

MODUL 9 TUGAS BESAR

Taufiq Hidayat (13221044)
Arif Firman Fadhilah (13221045)
Muhamad Fikri (13221046)
Ganesha Indrayana Kusuma (13221047)
Asisten: Muhammad Morteza Mudrick (13219061)

Kelompok: C2

EL2208-Praktikum Pemecahan Masalah dengan C

Laboratorium Dasar Teknik Elektro - Sekolah Teknik Elektro dan Informatika ITB

Abstrak

Modul 9 Tugas Besar ini meminta praktikan bekerja dalam kelompok dan menyelesaikan masalah pemrograman sesuai dengan kasus yang didapat per kelompok, kelompok kami mendapat soal 2 yaitu Convex Hull Algorithm. Kami diminta untuk membuat program mencari perbatasan markas efektif negara api yang ditarik dari satu markas ke markas lainnya. Pengerjaan tugas ini dimulai dengan mengurai lingkup masalah yang terdapat pada persoalan, kemudian mengidentifikasi fungsi-fungsi atau fitur yang diperlukan pada program. Untuk dapat menyelesaikan persoalan ini kami memilih metode Graham Scan dengan perhitungan jarak dengan pendekatan Haversine. Walau pengerjaan dilaksanakan di sela-sela kesibukan lain namun untungnya pekerjaan bisa diselesaikan.

Kata kunci: Convex Hull, Graham Scan, Haversine.

1. PENDAHULUAN

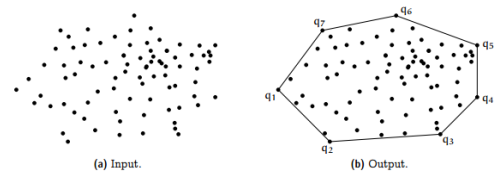
Pada tugas ini diberikan daftar markas militer dalam bentuk file .csv, kemudian praktikan diminta untuk menentukan perbatasan efektif dari markas-markas tersebut seluas mungkin namun masih dalam jarak efektif <2500 km. Kemudian akan dihitung jarak total perbatasan efektif negara api dan juga akan diberi keluaran markas-markas yang berada di luar perbatasan efektif.

Dalam prosesnya praktikan diminta untuk melakukan asistensi dengan asisten untuk memberitahu progress tugas dan menanyakan terkait asumsi-asumsi dan seputar lingkup masalah yang nantinya akan digunakan pada program.

Program harus dapat *dicompile* dengan perintah `make main` dan menghasilkan .exe file. Setelah semua selesai source code dikumpulkan di repository Git masing-masing kelompok.

2. STUDI PUSTAKA

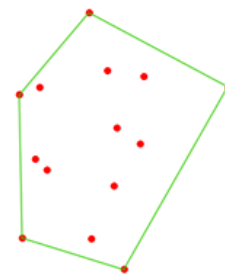
2.1 CONVEX HULL ALGORITHM



Gambar 2.1.0 Ilustrasi Convex Hull Algorithm

Convex Hull adalah struktur yang ada di mana-mana dalam geometri komputasi. Convex hull memiliki banyak aplikasi dalam ilmu data science seperti image processing.[1]

2.2 GRAHAM SCAN

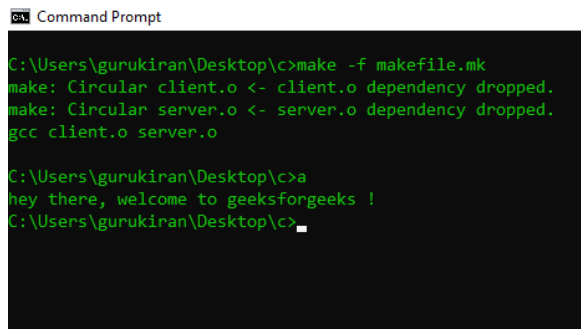


Gambar 2.2.0 Graham Scan

Graham Scan adalah salah satu metode untuk menghitung convex hull dari sekumpulan titik tertentu pada bidang. Untuk setiap titik, pertama-

tama ditentukan apakah scanning dari titik-titik ini belok ke kiri atau belok kanan.[1]

2.3 MAKEFILE



```
C:\Users\gurukiran\Desktop>make -f makefile.mk
make: Circular client.o <- client.o dependency dropped.
make: Circular server.o <- server.o dependency dropped.
gcc client.o server.o

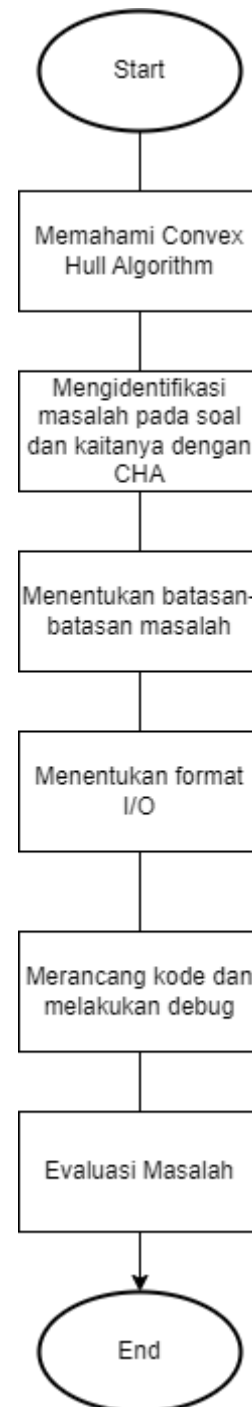
C:\Users\gurukiran\Desktop>a
hey there, welcome to geeksforgeeks !
C:\Users\gurukiran\Desktop>
```

Gambar 2.3 Makefile

Makefile adalah sekumpulan perintah (mirip dengan perintah terminal) dengan nama variabel dan target untuk membuat file objek dan menghapusnya. Dalam satu file make kita dapat

membuat beberapa target untuk dikompilasi dan untuk menghapus objek, file biner.[2]

3. METODOLOGI



Gambar 3.0 Flowchart Langkah Pengerjaan Tugas

Langkah satu sampai tiga merepresentasikan pendefinisian ruang lingkup masalah yang dihadapi, di mana kami harus mengetahui apa yang kami kerjakan, bagaimana cara kerjanya, apa saja yang dicari, dan menentukan apa saja yang harus dan tidak boleh dilakukan.

Kemudian langkah empat sampai enam mewakili perancangan *software*, implementasi rancangan, dan pengujian. Langkah-langkah tersebut adalah proses eksekusi dari penguraian lingkup masalah pada langkah sebelumnya yang diimplementasikan sesuai pemahaman kami dengan sebuah rancangan *software* yang diuji dan dilakukan debug sampai semua masalah teratasi dengan melakukan evaluasi.

4. HASIL DAN ANALISIS

4.1 RUANG LINGKUP MASALAH

Masalah yang akan diselesaikan berupa penentuan kombinasi titik terluar membentuk bangun datar tertutup dengan panjang sisi tidak melebihi 2500 km dari daftar koordinat titik yang diberikan, serta mencetak panjang keliling bangun datar tersebut beserta titik-titik yang ada di dalam dan luar bangun datar dalam artian markas.

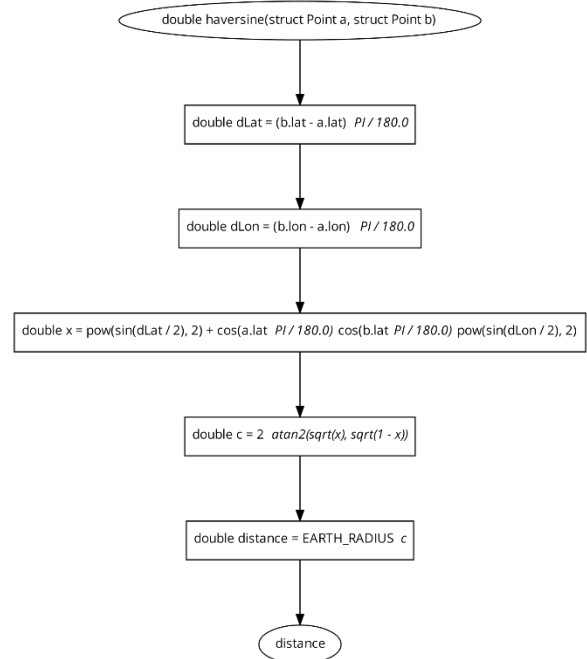
Asumsi yang digunakan dalam tugas ini di antaranya adalah nilai PI (π) sebesar 3.14159265358979323846, maksimal titik (markas) yang diproses sebanyak 100 titik, dan radius bumi yang bernilai 6371. Angka-angka ini sesuai dengan data aktual pada dunia nyata dengan galat yang insignifikan.

Program kami memiliki fitur sebagai berikut:

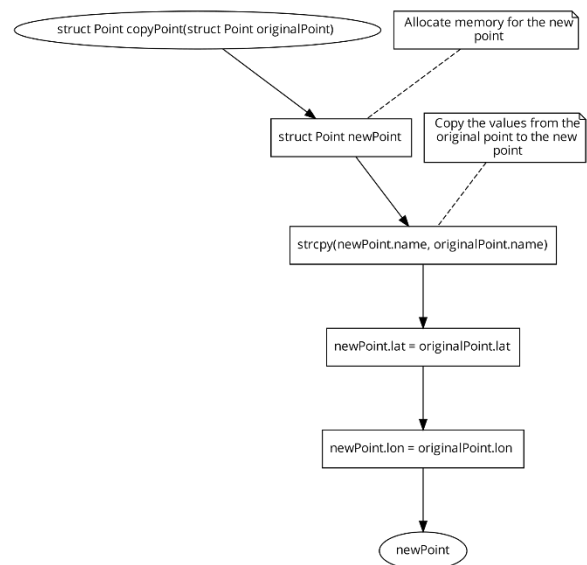
- Baca File CSV
- Menghitung jarak titik dengan Haversine formula
- Menentukan titik dengan koordinat terendah
- Menghitung posisi tiga titik secara clock/counter wise
- Memanipulasi stack dengan fungsi push dan pop.
- Menampilkan hasil convex hull yang terbentuk dan menghitung keliling dari convex hull.

4.2 RANCANGAN

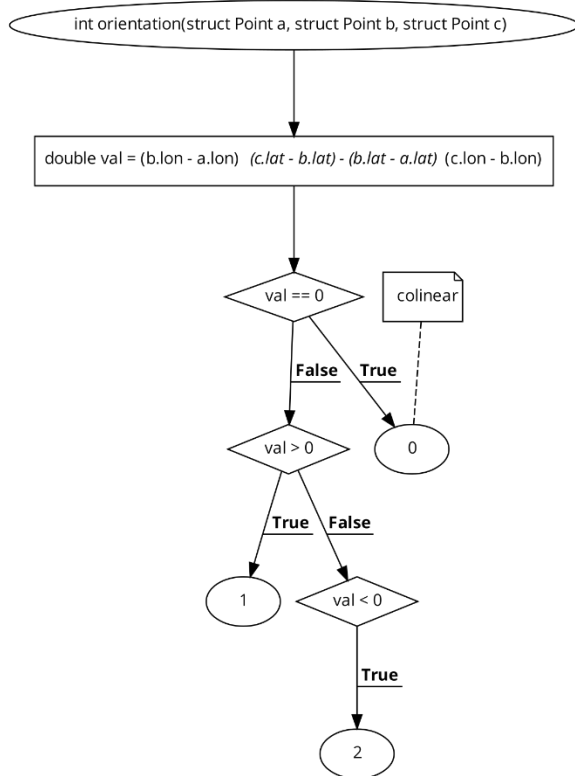
Flowchart point.c



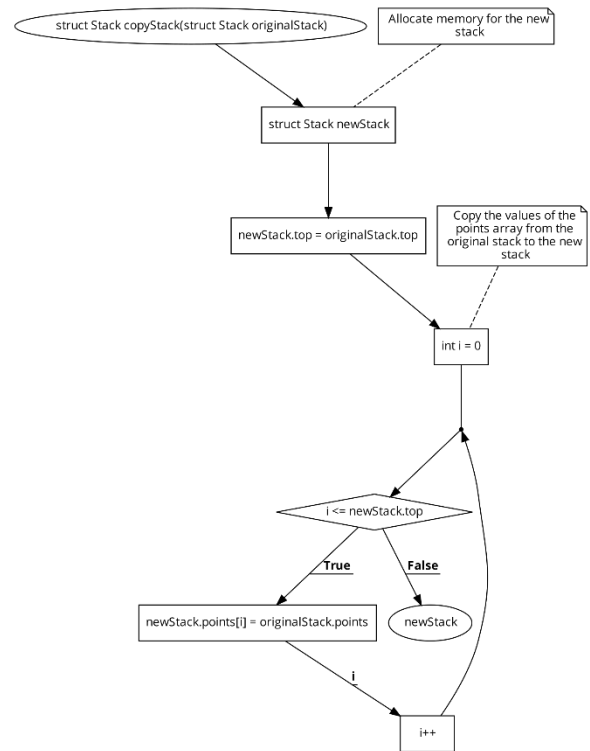
Gambar 4.1.0 Flowchart Function Haversine



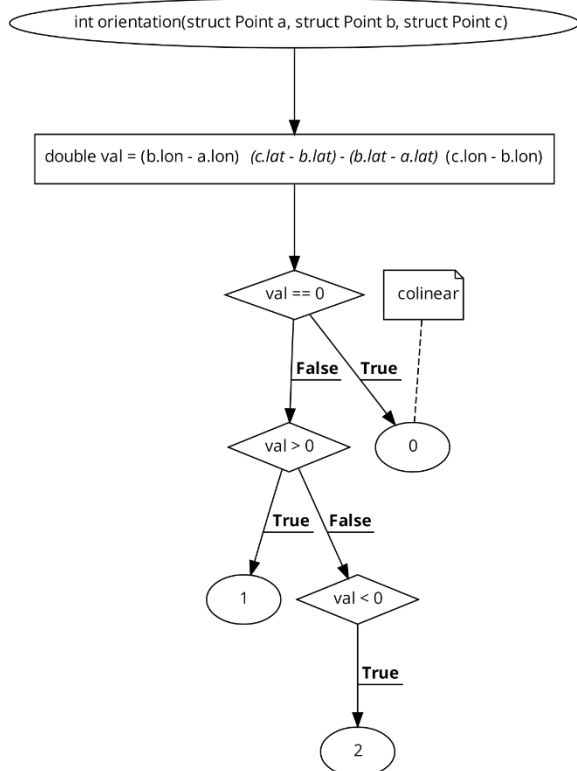
Gambar 4.1.1 Flowchart Function copyPoint



Gambar 4.1.2 Flowchart Function Orientation

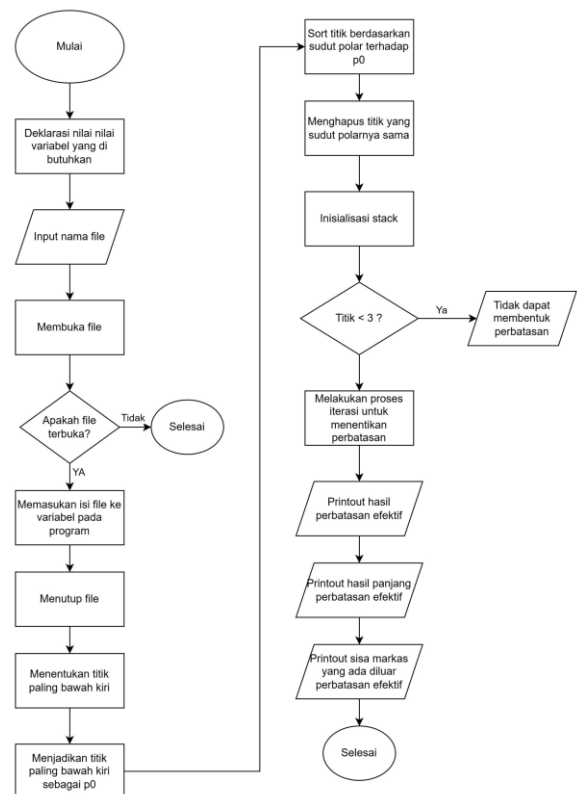


Gambar 4.1.4 Flowchart Function copyStack



Gambar 4.1.3 Flowchart Function Swap

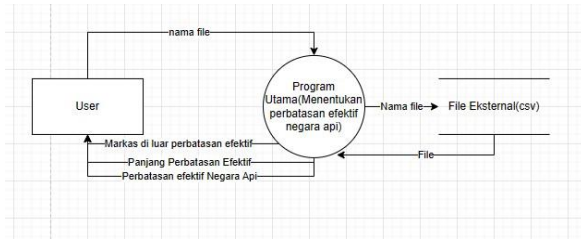
Flowchart main.c



Gambar 4.1.5 Flowchart main.c

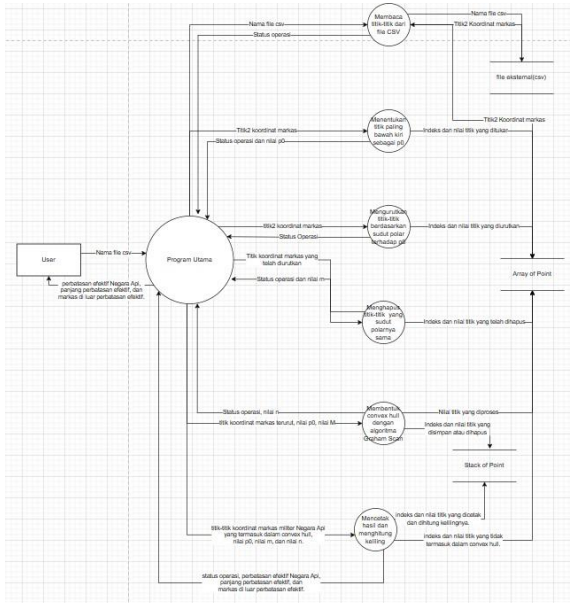
Flowchart Stack.c

DFD Level 0



Gambar 4.1.6 DFD level 0

DFD Level 1



Gambar 4.1.7 DFD level 1

Task	Pembagian Tugas
Pembacaan file (point.c, point.h),(stack.c, stack.h)	Designer: - 13221044 Implementer: - 13221044 - 13221047 Tester: - 13221047
Integrasi (main.c)	Implementer: - 13221044 - 13221045 - 13221046 - 13221047 Tester: - 13221047

Laporan	- 13221044 - 13221045 - 13221046 - 13221047
---------	--

4.3 IMPLEMENTASI

Bagian ini berisikan penjelasan kontribusi setiap anggota kelompok serta penjelasan dari kode yang Anda buat (struktur *source code*, penjelasan pemetaan kode ke rancangan, penjelasan fungsi, dll.). Jika ada perbedaan antara rancangan dan implementasi rancangan atau perbedaan antara pembagian tugas dan kontribusi anggota, jelaskan alasannya di bagian ini.

Point.c

```
#include "point.h"

#define BUFFER_SIZE 100
#define PI 3.14159265358979323846
#define EARTH_RADIUS 6371 // dalam Kilometer

struct Point copyPoint(struct Point originalPoint) {
    // Alokasi memori untuk titik baru
    struct Point newPoint;

    // Mengcopy nilai dari titik awal ke titik baru
    strcpy(newPoint.name, originalPoint.name);
    newPoint.lat = originalPoint.lat;
    newPoint.lon = originalPoint.lon;

    return newPoint;
}

double haversine(struct Point a, struct Point b) {
    double dLat = (b.lat - a.lat) * PI / 180.0;
    double dLon = (b.lon - a.lon) * PI / 180.0;
```

```

        double x = pow(sin(dLat / 2),
2) + cos(a.lat * PI / 180.0) *
cos(b.lat * PI / 180.0) *
pow(sin(dLon / 2), 2);
        double c = 2 * atan2(sqrt(x),
sqrt(1 - x));
        double distance = EARTH_RADIUS
* c;
        return distance;
}

// Deklarasi fungsi orientation
yang digunakan untuk menentukan
arah putaran tiga titik pada bidang
kartesius.
int orientation(struct Point a,
struct Point b, struct Point c) {
    double val = (b.lon - a.lon) *
(c.lat - b.lat) - (b.lat - a.lat) *
(c.lon - b.lon);
    if (val == 0) {
        return 0; // colinear
    }
    if (val > 0) {
        return 1;
    }
    if (val < 0) {
        return 2;
    }
}

void swap(struct Point *p1, struct
Point *p2) {
    struct Point temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

```

point.c adalah file header yang berisi fungsi-fungsi yang digunakan untuk mengolah titik-titik geografis pada file markas.csv.

Pertama-tama, terdapat function copyPoint yang digunakan untuk mengalokasikan memori untuk titik baru dan mengcopy nilai dari titik asli ke titik baru.

Kemudian, terdapat fungsi haversine yang menghitung jarak antara dua titik dalam satuan

kilometer dengan menggunakan formula haversine.

Fungsi ini digunakan untuk menghitung jarak antara dua titik pada bola bumi menggunakan rumus Haversine. Fungsi ini menerima dua parameter berupa titik-titik yang akan dihitung jaraknya dan mengembalikan hasil perhitungan jarak dalam satuan kilometer.

Pertama, fungsi ini menghitung perbedaan latitude dan longitude antara dua titik dan mengkonversinya ke dalam radian menggunakan konstanta π (PI). Selanjutnya, fungsi menghitung nilai x menggunakan formula Haversine. Nilai c dihitung dengan menggunakan nilai x dan kemudian dihitung jarak dengan mengalikan jari-jari bumi (yang didefinisikan dengan konstanta EARTH_RADIUS) dengan nilai c.

Formula Haversine ini dapat digunakan untuk menghitung jarak antara dua titik pada bola bumi karena bola bumi berbentuk seperti bola, bukan seperti dataran. Oleh karena itu, jarak antara dua titik tidak dapat dihitung dengan menganggap bola bumi sebagai dataran.

Selanjutnya, terdapat fungsi orientation yang digunakan untuk menentukan arah putaran tiga titik pada bidang kartesius. Secara lebih teknis, fungsi ini menggunakan tiga titik sebagai input, yaitu titik a, b, dan c yang masing-masing memiliki nilai latitude (lat) dan longitude (lon) dalam format desimal.

Cara kerja fungsi ini adalah dengan menghitung nilai determinan matriks yang dibentuk oleh koordinat tiga titik tersebut. Jika nilai determinan positif, maka arah putaran adalah searah jarum jam, dan jika negatif maka arah putarannya berlawanan arah jarum jam. Jika nilainya adalah 0, maka titik-titik tersebut sejajar (colinear).

Hasil return fungsi orientation berupa bilangan bulat, yang bernilai 0 jika titik-titik tersebut sejajar, 1 jika arah putarannya searah jarum jam, dan 2 jika arah putarannya berlawanan arah jarum jam.

Terakhir, terdapat fungsi swap yang digunakan untuk menukar nilai dua titik yang diwakili oleh pointer ke structure titik.

Di dalam file header ini juga didefinisikan beberapa konstanta seperti BUFFER_SIZE, PI, dan EARTH_RADIUS yang digunakan dalam fungsi-fungsi tersebut.

stack.c

```

#include "stack.h"

void push(struct Stack *stack,
struct Point point) {
    stack->points[++stack->top] =
point;
}

struct Point pop(struct Stack
*stack) {
    return stack->points[stack-
>top--];
}

struct Point nextToTop(struct Stack
*stack) {
    return stack->points[stack->top
- 1];
}

struct Point top(struct Stack
stack) {
    return stack.points[stack.top];
}

struct Stack copyStack(struct Stack
originalStack) {
    // Alokasi memori untuk stack
baru
    struct Stack newStack;
    newStack.top =
originalStack.top;

    // Mengcopy value dari array
points ke stack baru
    for (int i = 0; i <=
newStack.top; i++) {
        newStack.points[i] =
originalStack.points[i];
    }

    return newStack;
}

```

Struck Point merepresentasikan titik dengan koordinat (x,y) pada bidang kartesian. Struktur Stack yang digunakan untuk menyimpan titik-titik dalam algoritma Convex Hull juga

diimplementasikan dengan menggunakan array of struct Point.

Pada fungsi copyStack, dilakukan alokasi memori untuk stack baru dan dilakukan peng-copy-an elemen-elemen pada array of struct Point dari stack asli ke stack baru. Hal ini diperlukan karena ketika convex hull sudah ditemukan, maka stack yang berisi titik-titik sementara harus dikosongkan untuk mengembalikan memori yang digunakan pada stack tersebut.

Dalam implementasi algoritma Convex Hull dengan metode Graham Scan, stack ini digunakan sebagai wadah sementara untuk menyimpan titik-titik yang membentuk convex hull. Ketika algoritma berakhir, maka stack ini akan berisi titik-titik pada convex hull secara berurutan dari yang ter kiri ke kanan pada bidang kartesian.

Pada implementasi algoritma Convex Hull dengan metode Graham Scan, elemen yang ditambahkan adalah sebuah titik. Fungsi push akan menambahkan titik baru ke dalam stack, dan meng-update nilai variabel top pada stack untuk menunjukkan index dari elemen teratas pada stack.

Fungsi pop akan mengambil titik teratas pada stack, dan meng-update nilai variabel top pada stack untuk menunjukkan index dari elemen teratas pada stack setelah penghapusan.

Selanjutnya, fungsi nextToTop digunakan untuk mengambil titik yang berada di atas titik teratas pada stack. Fungsi ini sering digunakan pada algoritma Convex Hull dengan metode Graham Scan untuk menentukan orientasi dari 3 titik teratas pada stack.

Kemudian fungsi top digunakan untuk mengambil titik teratas pada stack tanpa menghapusnya. Fungsi ini digunakan pada algoritma Convex Hull dengan metode Graham Scan untuk mendapatkan informasi mengenai titik teratas pada stack, seperti posisinya pada bidang kartesian.

Terakhir adalah Fungsi copyStack yang digunakan untuk melakukan peng-copy-an stack asli ke dalam stack baru. Hal ini diperlukan pada algoritma Convex Hull dengan metode Graham Scan untuk menyimpan sementara titik-titik yang membentuk convex hull pada sebuah stack. Setelah convex hull selesai ditemukan, stack yang berisi titik-titik sementara ini perlu dikosongkan untuk mengembalikan memori yang digunakan pada stack tersebut. Fungsi copyStack akan melakukan alokasi memori untuk stack baru, meng-copy isi dari stack asli ke stack baru, dan mengembalikan stack baru yang telah di-copy dari stack asli.

main.c

```
/*EL2208 Praktikum Pemecahan
Masalah dengan C 2022/2023
*Modul:9-TugasBesar
*Kelompok:C2
*HaridanTanggal:27 April 2023
*Asisten(NIM):Muhammad Morteza
Mudrick (13219061)
*NamaFile:main.c
*Deskripsi:File main untuk
implementasi Convex Hull Algorithm
yang di dalamnya terdapat directive
file header stack dan point
*/

#include <stdio.h>
#include "lib/stack.h"
#include "lib/point.h"

// variabel global p0
struct Point p0;
// Deklarasi variabel p0 sebagai
titik paling bawah kiri yang akan
digunakan sebagai titik awal untuk
membentuk hull.

// Deklarasi fungsi compare yang
digunakan sebagai parameter fungsi
qsort untuk mengurutkan titik-titik
berdasarkan sudut yang dibentuk
dengan titik awal p0.
int compare(const void *p1, const
void *p2) {
    struct Point *a = (struct
Point *)p1;
    struct Point *b = (struct
Point *)p2;

    int o = orientation(p0, *a,
*b); //fungsi orientation digunakan
untuk menentukan arah putaran tiga
titik pada bidang kartesius.
    if (o == 0) {
        (haversine(p0, *a) >=
haversine(p0, *b)) ? -1 : 1;
    }

    return (o == 2) ? -1 : 1;
}
```

```
int main() {
    int n = 0;
    double perimeter = 0;
    struct Point points[100],
ineffective [100];
    char fname[100], path[106];

    // input filename
    printf("\nMasukkan file
markas: ");
    scanf("%s", fname);
    snprintf(path, sizeof(path),
"test/%s", fname);

    FILE *file = fopen(path,
"r");
    if (file == NULL) {
        printf("Tidak dapat
membuka file.\n");
        return 1;
    }

    // input titik dari file
    char line[1024];
    while (fgets(line,
sizeof(line), file)) {
        char name[50];
        double latitude,
longitude;
        if (sscanf(line,
"%[^,],%lf,%lf", name, &latitude,
&longitude) == 3) {
            n++;
            strcpy(points[n-
1].name, name);
            points[n-1].lat =
latitude;
            points[n-1].lon =
longitude;
        }
    }

    // close file
    fclose(file);
}
```



```

        // menentukan titik paling
bawah kiri
        int ymin = points[0].lat, min
= 0;
        for (int i = 1; i < n; i++)
        {
            int y = points[i].lat;
            if ((y < ymin) || (ymin
== y && points[i].lon <
points[min].lon)) {
                ymin = points[i].lat,
min = i;
            }
        }

        // meletakkan titik bawah
kiri sebagai p0
        swap(&points[0],
&points[min]);
        p0 = points[0];

        // sort titik berdasarkan
sudut polar terhadap p0
        qsort(&points[1], n-1,
sizeof(struct Point), compare);

        // menghapus titik yang sudut
polarnya sama
        int m = 1;
        for (int i=1; i<n; i++)
        {
            while (i < n-1 &&
orientation(points[0], points[i],
points[i+1]) == 0) {
                i++;
            }
            points[m] = points[i];
            m++;
        }
        n = m;

        // initialize stack
        struct Stack stack;
        stack.top = -1;

```

```

        // tidak dapat membentuk
convex hull jika titik < 3
        if (n < 3) {
            printf("Tidak dapat
membentuk perbatasan\n");
        }
        else {

            // melakukan loop hingga
titik dapat kembali ke titik awal
            while (1) {
                // push 3 titik
pertama
                push(&stack,
points[0]);
                push(&stack,
points[1]);
                push(&stack,
points[2]);

                m = 0;

                for (int i = 3; i <
n; i++) {
                    struct Stack temp
= copyStack(stack);

                    // lakukan pop
terhadap stack jika jarak 2 titik
baru > 2500 atau arah 2 titik baru
berlawanan arah jarum jam
                    while (stack.top
>= 1 &&
(orientation(nextToTop(&stack),
top(stack), points[i]) != 2 ||
haversine(top(stack), points[i]) >
2500)) {
                        struct Point
dump = pop(&stack);
                    }

                    // jika ukuran
stack kurang dari 1 maka titik
terakhir tidak efektif
                    if (stack.top <
1) {
                        stack =
copyStack(temp);

```

```

        m++;
        ineffective[m
-1] = copyPoint(points[i]);
        } else {
            push(&stack,
points[i]);
        }
    }

    // break while loop
jika convex hull terbentuk
    if (haversine(p0,
top(stack)) <= 2500) {
        push(&stack, p0);
        break;
    }
}

// print result dan
hitung keliling
printf("\nPerbatasan
Efektif Negara Api:\n");
for (int i = 0; i <=
stack.top; i++) {
    if (i < stack.top) {
        perimeter +=
haversine(stack.points[i],
stack.points[i+1]);
    }
    printf("%d. %s
(%.4f, %.4f)\n", i+1,
stack.points[i].name,
stack.points[i].lat,
stack.points[i].lon);
}

printf("\nPanjang
Perbatasan Efektif Negara Api: %.4f
km\n", perimeter);

printf("\nMarkas di Luar
Perbatasan Efektif: (%d)\n", m);
for (int i = 0; i < m;
i++) {

```

```

        printf("%s
(%.4f, %.4f)\n", ineffective[i].name,
ineffective[i].lat,
ineffective[i].lon);
    }
}

return 0;
}

```

Program membaca input data dari file yang berisi lintang dan bujur setiap titik. Program diawali dengan melakukan penentuan titik dengan garis lintang terendah dan garis bujur paling kiri, yang digunakan sebagai titik awal pembuatan convex hull.

Kemudian, program mengurutkan titik berdasarkan sudut kutub sehubungan dengan titik awal. Lalu program akan menghapus semua titik yang memiliki sudut kutub yang sama. Setelah itu, program menginisialisasi stack dan menggunakannya untuk meng-push tiga titik pertama.

Program kemudian memeriksa apakah titik berikutnya berada di luar convex hull saat ini dengan menghitung orientasi dan jarak antara titik saat ini dan dua titik teratas dalam stack. Jika titik berada di luar convex hull, program akan meng-pop titik teratas hingga titik berada di dalam convex hull. Jika ukuran stack kurang dari satu, berarti titik terakhir tidak efektif dan harus dihilangkan.

Terakhir, program menghitung keliling convex hull dengan melakukan iterasi melalui stack dan menjumlahkan jarak antara setiap pasang titik yang berdekatan.

Analisis Kompleksitas:

Program dimulai dengan beberapa operasi awal untuk mendeklarasikan variabel, membuka, membaca, dan menutup file input. Kompleksitas waktu dan ruang dari operasi ini tergantung pada ukuran file input, dengan kompleksitas $O(n)$ di mana n adalah jumlah baris pada file input. Setelah itu, program menentukan titik paling bawah kiri dari titik-titik input. Ini dilakukan dalam loop for dan kompleksitas waktu dari bagian ini adalah $O(n)$, karena hanya ada satu loop dan operasi di dalam loop memiliki kompleksitas konstan. Kemudian, program melakukan swap untuk menempatkan titik bawah kiri sebagai $p0$. Ini

hanya melibatkan pertukaran nilai pada array, sehingga kompleksitasnya adalah $O(1)$.

Selanjutnya, program melakukan pengurutan titik-titik berdasarkan sudut polar terhadap p_0 menggunakan fungsi `qsort`. Fungsi ini memiliki kompleksitas waktu rata-rata $O(n \log n)$ dan kompleksitas ruang $O(\log n)$ karena menggunakan algoritma quicksort. Setelah pengurutan, program menghapus titik yang sudut polarnya sama. Ini melibatkan satu loop `for` dan pengecekan kondisi dalam `while` loop, yang memiliki kompleksitas waktu $O(n)$.

Program selanjutnya melakukan inisialisasi stack dan menambahkan tiga titik awal ke dalamnya. Ini melibatkan tiga operasi `push`, yang memiliki kompleksitas waktu dan ruang $O(1)$. Kemudian, program melakukan loop untuk setiap titik input, menghitung jarak dan arah antara titik baru dan dua titik teratas pada stack, dan memutuskan apakah titik baru harus dimasukkan ke dalam stack atau tidak. Loop ini melibatkan satu loop `for` dan operasi `push`, `pop`, dan `copy` pada stack, yang semuanya memiliki kompleksitas waktu dan ruang $O(1)$.

Akhirnya, program mencetak hasilnya. Ini hanya melibatkan operasi cetak, yang memiliki kompleksitas waktu dan ruang $O(1)$. Secara keseluruhan, kompleksitas waktu dan ruang program ini bergantung pada ukuran file input, dengan kompleksitas waktu terbesar $O(n \log n)$ pada operasi pengurutan titik dan kompleksitas ruang terbesar $O(n)$ pada array titik. Namun, kompleksitas waktu dan ruang lainnya adalah konstan.

4.4 PENGUJIAN

Point.c (copyPoint,haversine,orientation,swap)	<input checked="" type="checkbox"/>
Stack.c (push,pop,nextToTop,top,copyStack(<input checked="" type="checkbox"/>
Main.c	<input checked="" type="checkbox"/>

Input output (read csv, output)	<input checked="" type="checkbox"/>
------------------------------------	-------------------------------------

Tabel 4.4 Checklist Keberhasilan File

Eksekusi File:

a) Dengan Command Prompt

```
C:\Wuliah\Semester 4\Lainya\PMC\TUBES\TUBES>make main
gcc main.c lib\point.c lib\stack.c -o main

C:\Wuliah\Semester 4\Lainya\PMC\TUBES\TUBES>./main
'.' is not recognized as an internal or external command,
operable program or batch file.

C:\Wuliah\Semester 4\Lainya\PMC\TUBES\TUBES>main
Masukkan file markas: markas.csv

Perbatasan Efektif Negara Api:
1. Markas Surabaya Kerajaan Bumi (-7.2575, 112.7521)
2. Markas Jakarta Kerajaan Bumi (-6.2888, 106.8456)
3. Markas Pulau Ember Langkawi (6.3500, 99.8000)
4. Markas Pulau Kyoshi (10.4744, 98.9305)
5. Markas Kyoshi Bangkok (13.7563, 100.5018)
6. Markas Senlin (23.5565, 117.6227)
7. Markas Ba Sing Se (25.0330, 121.5654)
8. Markas Davao Kerajaan Bumi (7.1907, 125.4553)
9. Markas Gaoling (-0.9116, 119.9004)
10. Markas Surabaya Kerajaan Bumi (-7.2575, 112.7521)

Panjang Perbatasan Efektif Negara Api: 9849.7916 km

Markas di Luar Perbatasan Efektif: (2)
Markas Hira'a Tahiti (15.2549, 145.8150)
Markas Makapu Oahu (21.3394, 157.7147)
```

Gambar 4.4.1 Eksekusi dengan CMD

b) Dengan Visual Studio Code

```
PS C:\kulia\Semester 4\Lainya\PMC\TUBES\TUBES> make main
gcc main.c lib\point.c lib\stack.c -o main
PS C:\kulia\Semester 4\Lainya\PMC\TUBES\TUBES> ./main
Masukkan file markas: markas.csv

Perbatasan Efektif Negara Api:
1. Markas Surabaya Kerajaan Bumi (-7.2575, 112.7521)

2. Markas Jakarta Kerajaan Bumi (-6.2888, 106.8456)
3. Markas Pulau Ember Langkawi (6.3500, 99.8000)
4. Markas Pulau Kyoshi (10.4744, 98.9305)
5. Markas Kyoshi Bangkok (13.7563, 100.5018)
6. Markas Senlin (23.5565, 117.6227)
7. Markas Ba Sing Se (25.0330, 121.5654)
8. Markas Davao Kerajaan Bumi (7.1907, 125.4553)
9. Markas Gaoling (-0.9116, 119.9004)

10. Markas Surabaya Kerajaan Bumi (-7.2575, 112.7521)

Panjang Perbatasan Efektif Negara Api: 9849.7916 km

Markas di Luar Perbatasan Efektif: (2)
Markas Hira'a Tahiti (15.2549, 145.8150)
Markas Makapu Oahu (21.3394, 157.7147)
```

Gambar 4.4.2 Eksekusi dengan VSCode

5. KESIMPULAN

- Telah berhasil dibuat program implementasi Convex Hull Algorithm dengan metode Graham Scan.
- Tugas ini utamanya banyak menggunakan konsep linkedlist, seperti stack.
- Teknik Graham Scan performanya lebih baik dari Jarvis March karena

kompleksitasnya lebih rendah sehingga relatif lebih cepat.

DAFTAR PUSTAKA

- [1] <https://towardsdatascience.com/convex-hull-an-innovative-approach-to-gift-wrap-your-data-899992881efc> 26/04/2023,7:15
- [2] <https://www.geeksforgeeks.org/convex-hull-using-graham-scan/?ref=gcse> 26/04/2023, 7:27

LAMPIRAN

link ke repository: <https://github.com/el2208-ppmc-2023/modul-9-team-c2>



-Taufiq Hidayat (Izin ke kamar mandi)