

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

## 1. Asisten Sherlock Holmes

### 1.1. Penulisan Kode beserta output

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 // Self-referential structure for each node
4 struct Node
5 {
6     char alphabet;
7     struct Node *link;
8 };
9
10 // Function to initialize the nodes as per provided rules
11 void initializeNodes(struct Node *l1, struct Node *l2, struct Node *l3, struct Node *l4, struct Node *l5,
12 struct Node *l6, struct Node *l7, struct Node *l8, struct Node *l9)
13 {
14     l1->link = NULL;
15     l1->alphabet = 'F';
16     l2->link = NULL;
17     l2->alphabet = 'M';
18     l3->link = NULL;
19     l3->alphabet = 'A';
20     l4->link = NULL;
21     l4->alphabet = 'I';
22     l5->link = NULL;
23     l5->alphabet = 'K';
24     l6->link = NULL;
25     l6->alphabet = 'T';
26     l7->link = NULL;
27     l7->alphabet = 'N';
28     l8->link = NULL;
29     l8->alphabet = 'O';
30     l9->link = NULL;
31     l9->alphabet = 'R';
32 }
33
34 // Function to push an item onto the stack
35 void push(struct Node **top, char data)
36 {
37     struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
38     if (!newNode)
39     {
40         printf("Memory allocation failed.\n");
41         exit(EXIT_FAILURE);
42     }
43     newNode->alphabet = data;
44     newNode->link = *top;
45     *top = newNode;
46 }
47
48 // Function to pop an item from the stack
49 char pop(struct Node **top)
50 {
51     if (*top == NULL)
52     {
53         return '\0';
54     }
55     char data = (*top)->alphabet;
56     (*top) = (*top)->link;
57     return data;
58 }
59
60 int main()
61 {
62     struct Node *l1, *l2, *l3, *l4, *l5, *l6, *l7, *l8, *l9;
63     initializeNodes(l1, l2, l3, l4, l5, l6, l7, l8, l9);
64     push(&l1, 'I');
65     push(&l1, 'N');
66     push(&l1, 'F');
67     push(&l1, 'M');
68     push(&l1, 'A');
69     push(&l1, 'O');
70     push(&l1, 'K');
71     push(&l1, 'R');
72     printf("The formed word: ");
73     while (l1->link != NULL)
74     {
75         printf("%c", pop(&l1));
76     }
77     printf("%c", pop(&l1));
78     return 0;
79 }
```

→ w-6 (UIP stable) ./asisten-sherlock-holmes  
The formed word: INFORMATIKA  
→ w-6 (UIP stable) |

```
#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
```

```

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;
}

```

```

// 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 11, 12, 13, 14, 15, 16, 17, 18, 19;
    initializeNodes(&11, &12, &13, &14, &15, &16, &17, &18, &19);

    // Link the nodes according to the provided rules
    11.link = &12;
    12.link = &13;
    13.link = &14;
    14.link = &15;
    15.link = &16;
    16.link = &17;

```

```

17.link = &18;
18.link = &19;
19.link = &11;

// Print the formed word
printf("The formed word: ");
printf("%c%c%c%c%c%c%c%c%c\n",
13.link->alphabet,
13.link->link->link->link->alphabet,
13.link->link->link->link->link->link->link->alphabet,
13.link->link->link->link->link->alphabet,
13.link->link->link->link->link->link->alphabet,
13.link->link->link->link->link->link->link->link->alphabet,
13.link->link->link->alphabet,
13.link->alphabet,
13.link->link->alphabet,
13.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

```
1 #include <stdio.h>
2
3 #define max(a, b) ((a) > (b) ? (a) : (b))
4
5 void printStack(int *stack, int count, char *name)
6 {
7     printf("Stack %s : ", name);
8     if (count > 0)
9     {
10         printf("[");
11         for (int i = 0; i < count; i++)
12         {
13             printf("%d ", stack[i]);
14         }
15         printf("]");
16     }
17     else
18     {
19         printf("[ ( Empty )");
20     }
21 }
22
23 int twoStacks(int maxSum, int *a, int a_count, int *b, int b_count)
24 {
25     int i = 0, j = 0, sum = 0, count = 0, maxCount = 0;
26
27     // Move pointer i to the maximum number of elements from stack a that can be included
28     while (i < a_count && sum + a[i] <= maxSum)
29     {
30         int old_sum = sum;
31         sum += a[i];
32         printf("Taking value %d from Stack A\n", a[i]);
33         printf("Current sum: %d as a result from ( %d + %d )\n", sum, old_sum, a[i]); // Print the curr
34         i++;
35         count++;
36         printf(" ");
37         printStack(a + i, a_count - i, "A");
38         printf("\n");
39     }
40     printf("\n");
41     maxCount = count;
42
43     // Try including elements from stack b one by one
44     while (j < b_count && i >= 0)
45     {
46         if (sum + b[j] <= maxSum)
47         {
48             int old_sum = sum;
49             sum += b[j];
50             printf("Taking value %d from Stack B\n", b[j]);
51             printf("Current sum: %d as a result from ( %d + %d )\n", sum, old_sum, b[j]); // Print the
52             j++;
53             count++;
54             maxCount = max(maxCount, count);
55             printf(" ");
56             printStack(b + j, b_count - j, "B");
57             printf("\n");
58         }
59         else if (i > 0)
60         {
61             sum -= a[i];
62             i--;
63             printf("Bring back value %d to Stack A\n", a[i]);
64             printf("Current sum: %d as a result from ( %d - %d )\n", sum, old_sum, a[i]); // Print the
65             count--;
66             printf(" ");
67             printStack(a + i, a_count - i, "A");
68             printf("\n");
69         }
70     }
71
72     printf("Maximum number of elements selected: %d\n", maxCount);
73 }
```

```
4 → w=4 (UIP 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

4 → w=6 (UIP stable)
```

```
asisten-sherlock- X two-game-stack.c X
1 #include <stdio.h>
2
3 #define max(a, b) ((a) > (b) ? (a) : (b))
4
5 void printStack(int *stack, int count, char *name)
6 {
7     printf("Stack %s : ", name);
8     if (count > 0)
9     {
10         printf("[ ");
11         for (int i = 0; i < count; i++)
12         {
13             printf("%d ", stack[i]);
14         }
15         printf("]");
16     }
17     else
18     {
19         printf("[ ] ( Empty )");
20     }
21 }
22
23 int twoStacks(int maxSum, int *a, int a_count, int *b, int b_count)
24 {
25     int i = 0, j = 0, sum = 0, count = 0, maxCount = 0;
26
27     // Move pointer i to the maximum number of elements from stack a that can be included
28     while (i < a_count & sum + a[i] <= maxSum)
29     {
30         int old_sum = sum;
31         sum += a[i];
32         printf("Taking value %d from Stack A\n", a[i]);
33         printf(" Current sum: %d as a result from ( %d + %d )\n", sum, old_sum, a[i]); // Print the curr
34         i++;
35         count++;
36         printStack(a + i, a_count - i, "A");
37         printf("\n");
38     }
39     printf("\n");
40     maxCount = count;
41
42     // Try including elements from stack b one by one
43     while (j < b_count & i >= 0)
44     {
45         if (sum + b[j] <= maxSum)
46         {
47             int old_sum = sum;
48             sum += b[j];
49             printf("Taking value %d from Stack B\n", b[j]);
50             printf(" Current sum: %d as a result from ( %d + %d )\n", sum, old_sum, b[j]); // Print the
51             j++;
52             count++;
53             maxCount = max(maxCount, count);
54             printf(" ");
55             printStack(b + j, b_count - j, "B");
56             printf("\n");
57         }
58         else if (i > 0)
59         {
60             i--;
61             sum -= a[i];
62             printf("Bring back value %d to Stack A\n", a[i]);
63             printf(" Current sum: %d as a result from ( %d - %d )\n", sum, old_sum, a[i]); // Print the curr
64             i--;
65             count--;
66             printStack(a + i, a_count - i, "A");
67             printf("\n");
68         }
69         else if (j > 0)
70         {
71             j--;
72             sum -= b[j];
73             printf("Bring back value %d to Stack B\n", b[j]);
74             printf(" Current sum: %d as a result from ( %d - %d )\n", sum, old_sum, b[j]); // Print the curr
75             j--;
76             count--;
77             printStack(b + j, b_count - j, "B");
78             printf("\n");
79         }
80     }
81     printf("\n");
82     printf("Maximum number of elements selected: %d\n", maxCount);
83 }
84
85 int main()
86 {
87     int n, m, maxSum;
88     printf("Enter the number of games (g): ");
89     scanf("%d", &n);
90     printf("Enter the number of Integers in Stack A (n), Stack B (m), and the maximum sum allowed (maxSum): ");
91     scanf("%d %d %d", &n, &m, &maxSum);
92     printf("Enter the Integers for Stack A: ");
93     int *a = (int *)malloc(n * sizeof(int));
94     for (int i = 0; i < n; i++)
95     {
96         scanf("%d", &a[i]);
97     }
98     printf("Enter the Integers for Stack B: ");
99     int *b = (int *)malloc(m * sizeof(int));
100    for (int i = 0; i < m; i++)
101    {
102        scanf("%d", &b[i]);
103    }
104    twoStacks(maxSum, a, n, b, m);
105    return 0;
106 }
```

```
#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.