Spesifikasi Tugas Besar 2
IF2111 Algoritma dan Struktur Data STI

PURRMART

Revisi

ver. 2 Desember 2024
Revisi 2/12/2024 - Penambahan *constraint* pada bonus OPTIMASIRUTE
Revisi 20/12/2024 - Perbaikan *output* contoh pada bonus OPTIMASIRUTE
Revisi 20/12/2024 - Penjelasan lebih lanjut pada flow CART PAY

Deadline

Milestone 2: ~~**20 Desember 2024 21.11 WIB**~~ **22 Desember 2024 21.11 WIB**

Daftar Isi

Latar Belakang

Agen Purry sedang menikmati tidur siangnya ketika dia tiba-tiba mendengar *alarm* dari belakang sofa. Suatu pintu rahasia terbuka di bawah dirinya dan ia jatuh ke ruang bawah tanah dan langsung disambut dengan misi terbarunya.

> “Ah Agen Purry, maaf harus mengganggu waktu tidur kamu tapi kami mendapatkan laporan bahwa Dr. Asep Spakbor sedang membuat suatu mesin yang dinamakan ‘Oppenheimer-inator’ yang akan menghancurkan wilayah tiga negara bagian. Aku membutuhkan bantuanmu untuk menghentikan Dr. Asep Spakbor, ini merupakan ancaman terbesar yang pernah ia buat.”

And indeed it was. Setelah pertarungan sengit selama 3 bulan 13 hari 2 jam 47 menit dan 2 detik, suplai senjata dan suplai peralatan yang dimiliki OWCA mulai menipis. Harapan kemenangan OWCA mulai memudar...

Tanpa disangka, Agen Purry mengeluarkan senjata rahasia miliknya: menjadi orang Bojongsoang yang memiliki kenalan pegawai Borma. Toko Borma adalah komponen penting yang dapat membawakan kemenangan untuk OWCA pada waktu-waktu kritis ini. Sebab, meskipun Borma terlihat seperti *supermarket* pada umumnya, mereka sebenarnya merupakan pemasok barang-barang perang. Namun terdapat satu masalah kecil, Borma masih beroperasi secara tatap muka dan OWCA tidak memiliki *transport* untuk pergi ke Bojongsoang.

(Reaksi Jujur Purry)

Untuk menyelesaikan permasalahan ini, OWCA mengontak tim *programmer* paling andalnya untuk merancang suatu sistem jual beli ke Borma dengan nama **PURRMART**! **Benar, tim tersebut adalah kalian!** Misi ini akan menantang dan menguji kalian. Namun, dengan kerja tim dan tekad yang kuat, kalian pasti dapat menghadapi tantangan ini.

Good luck, programmers! OWCA counts on you! **(•̀ω •́)✧**

Spesifikasi Umum

Buatlah sebuah aplikasi simulasi berbasis CLI (*command-line interface*). Sistem ini dibuat dalam **bahasa C** dengan menggunakan **struktur data yang sudah kalian pelajari** di mata kuliah ini. Kalian boleh menggunakan (atau memodifikasi) struktur data yang sudah kalian buat untuk praktikum pada tugas besar ini. Daftar ADT yang wajib digunakan dapat dilihat pada bagian Daftar ADT. *Library* yang boleh digunakan hanya **stdio.h**, **stdlib.h**, **time.h**, dan **math.h**.

System Mechanic

1. About the System

PURRMART adalah sebuah aplikasi yang dapat mensimulasikan aktivitas beli barang pada *e-commerce*. PURRMART memiliki beberapa fitur utama, yaitu:

- Menampilkan barang toko
- Meminta dan menyuplai barang baru ke toko
- Menyimpan dan membeli barang dalam keranjang
- Menampilkan barang yang sudah dibeli
- Membuat dan menghapus *wishlist*
- Bekerja untuk menghasilkan uang

2. Menu Program

Ketika program pertama kali dijalankan, PURRMART akan memperlihatkan *main menu* yang berisi ***welcome menu*** dan beberapa *command* yaitu **START**, **LOAD**, dan juga **HELP**.

Setelah itu, program akan memasuki ***login menu*** yang memiliki command **LOGIN**, **REGISTER**, dan juga **HELP**. Jika pengguna berhasil memasuki kredensial suatu akun, maka mereka akan masuk ke menu selanjutnya.

Main menu menerima masukan berupa *command* yang akan dijelaskan pada bagian berikutnya. Program akan terus menerima *command* sampai diberikan *command* **QUIT** yang berlaku pada seluruh menu.

3. Command

Pengguna dapat memasukkan *command-command* berikut. Seluruh *command* hanya berlaku pada menu utama.

a. PROFILE

PROFILE adalah *command* yang digunakan untuk melihat data diri pengguna. PROFILE hanya dapat dipanggil saat status pengguna telah login

```
>> PROFILE
Nama : Purry
Saldo: 2000

(Silahkan kreasikan atribut yang ditampilkan)
// Kembali ke menu utama
```

b. CART ADD <nama> <n>

CART ADD adalah *command* yang digunakan untuk menambahkan barang dengan kuantitas tertentu ke dalam keranjang belanja.

```
>> CART ADD AK47 20
Berhasil menambahkan 20 AK47 ke keranjang belanja!
// Kembali ke menu utama
```

```
>> CART ADD BebekKaliya 240
Barang tidak ada di toko!
// Perintah invalid; Kembali ke menu utama
```

c. CART REMOVE <nama> <n>

CART REMOVE adalah *command* yang digunakan untuk mengurangi barang sejumlah kuantitas tertentu dari keranjang belanja. Perlu dilakukan validasi terhadap kuantitas yang diberikan, bila kuantitas pada keranjang belanja lebih sedikit dari N maka perintah akan gagal.

```
>> CART REMOVE AK47 10
Berhasil mengurangi 10 AK47 dari keranjang belanja!
// Kembali ke menu utama
```

```
>> CART REMOVE AK47 70
```

```
Tidak berhasil mengurangi, hanya terdapat 10 AK47 pada keranjang!
// Asumsi di keranjang belanja jumlah AK47 <70 sehingga perintah invalid; Kembali ke menu utama
```

```
>> CART REMOVE BintangSkibidi 70
Barang tidak ada di keranjang belanja!
// Asumsi tidak ada Bintang Skibidi di keranjang belanja; Kembali ke menu utama
```

d. CART SHOW

CART SHOW adalah *command* yang digunakan untuk menunjukkan barang-barang yang sudah dimasukkan ke dalam keranjang.

```
// Contoh dimana keranjang memiliki isi
>> CART SHOW
Berikut adalah isi keranjangmu.
Kuantitas  Nama    Total
2          AK47    20
1          Lalabu  10
Total biaya yang harus dikeluarkan adalah 30.

// Command mati; Kembali ke menu utama
```

```
// Contoh yang kosong
>> CART SHOW
Keranjang kamu kosong!
```

e. CART PAY

CART PAY adalah *command* yang digunakan untuk membeli barang-barang yang sudah dimasukan ke dalam keranjang. Perlu dipastikan bahwa **pengguna memiliki uang yang cukup** untuk membeli seluruh barang keranjang. Pembelian akan mengurangi uang yang dimiliki pengguna dan menambahkan riwayat pembelian.

Nama barang yang dimasukan ke riwayat pembelian adalah barang dengan total harga (harga barang * kuantitas) terbesar. Jika terdapat lebih dari 1 barang dengan total yang sama, maka yang disimpan adalah barang dengan urutan lexical yang lebih besar. Dimasukan juga total harga pada pembelian tersebut. Jika terdapat barang dengan nama Zebra dan AK47

yang memiliki total sama, maka Zebra akan dimasukan ke *history* karena secara lexical z > a.

Jika tidak mengerjakan bonus, maka barang yang dimasukan ke history **hanya 1** dengan ketentuan di atas.

```
// Contoh pembayaran yang berhasil (Pengguna memasukan Ya)
>> CART PAY
Kamu akan membeli barang-barang berikut.
Kuantitas  Nama    Total
2          AK47    20
1          Lalabu  10
Total biaya yang harus dikeluarkan adalah 30, apakah jadi dibeli? (Ya/Tidak): Ya

Selamat kamu telah membeli barang-barang tersebut!

// Command mati; Kembali ke main menu
```

```
// Contoh pembayaran yang gagal (Pengguna tidak memiliki uang yang cukup)
>> CART PAY
Kamu akan membeli barang-barang berikut.
Kuantitas  Nama    Total
2          AK47    20
1          Lalabu  20
Total biaya yang harus dikeluarkan adalah 40, apakah jadi dibeli? (Ya/Tidak): Ya

Uang kamu hanya 15, tidak cukup untuk membeli keranjang!

// Command mati; Kembali ke main menu
```

```
// Contoh pembayaran yang gagal (Pengguna memasukan Tidak)
>> CART PAY
Kamu akan membeli barang-barang berikut.
Kuantitas  Nama    Total
2          AK47    20
1          Lalabu  10
Total biaya yang harus dikeluarkan adalah 30, apakah jadi dibeli? (Ya/Tidak): Tidak

// Command mati; Kembali ke main menu
```

```
// Contoh pembayaran yang gagal (Pengguna memasukan masukan aneh)
```

```
>> CART PAY
Kamu akan membeli barang-barang berikut.
Kuantitas  Nama    Total
2          AK47    20
1          Lalabu  10
Total biaya yang harus dikeluarkan adalah 30, apakah jadi dibeli?
(Ya/Tidak): Purry

// Command mati; Kembali ke main menu
```

```
// Contoh pembayaran yang gagal (Keranjang kosong)
>> CART PAY
Keranjang kamu kosong!
```

f. HISTORY <n>

HISTORY adalah *command* yang digunakan untuk menunjukan riwayat pembelian seorang pengguna. N merupakan jumlah riwayat yang ditampilkan, contoh N=3 maka akan menampilkan 3 riwayat pembelian terbaru. Jika N melebihi jumlah riwayat pembelian yang ada, maka seluruh riwayat pembelian akan ditampilkan. Urutan penunjukan adalah dari yang paling baru ke paling tua.

```
// Contoh menunjukan riwayat pembelian N < total riwayat
>> HISTORY 3
Riwayat pembelian barang:
1. AK47 40
2. AK47 100
3. Lalabu 35

// Command mati; Kembali ke main menu
```

```
// Contoh menunjukan riwayat pembelian N ⩾ total riwayat
>> HISTORY 10
Riwayat pembelian barang:
1. AK47 40
2. AK47 100
3. Lalabu 35
4. AK47 10
5. Meong 500
6. Ayam Goreng Crisbar 20

// Command mati; Kembali ke main menu
```

```
// Contoh riwayat pembelian kosong
Kamu belum membeli barang apapun!
```

g. WISHLIST ADD

WISHLIST ADD merupakan *command* yang digunakan untuk menambahkan suatau barang ke *wishlist*.

```
>> WISHLIST ADD
Masukkan nama barang: Ayam Geprek Bakar Crispy Besthal

Berhasil menambahkan Ayam Geprek Bakar Crispy Besthal ke wishlist!
```

```
>> WISHLIST ADD
Masukkan nama barang: Ayam Geprek Bakar Crispy Besthal

Ayam Geprek Bakar Crispy Besthal sudah ada di wishlist!
```

```
>> WISHLIST ADD
Masukkan nama barang: Ayam Geprek Sambalado Besthal

Tidak ada barang dengan nama Ayam Geprek Sambalado Besthal!
```

h. WISHLIST SWAP <i> <j>

WISHLIST SWAP merupakan *command* yang digunakan untuk menukar barang posisi ke-i dengan barang posisi ke-j pada *wishlist*. Posisi i dan j merupakan urutan barang pada *wishlist*, urutan dimulai dari 1.

```
>> WISHLIST SWAP 1 2
Berhasil menukar posisi Ayam Geprek Bakar Crispy Besthal dengan Ayam Mangut Besthal pada wishlist!
// Urutan Ayam Geprek Bakar Crispy Besthal berubah dari 1 menjadi 2. Sebaliknya, urutan Ayam Mangut Besthal berubah dari 2 menjadi 1
```

```
>> WISHLIST SWAP 1 2
Gagal menukar posisi Ayam Geprek Bakar Crispy Besthal!
// Hanya terdapat satu barang (Ayam Geprek Bakar Crispy Besthal) pada wishlist sehingga posisinya tidak dapat ditukar
```

i. WISHLIST REMOVE <i>

WISHLIST REMOVE adalah *command* yang digunakan untuk menghapus barang dengan posisi ke-i dari *wishlist*.

```
// Contoh menghapus barang ke-i dari WISHLIST
>> WISHLIST REMOVE 2
Berhasil menghapus barang posisi ke-2 dari wishlist!

// Command mati; Kembali ke main menu
```

```
// Contoh penghapusan wishlist yang gagal (Pengguna memasukkan urutan barang yang tidak ada)

//Misalnya dalam kasus ini, hanya terdapat 5 barang di dalam wishlist
>> WISHLIST REMOVE 10
Penghapusan barang WISHLIST gagal dilakukan, Barang ke-10 tidak ada di WISHLIST!

// Command mati; Kembali ke main menu
```

```
// Contoh penghapusan wishlist yang gagal (Wishlist kosong)

//Misalnya dalam kasus ini, tidak ada barang di dalam wishlist
>> WISHLIST REMOVE 1
Penghapusan barang WISHLIST gagal dilakukan, WISHLIST kosong!

// Command mati; Kembali ke main menu
```

```
// Contoh penghapusan wishlist yang gagal (Pengguna memasukkan perintah yang tidak valid)
>> WISHLIST REMOVE XY
Penghapusan barang WISHLIST gagal dilakukan, command tidak valid!

// Command mati; Kembali ke main menu
```

j. WISHLIST REMOVE

WISHLIST REMOVE adalah *command* yang digunakan untuk menghapus barang dari wishlist berdasarkan nama barang yang dimasukkan pengguna.

```
// Contoh menghapus barang “Lalabu” dari WISHLIST
>> WISHLIST REMOVE
Masukkan nama barang yang akan dihapus : Lalabu
Lalabu berhasil dihapus dari WISHLIST!

// Command mati; Kembali ke main menu
```

```
//  Contoh  penghapusan  wishlist  yang  gagal  (Barang  tidak  ada  di
WISHLIST)

>> WISHLIST REMOVE
Masukkan nama barang yang akan dihapus : LoremIpsum
Penghapusan barang WISHLIST gagal dilakukan, LoremIpsum tidak ada di
WISHLIST!

// Command mati; Kembali ke main menu
```

k. WISHLIST CLEAR

WISHLIST CLEAR adalah *command* yang digunakan untuk menghapus semua barang yang terdapat di dalam WISHLIST.

```
>> WISHLIST CLEAR
Wishlist telah dikosongkan.
```

l. WISHLIST SHOW

WISHLIST SHOW adalah *command* yang digunakan untuk menunjukkan barang-barang yang sudah dimasukkan ke dalam wishlist.

```
>> WISHLIST SHOW
Berikut adalah isi wishlist-mu:
1 Ayam Geprek Bakar Crispy Besthal
2 Ayam Mangut Besthal
3 Karaage Don
4 Torikatsu Don
```

```
>> WISHLIST SHOW
Wishlist kamu kosong!
```

4. Perubahan Command

Terdapat beberapa *command* iterasi sebelumnya yang berubah, yaitu sebagai berikut.

a. START, LOAD, dan SAVE

Terdapat perubahan konfigurasi yang harus ditambahkan dalam implementasi *command* START, LOAD, dan SAVE. **BACA KONFIGURASI SISTEM UNTUK PEMBAHARUAN SAVE FILE!**

b. STORE LIST

STORE LIST akan menampilkan nama barang yang dijual **beserta harganya**.

```
>> STORE LIST
List barang yang ada di toko :
- Platypus Laser - Harga: 100
- Shrink Ray - Harga: 500
- Net Shooter - Harga: 250
- Camouflage Cloak - Harga: 150
- Sleep Dart Gun - Harga: 300
- Bubble Blaster - Harga: 200
```

```
>> STORE LIST
TOKO KOSONG
```

Konfigurasi Sistem

File konfigurasi akan dibaca saat memulai permainan. File ini menyimpan data-data yang disimpan ketika sistem dijalankan sebelumnya atau data-data *default*. Spesifikasi dari *file* konfigurasi adalah sebagai berikut:

1. Barisan pertama adalah bilangan bulat positif **N** yang menunjukan banyaknya barang di dalam sistem
2. Selanjutnya, sejumlah **N** baris menyatakan nama barang beserta harganya dengan format **<Harga barang> <Nama barang>**
3. Baris selanjutnya adalah bilangan bulat positif **M** yang menunjukan banyaknya pengguna di dalam sistem
4. Selanjutnya, terdapat data M buah pengguna dengan masing-masing spesifikasi berikut:

a. Baris pertama adalah bilangan bulat positif **K** yang menunjukan banyaknya riwayat pengguna tersebut
b. Selanjutnya, sejumlah **K** baris menyatakan nama barang beserta total biaya dengan format **<Total biaya> <Nama barang>**
c. Baris selanjutnya adalah bilangan bulat positif **J** yang menunjukan banyaknya *wishlist* pengguna tersebut
d. Selanjutnya, sejumlah **J** baris menyatakan nama barang dengan format **<Nama barang>**

Berikut adalah contoh file konfigurasi yang dimuat di awal sebuah *session* sebagai inisialisasi:

Contoh	Penjelasan baris di kanan
3 10 AK47 20 Lalabu 20 Ayam Goreng Crisbar 500 Meong	# Banyaknya barang di dalam toko
2 100 user1 alstrukdatkeren 6 40 AK47 100 AK47 35 Lalabu 10 AK47 500 Meong 20 Ayam Goreng Crisbar	# Banyaknya pengguna di dalam program # Data pengguna “user1” # Banyaknya riwayat pembelian pengguna “user1”
2 Ayam Goreng Crisbar AK47	# Banyaknya *wishlist* pengguna “admin”
25 user2 kerenbangetkak 0 1 Meong	# Data pengguna “user1” # Banyaknya riwayat pembelian pengguna “user2” # Banyaknya *wishlist* pengguna “user2”

Perlu diperhatikan bahwa antrian permintaan barang DAN keranjang tiap pengguna tidak disimpan di konfigurasi! Jika sistem dimulai, maka antrian dan keranjang akan dibuat lagi dari 0.

Daftar ADT

1. **ADT Kustom**
   Terdapat **perubahan** pada ADT pengguna (User). ADT akan menyimpan keranjang, riwayat pembelian, dan *wishlist* yang dimiliki oleh seorang user.

```
typedef struct {
  char name[MAX_LEN];
  char password[MAX_LEN];
  integer money;
  Map keranjang;
  Stack riwayat_pembelian;
  Linkedlist wishlist;
} User;
```

2. **ADT Stack**
   ADT ini digunakan untuk menyimpan riwayat pembelian seorang pengguna. Riwayat pembelian akan menyimpan nama barang dengan ketentuan di bawah dan total biaya seluruh barang.

```
typedef struct {
  char nama_barang[MAX_LEN];  // array of char dapat digantikan
                                 dengan Word
  int total_biaya_cart
} ElTypeStack;
```

   Nama barang yang disimpan adalah barang dengan total harga (harga barang * kuantitas) terbesar. Jika terdapat lebih dari 1 barang dengan total yang sama, maka yang disimpan adalah barang dengan urutan lexical yang lebih besar. Total biaya dari seluruh pembelian juga disimpan pada suatu elemen. Contoh, terdapat *cart* sebagai berikut.

```
Kuantitas  Nama    Total
2          AK47    20
1          Zebra   20
1          Lalabu  10
```

   Karena barang Zebra dengan AK47 memiliki total harga yang sama, maka dipilih Zebra karena secara lexical lebih besar (z > a). Pada history akan

ditambah entri baru dengan data <"Zebra", 50> (nama barang dan total harga)

3. **ADT Setmap**
   ADT ini digunakan untuk menyimpan keranjang pembelian seorang pengguna. Keranjang akan menyimpan nama beserta kuantitas barang yang ingin dibeli. Key elemen dalam setmap berupa barang yang pasti unik sementara value adalah kuantitas dari elemen tersebut.

4. **ADT Linked List**
   ADT ini digunakan untuk menyimpan *wishlist* seorang pengguna. Elemen yang disimpan di dalam *wishlist* adalah nama barang yang ingin dibeli.

Bonus

Pada tugas besar ini, terdapat beberapa fitur tambahan yang bisa diimplementasikan. Fitur-fitur ini tidak wajib untuk dikerjakan dan bobotnya lebih kecil dibanding fitur utama. **Utamakan fitur-fitur utama yang diminta sebelum mengerjakan bonus.** Berikut adalah penjelasan dari masing-masing fitur bonus:

1. **Store List Gacor**
   Pada iterasi sebelumnya, barang yang masuk ke toko harus bersifat **unique.** Sebelumnya, kalian menggunakan ADT List Dinamis untuk menyimpan barang-barang tersebut. Apakah terdapat ADT lain yang lebih efisien untuk mengimplementasikan fungsionalitas tersebut? Jika ada, ubah kode *store* kalian untuk menggunakan ADT-nya.
   Catatan: *Insertion* dan *deletion* dengan kompleksitas ~O(1) akan diberikan nilai tambahan.

2. **Riwayat Maksimal**
   ADT Stack digunakan untuk menyimpan riwayat pembelian seorang pengguna. Namun, data yang disimpan hanya nama barang dengan total harga terbesar. Petinggi OWCA ingin memiliki detail setiap pembelian ~~untuk keperluan *tax fraud*~~, mereka ingin *command* HISTORY untuk menunjukan seluruh barang yang dibeli.

```
// Contoh menunjukan riwayat pembelian N < total riwayat
```

```
>> HISTORY 3
Riwayat pembelian barang:
Pembelian 1 - Total 40
Kuantitas  Nama    Total
2          AK47    20
1          Lalabu  20

Pembelian 2 - Total 100
Kuantitas  Nama    Total
8          AK47    80
1          Lalabu  20

Pembelian 3 - Total 35
Kuantitas  Nama    Total
2          M14     15
1          Lalabu  20

// Command mati; Kembali ke main menu
```

Agar detail riwayat pembelian *persistent*, terdapat juga perubahan pada konfigurasi file, yaitu:

a. Tidak terdapat **K** baris riwayat, tetapi elemen riwayat.
b. Baris pertama setiap elemen riwayat adalah bilangan positif tidak nol **L** yang menunjukan banyaknya barang yang dibeli dalam pembelian dan bilangan **X** yaitu total harga pada pembelian tersebut
c. Selanjutnya, sejumlah **L** baris menyatakan barang yang dibeli dengan format **<Total biaya> <Kuantitas barang> <Nama barang>**

Contoh konfigurasi baru adalah sebagai berikut.

```
/** Konfigurasi barang toko */
/** Konfigurasi User lainnya */
100 user1 alstrukdatkeren
6
2 40
20 2 AK47
20 1 Lalabu
2 100
80 8 AK47
20 1 Lalabu
2 35
15 2 M14
20 1 Lalabu
```

```
/** Dibawah adalah penjelasan
tulisan di kanan */
# Data user
# Banyaknya elemen riwayat pengguna
# Banyaknya riwayat-1 dan total


# Banyaknya riwayat-2 dan total


# Banyaknya riwayat-3 dan total
```

1 10 10 1 AK47 1 500 500 1 Meong 1 20 20 1 Ayam Goreng Crisbar /** Konfigurasi wishlist user*/ /** Konfigurasi User lainnya */	# Banyaknya riwayat-4 dan total # Banyaknya riwayat-5 dan total # Banyaknya riwayat-6 dan total

3. **Deteksi Kebocoran Senjata Biologis**

   Pada *milestone* sebelumnya, beberapa dari Anda telah sukses membuat sistem untuk mendeteksi kode pada DNA dari pabrik untuk mencegah sabotase musuh. Namun, OWCA mencurigai adanya kebocoran pada gudang PURRMART akibat penyimpanan yang tidak sesuai prosedur operasional baku (POB). Untuk menyelidiki hal tersebut, OWCA mencoba melakukan metagenomik untuk mengetahui spesies yang terdapat pada sampel dari lingkungan. Proses tersebut membutuhkan proses *sequence alignment* untuk mengecek kesamaan antara sekuens senjata biologis dengan sekuens hasil metagenomik. Oleh karena itu, Anda diminta mengimplementasikan kakas *global alignment* dengan algoritma Needleman-Wunsch. Apabila skor akhir lebih besar dari 80% panjang sekuens yang lebih panjang (jumlah karakter/basa nukleotida), maka dapat disimpulkan bahwa terjadi kebocoran senjata biologis. Panjang sekuens maksimum 50 karakter. Panjang kedua sekuens tidak harus sama, tetapi cukup mirip (misal 48 karakter dan 44 karakter). **Perhatikan kompleksitas algoritma, tidak boleh lebih dari ~O(2^n).**

   Ketentuan *scoring*:
   - *Match:* +1
   - *Mismatch:* 0
   - *Gap penalty:* -1

```
>> GLOBALALIGNMENT
Masukan sekuens referensi: TAGTAGAATGGGAGAGGTT // Panjang: 19 karakter
Masukan sekuens query:     TAGTAGGGTTAATGTT // Panjang: 16 karakter

Skor: 9
Sekuens yang telah disejajarkan:
TAGTAGAATGGGAGAGGTT
```

```
TAGTAG---GGTTAATGTT

Woah! Tidak ada kebocoran (≧◡≦) // 19*80% = 15.20 > 9 (lebih rendah)
```

```
>> GLOBALALIGNMENT
Masukan sekuens referensi: TAGTAGAATGGGAGAGGC // Panjang: 18 karakter
Masukan sekuens query:     TAGTAGAATGGGTAAGTC // Panjang: 18 karakter

Skor: 15
Sekuens yang telah disejajarkan:
TAGTAGAATGGGAGAGGC
TAGTAGAATGGGTAAGTC

Nawh! Ada kebocoran... (ಥ﹏ಥ) // 18*80% = 14.4 < 15 (lebih tinggi)
```

Fun Fact: Di dunia nyata, penggunaan *global alignment* untuk menjajarkan hasil metagenomik dengan tujuan identifikasi organisme bersifat tidak lazim. Kakas seperti BLAST yang menerapkan *local alignment* lebih tepat dan lazim digunakan. Namun, implementasinya rumit sehingga tidak dipilih dalam tugas ini.

4. Optimasi Rute Ekspedisi

Toko PURRMART memiliki banyak klien sehingga perusahaan SiLambat, rekan toko PURRMART, membutuhkan cara untuk mengirim barang dengan rute yang paling efisien. Seluruh titik harus dikunjungi dan suatu titik ke titik lainnya memiliki jarak tertentu. Awalnya, salah satu karyawan PURRMART, menggunakan algoritma BFS untuk menyelesaikan permasalahan ini. Namun, ternyata algoritma tersebut tidak efisien dan memerlukan kemampuan komputasi yang besar. Anda diminta menggunakan algoritma alternatif yang lebih efisien untuk menyelesaikan masalah ini, semakin efisien semakin baik. **Jelaskan alasan pemilihan algoritmanya di laporan.**

```
>> OPTIMASIRUTE
Masukkan jumlah lokasi pengiriman (node): 4
Masukkan jumlah rute (edge): 5
Masukkan jarak antarlokasi (weight):
0 1 10
0 2 15
0 3 20
1 2 35
1 3 25

Data diterima, silakan tunggu…

A-ha! Rute paling efektif adalah 0-2-1-3 dengan total jarak 75.
```

5. **Pencarian Barang**

   Toko PURRMART memiliki terlalu banyak barang sehingga kamu diminta untuk membantu membuat sebuah mesin pencari baru. Mesin pencari yang lama tidak efektif dikarenakan mesin tersebut mencari barang yang memiliki nama secara *exact match*. Karena manusia tempatnya salah dan lupa, barang yang dicari sering tidak ketemu karena nama barang yang dicari seringlah sekelumit berbeda. Oleh karena itu, kamu diminta untuk membuat mesin pencari kuasi *match* menggunakan jarak Levenshtein dengan *threshold distance default* adalah 10. *Threshold distance default* dapat diganti.

- *Command* akan diterima dengan format seperti berikut
- STORE FIND <nama-barang> --distance=<X> --ignore-casing --ignore-space
- Flag - flag berupa --distance, --ignore-casing, --ignore-space bersifat opsional
- Flag --distance=X menyatakan untuk mengganti *threshold default* menjadi X
- Flag --ignore-casing untuk menganggap bahwa 2 character yang hanya berbeda dalam casingnya dapat dianggap sebagai huruf yang sama. Lowercase letter and uppercase letter dianggap sebagai huruf yang sama.
- Flag --ignore-space menyatakan bahwa spasi di dalam pencarian dapat di ignore pada semua string sehingga matching akan dilakukan antar 2 string yang tidak memiliki whitespace
- Urutan order dari *command* tidaklah *strict*. Ketiga *command* tersebut akan mengeluarkan hasil yang sama.
  - STORE FIND AK47 --distance=10 --ignore-casing
  - STORE FIND --distance10 AK47 --ignore-casing
  - STORE FIND --ignore-casing AK47 -distance10

Asumsi store sudah memiliki barang-barang ini
- **Wobbulous X200 Supreme**
- **GrumblySnort Extreme**
- **Zizzle Pop Max Plus**
- **Floombastic Ultra Boost**
- **WubbleTronix 3000 Deluxe**
- **Zizzle Zap UltraMax**

>> **STORE FIND Zarflob Supreme Edition**
Barang tidak ditemukan.

>> **STORE FIND Floompbastic Ultrablast**
Berikut adalah barang yang mirip:
- Floombastic Ultra Boost

```
>> STORE FIND --distance=7 Zizzle Zap Plus
Berikut adalah barang yang mirip:
   -  Zizzle Pop Max Plus
   -  Zizzle Zap UltraMax
```

```
>> STORE FIND WobbulousX202Supreme --ignore-space
Berikut adalah barang yang mirip:
   -  Wobbulous X200 Supreme
```

```
>> STORE FIND --ignore-casing zizzle zap ultraamax
Berikut adalah barang yang mirip:
   -  Zizzle Zap UltraMax
```

Catatan Tambahan

1. Tampilan program boleh dibuat sesuai keinginan kalian, tampilan pada spesifikasi ini hanya merupakan contoh.
2. **Diwajibkan** untuk membuat **driver** untuk masing-masing ADT. *Driver* berisi sebuah *main file* yang memanggil fungsi/prosedur yang ada di ADT tersebut. Kegunaan *driver* adalah untuk *testing* ADT yang sudah dibuat.
3. Sebagai saran, manfaatkan **Makefile** untuk mempermudah proses kompilasi dan penjalanan program. Bila sulit dalam menggunakan **Makefile**, bisa diakali dengan menggunakan **shell *script*/batch *file***.
4. Gunakan **GitHub** sebagai *version control*, lalu undang asisten kalian sebagai *collaborator*. Pastikan asisten sudah masuk ke dalam *repository* sebelum asistensi pertama.
5. Buat *file* README yang minimal mengandung deskripsi singkat program, identitas anggota kelompok dan cara kompilasi program. *Readme* dapat
6. dibuat dengan menggunakan markdown.
7. Buat struktur program yang serapi mungkin. Jangan buat semuanya pada *file* yang sama. Contoh struktur program (tidak harus diikuti):

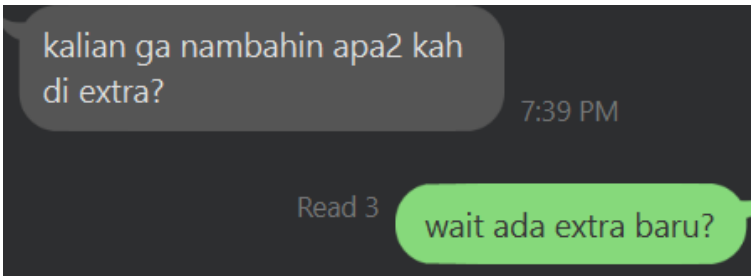
```
.
├── bin
│   ├── Makefile
│   └── sample_savedata
├── save
│   ├── save1.txt        # Konfigurasi yang pernah disimpan
│   ├── saveberapa.txt   # Konfigurasi lain yang pernah disimpan
│   └── default.txt      # Konfigurasi awal
├── docs
│   ├── Spesifikasi....pdf
│   ├── IF2111_Form..._01.xlsx
│   └── IF2111_TB..._01.pdf
├── src
│   ├── boolean.h
```

```
│   ├── console.c         # Implementation of all commands
│   ├── console.h
│   ├── main.c            # Simply calls the Game function in console
│   ├── ADT               # Location of all the data structures used by this game.
│   │                     # Every subfolder will have a header, a c implementation and a driver.
│   │   ├── kata
│   │   ├── listlinier
│   │   ├── mesinkata
└── └── └── stack
README.MD
```

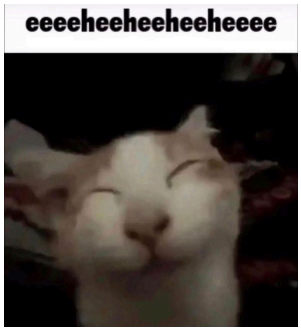
8. Manfaatkan ADT yang sudah kalian buat dalam praktikum semaksimal mungkin.
9. Perhatikan bahwa nilai untuk bonus akan **lebih kecil** dibandingkan dengan fitur utama. Silakan prioritaskan fitur-fitur yang lebih penting terlebih dahulu.
10. Jika ada yang kurang jelas, silahkan bertanya melalui QnA.

Extras

"Ketika kamu dihantam oleh bertubi-tubi tubes,
tapi kamu adalah orang yang *chill*"
- Cipher (*probably*)



"Maaf kami lupa ada extras baru 😭"
– 5 dari 6 asisten



"Anyway, tetap semangat ya ciphers lop yu all! (ノ◕ワ◕)ノ*:･ﾟ✧"
– Aul

~ Semangat guys kalian pasti bisa! ~