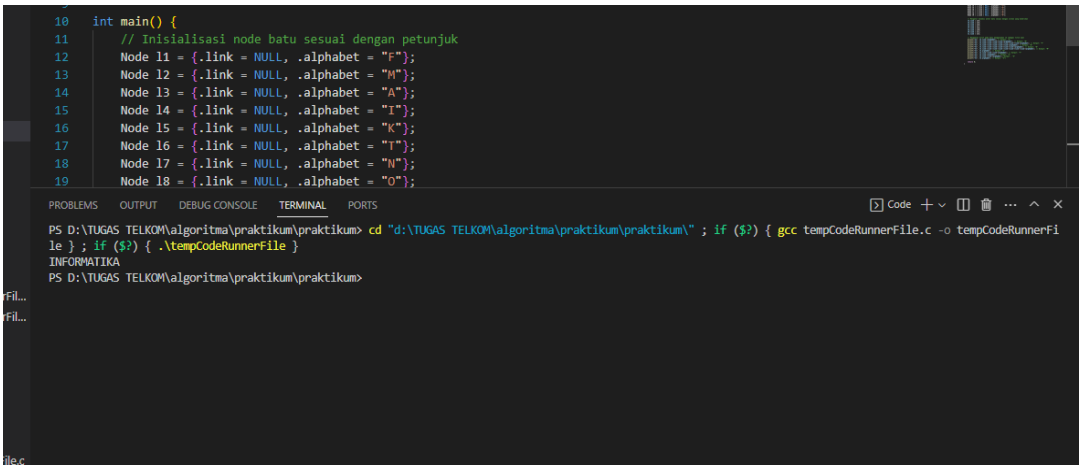
Nama : fitria nur rofika
Nim : 1203230097
IF 03-02

SOURCE CODE

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

// Define the structure for the stone node
typedef struct Node {
    char *alphabet;  // Menyimpan huruf pada batu
    struct Node *link;  // Pointer ke node berikutnya
} Node;

int main() {
    // Inisialisasi node batu sesuai dengan petunjuk
    Node I1 = {.link = NULL, .alphabet = "F"};
    Node I2 = {.link = NULL, .alphabet = "M"};
    Node I3 = {.link = NULL, .alphabet = "A"};
    Node I4 = {.link = NULL, .alphabet = "I"};
    Node I5 = {.link = NULL, .alphabet = "K"};
    Node I6 = {.link = NULL, .alphabet = "T"};
    Node I7 = {.link = NULL, .alphabet = "N"};
    Node I8 = {.link = NULL, .alphabet = "O"};
    Node I9 = {.link = NULL, .alphabet = "R"};

    // Mengatur koneksi antar batu sesuai dengan urutan yang diberikan
    I3.link = &I4;
    I4.link = &I5;
    I5.link = &I6;
    I6.link = &I7;
    I7.link = &I8;
    I8.link = &I9;
    I9.link = &I1;
    I1.link = &I2;

    // Mengakses huruf pada batu menggunakan I3 sebagai titik awal
    printf("%s", I3.link->alphabet); // Output: "I"
    printf("%s", I3.link->link->link->link->alphabet); // Output: "N"
    printf("%s", I3.link->link->link->link->link->link->link->alphabet); // Output: "F"
    printf("%s", I3.link->link->link->link->link->alphabet); // Output: "O"
    printf("%s", I3.link->link->link->link->link->link->alphabet); // Output: "R"
    printf("%s", I3.link->link->link->link->link->link->link->link->alphabet); // Output: "M"
    printf("%s", I3.alphabet); // Output: "A"
    printf("%s", I3.link->link->link->alphabet); // Output: "T"
    printf("%s", I3.link->alphabet); // Output: "I"
```

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

    return 0;
}
```

Penjelasan :
1. Pertama-tama, struktur data Node didefinisikan untuk merepresentasikan sebuah node dalam
linked list. Setiap node memiliki dua atribut: alphabet untuk menyimpan huruf pada batu, dan
link untuk menunjukkan ke node berikutnya dalam linked list.
2. Di dalam fungsi main(), beberapa node batu (l1, l2, ..., l9) diinisialisasi sesuai dengan petunjuk.
Setiap node direpresentasikan oleh sebuah variabel dari tipe Node, dengan huruf pada batu
disimpan dalam atribut alphabet, dan link diatur menjadi NULL pada awalnya.
3. Kemudian, koneksi antara node-node batu diatur sesuai dengan urutan yang diberikan dalam
petunjuk. Ini dilakukan dengan mengatur pointer link dari setiap node untuk menunjuk ke node
batu berikutnya dalam urutan yang benar.
4. Setelah koneksi antara node-node batu diatur, dilakukan pencetakan beberapa huruf pada
batu dengan mengakses linked list dari node l3 sebagai titik awal.
5. Setiap printf() digunakan untuk mencetak huruf pada batu yang diakses dengan mengikuti
pointer link dari node yang ditentukan. Misalnya, l3.link->alphabet digunakan untuk mencetak
huruf pada batu yang dihubungkan ke l3, yaitu huruf "I".
6. Output dari setiap printf() menunjukkan huruf pada batu yang diakses sesuai dengan urutan
yang diberikan dalam kode program.

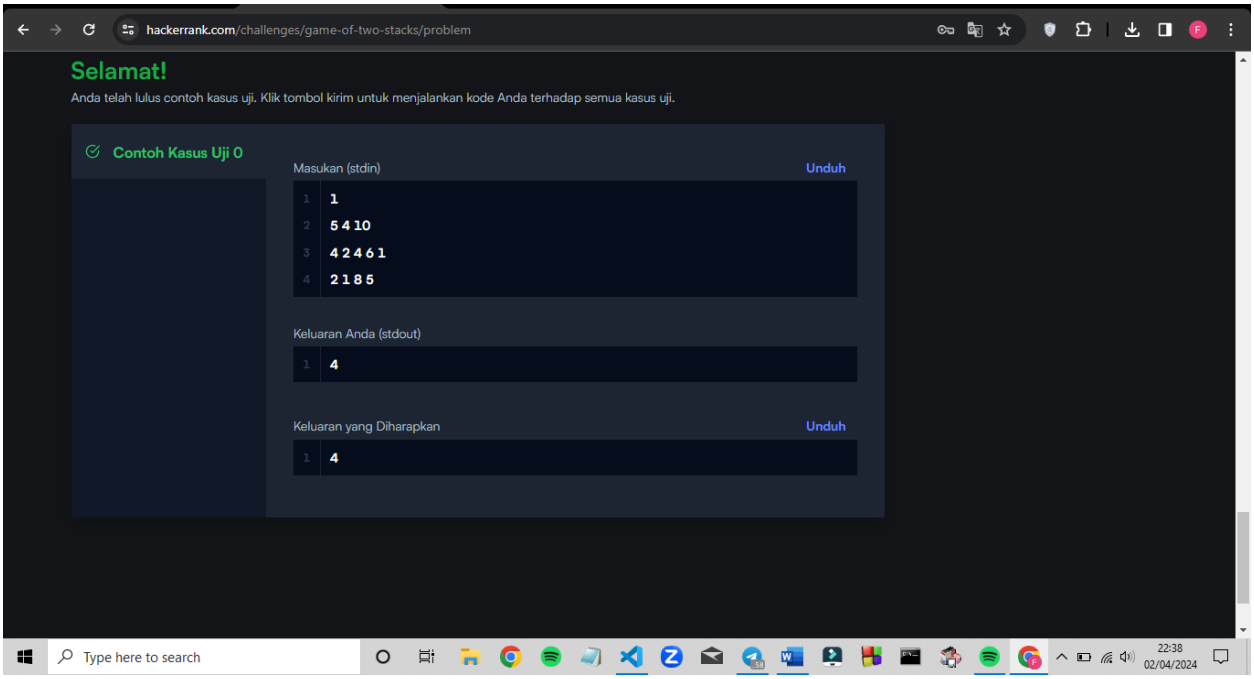


s

HACKERANK

```c
#include <stdio.h>
int twoStacks(int maxSum, int n, int m, int *a, int *b) {
 int i = 0, j = 0, count = 0, sum = 0;
 while (i < n && sum + a[i] <= maxSum) {
 sum += a[i++];
 count++;
 }
 int maxCount = count;
 while (j < m && i >= 0) {
 sum += b[j++];
 count++;
 while (sum > maxSum && i > 0) {
 sum -= a[--i];
 count--;
 }
 if (sum <= maxSum && count > maxCount)
 maxCount = count;
 }
 return maxCount;
}
int main() {
 int games;
 scanf("%d", &games);
 while (games--) {
 int n, m, maxSum;
 scanf("%d %d %d", &n, &m, &maxSum);
 int a[n], b[m];
 for (int i = 0; i < n; i++)
```

```
 scanf("%d", &a[i]);
 for (int i = 0; i < m; i++)
 scanf("%d", &b[i]);
 int result = twoStacks(maxSum, n, m, a, b);
 printf("%d\n", result);
 }
 return 0;
}
```



1. Pada awal program, kita memiliki fungsi twoStacks yang mengambil empat parameter: maxSum (jumlah maksimum yang dapat ditoleransi), n (ukuran stack pertama), m (ukuran stack
kedua), serta dua array a dan b yang mewakili nilai-nilai pada masing-masing stack.
2. Di dalam fungsi twoStacks, dua indeks i dan j diinisialisasi untuk masing-masing stack. Juga,
dua variabel count dan sum diinisialisasi sebagai 0, untuk menghitung jumlah elemen yang diproses dan jumlah total dari elemen-elemen tersebut.
3. Pertama, dilakukan iterasi pada stack pertama (a) dengan menggunakan loop while. Setiap
elemen stack pertama akan ditambahkan ke sum, dan jika sum tidak melebihi maxSum, indeks i
dan count akan ditingkatkan.
4. Setelah iterasi pada stack pertama selesai, kita memiliki jumlah maksimum elemen yang
dapat diambil dari stack pertama tanpa melebihi maxSum.

5. Selanjutnya, dilakukan iterasi pada stack kedua (b) dengan menggunakan loop while. Setiap
elemen stack kedua akan ditambahkan ke sum dan count.
6. Saat sum melebihi maxSum, dilakukan pengurangan elemen-elemen dari stack pertama (a)
secara berurutan sampai sum tidak melebihi maxSum lagi atau sampai stack pertama kosong.
7. Selama iterasi pada stack kedua, kita selalu memeriksa apakah count saat ini melebihi maxCount (jumlah maksimum elemen yang telah diproses sebelumnya). Jika ya, kita perbarui
maxCount.
8. Setelah iterasi pada kedua stack selesai, kita kembalikan maxCount yang merupakan jumlah
maksimum total elemen yang dapat diambil dari kedua stack tanpa melebihi maxSum.
9. Di dalam fungsi main, kita membaca jumlah permainan (games) yang akan dilakukan. Kemudian, untuk setiap permainan, kita membaca ukuran stack pertama (n), ukuran stack kedua
(m), dan maxSum. Selanjutnya, kita membaca nilai-nilai dari masing-masing stack.
10. Setelah membaca nilai-nilai stack, kita panggil fungsi twoStacks untuk menghitung jumlah
maksimum elemen yang dapat diambil dari kedua stack tanpa melebihi maxSum, dan kemudian
cetak hasilnya.