;
}

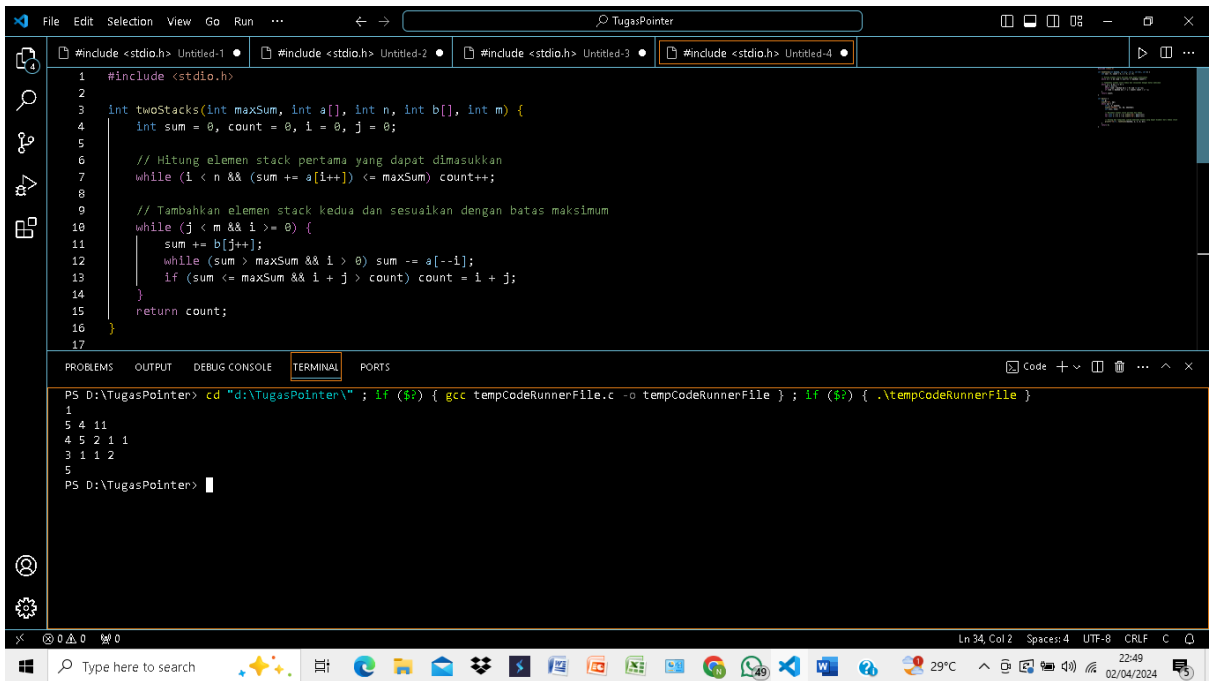
int main() {
    int g;
    scanf("%d", &g);
    while (g--) {
        int n, m, maxSum;
        scanf("%d%d%d", &n, &m, &maxSum);
        int a[n], b[m];

        // Masukkan elemen stack pertama dan kedua
        for (int i = 0; i < n; scanf("%d", &a[i++]));
        for (int i = 0; i < m; scanf("%d", &b[i++]));

        // Hitung dan tampilkan jumlah maksimum elemen yang dapat diambil dari kedua stack
        printf("%d\n", twoStacks(maxSum, a, n, b, m));
    }

    return 0;
}
```

Output



```
1 #include <stdio.h>
2
3 int twoStacks(int maxSum, int a[], int n, int b[], int m) {
4     int sum = 0, count = 0, i = 0, j = 0;
5
6     // Hitung elemen stack pertama yang dapat dimasukkan
7     while (i < n && (sum += a[i++]) <= maxSum) count++;
8
9     // Tambahkan elemen stack kedua dan sesuaikan dengan batas maksimum
10    while (j < m && i >= 0) {
11        sum += b[j++];
12        while (sum > maxSum && i > 0) sum -= a[--i];
13        if (sum <= maxSum && i + j > count) count = i + j;
14    }
15    return count;
16 }
17
18 int main() {
19     int a[] = {5, 4, 11};
20     int b[] = {4, 5, 2, 1, 1};
21     int n = 3, m = 5, maxSum = 10;
22     printf("%d", twoStacks(maxSum, a, n, b, m));
23     return 0;
24 }
```

```
PS D:\TugasPointer> cd "d:\TugasPointer\"; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile }; if ($?) { .\tempCodeRunnerFile }
1
5 4 11
4 5 2 1 1
3 1 1 2
5
PS D:\TugasPointer>
```

4
5
2
1
1

3
1
1
2

Move 1

4
5
2
1
1

3
1
1
2

Move 2

Move 3

Move 4

Move 5

Output = 5