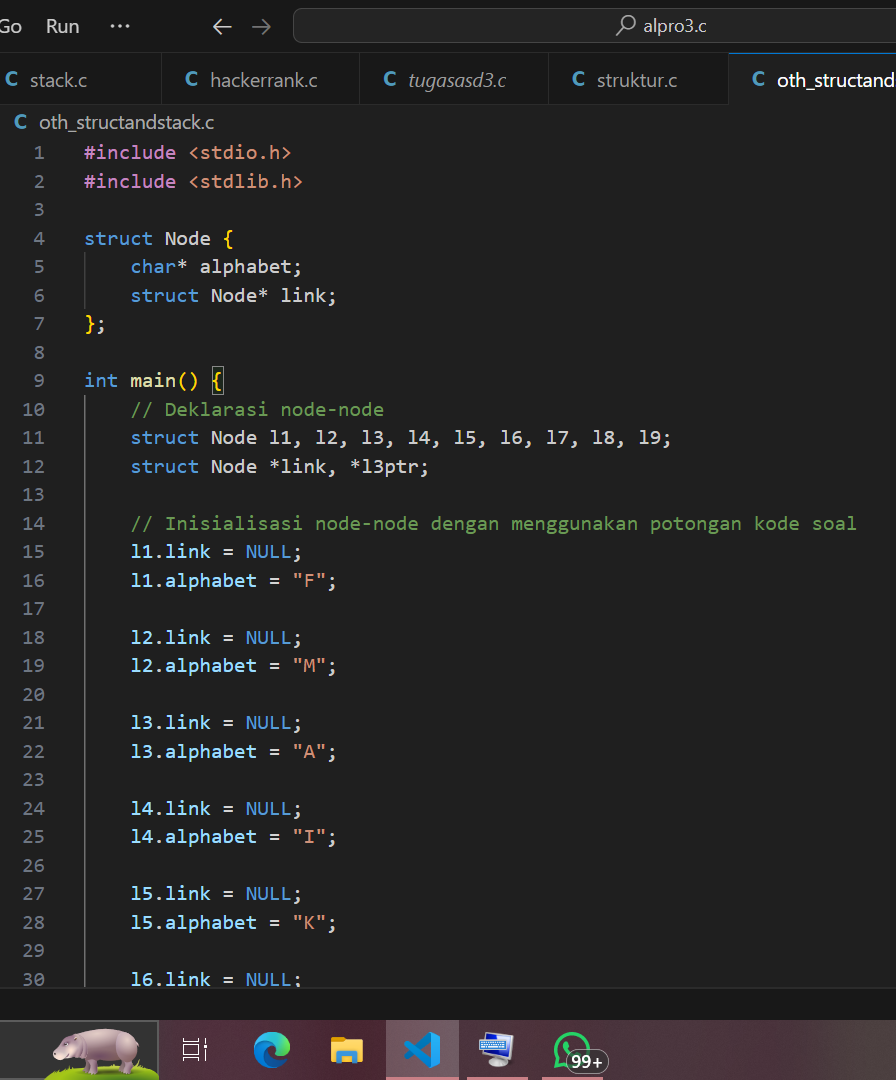
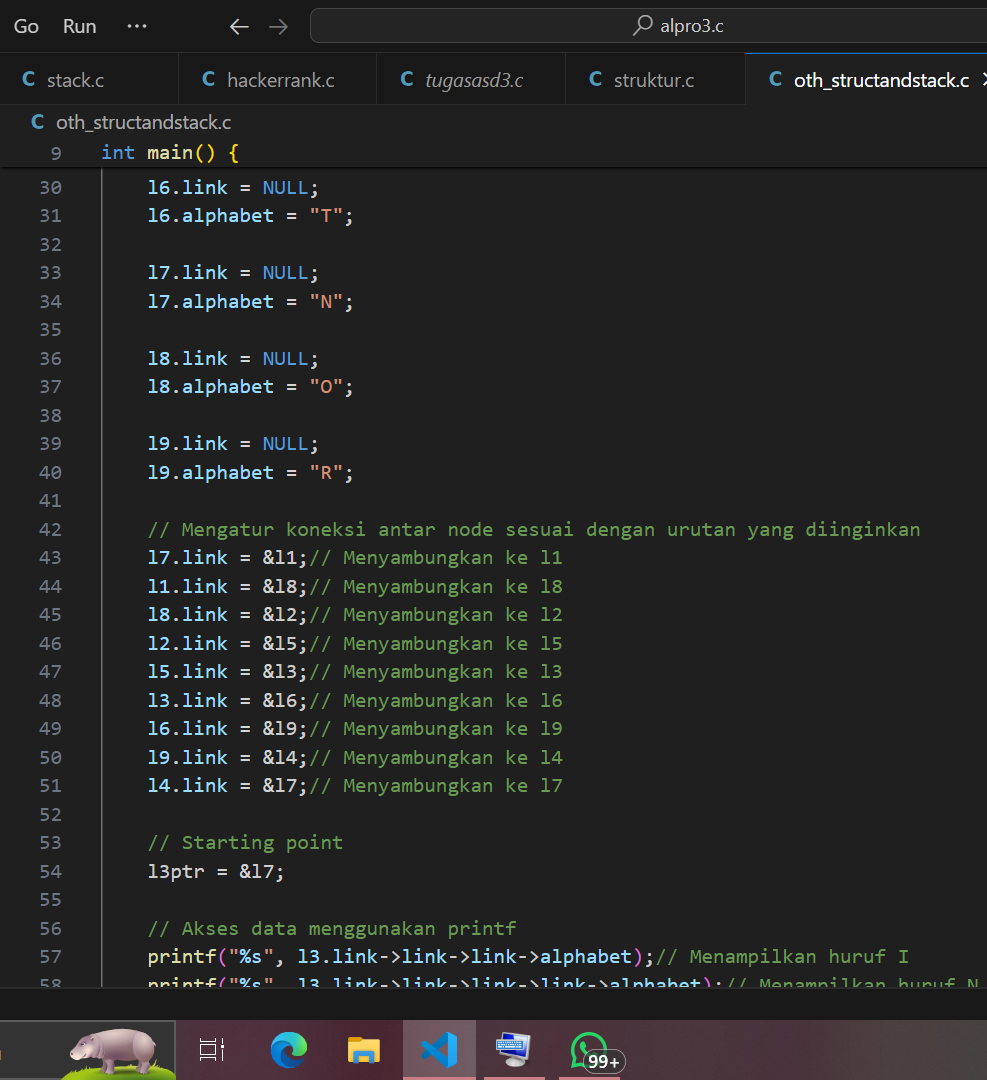
Neisya nurul alyazara

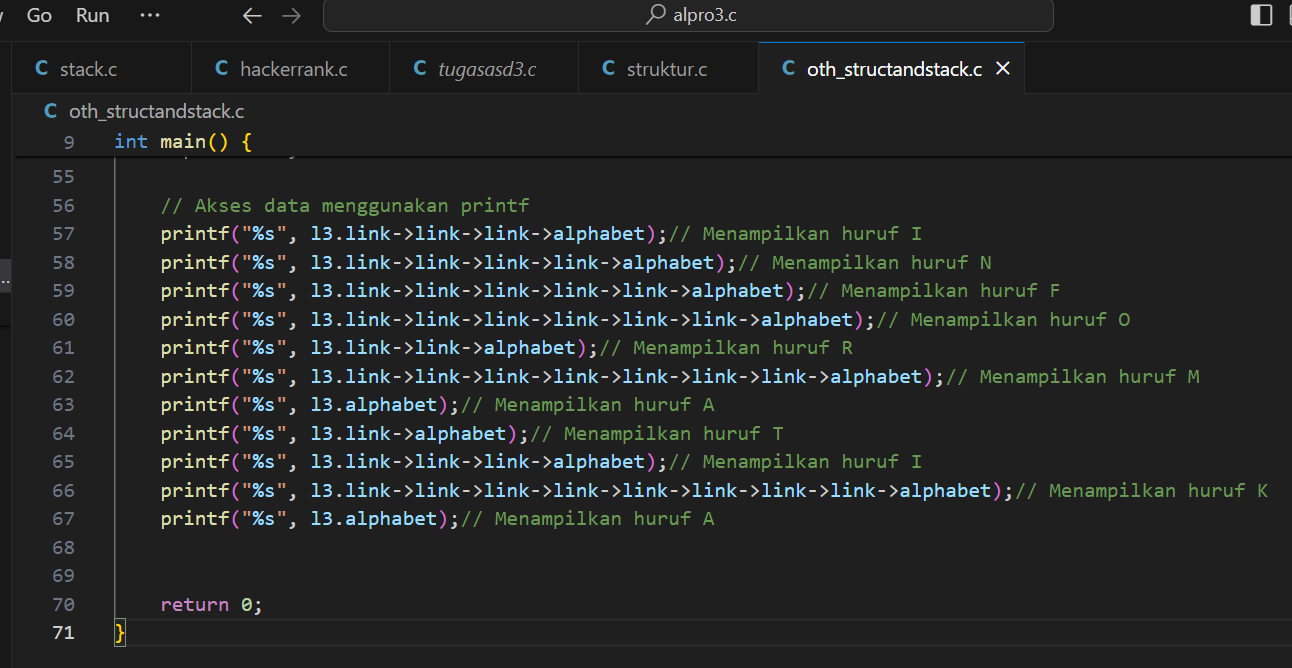
If 03-01

1203230098

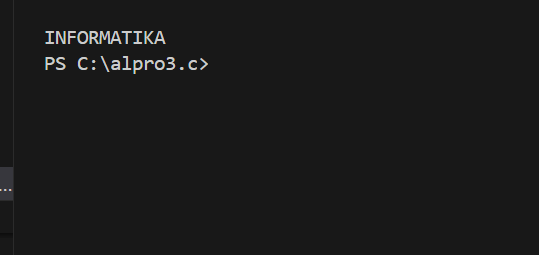
1. Input:







Output:



1. Hackerrank

input:

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

char\* readline();

char\* ltrim(char\*);

char\* rtrim(char\*);

char\*\* split\_string(char\*);

int parse\_int(char\*);

int twoStacks(int maxSum, int a\_count, int\* a, int b\_count, int\* b) {

    int count = 0;

    int sum = 0;

    int idx\_a = 0, idx\_b = 0;

    while (idx\_a < a\_count && sum + a[idx\_a] <= maxSum) {

        sum += a[idx\_a];

        idx\_a++;

        count++;

    }

    int max\_count = count;

    while (idx\_b < b\_count && idx\_a >= 0) {

        sum += b[idx\_b];

        idx\_b++;

        count++;

        while (sum > maxSum && idx\_a > 0) {

            idx\_a--;

            sum -= a[idx\_a];

            count--;

        }

        if (sum <= maxSum && count > max\_count) {

            max\_count = count;

        }

    }

    return max\_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);

        free(a);

        free(b);

    }

    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') {

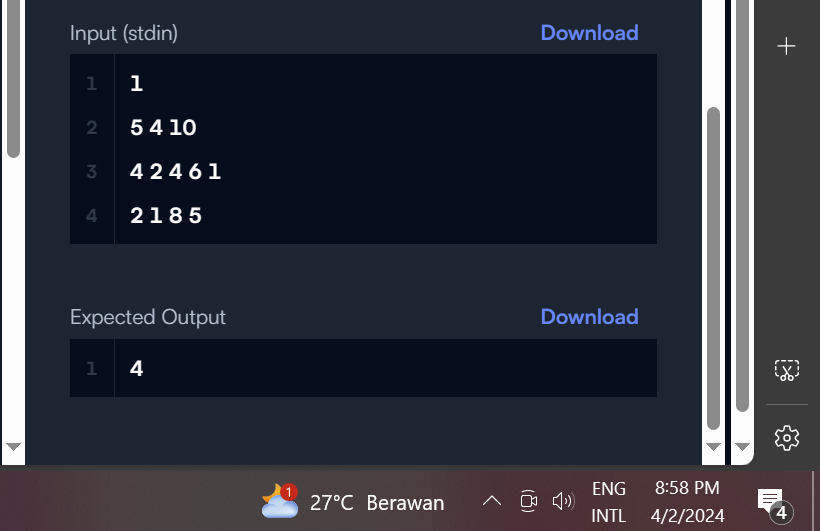
        exit(EXIT\_FAILURE);

    }

    return value;

}

Output:



Penjelasan:

readline(): Membaca input dari stdin (standar input) secara dinamis, mengalokasikan memori seiring dengan pertambahan panjang input.

ltrim() dan rtrim(): Menghapus spasi atau karakter kosong dari awal (kiri) dan akhir (kanan) sebuah string.

split\_string(): Memisahkan string input menjadi token-token berdasarkan spasi, kemudian menyimpannya dalam array strings.

parse\_int(): Mengubah string menjadi bilangan integer.

twoStacks(): Fungsi utama yang memecahkan masalah dua tumpukan. Logikanya adalah sebagai berikut:

Mengambil sebanyak mungkin elemen dari tumpukan pertama (a) tanpa melebihi batas sum maksimum.

Kemudian, menambahkan elemen-elemen dari tumpukan kedua (b) satu per satu ke dalam tumpukan pertama, sambil menghapus elemen-elemen dari tumpukan pertama jika total sum melebihi batas sum maksimum.

Menyimpan jumlah maksimum elemen yang diambil dari kedua tumpukan.

main(): Fungsi utama program, membaca input dari stdin, memanggil twoStacks() untuk setiap kasus uji, dan menuliskan output ke stdout (standar output).

Terdapat alokasi memori dinamis menggunakan malloc() untuk menyimpan array dari elemen-elemen tumpukan. Kemudian, membebaskan memori menggunakan free() setelah penggunaan selesai untuk menghindari kebocoran memori.

Program juga membaca input dan menulis output menggunakan file untuk keperluan uji kasus yang lebih besar, dengan menggunakan variabel OUTPUT\_PATH dari lingkungan.

Program menggunakan stdlib.h, stdio.h, stdbool.h, dan string.h untuk fungsi-fungsi standar dan lain-lain.