

Nama : Ajax Amstermarstama Januariel

IF03-03

NIM : 1203230091

Source Code no.1

```
C:\Users\user> OneDrive\Documents > C:\alpha.c > main()
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Stone {
5      char letter;
6      struct Stone *next;
7  };
8
9  void addStone(struct Stone **head, char letter) {
10     struct Stone *newStone = (struct Stone *)malloc(sizeof(struct Stone));
11     newStone->letter = letter;
12     newStone->next = NULL;
13
14     if (*head == NULL) {
15         *head = newStone;
16         return;
17     }
18
19     struct Stone *lastStone = *head;
20     while (lastStone->next != NULL) {
21         lastStone = lastStone->next;
22     }
23     lastStone->next = newStone;
24 }
25
26 void printStones(struct Stone *head) {
27     struct Stone *current = head;
28     while (current != NULL) {
29         printf("%c", current->letter);
30         current = current->next;
31     }
32 }
33
34 int main() {
35     struct Stone *head = NULL;
```

```
C:\Users\user> OneDrive\Documents > C:\alpha.c > main()
36
37     // Menambahkan huruf-huruf ke dalam linked list
38     addStone(&head, 'I');
39     addStone(&head, 'N');
40     addStone(&head, 'F');
41     addStone(&head, 'O');
42     addStone(&head, 'R');
43     addStone(&head, 'M');
44     addStone(&head, 'A');
45     addStone(&head, 'T');
46     addStone(&head, 'I');
47     addStone(&head, 'K');
48     addStone(&head, 'A');
49
50     // Mencetak huruf-huruf dari linked list
51     printStones(head);
52
53     // Membersihkan memori yang dialokasikan
54     struct Stone *current = head;
55     struct Stone *next;
56     while (current != NULL) {
57         next = current->next;
58         free(current);
59         current = next;
60     }
61
62     return 0;
63 }
```

Penjelasan source code

```
struct Stone {
    char letter;
    struct Stone *next;
};
```

Kode tersebut mendefinisikan struktur bernama **“Stone”** dengan dua anggota, yaitu **letter**(karakter) dan **next**(pointer ke Stone berikutnya). Ini bisa digunakan untuk membuat

daftar terhubung di mana setiap elemen menyimpan karakter dan pointer ke elemen berikutnya.

```
void addStone(struct Stone **head, char letter) {
    struct Stone *newStone = (struct Stone *)malloc(sizeof(struct Stone));
    newStone->letter = letter;
    newStone->next = NULL;
```

Fungsi “**addStone**” ini menerima parameter **head**, yang merupakan pointer ke pointer dari struktur **Stone** dan **Letter**, yang merupakan karakter yang akan ditambahkan ke dalam struktur. Selanjutnya, fungsi ini melakukan alokasi memori dinamis untuk membuat sebuah simpul baru dari struktur **Stone**, mengatur karakter **Letter** dari simpul baru tersebut, dan mengatur pointer **next** menjadi NULL.

```
    if (*head == NULL) {
        *head = newStone;
        return;
    }
```

Pada bagian ini, kode memeriksa apakah **head** (pointer ke pointer Stone) adalah NULL. Jika iya, artinya tidak ada elemen dalam daftar (atau “**head**” dari daftar adalah NULL), maka **head** diatur menjadi **newStone**, yang merupakan simpul baru yang telah dibuat, dan fungsi tersebut kemudian selesai (return)

```
    struct Stone *lastStone = *head;
    while (lastStone->next != NULL) {
        lastStone = lastStone->next;
    }
    lastStone->next = newStone;
}
```

Kode ini mencari elemen terakhir dalam daftar dengan menggunakan **loop while**, kemudian menambahkan simpul baru ke akhir daftar dengan mengatur pointer **next** dari elemen terakhir tersebut menjadi **newStone**.

```
void printStones(struct Stone *head) {
    struct Stone *current = head;
    while (current != NULL) {
        printf("%c", current->letter);
        current = current->next;
    }
}
```

Fungsi di atas adalah **printStones**, yang menerima pointer ke kepala dari linked list yang berisi struktur **Stone**. Fungsi ini mengiterasi melalui setiap node dalam linked list, mencetak karakter **letter** dari setiap node ke layar, dan kemudian maju ke node berikutnya menggunakan pointer **next**. Fungsi ini akan terus mencetak karakter sampai mencapai akhir dari linked list (sampai pointer **current** menjadi NULL.)

*ket : Node berasal dari struktur **Stone** yang merupakan bagian dari sebuah linked list. Dalam konteks diatas, setiap node merepresentasikan satu elemen dari linked list dan berisi informasi seperti karakter (**letter**) dan pointer ke node berikutnya (**next**).

```
int main() {
```

```
struct Stone *head = NULL;
```

Kode tersebut menginisialisasi pointer *head* ke *NULL*, menandakan bahwa linked list kosong pada awalnya.

```
addStone(&head, 'I');
addStone(&head, 'N');
addStone(&head, 'F');
addStone(&head, 'O');
addStone(&head, 'R');
addStone(&head, 'M');
addStone(&head, 'A');
addStone(&head, 'T');
addStone(&head, 'I');
addStone(&head, 'K');
addStone(&head, 'A');
```

Kode tersebut menambahkan beberapa node ke linked list. Setiap pemanggilan *addStone(&head, 'x')* menambahkan sebuah node baru dengan karakter 'x' ke linked list yang diarahkan oleh pointer *head*.

```
printStones(head);
```

Pemanggilan fungsi *printStones(head)* akan mencetak karakter dari setiap node dalam linked list yang dimulai dari *head*, sesuai dengan urutan mereka dalam linked list.

```
struct Stone *current = head;
struct Stone *next;
while (current != NULL) {
    next = current->next;
    free(current);
    current = next;
}
```

Kode tersebut melakukan setiap kode dalam linked list yang dimulai dari *head*, menghapus setiap node satu per satu, dan mengosongkan memori yang dialokasikan untuk setiap kode.

Output

```
PS C:\Users\user> cd "c:\Users\user\OneDrive\Documents\" ; if ($?) { gcc alpha.c -o alpha } ; if ($?) { .\alpha }
INFORMATIKA
PS C:\Users\user\OneDrive\Documents>
```

```
Change Theme Language C

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

int* read_integers(int size) {
    int* arr = malloc(size * sizeof(int));
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    return arr;
}

int twoStacks(int maxSum, int a_count, int* a, int b_count, int* b) {
    int i = 0, j = 0, count = 0, sum = 0;

    // Simulate removing elements from stack a
    while (i < a_count && sum + a[i] <= maxSum) {
        sum += a[i];
        i++;
        count++;
    }

    // Simulate removing elements from stack b, while adjusting sum
    while (j < b_count && i >= 0) {
        sum += b[j];
        j++;

        // If sum exceeds maxSum, remove elements from stack a until it's within limit
        while (sum > maxSum && i > 0) {
            sum -= a[i];
            i--;
            count--;
        }
    }

    return count;
}

int main() {
    int g;
    scanf("%d", &g); // Read the number of test cases

    for (int g_itr = 0; g_itr < g; g_itr++) {
        int n, m, maxSum;
        scanf("%d %d %d", &n, &m, &maxSum);

        int* a = read_integers(n);
        int* b = read_integers(m);

        int result = twoStacks(maxSum, n, a, m, b);
        printf("%d\n", result);

        free(a);
        free(b);
    }
}
```

Penjelasan source code no.2

```
int* read_integers(int size) {
    int* arr = malloc(size * sizeof(int));
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    return arr;
}
```

Fungsi *read_integers* mengalokasikan memori untuk array integer sebesar *size*, membaca *size* angka dari input pengguna dan mengembalikan pointer ke array berikutnya.

*note: *read_integers* merupakan fungsi yang bertugas untuk membaca sejumlah bilangan bulat dari input pengguna dan mengembalikan array yang berisi bilangan-bilangan tersebut.

```
int twoStacks(int maxSum, int a_count, int* a, int b_count, int* b) {
```

```
int i = 0, j = 0, count = 0, sum = 0;
```

Kode tersebut adalah awal dari fungsi *twoStacks*, yang bertujuan menghitung jumlah elemen dari dua stack (*a* dan *b*) yang bisa diambil sehingga total nilainya tidak melebihi *maxSum*. Variabel *i* dan *j* menandai posisi dalam masing-masing stack, *count* menghitung elemen yang diambil, dan *sum* menyimpan total nilai yang diambil.

```
while (i < a_count && sum + a[i] <= maxSum) {  
    sum += a[i];  
    i++;  
    count++;  
}
```

Baris kode ini bertujuan untuk menambahkan elemen dari stack *a* ke total *sum* hingga total *sum* tidak melebihi *maxSum* atau tidak ada elemen lagi dalam stack *a* yang dapat ditambahkan.

```
while (j < b_count && i >= 0) {  
    sum += b[j];  
    j++;  
}
```

Loop tersebut menambahkan elemen dari stack *b* ke total *sum* sampai mana total *sum* mencapai atau mendekati *maxSum*, atau tidak ada elemen lagi dalam stack *b* yang dapat ditambahkan.

```
while (sum > maxSum && i > 0) {  
    i--;  
    sum -= a[i];  
}
```

Loop tersebut mengurangi elemen dari stack *a* dari total *sum* jika total *sum* melebihi *maxSum*, sampai tidak ada elemen lagi yang dapat dihapus atau total *sum* tidak melebihi *maxSum*.

```
if (sum <= maxSum && i + j > count) {  
    count = i + j;  
}
```

Kode ini memperbarui nilai *count* dengan jumlah elemen yang diambil (*i + j*) jika total *sum* tidak melebihi *maxSum* dan jumlah elemen total *sum* tidak melebihi *maxSum* dan jumlah elemen yang diambil lebih besar dari nilai *count* saat ini.

```
int main() {  
    int g;  
    scanf("%d", &g); // Read the number of test cases  
  
    for (int g_itr = 0; g_itr < g; g_itr++) {  
        int n, m, maxSum;  
        scanf("%d %d %d", &n, &m, &maxSum);  
  
        int* a = read_integers(n);
```

```

    int* b = read_integers(m);

    int result = twoStacks(maxSum, n, a, m, b);
    printf("%d\n", result);

    free(a);
    free(b);
}

```

Program ini membaca jumlah kasus uji (*g*), kemudian untuk setiap uji, membaca ukuran stack *a* dan *b* serta *maxSum*. Selanjutnya, program membaca elemen-elemen dari kedua stack tersebut, menghitung hasilnya menggunakan fungsi *twoStacks*, dan mencetak hasilnya. Setelah itu, membebaskan memori yang dialokasikan untuk kedua stack, program berakhir setelah semua kasus uji selesai dievaluasi.

Output

The screenshot shows a web interface for a coding challenge. At the top, a green banner says "Congratulations" and "You solved this challenge. Would you like to challenge your friends?" with social media icons and a "Next Challenge" button. On the left, a list of test cases from 7 to 13 is shown, all with green checkmarks. The main area displays the "Compiler Message" as "Success". Below this, the "Input (stdin)" is shown as a 4x4 grid of numbers:

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

 The "Expected Output" is shown as a single line:

1	4
---	---

 Download links are provided for both the input and output sections.