NAMA : I MADE SUTA EKA DHARMA

KELAS : IF 03-01

NIM : 1203230072

OTH STRUCT DAN STAK

1. SOURCE CODE

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

// Definisikan struktur Stone
struct Stone {
    char* alphabet;
    struct Stone* link;
};

int main() {
    // Inisialisasi node-node sesuai dengan tabel yang diberikan
    struct Stone l1, l2, l3, l4, l5, l6, l7, l8, l9;

    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";

```

```
42.l3.link = &l6; //A->T
43.l6.link = &l9; //T->R
44.l9.link = &l4; //R->I
45.l4.link = &l7; //I->N
46.l7.link = &l1; //N->F
47.l1.link = &l8; //T->O
48.l8.link = &l2; //O->M
49.l2.link = &l5; //M->K
50.l5.link = &l3; //K->A
51.     // Akses data menggunakan l3 sebagai titik awal
52.     printf("%s", l3.link->link->link->alphabet);
53.     printf("%s", l3.link->link->link->link->alphabet);
54.     printf("%s", l3.link->link->link->link->link->alphabet);
55.     printf("%s", l3.link->link->link->link->link->link->alphabet);
56.     printf("%s", l3.link->link->alphabet);
57.     printf("%s", l3.link->link->link->link->link->link->link-
   >alphabet);
58.     printf("%s", l3.alphabet);
59.     printf("%s", l3.link->alphabet);
60.     printf("%s", l3.link->link->link->alphabet);
61.     printf("%s", l3.link->link->link->link->link->link->link->link-
   >alphabet);
62.     printf("%s", l3.alphabet);
63.
64.     printf("\n\nStack Visualization:\n");
65.
66.     struct Stone* current = &l3;
67.
68.
69.     return 0;
70.}
```

OUTPUT

```
PS C:\Users\Asus\vscode> cd "c:\Users\Asus\vscode\eko\" ; if ($?) { gcc holmes.c -o holmes } ; if ($?) { .\holmes }
INFORMATIKA

Stack Visualization:
PS C:\Users\Asus\vscode\eko>
```

PENJELASAN KODE

```
// Definisikan struktur Stone
struct Stone {
    char* alphabet;
    struct Stone* link;
};
```

- alphabet: Sebuah pointer yang menunjuk ke tipe data char. Ini digunakan untuk menyimpan sebuah karakter atau string yang merepresentasikan huruf.
- link: Sebuah pointer yang menunjuk ke struktur Stone itu sendiri. Ini digunakan untuk membuat sebuah linked list, di mana setiap node (atau elemen) memiliki referensi atau "link" ke node berikutnya.
- setiap objek dari tipe Stone memiliki kemampuan untuk menyimpan sebuah karakter atau string (alphabet) dan juga memiliki kemampuan untuk menghubungkan dirinya ke objek Stone lainnya (link).

```
int main() {
    struct Stone l1, l2, l3, l4, l5, l6, l7, l8, l9;

    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";

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

- Deklarasi Struktur: Terdapat 9 variabel bertipe Stone yang dideklarasikan: l1 sampai l9.

- Inisialisasi Node: Setiap node (variabel) diinisialisasi dengan member link yang menunjuk ke NULL dan member alphabet yang menunjuk ke karakter atau string tertentu.

- Penghubungan Node: Node-node ini kemudian dihubungkan satu sama lain untuk membentuk sebuah linked list. Misalnya, l3.link = &l6 berarti node l3 mengarah ke node l6, yang berarti urutan A->T.

- Urutan Penghubungan: Berdasarkan kode, urutan penghubungan adalah sebagai berikut:
  A -> T -> R -> I -> N -> F -> O -> M -> K -> A
- Jadi, setiap node memiliki informasi tentang karakter (alphabet) dan referensi ke node berikutnya (link), membentuk struktur data linked list.

```
printf("%s", l3.link->link->link->alphabet);
printf("%s", l3.link->link->link->link->alphabet);
printf("%s", l3.link->link->link->link->link->alphabet);
printf("%s", l3.link->link->link->link->link->link->alphabet);
printf("%s", l3.link->link->alphabet);
printf("%s", l3.link->link->link->link->link->link->link->alphabet);
printf("%s", l3.alphabet);
printf("%s", l3.link->alphabet);
printf("%s", l3.link->link->link->alphabet);
printf("%s", l3.link->link->link->link->link->link->link->link->alphabet);
printf("%s", l3.alphabet);


printf("\n\nStack Visualization:\n");

struct Stone* current = &l3;



return 0;
```

- Printf Statements: Terdapat serangkaian pernyataan printf yang mencetak karakter dari linked list l3:

- printf("%s", l3.link->link->link->alphabet);: Mencetak karakter dari node ke-4 dari l3.
- printf("%s", l3.link->link->link->link->alphabet);: Mencetak karakter dari node ke-5 dari l3.
- printf("%s", l3.link->link->link->link->link->alphabet);: Mencetak karakter dari node ke-6 dari l3.

- printf("%s", l3.link->link->link->link->link->link->alphabet);: Mencetak karakter dari node ke-7 dari l3.
- printf("%s", l3.link->link->alphabet);: Mencetak karakter dari node ke-3 dari l3.
- printf("%s", l3.link->link->link->link->link->link->link->alphabet);: Mencetak karakter dari node ke-8 dari l3.
- printf("%s", l3.alphabet);: Mencetak karakter dari node l3.
- printf("%s", l3.link->alphabet);: Mencetak karakter dari node ke-2 dari l3.
- printf("%s", l3.link->link->link->alphabet);: Mencetak karakter dari node ke-4 dari l3.
- printf("%s", l3.link->link->link->link->link->link->link->link->alphabet);: Mencetak karakter dari node ke-9 dari l3.
- printf("%s", l3.alphabet);: Mencetak karakter dari node l3.
- Stack Visualization: Setelah mencetak semua karakter, program mencetak label "Stack Visualization:".
- Pointer Current: struct Stone* current = &l3; adalah deklarasi pointer current yang menunjuk ke node pertama dari linked list l3.

2. SOURCE CODE

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

#define MAX_SIZE 50

typedef struct {
    int data[MAX_SIZE];
    int top;
} Stack;

void initializeStack(Stack *stack) {
    stack->top = -1;
}

void push(Stack *stack, int value) {
    stack->data[++stack->top] = value;
}

int pop(Stack *stack) {
    return stack->data[stack->top--];
}

int peek(Stack *stack) {
    return stack->data[stack->top];
}

int isStackEmpty(Stack *stack) {
    return stack->top == -1;
}

int isStackFull(Stack *stack) {
    return stack->top == MAX_SIZE - 1;
}

int processTwoStacks(int maxSum, int stackA[], int stackB[], int n, int m) {
    Stack sA, sB;
    initializeStack(&sA);
    initializeStack(&sB);

    // Push elements to stacks
    for (int i = n - 1; i >= 0; i--) {
        push(&sA, stackA[i]);
    }
    for (int i = m - 1; i >= 0; i--) {
        push(&sB, stackB[i]);
    }
```

```
    int total = 0, count = 0;
    int length = n >= m ? m : n;

    for (int i = 0; i < length; i++) {
        if (total + peek(&sA) <= maxSum) {
            total += pop(&sA);
            count++;
        }
        if (total + peek(&sB) <= maxSum) {
            total += pop(&sB);
            count++;
        }
    }
    return count;
}

int main() {
    int numGames;
    scanf("%d", &numGames);

    while (numGames--) {
        int n, m, maxSum;
        scanf("%d %d %d", &n, &m, &maxSum);

        int stackA[n], stackB[m];
        for (int i = 0; i < n; i++) {
            scanf("%d", &stackA[i]);
        }
        for (int i = 0; i < m; i++) {
            scanf("%d", &stackB[i]);
        }

        int result = processTwoStacks(maxSum, stackA, stackB, n, m);
        printf("%d\n", result);
    }

    return 0;
}
```

OUTPUT

```
PS C:\Users\Asus\vscode\eko> cd "c:\Users\Asus\vscode\eko\" ; if ($?) { gcc staqk.c -o staqk } ; if ($?) { .\staqk }
1
5 4 11
4 5 2 1 1
3 1 1 2
5
PS C:\Users\Asus\vscode\eko>
```

PENJELASAN KODE

```c
typedef struct {
    int data[MAX_SIZE];
    int top;
} Stack;

void initializeStack(Stack *stack) {
    stack->top = -1;
}

void push(Stack *stack, int value) {
    stack->data[++stack->top] = value;
}

int pop(Stack *stack) {
    return stack->data[stack->top--];
}

int peek(Stack *stack) {
    return stack->data[stack->top];
}

int isStackEmpty(Stack *stack) {
    return stack->top == -1;
}

int isStackFull(Stack *stack) {
    return stack->top == MAX_SIZE - 1;
}
```

- Struktur Stack memiliki dua anggota: data yang merupakan array untuk menyimpan elemen-elemen tumpukan dan top yang merupakan indeks dari elemen teratas tumpukan.
- Fungsi ini menginisialisasi tumpukan dengan mengatur top ke -1, menandakan tumpukan kosong.
- Fungsi push menambahkan elemen baru ke tumpukan dengan meningkatkan top dan menyimpan nilai di indeks yang baru ditunjuk.
- Fungsi pop menghapus dan mengembalikan elemen teratas dari tumpukan dengan mengurangi top dan mengakses elemen yang sesuai.
- Fungsi peek mengembalikan nilai dari elemen teratas tumpukan tanpa menghapusnya.
- Fungsi isStackEmpty mengembalikan 1 jika tumpukan kosong, dan 0 jika tidak.
- Fungsi isStackFull mengembalikan 1 jika tumpukan penuh, dan 0 jika tidak.

```
int processTwoStacks(int maxSum, int stackA[], int stackB[], int n, int m) {
    Stack sA, sB;
    initializeStack(&sA);
    initializeStack(&sB);

    // Push elements to stacks
    for (int i = n - 1; i >= 0; i--) {
        push(&sA, stackA[i]);
    }
    for (int i = m - 1; i >= 0; i--) {
        push(&sB, stackB[i]);
    }

    int total = 0, count = 0;
    int length = n >= m ? m : n;

    for (int i = 0; i < length; i++) {
        if (total + peek(&sA) <= maxSum) {
            total += pop(&sA);
            count++;
        }
        if (total + peek(&sB) <= maxSum) {
            total += pop(&sB);
            count++;
        }
    }
    return count;
}
```

- Fungsi processTwoStacks dirancang untuk memproses dua tumpukan stackAdan stackBdengan jumlah maksimum tertentu maxSum. Fungsi ini mengembalikan jumlah maksimum elemen yang dapat dikeluarkan dari dua tumpukan sehingga jumlah elemen yang muncul kurang dari atau sama dengan maxSum.
- Dua tumpukan sAdan sBdiinisialisasi.
- Elemen stackA dan stackB didorong ke tumpukan masing-masing sAdan sB.
- Variabel ini total digunakan untuk melacak jumlah elemen yang muncul saat ini, dan count digunakan untuk melacak jumlah elemen yang muncul.
- Variabel length diatur ke panjang yang lebih kecil stackAdan stackBuntuk memastikan bahwa kedua tumpukan diproses secara merata.
- Fungsi ini mengulangi setiap indeks tumpukan hingga length.
- Jika jumlah dari total dan elemen teratas dari sAkurang dari atau sama dengan maxSum, elemen teratas dari sAakan dimunculkan, dan countbertambah.
- Proses yang sama diulangi untuk sB.
- Terakhir, fungsi tersebut mengembalikan nilai count

```c
int main() {
    int numGames;
    scanf("%d", &numGames);

    while (numGames--) {
        int n, m, maxSum;
        scanf("%d %d %d", &n, &m, &maxSum);

        int stackA[n], stackB[m];
        for (int i = 0; i < n; i++) {
            scanf("%d", &stackA[i]);
        }
        for (int i = 0; i < m; i++) {
            scanf("%d", &stackB[i]);
        }

        int result = processTwoStacks(maxSum, stackA, stackB, n, m);
        printf("%d\n", result);
    }

    return 0;
}
```

- processTwoStacks fungsi untuk menentukan jumlah poin maksimum yang dapat diperoleh dengan memainkan permainan tersebut, dan mencetak hasilnya.
- Fungsi main mendeklarasikan variabel integer numGamesuntuk menyimpan jumlah permainan yang akan dimainkan, dan membaca nilainya dari input standar menggunakan scanffungsi tersebut.
- Fungsi main kemudian memasuki perulangan yang menjalankan numGameswaktu, dan untuk setiap perulangan perulangan, ia melakukan hal berikut:
- Ini mendeklarasikan tiga variabel bilangan bulat n, m, dan maxSumuntuk menyimpan jumlah kartu di setiap tumpukan, dan jumlah maksimum yang dapat diperoleh dengan memainkan permainan. Itu membaca nilainya dari input standar menggunakan scanffungsi.
- Ini mendeklarasikan dua array stackAdan stackBbilangan bulat dengan ukuran ndan mmasing-masing, untuk menyimpan nilai kartu di setiap tumpukan. Itu membaca nilainya dari input standar menggunakan scanffungsi.
- Ia memanggil processTwoStacks fungsi dengan argumen maxSum, stackA, stackB, n, dan msebagai , dan menyimpan hasilnya dalam resultvariabel.
- Ini mencetak resultmenggunakan printf fungsi.

- Fungsi main mengembalikan 0 untuk menunjukkan bahwa fungsi tersebut telah selesai dengan sukses.