

Nama : Haidar Fulca Kurniawan
NIM : 1203230077
Kelas : IF 03-02

TUGAS PRAKTIKUM ALGORITMA STRUKTUR DATA

SOAL 1

- Source Code

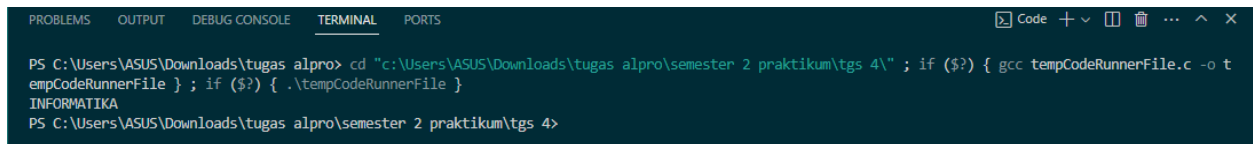
```
• #include <stdio.h>
•
• struct huruf
• {
•     struct huruf *link;
•     char alphabet;
• };
•
• int main()
• {
•     struct huruf l1, l2, l3, l4, l5, l6, l7, l8, l9;
•
•     l1.link = NULL;
•     l1.alphabet = 'F';
•
•     l2.link = NULL;
•     l2.alphabet = 'M';
•
•     l3.link = NULL;
•     l3.alphabet = 'A';
•
•     l4.link = NULL;
•     l4.alphabet = 'I';
•
•     l5.link = NULL;
•     l5.alphabet = 'K';
•
•     l6.link = NULL;
•     l6.alphabet = 'T';
•
```

```

•     17.link = NULL;
•     17.alphabet = 'N';
•
•     18.link = NULL;
•     18.alphabet = 'O';
•
•     19.link = NULL;
•     19.alphabet = 'R';
•
•     14.link = &l7;
•     17.link = &l1;
•     11.link = &l8;
•     18.link = &l9;
•     19.link = &l2;
•     12.link = &l3;
•     13.link = &l6;
•     16.link = &l4;
•
•     printf("%c",
14.alphabet);
•     printf("%c", 14.link-
>alphabet);
•     printf("%c", 14.link->link-
>alphabet);
•     printf("%c", 14.link->link->link-
>alphabet);
•     printf("%c", 14.link->link->link->link-
>alphabet);
•     printf("%c", 14.link->link->link->link->link-
>alphabet);
•     printf("%c", 14.link->link->link->link->link->link-
>alphabet);
•     printf("%c", 14.link->link->link->link->link->link->link-
>alphabet);
•     printf("%c", 14.link->link->link->link->link->link->link->link-
>alphabet);
•
•     14.link = &l5;
•     15.link = &l3;
•
•     printf("%c", 14.link->alphabet);
•     printf("%c", 14.link->link->alphabet);
•
•     return 0;
• }

```

• Output



```
PS C:\Users\ASUS\Downloads\tugas alpro> cd "c:\Users\ASUS\Downloads\tugas alpro\semester 2 praktikum\tgs 4\" ; if ($?) { gcc tempCodeRunnerFile.c -o t
empCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
INFORMATIKA
PS C:\Users\ASUS\Downloads\tugas alpro\semester 2 praktikum\tgs 4>
```

• Penjelasan

1. Program Code C ini berfungsi untuk memberikan output “INFORMATIKA” dari karakter karakter yang acak yang telah ditemukan di sebuah batu.

2. Pada **Main Program** pertama tama dideklarasikan variabel 11, 12, 13, 14, 15, 16, 17, 18, 19 dengan struktur data **struct huruf**.

- **Struct huruf** sudah dideklarasikan diatas sebelum **main**. Dengan elemen **struct huruf *link** (**nested**) dan elemen **alphabet**.

3. Kemudian di inisiasikan setiap variabel l (1 sampai 9) dengan elemen **link** ialah NULL. Lalu dinisiasikan juga elemen alphabet variabel 11 ialah F; 12 ialah M; 13 ialah A; 14 ialah I; 15 ialah K; 16 ialah T; 17 ialah N; 18 ialah O; 19 ialah R.

4. Setelah itu dibuatlah sebuah link berantai untuk mengakses huruf. Pertama diinisiasikan bahwa 14.link ialah huruf dari 17, 17.link ialah huruf dari 11, 11.link ialah huruf dari 18, 18.link ialah huruf dari 19, 19.link ialah huruf dari 12, 12.link ialah huruf dari 13, 13.link ialah huruf dari 16, 16.link ialah huruf dari 14.

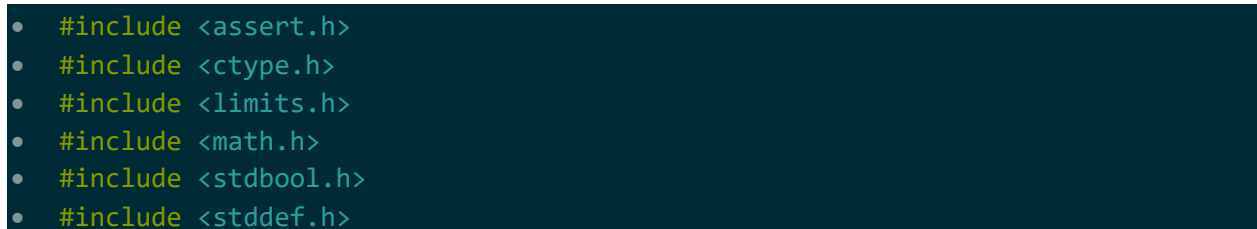
5. Selanjutnya akan di print dari masing masing huruf. Yang pertama ialah mengoutputkan huruf I dengan memanggil 14.alphabet. lalu mengoutputkan huruf N dengan meanggil 14.link->alphabet. Cara itu digunakan untuk dapat mengakses huruf N dengan inisiasi sebelumnya. Begitu seterusnya sampai output huruf I yaitu dengan memanggil 14.link->link->link->link->link->link->link->alphabet. Dengan begitu output sudah keluar I N F O R M A T I.

5. Jika sudah kita harus menginisiasikan kembali bahwa 14.link ialah huruf dari 15 yaitu K dan 15.link ialah 13 yaitu A. dengan itu maka kita bisa melakukan printf kembali seperti tadi. Meng output kan K dengan memanggil 14.link->alphabet dan A dengan memanggil 14.link->link->alphabet.

6. Setelah semua dijalankan, maka program akan memberikan output INFORMATIKA.

SOAL 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 *getline();
• 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;
•     while (i < a_count && sum + a[i] <= maxSum)
•     {
•         sum += a[i];
•         i++;
•     }
•     count = i;
•     while (j < b_count && i >= 0)
•     {
•         sum += b[j];
•         j++;
•         while (sum > maxSum && i > 0)
•         {
•             i--;
•             sum -= a[i];
•         }
•         if (sum <= maxSum && i + j > count)
•         {
•             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(fptr, "%d\n", result);
•     }

```

```

•
•     fclose(fptr);
•
•     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;
• }

```

• Output

☒ Test against custom input

```

1
5 4 11
4 5 2 1 1
3 1 1 2
    
```

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
---	---

• Penjelasan

1. Program Code C ini berfungsi untuk menentukan jumlah maksimum elemen yang dapat diambil dari kedua tumpukan (tumpukan A dan B) sedemikian sehingga total nilai elemennya tidak melebihi batas maksimum.

2. Pertama tama diinisiasi variabel `i`, `j`, `sum`, dan `count` yang semuanya dimulai dari 0.
3. Selanjutnya ditelusuri tumpukan A. Ditambahkan elemen-elemen dari tumpukan A ke dalam tumpukan sementara (`sum`) selama total nilai elemen tidak melebihi `maxSum`.
 - Iterasi pertama: `sum` = 4 (elemen pertama dari tumpukan A)
 - Iterasi kedua: `sum` = 6
 - Iterasi ketiga: `sum` = 10
 - Iterasi keempat: `sum` = 16 (lebih besar dari `maxSum`, berhenti)
4. Simpan jumlah elemen yang dapat diambil dari tumpukan A ke dalam `count`. Dalam kasus ini, `count` adalah 4 karena hanya 4 elemen pertama dari tumpukan A yang dapat diambil tanpa melebihi `maxSum`.
5. Setelah itu mulai menelusuri tumpukan B. Kita tambahkan elemen-elemen dari tumpukan B ke dalam tumpukan sementara (`sum`) sambil mencoba menghapus elemen-elemen dari tumpukan A jika total nilai elemen melebihi `maxSum`.
 - Iterasi pertama: `sum` = 18 (elemen pertama dari tumpukan B + 4 elemen pertama dari tumpukan A)
 - Iterasi kedua: `sum` = 19
 - Iterasi ketiga: `sum` = 27 (lebih besar dari `maxSum`, hapus elemen dari tumpukan A)
 - Iterasi keempat: `sum` = 15 (elemen terakhir dari tumpukan B + 3 elemen pertama dari tumpukan A)
 - Iterasi kelima: `sum` = 21
 - Iterasi keenam: `sum` = 26
 - Iterasi ketujuh: `sum` = 31 (lebih besar dari `maxSum`, hapus elemen dari tumpukan A)
 - Iterasi kedelapan: `sum` = 19 (elemen terakhir dari tumpukan B + 1 elemen pertama dari tumpukan A)
6. Lalu cek apakah jumlah elemen yang dapat diambil dari kedua tumpukan (`count`) saat ini lebih besar dari jumlah sebelumnya. Jika ya, kita perbarui `count`.
7. Kemudian kembalikan nilai `count` sebagai hasil dari fungsi `twoStacks`. Dalam kasus ini, `count` akan bernilai 4, yang merupakan jumlah maksimum elemen yang dapat diambil dari kedua tumpukan tanpa melebihi `maxSum`. Oleh karena itu, output dari program akan menjadi 4.