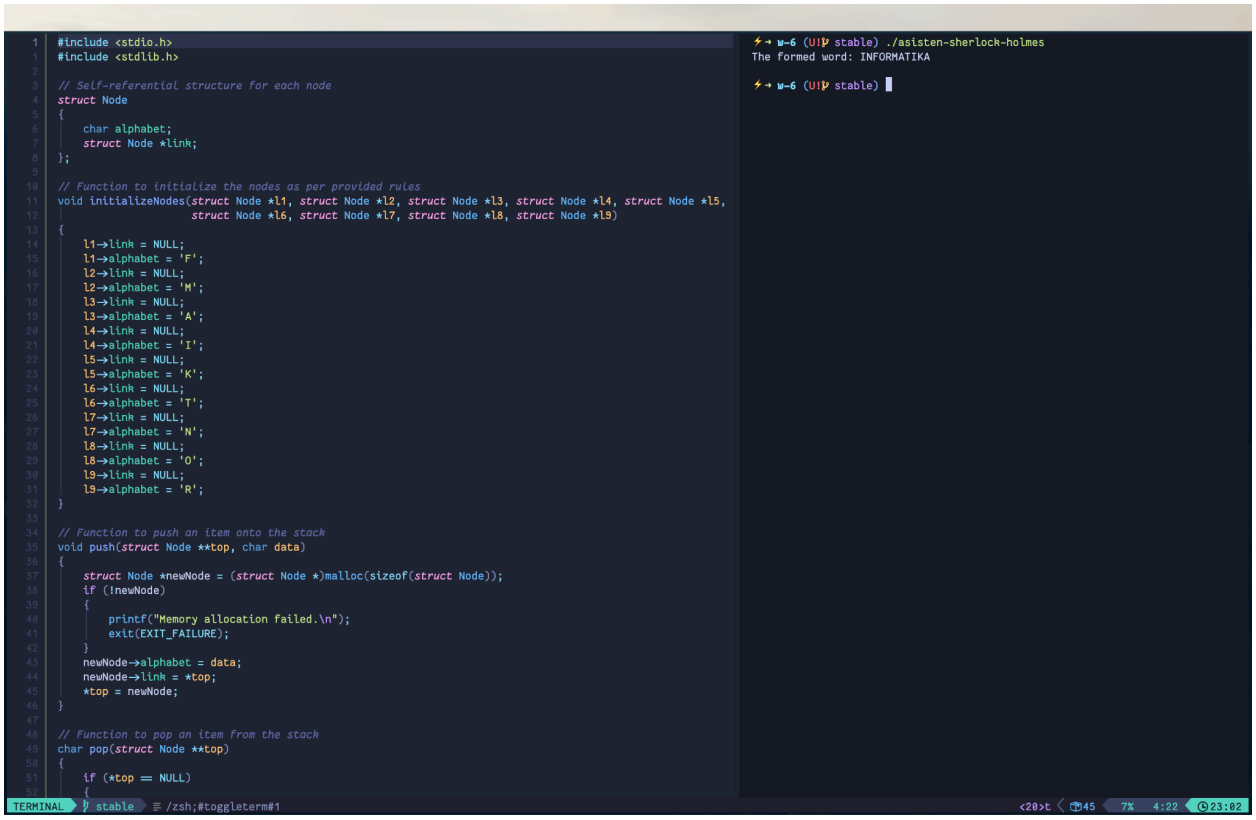
Nama : Arif Widianto
NIM : 1203230051
Tugas : Tugas OTH Week 6
Mata Kuliah : Algoritma & Struktur Data

1. Asisten Sherlock Holmes

1.1. Penulisan Kode beserta output




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

// Self-referential structure for each node
struct Node
{
    char alphabet;
    struct Node *link;
};

// Function to initialize the nodes as per provided rules
```

```c
void initializeNodes(struct Node *l1, struct Node *l2, struct Node *l3, struct Node *l4,
struct Node *l5,
                     struct Node *l6, struct Node *l7, struct Node *l8, struct Node *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';
}

// Function to push an item onto the stack
void push(struct Node **top, char data)
{
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode)
    {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->alphabet = data;
    newNode->link = *top;
    *top = newNode;
}
```

```c
// Function to pop an item from the stack
char pop(struct Node **top)
{
    if (*top == NULL)
    {
        printf("Stack underflow\n");
        exit(EXIT_FAILURE);
    }
    struct Node *temp = *top;
    char popped = temp->alphabet;
    *top = temp->link;
    free(temp);
    return popped;
}

// Function to peek the top element of the stack
char peek(struct Node *top)
{
    if (top == NULL)
    {
        printf("Stack is empty\n");
        exit(EXIT_FAILURE);
    }
    return top->alphabet;
}

// Main function
int main()
{
    // Initialize nodes as per provided rules
    struct Node l1, l2, l3, l4, l5, l6, l7, l8, l9;
    initializeNodes(&l1, &l2, &l3, &l4, &l5, &l6, &l7, &l8, &l9);

    // Link the nodes according to the provided rules
    l1.link = &l2;
    l2.link = &l3;
    l3.link = &l4;
    l4.link = &l5;
    l5.link = &l6;
    l6.link = &l7;
```

```
    l7.link = &l8;
    l8.link = &l9;
    l9.link = &l1;

    // Print the formed word
    printf("The formed word: ");
    printf("%c%c%c%c%c%c%c%c%c%c%c\n",
           l3.link->alphabet,
           l3.link->link->link->link->alphabet,
           l3.link->link->link->link->link->link->link->alphabet,
           l3.link->link->link->link->link->alphabet,
           l3.link->link->link->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,
           l3.link->link->link->alphabet,
           l3.link->alphabet,
           l3.link->link->alphabet,
           l3.alphabet);
    printf("\n");

    return 0;
}
```

1.2. Penjelasan Kode

1. `struct Node`
   - Struktur ini merupakan representasi dari *node* dalam *linked-list*.
   - Setiap *node* memiliki dua anggota: `alphabet` untuk menyimpan karakter dan `link` untuk menunjukkan ke *node* selanjutnya dalam *linked-list*.
2. `initializeNodes`
   - Fungsi ini bertujuan untuk menginisialisasi *nodes* sesuai dengan aturan yang diberikan.
   - Setiap *node* diinisialisasi dengan karakter tertentu dan `link`-nya diatur menjadi `NULL`.
3. `push`

   - Fungsi ini digunakan untuk menambahkan elemen ke dalam stack.
   - Ketika dipanggil, fungsi akan membuat *node* baru, mengatur nilai data-nya, dan menyesuaikan `link`-nya untuk menunjuk ke elemen sebelumnya.

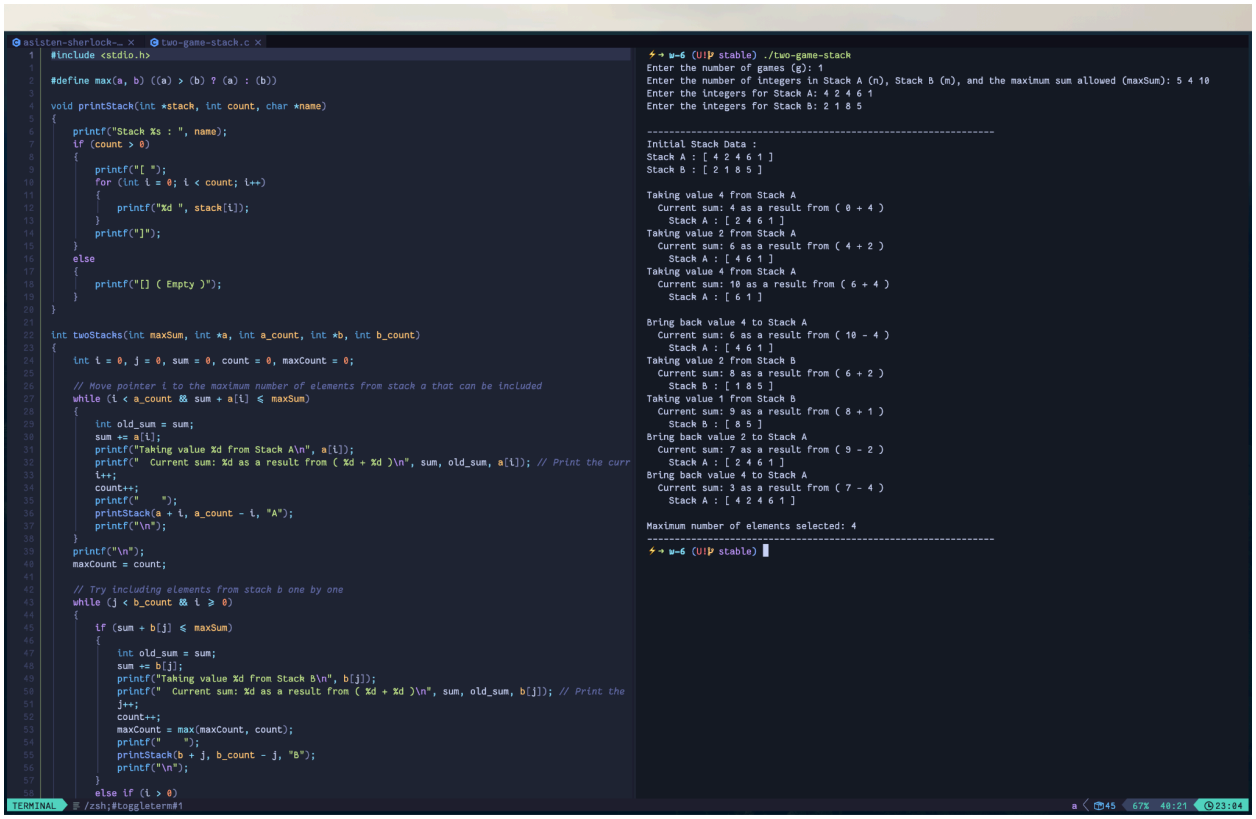
4. `pop`
   - Fungsi ini digunakan untuk menghapus elemen dari *stack*.
   - Saat dipanggil, fungsi akan menghapus elemen teratas dari *stack*, mengembalikan nilainya, dan mengatur link-nya untuk menunjuk ke elemen sebelumnya.
5. `peek`
   - Fungsi ini digunakan untuk melihat elemen teratas dari *stack* tanpa menghapusnya.
   - Saat dipanggil, fungsi akan mengembalikan nilai dari elemen teratas *stack*.
6. `main`
   - Fungsi utama yang melakukan inisialisasi dan penggunaan stack.
   - Pertama, *nodes* diinisialisasi sesuai dengan aturan yang diberikan.
   - Kemudian, *nodes* dihubungkan satu sama lain sesuai dengan aturan.
   - Kata yang terbentuk dari *linked-list* ini kemudian dicetak menggunakan `printf` dengan mengakses *nodes* tertentu sesuai dengan aturan yang telah ditentukan.

1.3. Kesimpulan

Kode tersebut adalah implementasi dari struktur data *stack* menggunakan *linked-list* dalam bahasa pemrograman C. Awalnya, *nodes linked-list* diinisialisasi dengan karakter-karakter tertentu yang membentuk urutan kata tertentu. Setiap *node* kemudian dihubungkan satu sama lain sesuai dengan aturan yang telah ditetapkan. Setelah *linked-list* terbentuk, program mencetak sebuah kata dengan mengambil karakter-karakter dari *nodes* tertentu dalam urutan yang telah ditentukan. Implementasi ini menggabungkan konsep stack untuk menambahkan, menghapus, dan melihat elemen-elemen dalam *linked-list*, sehingga memungkinkan pembentukan kata dengan urutan karakter yang sesuai.

2. Game of Two Stacks Problem

2.1. Penulisan Kode beserta output



```c
#include <stdio.h>

#define max(a, b) ((a) > (b) ? (a) : (b))

void printStack(int *stack, int count, char *name)
{
    printf("Stack %s : ", name);
    if (count > 0)
    {
        printf("[ ");
        for (int i = 0; i < count; i++)
        {
            printf("%d ", stack[i]);
        }
        printf("]");
    }
    else
    {
        printf("[] ( Empty )");
    }
}

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

    // Move pointer i to the maximum number of elements from stack a that can be included
    while (i < a_count && sum + a[i] <= maxSum)
    {
        int old_sum = sum;
        sum += a[i];
        printf("Taking value %d from Stack A\n", a[i]);
        printf("  Current sum: %d as a result from ( %d + %d )\n", sum, old_sum, a[i]); // Print the curr
        i++;
        count++;
        printf("    ");
        printStack(a + i, a_count - i, "A");
        printf("\n");
    }
    printf("\n");
    maxCount = count;

    // Try including elements from stack b one by one
    while (j < b_count && i >= 0)
    {
        if (sum + b[j] <= maxSum)
        {
            int old_sum = sum;
            sum += b[j];
            printf("Taking value %d from Stack B\n", b[j]);
            printf("  Current sum: %d as a result from ( %d + %d )\n", sum, old_sum, b[j]); // Print the
            j++;
            count++;
            maxCount = max(maxCount, count);
            printf("    ");
            printStack(b + j, b_count - j, "B");
            printf("\n");
        }
        else if (i > 0)
```

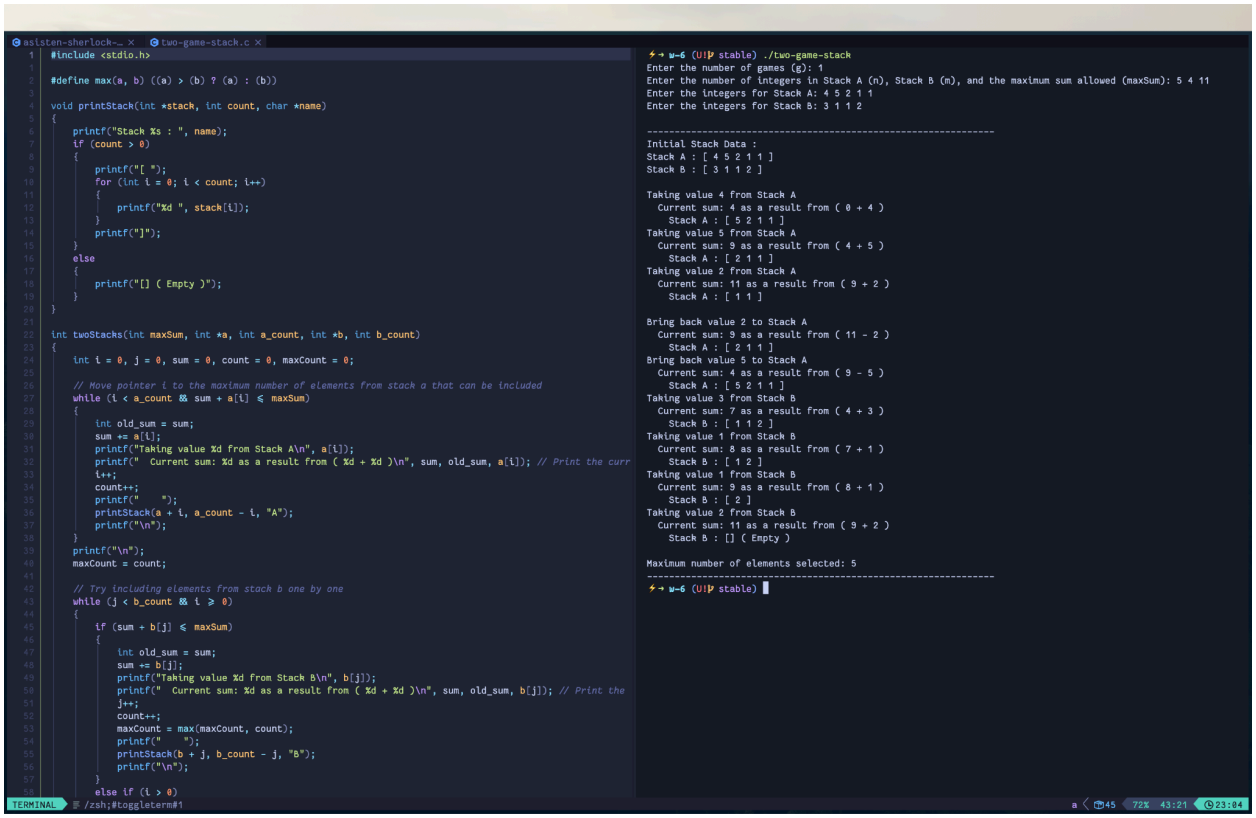
```
⚡→ w-6 (U!P stable) ./two-game-stack
Enter the number of games (g): 1
Enter the number of integers in Stack A (n), Stack B (m), and the maximum sum allowed (maxSum): 5 4 10
Enter the integers for Stack A: 4 2 4 6 1
Enter the integers for Stack B: 2 1 8 5

----------------------------------------------------------------
Initial Stack Data :
Stack A : [ 4 2 4 6 1 ]
Stack B : [ 2 1 8 5 ]

Taking value 4 from Stack A
  Current sum: 4 as a result from ( 0 + 4 )
    Stack A : [ 2 4 6 1 ]
Taking value 2 from Stack A
  Current sum: 6 as a result from ( 4 + 2 )
    Stack A : [ 4 6 1 ]
Taking value 4 from Stack A
  Current sum: 10 as a result from ( 6 + 4 )
    Stack A : [ 6 1 ]

Bring back value 4 to Stack A
  Current sum: 6 as a result from ( 10 - 4 )
    Stack A : [ 4 6 1 ]
Taking value 2 from Stack B
  Current sum: 8 as a result from ( 6 + 2 )
    Stack B : [ 1 8 5 ]
Taking value 1 from Stack B
  Current sum: 9 as a result from ( 8 + 1 )
    Stack B : [ 8 5 ]
Bring back value 2 to Stack A
  Current sum: 7 as a result from ( 9 - 2 )
    Stack A : [ 2 4 6 1 ]
Bring back value 4 to Stack A
  Current sum: 3 as a result from ( 7 - 4 )
    Stack A : [ 4 2 4 6 1 ]

Maximum number of elements selected: 4
----------------------------------------------------------------
⚡→ w-6 (U!P stable)
```




```c
#include <stdio.h>

#define max(a, b) ((a) > (b) ? (a) : (b))

void printStack(int *stack, int count, char *name)
{
    printf("Stack %s : ", name);
    if (count > 0)
    {
        printf("[ ");
        for (int i = 0; i < count; i++)
        {
            printf("%d ", stack[i]);
        }
        printf("]");
    }
    else
    {
        printf("[] ( Empty )");
```

```
    }
}

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

    // Move pointer i to the maximum number of elements from stack a that can be included
    while (i < a_count && sum + a[i] <= maxSum)
    {
        int old_sum = sum;
        sum += a[i];
        printf("Taking value %d from Stack A\n", a[i]);
        printf("  Current sum: %d as a result from ( %d + %d )\n", sum, old_sum, a[i]); //
Print the current value of sum
        i++;
        count++;
        printf("    ");
        printStack(a + i, a_count - i, "A");
        printf("\n");
    }
    printf("\n");
    maxCount = count;

    // Try including elements from stack b one by one
    while (j < b_count && i >= 0)
    {
        if (sum + b[j] <= maxSum)
        {
            int old_sum = sum;
            sum += b[j];
            printf("Taking value %d from Stack B\n", b[j]);
            printf("  Current sum: %d as a result from ( %d + %d )\n", sum, old_sum, b[j]);
// Print the current value of sum
            j++;
            count++;
            maxCount = max(maxCount, count);
            printf("    ");
            printStack(b + j, b_count - j, "B");
            printf("\n");
```

```
        }
        else if (i > 0)
        {
            int old_sum = sum;
            i--;
            sum -= a[i];
            printf("Bring back value %d to Stack A\n", a[i]);
            printf("  Current sum: %d as a result from ( %d - %d )\n", sum, old_sum, a[i]);
// Print the current value of sum
            count--;
            printf("    ");
            printStack(a + i, a_count - i, "A");
            printf("\n");
        }
        else
        {
            break;
        }
    }

    return maxCount;
}

int main()
{
    int g;
    printf("Enter the number of games (g): ");
    scanf("%d", &g);

    while (g--)
    {
        int n, m, maxSum;
        printf("Enter the number of integers in Stack A (n), Stack B (m), and the maximum sum
allowed (maxSum): ");
        scanf("%d %d %d", &n, &m, &maxSum);

        int a[n];
        printf("Enter the integers for Stack A: ");
        for (int i = 0; i < n; i++)
            scanf("%d", &a[i]);
```

```
        int b[m];
        printf("Enter the integers for Stack B: ");
        for (int i = 0; i < m; i++)
            scanf("%d", &b[i]);

        printf("\n");
        printf("----------------------------------------------------------------\n");
        printf("Initial Stack Data : \n");
        printStack(a, n, "A");
        printf("\n");
        printStack(b, m, "B");
        printf("\n\n");
        printf("\nMaximum number of elements selected: %d\n", twoStacks(maxSum, a, n, b, m));
        printf("----------------------------------------------------------------\n");
    }

    return 0;
}
```

2.2. Penjelasan Kode

1. `printStack`
   - Fungsi ini bertujuan untuk mencetak isi dari sebuah stack.
   - Parameter `stack` merupakan array yang merepresentasikan *stack*, `count` adalah jumlah elemen dalam *stack*, dan `name` adalah string yang menunjukkan nama dari *stack* tersebut.
   - Fungsi ini akan mencetak nama *stack*, diikuti dengan elemen-elemennya yang ditempatkan di dalam kurung siku. Jika *stack* kosong, maka akan mencetak pesan "`[ ] ( Empty )`" untuk menandakan bahwa *stack* tersebut tidak memiliki elemen.

2. `twoStacks`
   - Fungsi ini bertujuan untuk menghitung jumlah maksimum elemen yang dapat dipilih dari dua *stack* sedemikian rupa sehingga jumlahnya tidak melebihi jumlah maksimum yang diizinkan (`maxSum`).
   - Parameter `maxSum` adalah jumlah maksimum yang diizinkan, `a` adalah *array* yang merepresentasikan *Stack* A, `a_count` adalah jumlah elemen

dalam *Stack* A, `b` adalah *array* yang merepresentasikan Stack B, dan `b_count` adalah jumlah elemen dalam *Stack* B.

- Fungsi ini menggunakan dua pointer `i` dan `j` untuk mengiterasi melalui elemen-elemen *Stack* A dan *Stack* B.
- Pertama-tama, fungsi ini akan mencoba untuk memasukkan sebanyak mungkin elemen dari *Stack* A ke dalam jumlah yang dimungkinkan tanpa melebihi `maxSum`.
- Kemudian, fungsi ini akan mencoba untuk memasukkan elemen-elemen dari *Stack* B ke dalam jumlah yang dimungkinkan, sambil tetap memperhatikan agar total sum tidak melebihi `maxSum`,.
- Jika total `sum` melebihi `maxSum`, fungsi ini akan mencoba untuk mengurangi jumlah elemen dari *Stack* A sehingga total sum-nya menjadi lebih kecil.
- Setelah proses selesai, fungsi ini akan mengembalikan jumlah maksimum elemen yang berhasil dipilih dari kedua *stack*.

3. `main`
    - Fungsi ini merupakan fungsi utama yang akan dieksekusi pertama kali ketika program dijalankan.
    - Fungsi ini bertanggung jawab untuk membaca jumlah permainan (`g`) dari pengguna dan menjalankan logika permainan sebanyak `g` kali.
    - Untuk setiap permainan, fungsi ini akan membaca jumlah elemen dalam *Stack* A (`n`), *Stack* B (`m`), dan jumlah maksimum yang diizinkan (`maxSum`).
    - Selanjutnya, fungsi ini akan membaca elemen-elemen untuk *Stack* A dan *Stack* B dari pengguna.
    - Setelah itu, fungsi ini akan mencetak data awal dari kedua *stack* menggunakan fungsi `printStack`.
    - Kemudian, fungsi ini akan memanggil fungsi `twoStacks` untuk menghitung jumlah maksimum elemen yang dapat dipilih dan mencetak hasilnya.
    - Proses ini akan diulangi untuk setiap permainan hingga semua permainan selesai.

2.3. Kesimpulan

Kode tersebut merupakan implementasi solusi untuk masalah "Two Stacks Problem". Masalah ini melibatkan dua *stack*, di mana elemen-elemen dari kedua *stack* tersebut dimasukkan ke dalam susunan tertentu dengan tujuan memaksimalkan jumlah elemen yang dapat dipilih tanpa melebihi jumlah maksimum yang diizinkan.

Pertama, program memasukkan sebanyak mungkin elemen dari *Stack* A ke dalam *stack*, memperhatikan agar total jumlahnya tidak melebihi batas maksimum yang ditentukan (`maxSum`). Setelah itu, program mencoba untuk memasukkan elemen-elemen dari *Stack* B satu per satu ke dalam *stack*, sambil memastikan bahwa total jumlahnya tidak melebihi batas maksimum. Jika total jumlah melebihi batas maksimum, program akan mengurangi elemen-elemen dari *Stack* A sampai total jumlahnya kembali di bawah batas maksimum.

Setelah semua elemen telah diproses, program mengembalikan jumlah maksimum elemen yang berhasil dipilih dari kedua *stack*. Selama proses tersebut, setiap langkah yang dilakukan untuk memasukkan atau mengurangi elemen dari *stack* dicatat dan dicetak untuk membantu pemahaman algoritma tersebut. Dengan demikian, kode memberikan visualisasi yang jelas tentang bagaimana elemen-elemen diproses dan dipilih dari kedua *stack* untuk mencapai jumlah maksimum yang diinginkan tanpa melampaui batas maksimum yang ditetapkan.