MODUL 9 TUGAS BESAR

Gamaliel Adaran Nadiuarihon (13221018)
Jan Patrick Yeremia (13221019)
Vanny Alviolani Indriyani (13221020)
Hiradini Rahmah (13221021)
Asisten: Muhammad Daffa Rasyid (13220059)
Kelompok: Kelompok B2
EL2208-Praktikum Pemecahan Masalah dengan C
Laboratorium Dasar Teknik Elektro - Sekolah Teknik Elektro dan Informatika ITB

Abstrak

Pengimplementasian bahasa C dapat dilihat dalam berbagai aspek kehidupan, baik dari penggunaan arduino sampai program-program aplikasi komputer dan handphone yang digunakan sehari-hari. Oleh karena itu, sangat penting untuk kita agar memahami berbagai teknik dan algoritma dalam bahasa pemrograman C, seperti jenis-jenis struktur penyimpanan data dan berbagai problem solving algorithm agar kita dapat menyelesaikan berbagai masalah dengan memanfaatkan bahasa C.

Kata kunci: algoritma, struktur data, problem solving

1. PENDAHULUAN

Pada modul ini, kami membuat sebuah program untuk membuat sebuah perbatasan efektif dengan *input* lokasi koordinat bujur dan lintang dari berbagai markas yang perlu dicakup di dalam perbatasan tersebut serta dengan syarat jarak antar markas agar tidak melebihi 2500 km, tujuan dari praktikum kali ini adalah:

- Memahami pengaplikasian berbagai jenis struktur data
- Memahami penggunaan algoritma *problem solving* dalam menyelesaikan sebuah masalah
- Mengetahui cara pengimplementasian berbagai rumus dan perhitungan dalam bentuk fungsi
- Memahami pentingnya penentuan parameter dalam pembuatan sebuah fungsi
- Memahami pentingnya penentuan parameter syarat dalam menyelesaikan suatu masalah dengan algoritma-algoritma *problem solving*

2. STUDI PUSTAKA

2.1 STRUCT

Struct adalah tipe bentukan oleh pengguna yang digunakan untuk menggabungkan beberapa tipe data [1]. *Struct* didefinisikan menggunakan *struct statement* yang memiliki format sebagai berikut [1]

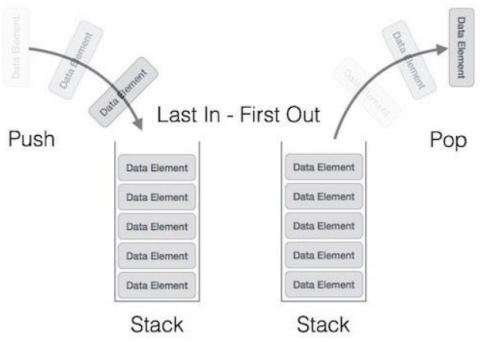
```
struct [structure tag] {

   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

structure tag adalah nama dari tipe data yang baru dibentuk, *member definition* adalah deklarasi variabel, dan *structure variable* adalah nama variabel yang dibentuk dengan tipe data bentukan tersebut [2].

Terdapat dua cara mengakses *structure*. Jika penggunaan *structure* sebagai sebuah variabel, digunakan tanda titik (.) untuk mengakses anggotanya [2]. Namun, jika penggunaan *structure* sebagai sebuah *pointer*, digunakan tanda panah (→) untuk mengakses anggotanya [2].

2.2 STACK

Stack adalah salah satu tipe data abstrak yang sering digunakan pada berbagai Bahasa pemrograman [2]. *Stack* akan menyimpan data dalam sebuah "tumpukan" yang hanya dapat diakses dari satu buah sisi saja [2]. Ilustrasi dari *stack* dapat dilihat pada gambar berikut.

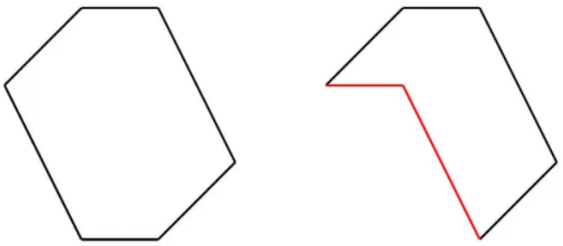


Gambar 2-1 Ilustrasi *Stack* [3]

Stack dapat diimplementasikan menggunakan *array*, *structure*, atau *linked list* [2]. Terdapat beberapa operasi dasar yang sering digunakan dalam mengolah *stack* yaitu *push*, *pop*, *peek*, dan *isEmpty* [2].

2.3 Convex Hull

Convex hull dari himpunan titik-titik didefinisikan sebagai poligon konveks terkecil yang melingkupi semua titik pada himpunan tersebut. *Convex* berarti poligon tersebut tidak memiliki sudut yang membengkok ke arah dalam [4]. Terdapat beberapa algoritma yang dapat digunakan untuk mengimplementasikan permasalahan *convex hull* ini, yaitu Jarvis March, Graham scan, Quickhull, Divide and Conquer, Monotone chain, Incremental convex hull algorithm, Kirkpatrick–Seidel algorithm, dan Chan's algorithm [5].

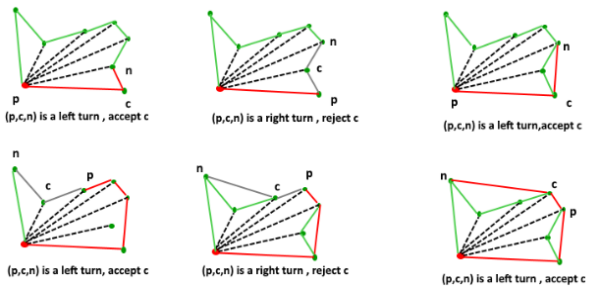


Gambar 2-2 Ilustrasi Graham Scan [4]

2.4 Convex Hull Algorithm : Graham Scan

Graham scan adalah salah satu algoritma yang digunakan untuk menyelesaikan permasalahan *convex hull*.

Cara kerja dari Graham scan adalah sebagai berikut. Pertama-tama akan dicari terlebih dahulu titik yang terendah. Titik ini akan dijadikan sebagai titik awal dari *convex hull*. Kemudian, sebanyak n-1 titik yang tersisa akan diurutkan berdasarkan sudut *counterclockwise* yang terbentuk terhadap titik awal. Jika terdapat dua atau lebih titik yang sudutnya bernilai sama, singkirkan titik tersebut kecuali titik yang memiliki jarak terjauh dari titik awal. Setelah pengurutan selesai, tambahkan titik yang tersisa ke dalam *stack*. Jika titik antara elemen teratas *stack*, elemen kedua teratas *stack*, dan titik ke-n membentuk sudut *counterclockwise*, tambahkan titik ke-n tersebut ke dalam *stack*. Namun, jika sudut yang terbentuk adalah *clockwise*, singkirkan titik tersebut [6].

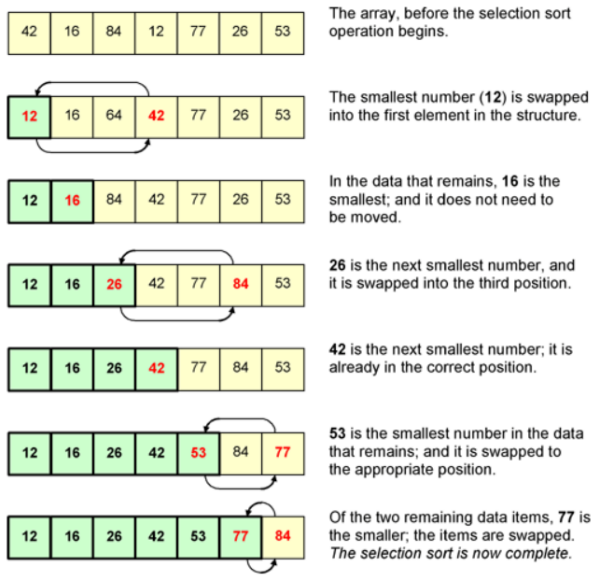


Gambar 2-3 Ilustrasi Graham Scan [7]

2.5 Selection Sort

Selection sort adalah sebuah algoritma pengurutan yang sederhana. Algoritma ini membagi elemen yang akan diurutkan menjadi dua bagian, yaitu elemen terurut pada bagian kiri dan elemen yang belum terurut pada bagian kanan [8].

Berikut adalah cara kerja dari algoritma *selection sort*. Lakukan pencarian elemen terkecil kemudian pindahkan menjadi elemen pertama. Langkah selanjutnya adalah melakukan pencarian elemen terkecil mulai dari elemen kedua sampai dengan elemen ke-n kemudian pindahkan menjadi elemen kedua. Hal ini terus dilakukan sampai dengan elemen yang terakhir. Berikut adalah ilustrasi dari cara kerja *selection sort*.



Gambar 2-4 Ilustrasi *Selection Sort* [9]

2.6 Haversine Formula

Formula Haversine adalah sebuah cara yang akurat untuk mengukur jarak antara dua titik pada permukaan sebuah bola menggunakan nilai lintang dan bujur [10].

Berikut adalah formula Haversine

$$a = sin^2(\phi_B - \frac{\phi_A}{2}) + cos(\phi_A * cos(\phi_B) * sin^2(\lambda_B - \frac{\lambda_A}{2})$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R * c$$

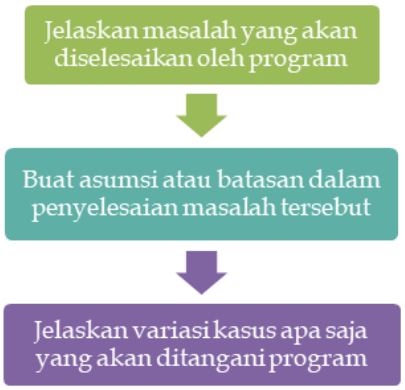
Keterangan :

- Φ adalah lintang
- λ adalah bujur
- R adalah jari-jari Bumi
- Sudut dalam satuan radian

3. METODOLOGI

Langkah-langkah pengerjaan tugas besar ini adalah sebagai berikut.

3.1 RUANG LINGKUP MASALAH

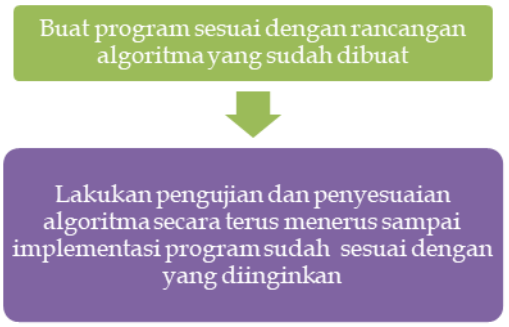


Gambar 3-1 Diagram alur pendefinisian ruang lingkup masalah

3.2 RANCANGAN

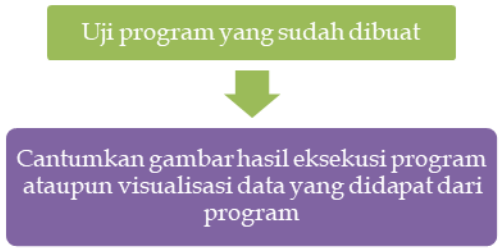
Gambar 3-2 Diagram alur rancangan program

3.3 IMPLEMENTASI



Gambar 3-3 Diagram alur implementasi program

3.4 PENGUJIAN



Gambar 3-4 Diagram alur pengujian program

4. HASIL DAN ANALISIS

4.1 RUANG LINGKUP MASALAH

Masalah yang diselesaikan oleh program pada tugas besar ini adalah masalah penentuan perbatasan efektif negara api yang ditarik dari satu markas ke markas lainnya. Perbatasan efektif ini harus dibuat seminimal mungkin dengan jarak antarmarkas tidak lebih dari 2500 km. Selain itu, program juga harus bisa menentukan keliling dari perbatasan efektif yang dibentuk dan markas mana saja yang terletak di luar perbatasan efektif.

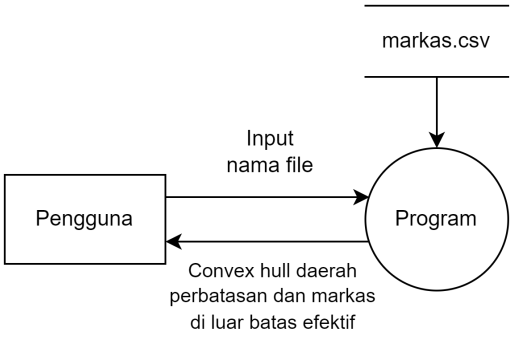
Input yang diterima oleh program berupa nama *file* yang berisikan daftar nama serta koordinat markas negara api. Praktikan mengasumsikan file yang akan dibaca akan selalu memiliki format csv dengan data di dalamnya adalah nama, posisi lintang, dan posisi bujur dari tiap markas. *Output* dari program ini adalah perbatasan efektif negara api berupa nama-nama tiap markas negara api yang menjadi batas perbatasan efektif. Markas pertama yang ditulis adalah markas yang terletak paling bawah pada sumbu koordinat kemudian dilanjutkan ke markas selanjutnya sampai kembali lagi di markas pertama. Setelah itu, program juga menuliskan panjang perbatasan efektif negara api serta daftar markas yang berada di luar perbatasan efektif.

Variasi *testcase* yang ditangani oleh program ini antara lain adalah ketika nama *file* tidak ditemukan atau pengguna meng-*input* nama *file* yang tidak sesuai format. Pada kasus ini, program akan memberitahu bahwa *file* yang tidak ditemukan dan program berakhir. *Testcase* selanjutnya adalah ketika perbatasan efektif yang sudah dibuat program tidak ada. Hal ini dapat disebabkan *file* yang kosong karena negara api belum memiliki markas, jumlah markas kurang dari tiga buah, ataupun jarak antarmarkas yang melebihi batas (>2500 km). Pada kasus ini, program akan mengeluarkan pesan bahwa perbatasan efektif negara api tidak bisa dibuat.

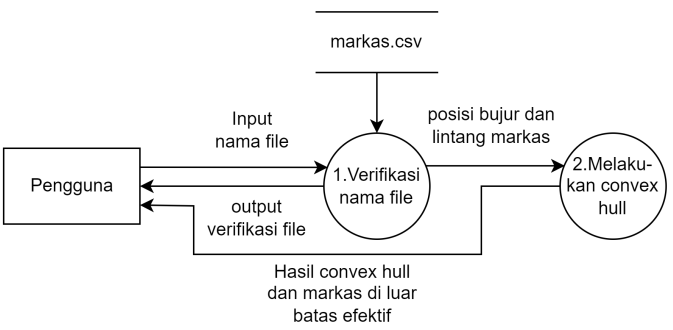
4.2 Rancangan

Program akan terdiri dari empat proses, yaitu proses meminta nama *file* dari pengguna dan membuka *file*, proses memindahkan data, proses pembuatan *convex hull*, dan terakhir adalah proses menampilkan *output*.

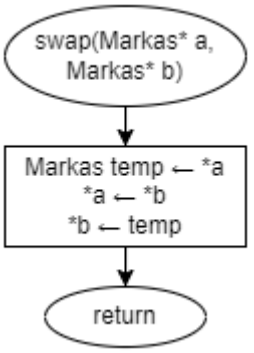
Pertama-tama, program akan meminta nama *file* dengan format csv kepada pengguna. Setelah menerima nama *file*, program akan membaca file tersebut. Akan dilakukan validasi apakah file tersebut benar atau tidak. Jika benar, program akan melakukan pemindahan data. Proses selanjutnya adalah proses pembuatan *convex hull*. Proses terakhir adalah menampilkan hasil yang didapat. Berikut adalah DFD dan *flowchart* dari fungsi main serta fungsi-fungsi operasional dari program ini (untuk gambar yang lebih jelas, lihat lampiran).



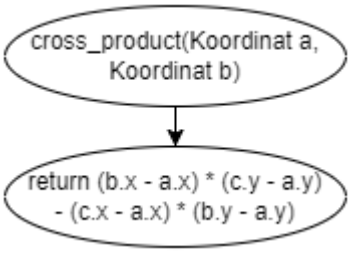
Gambar 4-1 DFD Level 0



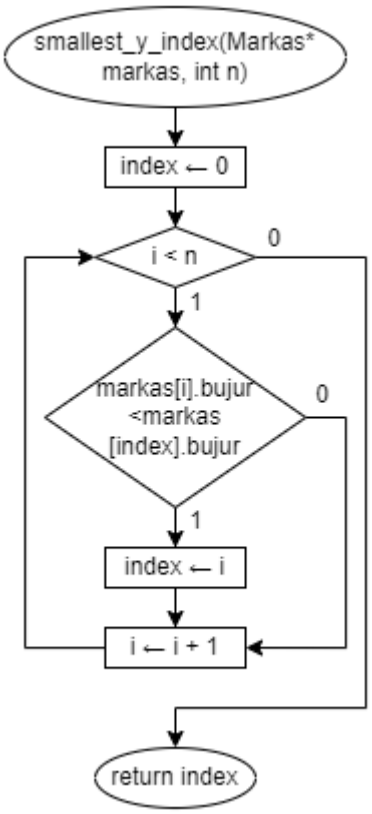
Gambar 4-2 DFD Level 1



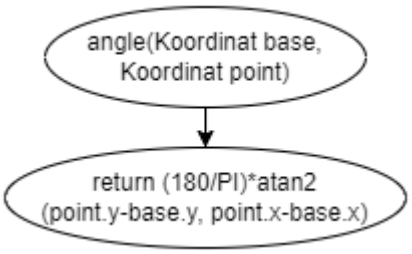
Gambar 4-3 *Flowchart* fungsi `swap()`



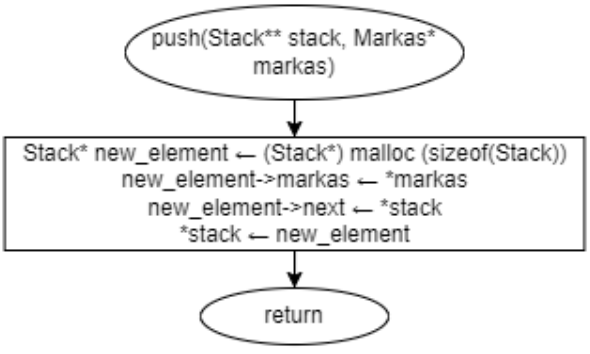
Gambar 4-4 *Flowchart* Fungsi `cross_product()`



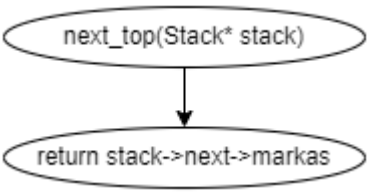
Gambar 4-5 *Flowchart* fungsi **`smallest_y_index()`**



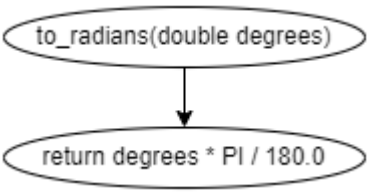
Gambar 4-6 *Flowchart* fungsi **`angle()`**



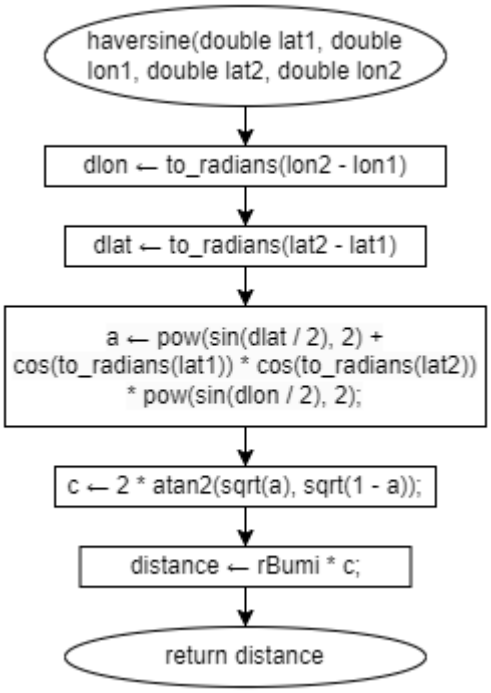
Gambar 4-7 *Flowchart* fungsi **`push()`**



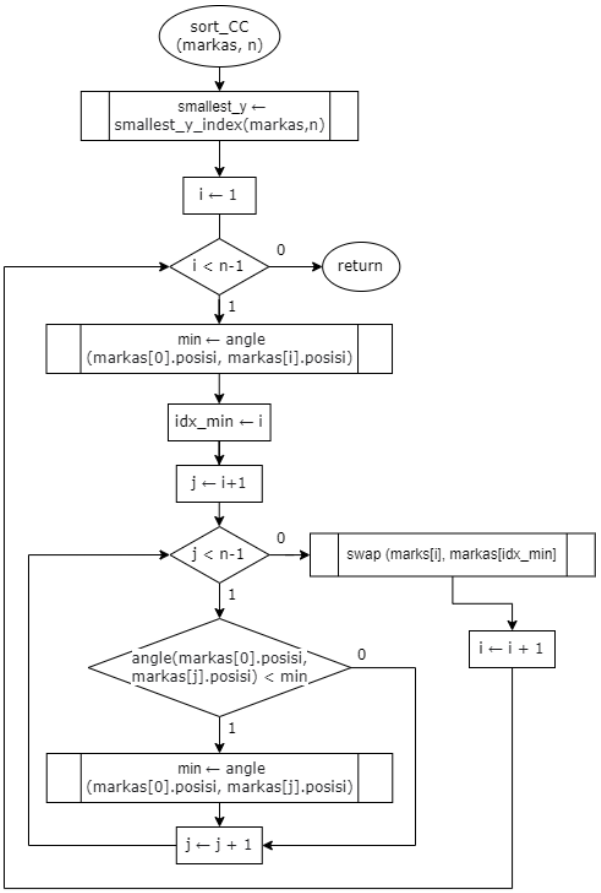
Gambar 4-8 *Flowchart* fungsi **`next_top()`**



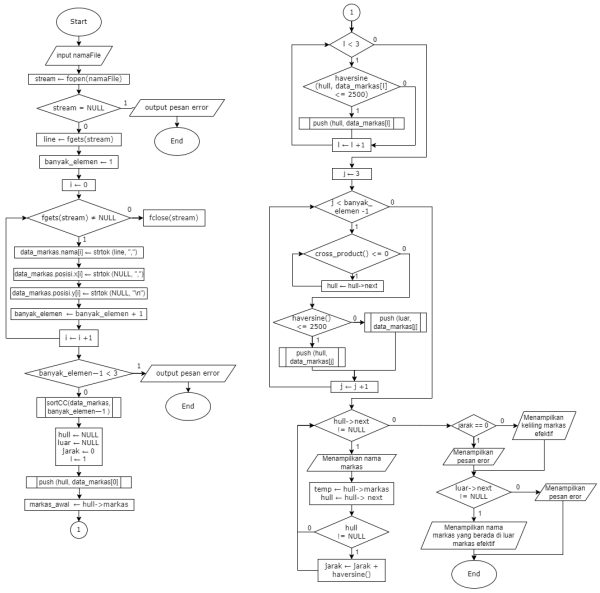
Gambar 4-9 *Flowchart* fungsi **`to_radians()`**



Gambar 4-10 *Flowchart* fungsi **`haversine()`**



Gambar 4-11 *Flowchart* fungsi `sort_CC()`



Gambar 4-12 *Flowchart* fungsi `main()`

Tabel 4-1 Pembagian Tugas

Task	Pembagian Tugas
Pembacaan file (files.c, files.h)	Designer: - 13221020 Implementer - 13221020 Tester - 13221018 - 13221019 - 13221021
Pemindahan data dari file ke program	Designer: - 13221020 Implementer - 13221020 Tester - 13221018 - 13221019 - 13221021
Convex hull	Designer: - 13221018 Implementer - 13221018 Tester - 13221019 - 13221020 - 13221021
Fungsi-fungsi operasional	Designer: - 13221019 - 13221021 Implementer - 13221019 - 13221021 Tester - 13221018 - 13221020
Integrasi (main.c, lib.h)	Implementer: - 13221018 Tester: - 13221019 - 13221020 - 13221021

4.3 IMPLEMENTASI

4.3.1 Kode main.h

```
// Tipe bentukan untuk posisi. Variabel x untuk
lintang dan variabel y untuk bujur.
typedef struct{
    double x;
    double y;
} Koordinat;
```

Struct `Koordinat` digunakan untuk menyimpan nilai lintang dan nilai bujur. Nilai lintang disimbolkan dengan x dan nilai bujur disimbolkan dengan y.

```
// Tipe bentukan untuk identitas sebuah markas.
typedef struct {
    char nama[max_char];
    Koordinat posisi;
} Markas;
```

Struct `Markas` digunakan untuk menyimpan identitas dari markas. *Struct* ini terdiri atas *string* untuk nama markas dan tipe bentukan koordinat.

```
// Stack untuk menyimpan convex hull.
typedef struct Stack{
    Markas markas;
    struct Stack* next;
} Stack;
```

Struct `Stack` digunakan untuk menyimpan data markas hasil dari operasi *convex hull*. *Stack* terdiri atas tipe bentukan Markas dan pointer terhadap *stack*.

```
// Fungsi untuk menukar posisi.
void swap(Markas* a, Markas* b){
    Markas temp = *a;
    *a = *b;
    *b = temp;
}
```

Fungsi `swap()` menerima dua parameter bertipe Markas. Fungsi ini digunakan untuk menukar posisi dari kedua variabel tersebut.

```
// Fungsi yang digunakan untuk menghitung nilai
perkalian cross dari tiga titik.
double cross_product(Koordinat a, Koordinat b,
Koordinat c) {
    return (b.x - a.x) * (c.y - a.y) - (c.x - a.x) *
(b.y - a.y);
}
```

Fungsi `cross_product()` menerima tiga parameter bertipe Koordinat. Fungsi ini digunakan untuk menghitung nilai perkalian silang antara tiga titik. Fungsi ini digunakan untuk mencari letak sebuah titik relatif kepada sebuah garis, dimana jika hasil fungsi `cross_product()` bernilai positif, maka titik c berada di samping kanan dari garis ab, dan jika bernilai negatif, maka titik berada pada arah kiri dari garis.

```
// Fungsi untuk mencari nilai bujur yang terkecil
dari sebuah array bertipe Markas.
int smallest_y_index(Markas* markas, int n){
    int index = 0;
    for(int i = 0; i < n; i++){

if(markas[i].posisi.y<markas[index].posisi.y){
            index = i;
        }
    }
    return index;
}
```

Fungsi `smallest_y_index()` menerima dua parameter, yaitu elemen pertama dari sebuah *array* bertipe Markas dan banyaknya elemen pada *array* tersebut. Fungsi ini digunakan untuk mencari indeks dari elemen dengan nilai y atau nilai bujur yang terendah. Dengan kata lain, fungsi ini digunakan untuk mencari indeks dari elemen yang memiliki posisi paling bawah.

```
// Fungsi untuk mencari sudut dari dua titik.
double angle(Koordinat base, Koordinat point){
    return
(180/PI)*atan2(point.y-base.y,point.x-base.x);
}
```

Fungsi `angle()` menerima dua parameter yang bertipe Koordinat. Fungsi ini digunakan untuk mencari sudut dalam satuan derajat antara kedua titik tersebut.

```
// Fungsi untuk mengurutkan array bertipe markas
berdasarkan sudutnya terhadap titik yang letaknya
berada paling bawah.
// Urutan dimulai dari yang paling kecil menuju yang
paling besar.
void sort_CC(Markas* markas, int n){
    int smallest_y = smallest_y_index(markas, n);
    swap(&markas[0], &markas[smallest_y]);

    for(int i = 1; i < n-1; i++){
        double min = angle(markas[0].posisi,
markas[i].posisi);
        int idx_min = i;
        for(int j = i+1; j < n-1; j++){
            if(angle(markas[0].posisi,
markas[j].posisi)<min){
                min = angle(markas[0].posisi,
markas[j].posisi);
                idx_min = j;
            }
        }
        swap(&markas[i],&markas[idx_min]);
    }
}
```

Fungsi `sort_cc()` menerima dua parameter, yaitu elemen pertama dari sebuah *array* bertipe Markas dan banyaknya elemen pada *array* tersebut. Fungsi ini digunakan untuk mencari elemen yang memiliki posisi paling bawah kemudian memindahkannya menjadi elemen pertama dari *array*. Prinsip kerja pengurutan pada fungsi ini menggunakan prinsip kerja dari fungsi *selection sort*. Setiap elemen akan dibandingkan satu sama lain berdasarkan sudut yang dibentuk dengan elemen pertama *array*. Hasil akhir dari fungsi ini adalah sebuah *array* bertipe Markas yang memiliki elemen pertama berupa markas dengan posisi terbawah dan elemen yang lainnya terurut membesar berdasarkan sudut yang dibentuk dengan elemen pertama *array*.

```
// Fungsi untuk menambahkan elemen pada stack.
void push(Stack** stack, Markas* markas){
```

```
    Stack* new_element = (Stack*) malloc
(sizeof(Stack));
    new_element->markas = *markas;
    new_element->next = *stack;
    *stack = new_element;
}
```

Fungsi `push()` menerima dua parameter, yaitu sebuah pointer terhadap *stack* dan sebuah variabel bertipe Markas. Fungsi ini digunakan untuk menambahkan elemen markas ke dalam sebuah *stack*.

```
// Fungsi untuk mengambil elemen kedua dari atas
sebuah stack.
Markas next_top(Stack* stack){
    return stack->next->markas;
}
```

Fungsi `next_top()` menerima sebuah parameter, yaitu sebuah *stack*. Fungsi ini digunakan untuk melihat elemen kedua dari atas sebuah *stack*.

```
// Fungsi untuk mengubah derajat menjadi radian.
double to_radians(double degrees) {
    return degrees * PI / 180.0;
}
```

Fungsi `to_radians()` menerima sebuah parameter, yaitu derajat. Fungsi ini digunakan untuk mengubah satuan derajat menjadi satuan radian.

```
// Fungsi untuk menghitung jarak menurut formula
haversine.
double haversine(double lat1, double lon1, double
lat2, double lon2) {
   double dlon, dlat, a, c, distance;

    dlon = to_radians(lon2 - lon1);
    dlat = to_radians(lat2 - lat1);

    a = pow(sin(dlat / 2), 2) +
cos(to_radians(lat1)) * cos(to_radians(lat2)) *
pow(sin(dlon / 2), 2);
    c = 2 * atan2(sqrt(a), sqrt(1 - a));
    distance = rBumi * c;

    return distance;
}
```

Fungsi `haversine()` menerima empat parameter, yaitu nilai lintang dan bujur dari titik pertama serta nilai lintang dan bujur dari titik kedua. Fungsi ini digunakan untuk menghitung jarak antara dua titik menggunakan formula haversine.

4.3.2 Kode main.c

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

```
int main(){
    // Membaca file csv.
    Markas *data_markas;
    data_markas = (Markas*)
                  malloc(sizeof(Markas));
    char nama_file[max_char];

    printf("Masukkan nama file: ");
    scanf("%s", nama_file);

    FILE* stream = fopen(nama_file, "r");

    // Verifikasi nama file.
      if (stream == NULL){
        printf("File tidak dapat dibuka.
                Program Berakhir.");
        return 0;
    }
```

Potongan kode di atas adalah proses pertama, yaitu meminta nama *file* dari pengguna. Setelah program menerima nama *file*, program akan membuka *file* tersebut untuk dibaca isinya. Kemudian dilakukan validasi, jika *file* yang diberikan kosong, program akan menampilkan pesan eror dan program selesai. Jika ternyata *file* tidak kosong, program akan melanjutkan ke proses selanjutnya.

```
// Memindahkan data dari file eksternal ke array.
    char line[255];
    char* token;
    int banyak_elemen = 1;
    int i = 0;
    fgets(line, 255, stream);

    while(fgets(line, 255, stream)) {
      token = strtok(line,",");
      strcpy(data_markas[i].nama,token);
      token = strtok(NULL, ",");
      data_markas[i].posisi.x = atof(token);
      token = strtok(NULL, "\n");
      data_markas[i].posisi.y= atof(token);
      banyak_elemen++;
      i++;
      data_markas = realloc(data_markas,(banyak_
                    elemen)*sizeof(Markas));
    }

    fclose(stream);

    if (banyak_elemen-1 < 3){
        printf("Perbatasan efektif tidak bisa
                dibuat\n");
        return 0;
    }
```

Proses kedua adalah proses memindahkan data. Program akan melewati baris pertama dari data yang terdapat di *file* karena berdasarkan format csv yang diberikan, baris pertama adalah keterangan. Selanjutnya, program akan membaca setiap baris untuk mengambil nama markas, nilai lintang, dan nilai bujur yang dipisahkan oleh koma kemudian memindahkannya ke dalam *array* yang bernama data_markas. Hal ini terus dilakukan sampai baris terakhir. Bersamaan dengan hal tersebut, dilakukan juga pelacakan banyaknya elemen.

Setelah proses pemindahan data selesai, program akan menutup *file* eksternal tersebut dan

melakukan validasi. Validasi dilakukan terhadap banyaknya elemen dari *array* data_markas. Jika elemen *array* bernilai 3, program akan menampilkan pesan eror dan program selesai. Jika elemen sama dengan atau lebih dari 3, program akan melanjutkan ke proses pembuatan *convex hull*.

```
// Mencari markas yang memiliki posisi paling bawah
(bujur atau y paling rendah) dan mengurutkan array
berdasarkan sudut yang dibentuk
// secara counterclockwise terhadap markas yang
paling bawah tersebut.
    sort_CC(data_markas, banyak_elemen-1);

    // Melakukan Convex hull.
    Stack* hull = (Stack*) malloc(sizeof(Stack));
    hull = NULL;
    Stack* luar = (Stack*) malloc(sizeof(Stack));
    luar = NULL;
    double jarak = 0;
    Markas markas_awal;


    push(&hull, &data_markas[0]);
    markas_awal = hull->markas;
    for(int l = 1; l < 3; l++){
        if(haversine(hull->markas.posisi.x, hull->
           markas.posisi.y, data_markas[l].posi
           si.x, data_markas[l].posisi.y) <= 2500){
            push(&hull, &data_markas[l]);
        }
    }

    for(int j = 3; j < banyak_elemen-1; j++){
        while(cross_product(next_top(hull).posisi,
              hull->markas.posisi,data_markas
              [j].posisi) <= 0){
            hull = hull->next;
        }
        if(haversine(hull->markas.posisi.x, hull->
           markas.posisi.y, data_markas[j].posi
           si.x, data_markas[j].posisi.y) <= 2500){
            push(&hull, &data_markas[j]);
        } else {
            push(&luar, &data_markas[j]);
        }
    }
```

Proses selanjutnya adalah proses pembuatan *convex hull*. Pertama-tama, akan dicari titik terendah dari *array* data_markas kemudian menjadikan titik tersebut elemen pertama dari *array* data_markas. Setelah itu, elemen-elemen dari *array* data_markas akan diurut membesar menurut sudut *counterclockwise* yang dibentuk dengan elemen pertama tadi. Untuk melakukan hal ini, digunakan fungsi `sort_CC()`.

Proses ini kemudian dilanjutkan dengan mendeklarasikan dua buah *stack*, yaitu hull dan luar. *Stack* hull digunakan untuk menyimpan markas hasil dari pembuatan *convex hull*, sedangkan *stack* luar digunakan untuk menyimpan markas yang berada di luar *convex hull*.

Pertama-tama program akan menambahkan markas pertama ke dalam hull. Pada umumnya, tiga titik pertama akan membentuk *convex hull*. Akan tetapi, karena terdapat syarat tambahan bahwa jarak antarmarkas harus ≤ 2500, akan dilakukan pemeriksaan terlebih dahulu. Jika jarak antara elemen teratas dari hull dengan markas ke-l dari data_markas memiliki jarak ≤ 2500, tambahkan markas tersebut ke dalam hull menggunakan fungsi `push()`. Jika jaraknya lebih dari 2500, singkirkan. Hal ini dilakukan hanya untuk 2 elemen pertama dari *array* data_markas.

Elemen *array* data_markas yang tersisa akan dievaluasi sebagai berikut. Selama fungsi `cross_product()` elemen teratas dari hull, elemen kedua teratas dari hull, dan elemen ke-j data_markas memberikan hasil ≤ 0, singkirkan elemen teratas hull. Jika kondisi tersebut tidak terjadi, periksa jarak antara elemen teratas hull dan elemen ke-j data_markas. Jika jaraknya ≤ 2500, tambahkan elemen tersebut ke hull. Jika jaraknya > 2500, tambahkan elemen tersebut ke luar. Hal ini dilakukan terus-menerus sampai elemen terakhir *array* data_markas.

```
// Menampilkan hasil kepada pengguna.
    if(hull->next != NULL){
        printf("\nPerbatasan markas Efektif: \n");
        printf("%s -> ", markas_awal.nama);
        while(hull != NULL){
            if(hull->next == NULL){
                printf("%s\n", hull->markas.nama);
            } else{
                printf("%s -> ",hull->markas.nama);
            }
            Markas temp = hull->markas;
            hull = hull->next;
            if(hull != NULL){
                jarak = jarak +
                haversine(hull->markas.posisi.x,
                hull->markas.posisi.y, temp.posi
                si.x, temp.posisi.y);
            }
        }
    }

    if(jarak == 0){
        printf("Perbatasan efektif tidak
                bisa dibuat\n");
        return 0;
    } else{
        printf("\nPanjang Perbatasan Efektif Negara
                Api: %f km\n\n", jarak);
    }

    printf("Markas di Luar Perbatasan Efektif: ");
    int k = 1;
    if(luar != NULL){
        printf("\n");
        while(luar != NULL){
            printf("%d . %s\n", k ,luar->markas.na
                    ma);
            k++;
            luar = luar->next;
        }
    } else {
        printf("Tidak ada.");
    }
    return 0;
}
```

Proses terakhir adalah proses menampilkan hasil. Program akan memeriksa apakah hull kosong atau tidak. Jika tidak kosong, program akan menampilkan elemen teratas dari hull. Pada saat yang bersamaan akan dihitung juga jarak antarmarkas. Hal ini akan dilakukan sampai

elemen terakhir dari hull. Proses dilanjutkan dengan memeriksa apakah terdapat nilai jarak. Jika ternyata jarak bernilai 0, program akan menampilkan pesan eror dan program selesai. Jika jarak tidak bernilai 0, program akan menampilkan nilai jarak tersebut. Terakhir, akan diperiksa apakah *stack* luar kosong atau tidak. Jika tidak kosong, setiap elemennya akan ditampilkan. Jika kosong, program akan menampilkan pesan "tidak ada" kepada pengguna. Setelah proses menampilkan hasil ini selesai, program selesai.

4.3.3 Analisis Kompleksitas Waktu dan Ruang

Berikut adalah analisis ruang dan waktu dari program ini.

- Membaca dan memproses data input

Membaca *file* eksternal dan memindahkan isi datanya menjadi sebuah *array* memiliki kompleksitas waktu O(n) dengan n adalah jumlah baris pada *file* eksternal. Hal ini disebabkan *while loop* untuk membaca file dan penggunaan strtok dan atof dieksekusi untuk setiap baris. Kompleksitas ruang dari operasi ini juga bernilai O(n) karena data disimpan dalam sebuah array yang berjumlah n elemen.

- Mengurutkan array Markas

Fungsi `sort_cc()` memiliki kompleksitas waktu $O(n^2)$ karena fungsi ini menggunakan *nesting* untuk membandingkan setiap elemen dengan semua elemen lainnya. Kompleksitas ruang dari operasi ini adalah O(1) karena hanya menggunakan memori tambahan untuk menukar elemen.

- Algoritma convex hull

Proses pembuatan *convex hull* memiliki kompleksitas waktu $O(n^2)$ karena terdapat *nesting loop* yang mengiterasi semua elemen. Fungsi `push()` yang berfungsi untuk menambahkan elemen ke dalam *stack* memiliki kompleksitas waktu O(1). Kompleksitas ruang dari operasi ini adalah O(n) dengan n adalah banyaknya elemen dari hull.

- Rumus Haversine

Fungsi haversine memiliki kompleksitas waktu O(1) karena hanya melakukan sejumlah operasi aritmatika yang konstan. Kompleksitas ruang dari operasi ini juga O(1) karena hanya menggunakan jumlah memori tetap.

Secara keseluruhan, kompleksitas waktu dari program ini adalah O(n^2) dan kompleksitas ruangnya adalah O(n) karena penyimpanan data dalam *array* dan *stack*.

4.4 PENGUJIAN

4.4.1 *Testcase* 1

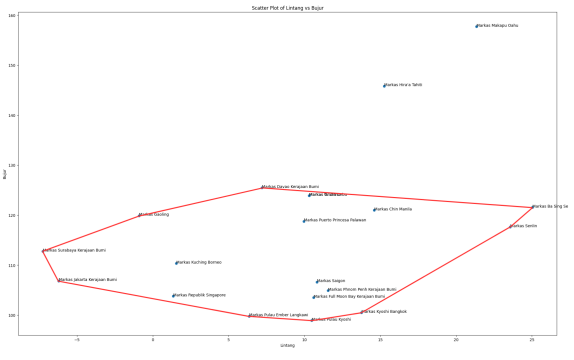
Kasus pertama adalah ketika input nama *file* dari *user* sudah benar dan data yang ada di dalam file ideal.

	A	B	C
1	Markas	Lintang	Bujur
2	Markas Ba Sing Se	25.033	121.5654
3	Markas Omashu	10.3157	123.8854
4	Markas Pulau Kyoshi	10.4744	98.9305
5	Markas Gaoling	-0.9116	119.9004
6	Markas Senlin	23.5565	117.6227
7	Markas Makapu Oahu	21.3394	157.7147
8	Markas Hira'a Tahiti	15.2549	145.815
9	Markas Chin Manila	14.5995	120.9842
10	Markas Tu Zin Cebu	10.3157	123.8854
11	Markas Saigon	10.8231	106.6297
12	Markas Kyoshi Bangkok	13.7563	100.5018
13	Markas Republik Singapore	1.3521	103.8198
14	Markas Jakarta Kerajaan Bumi	-6.2088	106.8456
15	Markas Surabaya Kerajaan Bumi	-7.2575	112.7521
16	Markas Davao Kerajaan Bumi	7.1907	125.4553
17	Markas Puerto Princesa Palawan	9.9672	118.7859
18	Markas Kuching Borneo	1.5497	110.3631
19	Markas Phnom Penh Kerajaan Bumi	11.5564	104.9282
20	Markas Full Moon Bay Kerajaan Bumi	10.6093	103.5297
21	Markas Pulau Ember Langkawi	6.35	99.8

Gambar 4-13 Data Markas Negara Api pada *File* markas.csv

```
Masukkan nama file: markas.csv

Perbatasan Efektif Negara Api:
Markas Pulau Kyoshi -> Markas Pulau Ember Langkawi -> Markas Jakarta Kerajaan Bumi
 -> Markas Surabaya Kerajaan Bumi -> Markas Gaoling -> Markas Davao Kerajaan Bumi
-> Markas Ba Sing Se -> Markas Senlin -> Markas Kyoshi Bangkok -> Markas Pulau Kyo
shi

Panjang Perbatasan Efektif Negara Api: 9381.316581 km

Markas di Luar Perbatasan Efektif:
1 . Markas Hira'a Tahiti
2 . Markas Makapu Oahu
```

Gambar 4-14 Output pada Terminal untuk *Testcase* 1



Gambar 4-15 Visualisasi *Output Testcase* 1

Pada kasus ini, terlihat dari gambar 4-2 dan 4-3 bahwa perbatasan efektif berhasil dibuat dengan markas-markas yang menjadi batasnya adalah Markas Pulau Kyoshi, Markas Pulau Ember Langkawi, Markas Jakarta Kerajaan Bumi, Markas Surabaya Kerajaan Bumi, Markas Gaoling, Markas Davao Kerajaan Bumi, Markas Ba Sing Se, Markas Senlin, dan Markas Kyoshi Bangkok. Sementara itu, terdapat dua markas yang tidak masuk ke dalam perbatasan efektif yaitu Markas Hira'a Tahiti dan Markas Makapu Oahu.

4.4.2 *Testcase* 2

Kasus kedua adalah ketika *user* memasukkan nama *file* yang salah sehingga program tidak bisa menemukan file tersebut.

```
Masukkan nama file: markas.txt
File tidak dapat dibuka. Program Berakhir.

Masukkan nama file: marka.csv
File tidak dapat dibuka. Program Berakhir.
```

Gambar 4-16 Output pada Terminal untuk *Testcase* 2

4.4.3 *Testcase* 3

Kasus ketiga adalah ketika *user* sudah meng-*input* nama *file* yang benar, namun data pada *file* tidak valid karena markas berjumlah kurang dari tiga buah sehingga perbatasan efektif tidak bisa dibuat.

	A	B	C
1	Markas	Lintang	Bujur
2			
3			
4			

Gambar 4-17 Data Markas Negara Api pada *File* markas2.csv

```
Masukkan nama file: markas2.csv
Perbatasan efektif tidak bisa dibuat
```

Gambar 4-18 Output pada Terminal untuk *Testcase* 3-1

	A	B	C
1	Markas	Lintang	Bujur
2	Markas Ba Sing Se	25.033	121.5654
3	Markas Omashu	10.3157	123.8854
4			

Gambar 4-19 Data Markas Negara Api pada *File* markas3.csv

```
Masukkan nama file: markas3.csv
Perbatasan efektif tidak bisa dibuat
```

Gambar 4-20 Output pada Terminal untuk *Testcase* 3-2

4.4.4 *Testcase* 4

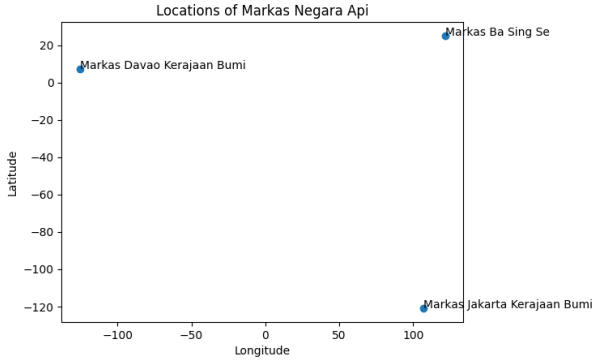
Kasus keempat adalah ketika nama *file* yang dimasukkan oleh *user* sudah benar dan data yang ada di dalam *file* tersebut valid. Namun, karena jarak antar markas yang terlalu jauh (>2500 km), perbatasan efektif negara api tidak bisa dibuat.

	A	B	C
1	Markas	Lintang	Bujur
2	Markas Ba Sing Se	25.033	121.5654
3	Markas Jakarta Kerajaan Bumi	-120.923	106.8456
4	Markas Davao Kerajaan Bumi	7.1907	-125.455

Gambar 4-20 Data Markas Negara Api pada *File* markas4.csv

```
Masukkan nama file: markas4.csv
Perbatasan efektif tidak bisa dibuat
```

Gambar 4-21 Output pada Terminal untuk *Testcase* 4



Gambar 4-22 Visualisasi data Markas Negara Api pada *File* markas4.csv

5. KESIMPULAN

Setelah melakukan mengerjakan tugas besar Praktikum Pemecahan Masalah dengan Bahasa C, didapat kesimpulan sebagai berikut.

- Convex hull algorithms dapat digunakan pada skema pembuatan perbatasan antar beberapa titik yang memiliki posisi lintang dan bujurnya masing-masing.
- Pembuatan convex hull dilakukan dengan mengaplikasikan stack berisi struct data identitas markas yang nantinya akan

diproses sehingga diperoleh urutan markas untuk dijadikan perbatasan.

- Proses pengolahan stack dimulai dengan pencarian titik koordinat paling bawah sehingga kemudian dapat dilakukan Graham Scan dengan syarat yang telah ditentukan. Selanjutnya akan dicari jarak antara dua titik menggunakan rumus haversine dengan syarat yang sama. Setelah itu barulah akan diperoleh convex hull dengan beberapa titik markas yang berada di luar daerah perbatasan efektif.
- Implementasi rumus dan metode tersebut dilakukan dalam bentuk fungsi dengan parameter-parameter yang sesuai dengan kebutuhannya.

Daftar Pustaka

[1] https://www.tutorialspoint.com/cprogramming/c_structures.htm, 26 April 2023, pukul 0.50 WIB.

[2] Dr. Reza Darmakusuma, S.T, M.T., Dismas Widyanto, Elkhan Julian Brillianshah, dan Irfan Tito Kurniawan, Modul Praktikum EL2208 Praktikum Pemecahan Masalah dengan C, ITB, Bandung, 2022.

[3] https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm, 26 April 2023, pukul 1.04 WIB.

[4] https://medium.com/@pascal.sommer.ch/a-gentle-introduction-to-the-convex-hull-problem-62dfcabee90c, 26 April 2023, pukul 1.38 WIB.

[5] https://en.wikipedia.org/wiki/Convex_hull_algorithms, 26 April 2023, pukul 1.39 WIB.

[6] https://www.tutorialspoint.com/Graham-Scan-Algorithm, 26 April 2023, pukul 1.13 WIB.

[7] https://iq.opengenus.org/graham-scan-convex-hull/, 26 April 2023, pukul 1.29 WIB.

[8] https://www.tutorialspoint.com/data_structures_algorithms/selection_sort_algorithm, 26 April 2023, pukul 1.47 WIB.

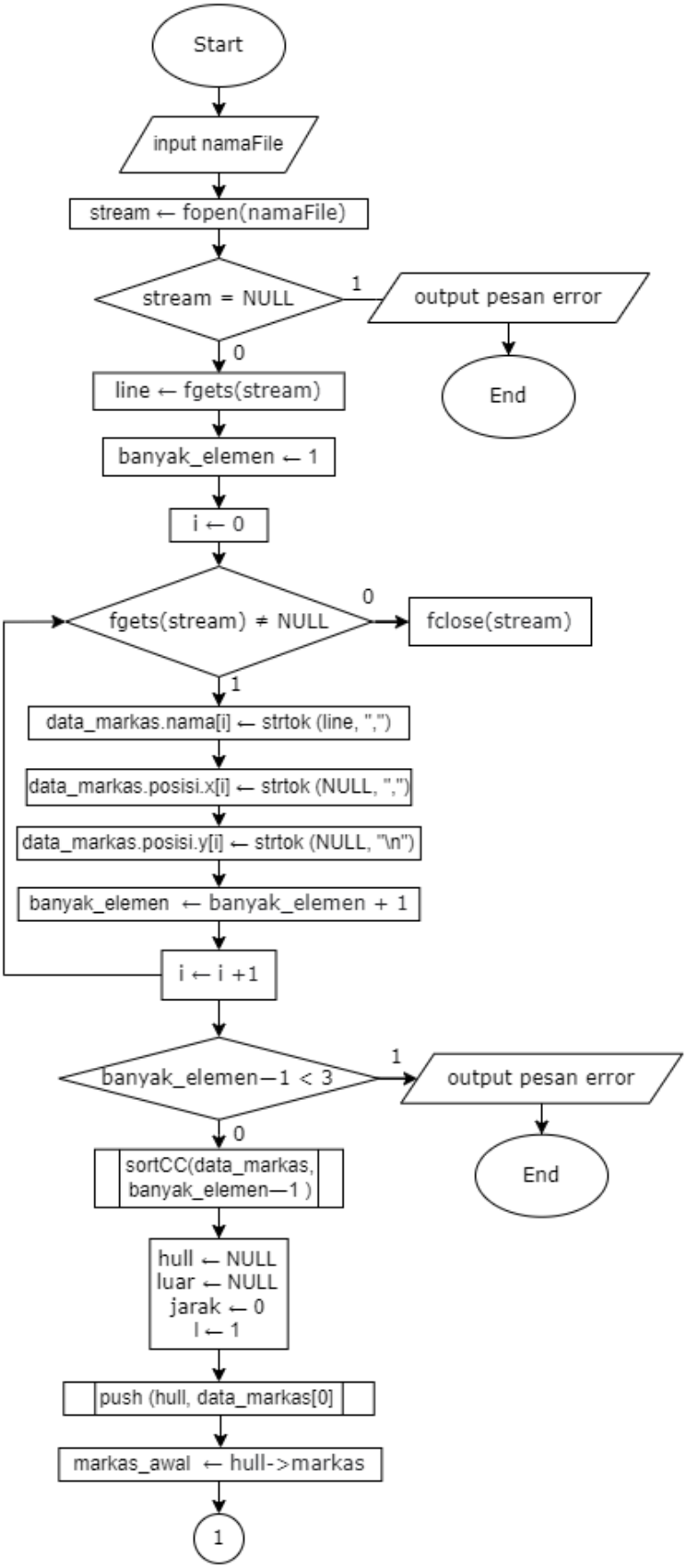
[9] https://stackoverflow.com/tags/selection-sort/info, 26 April 2023, pukul 2.02 WIB.

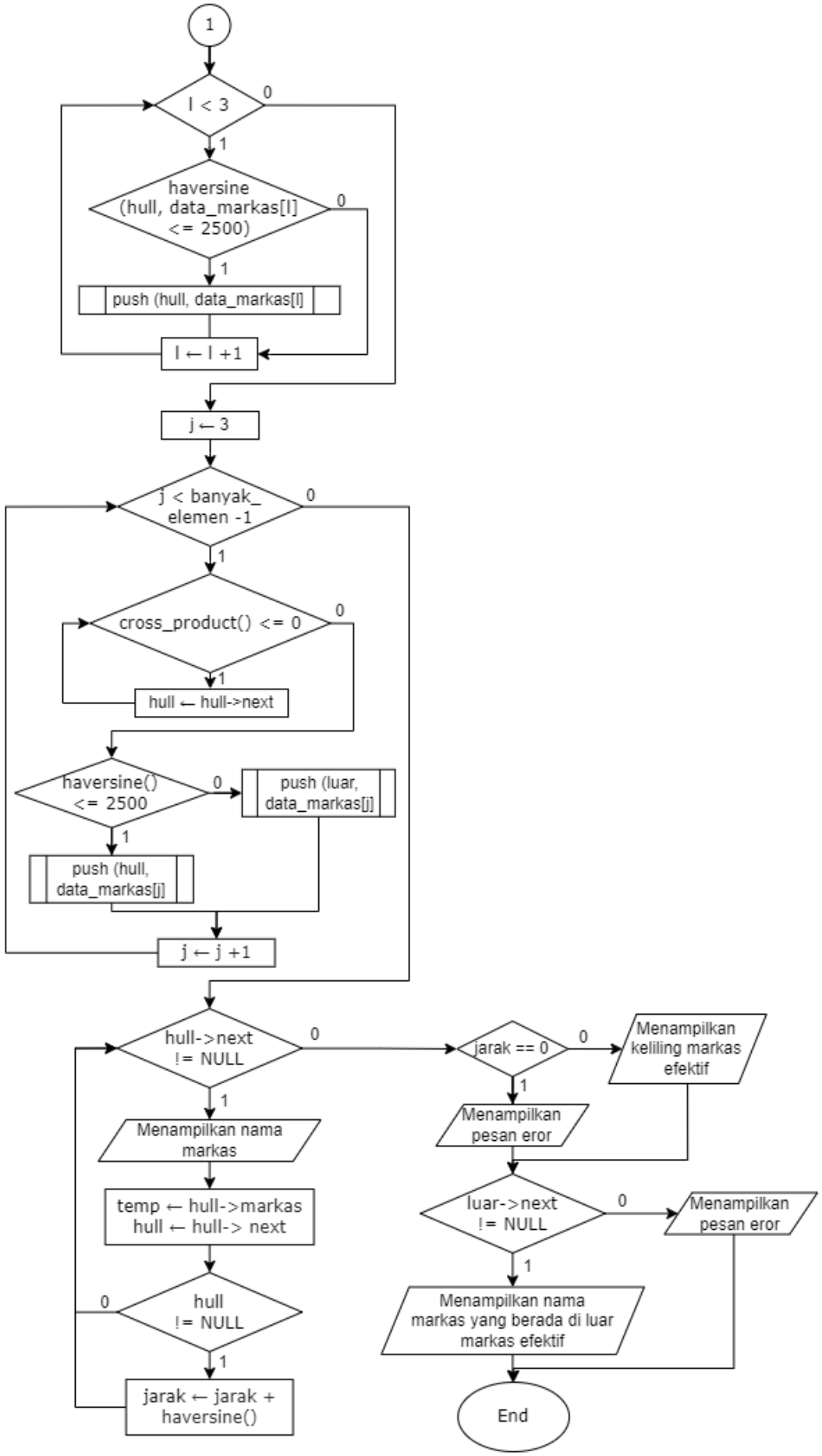
[10] https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128#:~:text=For%20example%2C%20haversine(%CE%B8),longitude%20of%20the%20two%20points, 26 April 2023, pukul 2.11 WIB.

Lampiran

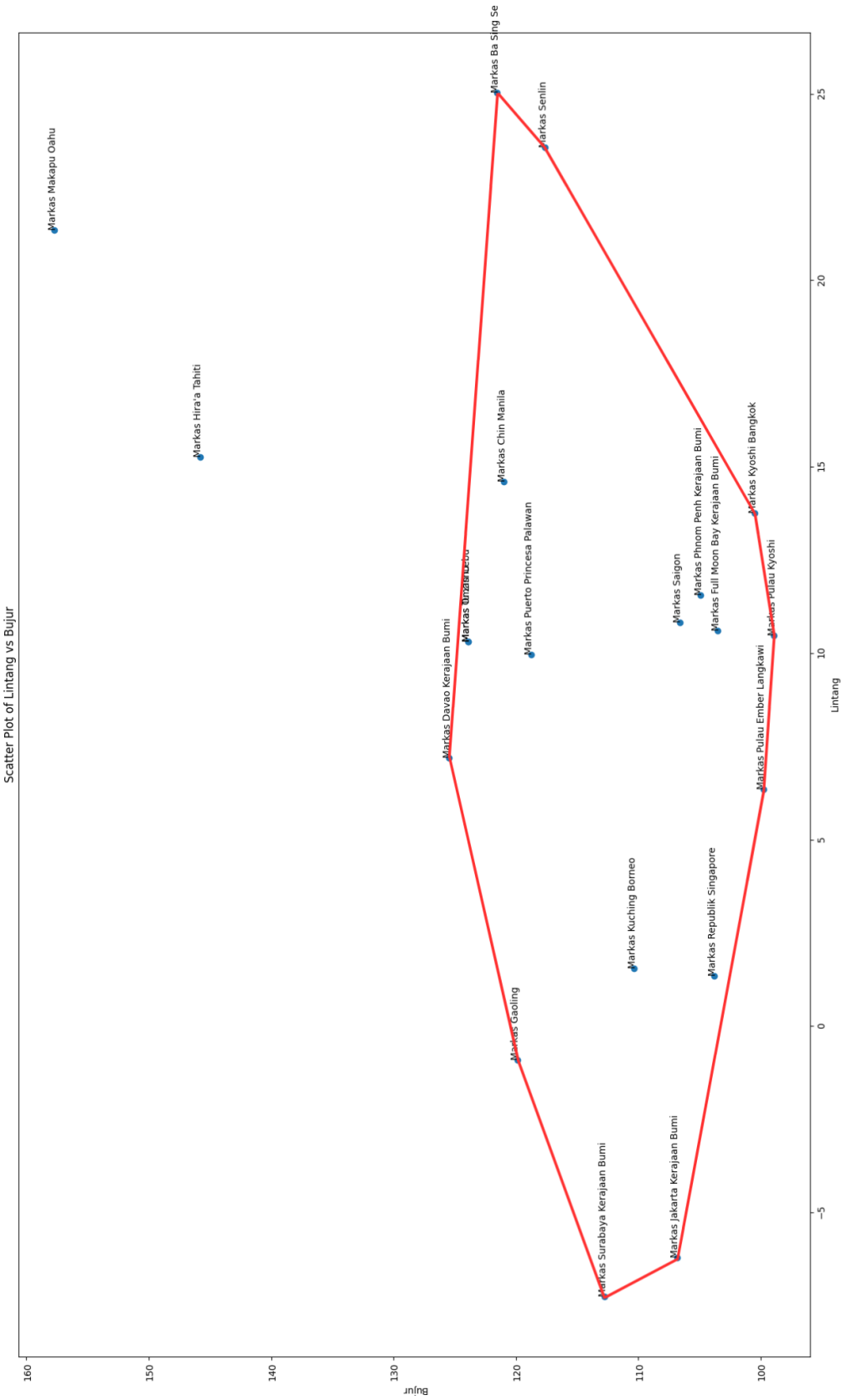
Repository github :
https://github.com/el2208-ppmc-2023/modul-9-team-b2

Flowchart fungsi main()




1
l < 3
0
1
haversine (hull, data_markas[l] <= 2500)
0
1
push (hull, data_markas[l]
l ← l +1
j ← 3
j < banyak_ elemen -1
0
1
cross_product() <= 0
0
1
hull ← hull->next
haversine() <= 2500
0
push (luar, data_markas[j]
1
push (hull, data_markas[j]
j ← j +1
hull->next != NULL
0
1
Menampilkan nama markas
temp ← hull->markas
hull ← hull-> next
0
hull != NULL
1
jarak ← jarak + haversine()
jarak == 0
0
Menampilkan keliling markas efektif
1
Menampilkan pesan eror
luar->next != NULL
0
Menampilkan pesan eror
1
Menampilkan nama markas yang berada di luar markas efektif
End

Visualisasi *Output Testcase* 1



Dokumentasi asistensi 24 April 2023