

LAPORAN PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

SOAL LATIHAN ASD – STRUCT DAN STACK



Oleh :

Muhammad Fajri Dwi Prasetya Subandi / 1203230076

IF-03-01

Program Studi Informatika

Fakultas Informatika

Universitas Telkom Surabaya

Tahun 2024

Jawaban!

1. Source Code

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

typedef struct Node {
    char alphabet;
    struct Node* link;
} Node;

typedef struct Stack {
    Node* top;
} Stack;

Stack* createStack() {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->top = NULL;
    return stack;
}

void push(Stack* stack, Node* node) {
    node->link = stack->top;
    stack->top = node;
}

Node* pop(Stack* stack) {
    if (stack->top == NULL) {
        return NULL;
    }
    Node* temp = stack->top;
    stack->top = stack->top->link;
    temp->link = NULL;
    return temp;
}

int main() {
    Node 11, 12, 13, 14, 15, 16, 17, 18, 19;

    11.alphabet = 'F';
    11.link = NULL;

    12.alphabet = 'M';
    12.link = NULL;
```

```
13.alphabet = 'A';
13.link = NULL;

14.alphabet = 'I';
14.link = NULL;

15.alphabet = 'K';
15.link = NULL;

16.alphabet = 'T';
16.link = NULL;

17.alphabet = 'N';
17.link = NULL;

18.alphabet = 'O';
18.link = NULL;

19.alphabet = 'R';
19.link = NULL;

Stack* stack = createStack();

int countI = 0, countA = 0, countK = 0;

push(stack, &l3);
push(stack, &l5);
push(stack, &l4);
push(stack, &l6);
push(stack, &l3);
push(stack, &l2);
push(stack, &l9);
push(stack, &l8);
push(stack, &l1);
push(stack, &l7);
push(stack, &l4);

printf("KATA: ");
while (stack->top != NULL) {
    Node* current = pop(stack);
    if (current->alphabet == 'I') {
        printf("%c", current->alphabet);
        countI++;
        if (countI == 2) {
            printf("%c%c", 'K', 'A');
            countK++;
        }
    }
}
```

```

        countA++;
    }
    } else {
        printf("%c", current->alphabet);
    }
}
printf("\n");

free(stack);

return 0;
}

```

SS Output

```

PS C:\Telkom University\Semester 2\Algoritma dan Struktur Data\latihan_soal_struct_dan_stack> cd "c:\Telkom University\Semester 2\Algoritma dan Struktur Data
\latihan_soal_struct_dan_stack\" ; if ($?) { gcc latsol1.c -o latsol1 } ; if ($?) { .\latsol1 }
KATA: INFORMATIKA
PS C:\Telkom University\Semester 2\Algoritma dan Struktur Data\latihan_soal_struct_dan_stack>

```

Penjelasan

1. Struktur Node: Mendefinisikan struktur Node yang berisi satu huruf abjad dan pointer ke node berikutnya dalam tumpukan.
2. Struktur Stack: Mendefinisikan struktur Stack yang memiliki pointer ke node paling atas (top) dari tumpukan.
3. Fungsi createStack(): Membuat tumpukan baru dengan mengalokasikan memori untuk Stack dan mengatur pointer top menjadi NULL.
4. Fungsi push(): Menambahkan node ke tumpukan. Node baru ditambahkan di atas node paling atas dan pointer top diubah sehingga menunjuk ke node baru.
5. Fungsi pop(): Menghapus node paling atas dari tumpukan dan mengembalikan node tersebut. Pointer top diperbarui untuk menunjuk ke node di bawahnya.
6. Fungsi main():
 - Menginisialisasi node-node dengan huruf abjad yang membentuk kata "INFORMATIKA".
 - Membuat tumpukan.
 - Menyimpan jumlah panggilan huruf I, A, dan K.
 - Menumpuk node-node sesuai dengan urutan yang diinginkan.
 - Mengeluarkan node dari tumpukan dan mencetak hurufnya.
 - Ketika huruf I kedua ditemukan, mencetak huruf K dan A.

- Menghapus tumpukan setelah selesai menggunakan.

2. Source Code

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);
```

```
int parse_int(char*);
```

```
/*
 * Complete the 'twoStacks' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 * 1. INTEGER maxSum
 * 2. INTEGER_ARRAY a
 * 3. INTEGER_ARRAY b
 */
```

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

```
    // Mengambil sebanyak mungkin elemen dari tumpukan a
    while (i < a_count && sum + a[i] <= maxSum) {
        sum += a[i];
        i++;
        count++;
    }
```

```
    // Mengambil elemen dari tumpukan b dan mengurangi elemen dari tumpukan a jika
    diperlukan
```

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

```

    j++;
    while (sum > maxSum && i > 0) {
        i--;
        sum -= a[i];
    }
    if (sum <= maxSum && count < i + j) {
        count = i + j;
    }
}

return count;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int g = parse_int(ltrim(rtrim(readline())));

    for (int g_itr = 0; g_itr < g; g_itr++) {
        char** first_multiple_input = split_string(rtrim(readline()));

        int n = parse_int(*(first_multiple_input + 0));

        int m = parse_int(*(first_multiple_input + 1));

        int maxSum = parse_int(*(first_multiple_input + 2));

        char** a_temp = split_string(rtrim(readline()));

        int* a = malloc(n * sizeof(int));

        for (int i = 0; i < n; i++) {
            int a_item = parse_int(*(a_temp + i));

            *(a + i) = a_item;
        }

        char** b_temp = split_string(rtrim(readline()));

        int* b = malloc(m * sizeof(int));

        for (int i = 0; i < m; i++) {
            int b_item = parse_int(*(b_temp + i));

            *(b + i) = b_item;
        }
    }
}

```

```

    }

    int result = twoStacks(maxSum, n, a, m, b);

    fprintf(fp, "%d\n", result);
}

fclose(fp);

return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';
    }
}

```

```

    data = realloc(data, data_length);

    if (!data) {
        data = '\0';
    }
} else {
    data = realloc(data, data_length + 1);

    if (!data) {
        data = '\0';
    } else {
        data[data_length] = '\0';
    }
}

return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {

```



```

        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

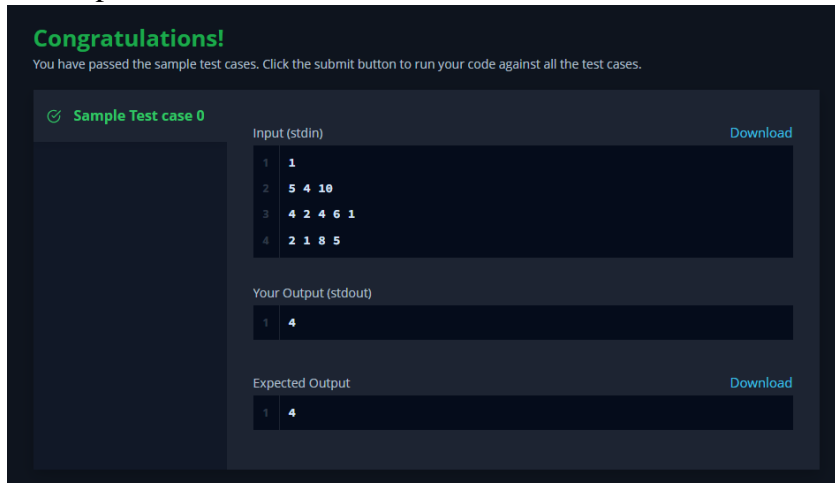
int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}

```

SS Output



Penjelasan

1. `int twoStacks(int maxSum, int a_count, int* a, int b_count, int* b) {`: Fungsi `twoStacks` menerima lima parameter: `maxSum` yang merupakan jumlah maksimum nilai yang diperbolehkan, `a_count` yang merupakan jumlah elemen dalam tumpukan `a`, `a` yang merupakan array yang berisi elemen-elemen tumpukan `a`, `b_count` yang merupakan jumlah elemen dalam tumpukan `b`, dan `b` yang merupakan array yang berisi elemen-elemen tumpukan `b`.
2. `int i = 0, j = 0, sum = 0, count = 0;`: Inisialisasi variabel `i`, `j`, `sum`, dan `count`. `i` dan `j` akan digunakan sebagai indeks untuk menelusuri tumpukan `a` dan `b`, `sum` akan menyimpan jumlah nilai elemen yang telah diambil, dan `count` akan menyimpan jumlah total elemen yang diambil dari kedua tumpukan.
3. `while (i < a_count && sum + a[i] <= maxSum) {`: Looping untuk mengambil sebanyak mungkin elemen dari tumpukan `a` selama total nilai elemen yang diambil belum melebihi `maxSum`.
4. `sum += a[i];`: Menambahkan nilai elemen ke `sum`.
5. `i++;`: Menambahkan indeks `i`.
6. `count++;`: Menambahkan `count` karena satu elemen telah diambil dari tumpukan `a`.
7. `while (j < b_count && i >= 0) {`: Looping untuk mengambil elemen dari tumpukan `b` dan mengurangi elemen dari tumpukan `a` jika total nilai elemen yang diambil melebihi `maxSum`.
8. `sum += b[j];`: Menambahkan nilai elemen dari tumpukan `b` ke `sum`.
9. `j++;`: Menambahkan indeks `j`.

10. ``while (sum > maxSum && i > 0) {``: Looping untuk mengurangi elemen dari tumpukan ``a`` jika total nilai elemen yang diambil melebihi ``maxSum``.
11. ``i--`;`: Mengurangi indeks ``i``.
12. ``sum -= a[i];``: Mengurangi nilai elemen dari tumpukan ``a`` dari ``sum``.
13. ``if (sum <= maxSum && count < i + j) {``: Memeriksa apakah jumlah total elemen yang diambil saat ini kurang dari jumlah sebelumnya. Jika iya, maka ``count`` akan diupdate dengan jumlah elemen yang baru.
14. ``count = i + j`;`: Mengupdate ``count`` dengan jumlah elemen yang diambil dari tumpukan ``a`` dan ``b``.
15. ``return count`;`: Mengembalikan jumlah total elemen yang dapat diambil dari kedua tumpukan.

Tambahkan **visualisasi** alur penyelesaiannya (per langkah) dalam bentuk stack pada laporan apabila input diubah menjadi sebagai berikut.

1

5 4 11

4 5 2 1 1

3 1 1 2

Visualisasi dalam bentuk stack:

Tumpukan a:

4 5 2 1 1

Tumpukan b:

3 1 1 2

Langkah 1:

$i = 4, \text{sum} = 4, \text{count} = 1$

Tumpukan a:

5 2 1 1

Tumpukan b:

3 1 1 2

Langkah 2:

$i = 3$, $sum = 9$, $count = 2$

Tumpukan a:

2 1 1

Tumpukan b:

3 1 1 2

Langkah 3:

$i = 2$, $sum = 11$, $count = 3$

Tumpukan a:

1 1

Tumpukan b:

3 1 1 2

Langkah 4:

$i = 1$, $sum = 12$, $count = 4$

Tumpukan a:

1

Tumpukan b:

3 1 1 2

Langkah 5:

$i = 0$, $sum = 13$, $count = 5$

Tumpukan a:

(empty)

Tumpukan b:

3 1 1 2

Langkah 6:

$j = 0$, $\text{sum} = 16$, $\text{count} = 5$

Tumpukan a:

(empty)

Tumpukan b:

1 1 2

Langkah 7:

$j = 1$, $\text{sum} = 17$, $\text{count} = 5$

Tumpukan a:

(empty)

Tumpukan b:

1 2

Langkah 8:

$j = 2$, $\text{sum} = 18$, $\text{count} = 5$

Tumpukan a:

(empty)

Tumpukan b:

2

Langkah 9:

$j = 3$, $\text{sum} = 20$, $\text{count} = 5$

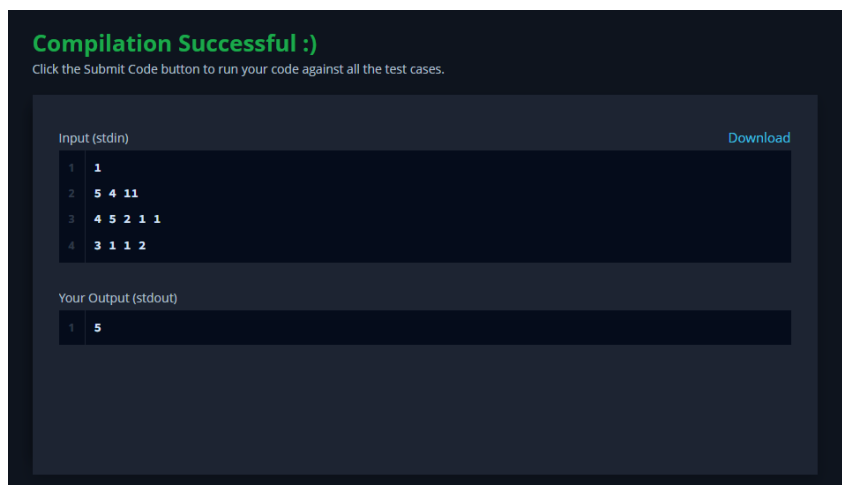
Tumpukan a:

(empty)

Tumpukan b:

(empty)

Hasil akhir: $\text{count} = 5$



Compilation Successful :)
Click the Submit Code button to run your code against all the test cases.

Input (stdin)		Download
1	1	
2	5 4 11	
3	4 5 2 1 1	
4	3 1 1 2	

Your Output (stdout)	
1	5