

**SKRIPSI**

**PERKAKAS COMMAND LINE KIRI**



**Alfred Aprianto Liaunardi**

**NPM: 6181801014**

**PROGRAM STUDI TEKNIK INFORMATIKA  
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS  
UNIVERSITAS KATOLIK PARAHYANGAN  
2022**



# DAFTAR ISI

<b>DAFTAR ISI</b>	<b>iii</b>
<b>DAFTAR GAMBAR</b>	<b>v</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	2
1.3 Tujuan . . . . .	2
1.4 Batasan Masalah . . . . .	2
1.5 Metodologi . . . . .	2
1.6 Sistematika Pembahasan . . . . .	3
<b>2 LANDASAN TEORI</b>	<b>5</b>
2.1 <i>Command Line</i> . . . . .	5
2.1.1 <i>Command Line Interface</i> dan <i>Graphical User Interface</i> . . . . .	5
2.1.2 <i>Command Line</i> di Linux . . . . .	6
2.1.3 <i>Command Line</i> di Windows . . . . .	7
2.2 KIRI . . . . .	10
2.2.1 Tampilan . . . . .	11
2.2.2 API . . . . .	12
<b>3 ANALISIS</b>	<b>15</b>
3.1 Analisis Aplikasi Sejenis . . . . .	15
3.1.1 <i>Chrome Web Store Item Property CLI</i> . . . . .	15
3.1.2 <i>iTunes Search API</i> . . . . .	16
3.2 Analisis Modul dan Fungsi Bahasa C . . . . .	17
3.2.1 <i>getopt</i> . . . . .	18
3.3 Analisis API KIRI . . . . .	19
<b>DAFTAR REFERENSI</b>	<b>21</b>
<b>A KODE PROGRAM</b>	<b>23</b>
<b>B HASIL EKSPERIMEN</b>	<b>25</b>



## DAFTAR GAMBAR

1.1	Tampilan halaman web KIRI . . . . .	1
2.1	Dua jenis tampilan perangkat lunak . . . . .	5
2.2	Baris <i>shell prompt</i> terminal di sistem operasi Linux. . . . .	6
2.3	Tampang kedua antarmuka <i>command line</i> bawaan di sistem operasi Windows. . . . .	8
2.4	Tampilan awal halaman web KIRI . . . . .	11
2.5	Tampilan akhir halaman web KIRI . . . . .	12
3.1	Contoh penggunaan perkakas <i>Chrome Web Store Item Property CLI</i> . . . . .	16
3.2	Contoh penggunaan perkakas <i>iTunes Search API</i> . . . . .	18
B.1	Hasil 1 . . . . .	25
B.2	Hasil 2 . . . . .	25
B.3	Hasil 3 . . . . .	25
B.4	Hasil 4 . . . . .	25

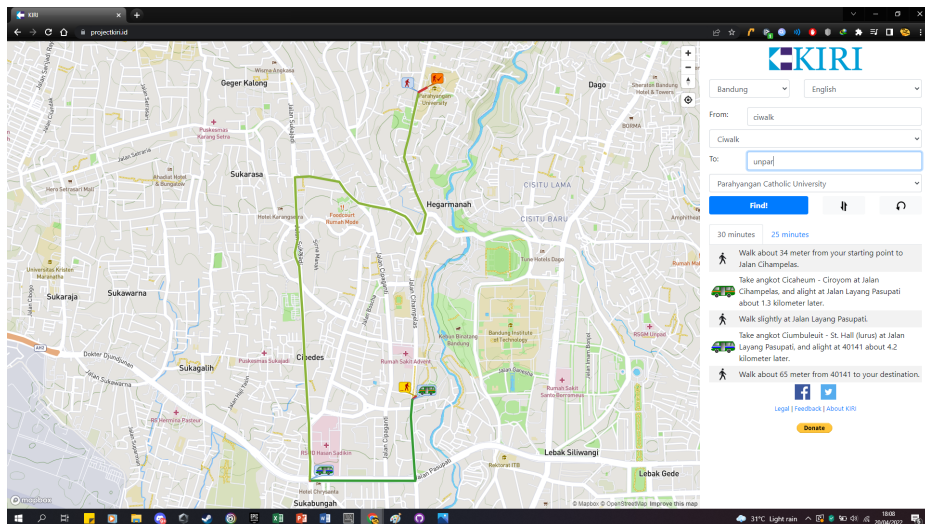


# BAB 1

## PENDAHULUAN

### 1.1 Latar Belakang

Project KIRI<sup>1</sup> (akan disingkat sebagai KIRI dalam dokumen ini) adalah sebuah perangkat lunak berbasis web yang dibuat untuk membantu mengurangi efek dari kemacetan. KIRI mengurangi dampak kemacetan dengan membantu penggunanya, baik masyarakat maupun turis, dalam menggunakan salah satu sarana transportasi umum yang ada di Indonesia, yaitu angkutan kota (angkot). Cara KIRI mempermudah penggunaan angkot adalah dengan menunjukkan rute yang akan ditempuh, beserta langkah-langkah yang harus dilakukan oleh pengguna yang ingin berpergian dari satu titik ke titik lain, mulai dari seberapa jauh pengguna harus berjalan untuk menaiki angkot yang bersangkutan, di mana pengguna harus naik atau turun, seberapa jauh lagi pengguna harus berjalan sampai ke titik tujuan, dan seberapa lama estimasi waktu perjalanan yang akan ditempuh. Untuk kebutuhan pembuatan perangkat lunak yang memanfaatkan fitur dari KIRI, tersedia juga REST API KIRI yang dapat digunakan secara praktis. Adapun tampilan dari halaman web ini dapat dilihat di gambar 1.1.



Gambar 1.1: Tampilan halaman web KIRI, yang menunjukkan rute dari Cihampelas Walk ke Universitas Katolik Parahyangan.

Sementara itu, dalam komputer, salah satu dari sekian banyak tipe perangkat lunak adalah *command line*. *Command line* (*command line interpreter*, atau *command line interface*) adalah

<sup>1</sup><https://projectkiri.id>

sebuah perangkat lunak berupa sebuah kotak/*window* yang memuat teks berupa perintah-perintah,<sup>2</sup> yang menerima masukan dari pengguna dan menjalankannya.[1] Perintah-perintah ini hanya berupa gabungan dari teks and simbol-simbol berupa karakter, tanpa ada tambahan gambar grafis apapun. Singkatnya, tipe perangkat lunak ini bukan merupakan tipe yang paling indah untuk dilihat oleh para pengguna, tetapi jika digunakan dengan tepat, maka jenis perangkat lunak ini bisa menyuruh komputer untuk melakukan banyak sekali perintah-perintah dengan sangat cepat dan sangat efektif. Pada skripsi ini akan dibuat sebuah perangkat lunak berupa perkakas *command line* (*command line tool*) yang dapat menjalankan fungsi-fungsi API dari KIRI. Perangkat lunak ini, seperti jenisnya, akan dibuat murni sebagai perkakas yang dijalankan dari *command line* (terminal, cmd, PowerShell, dll.), dan tampilan akhir dari perangkat lunak akan berupa *command line interface* tanpa tambahan *graphical user interface*. Keseluruhan dari perangkat lunak ini akan dibangun dalam bahasa C.

## 1.2 Rumusan Masalah

1. Bagaimana membangun perkakas *command line* yang dapat mengimplementasikan fitur-fitur API KIRI dalam bahasa C?
2. Bagaimana integrasi perkakas *command line* KIRI dapat dilakukan dengan perkakas-perkakas *command line* lainnya di Linux?

## 1.3 Tujuan

Batasan masalah dalam skripsi ini adalah sebagai berikut:

1. Membangun perkakas *command line* yang dapat mengimplementasikan fitur-fitur API KIRI dalam bahasa C.
2. Melakukan integrasi perkakas *command line* KIRI dengan perkakas-perkakas *command line* lainnya di Linux.

## 1.4 Batasan Masalah

Batasan masalah dalam skripsi ini adalah sebagai berikut:

1. Perangkat lunak dibuat murni dalam bentuk CLI, tanpa tambahan GUI.
2. Perangkat lunak yang dibuat tidak menyelesaikan batasan (lokasi tidak terdeteksi, rute tidak berhasil ditemukan, dsb.) yang sudah sejak awal terdapat dalam KIRI.

## 1.5 Metodologi

Metodologi yang akan diikuti dalam skripsi ini adalah sebagai berikut:

1. Melakukan studi dan eksplorasi terhadap fungsi-fungsi yang dimiliki perangkat lunak KIRI serta cara implementasi API KIRI.
2. Melakukan analisis dan desain perangkat lunak yang akan dibangun.

---

<sup>2</sup>[Ubuntu Tutorials - The Linux command line for beginners: 3. Opening a Terminal](#)



3. Melakukan studi dan eksplorasi terhadap seluruh kemungkinan *library-library* yang memenuhi spesifikasi dalam pembuatan perangkat lunak, berdasarkan analisis dan desain yang telah dilakukan sebelumnya.
4. Melakukan analisis kebutuhan fitur-fitur perangkat lunak dan melakukan eksplorasi *library* yang dapat digunakan dan memenuhi spesifikasi dalam pembuatan perangkat lunak.
5. Membangun perangkat lunak berdasarkan rancangan yang sudah dibuat, dengan mengimplementasikan seluruh modul dan *library* yang telah ditentukan di tahap sebelumnya dalam bahasa C.
6. Melakukan pengujian fungsional, perbaikan *bug*, serta rekomendasi perbaikan berdasarkan pengujian yang sudah dilakukan.
7. Menyelesaikan pembuatan dokumen-dokumen yang berkaitan, seperti dokumen skripsi dan dokumentasi perangkat lunak.

## 1.6 Sistematika Pembahasan

Setiap bab dalam skripsi ini memiliki sistematika pembahasan dalam poin-poin sebagai berikut:

1. Bab 1: Pendahuluan

Bab ini berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi penelitian, dan sistematika pembahasan.

2. Bab 2: Dasar Teori

Bab ini berisi pembahasan-pembahasan teoritis mengenai aspek-aspek yang akan dirujuk di dalam skripsi ini, seperti *command line*, bahasa C, dan juga KIRI.

3. Bab 3: Analisis dan Perancangan

Bab ini berisi pembahasan mengenai rancangan perangkat lunak serta seluruh analisis yang dilakukan terhadap kebutuhan fitur perangkat lunak.

4. Bab 4: Implementasi dan Pengujian

Bab ini berisi pembahasan mengenai pembuatan perangkat lunak, implementasi seluruh modul-modul yang telah ditentukan di bab 3, serta pengujian fitur-fitur dari perangkat lunak.

5. Bab 5: Kesimpulan dan Saran

Bab ini berisi kesimpulan hasil pembuatan perangkat lunak dan saran-saran terhadap hasil perangkat lunak yang diberikan selama pengerjaan skripsi.



## BAB 2

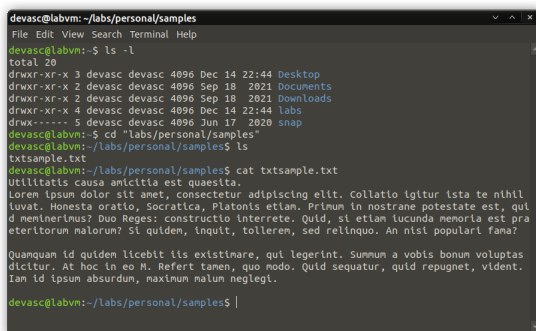
## LANDASAN TEORI

### 2.1 *Command Line*

*Command line* (atau *command line interface*) dapat diartikan sebagai tampilan antarmuka/*interface* yang memproses perintah dari pengguna dan meneruskannya langsung ke sistem operasi untuk dijalankan.[2] Seluruh sistem operasi komputer yang ada memiliki sebuah *command line interface* dalam bentuk *shell*, yang dapat digunakan oleh penggunanya untuk langsung mengakses fungsi atau servis yang disediakan oleh sistem operasi.[3]

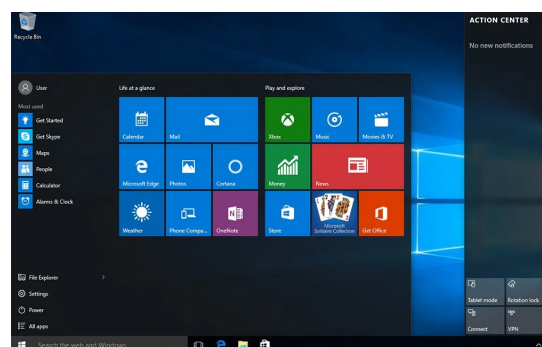
#### 2.1.1 *Command Line Interface dan Graphical User Interface*

Ada beberapa dari tipe antarmuka yang masih banyak digunakan di zaman sekarang, tetapi dua tipe yang paling banyak muncul adalah *command line interface* dan *graphical user interface*. Perangkat lunak berbasis *command line* sendiri bisa memiliki berbagai macam tampilan, tetapi semuanya selalu mengikuti satu bentuk antarmuka umum. Bentuk yang dimaksud adalah sebuah area/*window* yang memuat teks berupa perintah-perintah dari user untuk dilakukan oleh komputer, beserta keluarannya yang juga berupa teks, seperti dapat dilihat pada gambar 2.1a. Jenis perangkat lunak seperti ini disebut memiliki antarmuka jenis *command line interface* (CLI). Adapun dekorasi visual yang dimiliki oleh jenis tampilan ini hanya berupa warna pada teks-teks yang ada, tanpa tambahan gambar apapun. Jika perangkat lunak tersebut memiliki dekorasi dan/atau tombol interaktif berupa gambar grafis, seperti pada gambar 2.1b, maka perangkat lunak tersebut dikategorikan sebagai perangkat lunak berbasis *graphical user interface*.



```
devasc@labvm: ~/labs/personal/samples
File Edit View Search Terminal Help
devasc@labvm:~$ ls -l
total 20
drwxr-xr-x 3 devasc devasc 4096 Dec 14 22:44 Desktop
drwxr-xr-x 2 devasc devasc 4096 Sep 18 2021 Documents
drwxr-xr-x 2 devasc devasc 4096 Sep 18 2021 Downloads
drwxr-xr-x 4 devasc devasc 4096 Dec 14 22:44 Labs
drwxr-xr-x 5 devasc devasc 4096 Jun 17 2020 snap
devasc@labvm:~$ cd ~/labs/personal/samples
devasc@labvm:~/labs/personal/samples$ ls
txtsample.txt
devasc@labvm:~/labs/personal/samples$ cat txtsample.txt
Utilitatis causa amicitia est quaestita.
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Collatio igitur ista te nihil
luvat. Honestas oratio, Socratica, Platonis etiam. Primum in nostram potestatem est, qui
d meminerimus? Duo Reges: constructio interrete. Quid, si etiam lucunda memoria est pra
eteritorum malorum? Si quidem, inquit, tolerem, sed relinquo. An nisi populari fano?
Quamquam id quidem licebit illis existinare, qui legerint. Summun a vobis bonum voluptas
dicitur. At hoc in eo M. Refert tamen, quo modo. Quid sequatur, quid repugnet, vident.
Iam id ipsum absurdum, maximum malum neglegi.
```

(a) Antarmuka perangkat lunak berbasis *command line interface*.



(b) Antarmuka perangkat lunak berbasis *graphical user interface*.

Gambar 2.1: Contoh dua jenis antarmuka (*interface*) perangkat lunak.

Selain dari tampilannya sendiri, ada beberapa perbedaan utama lain antara perangkat-perangkat lunak berbasis *command line interface* dengan perangkat lunak berbasis *graphical user interface*. Adapun perbedaan-perbedaan utama dari kedua jenis antarmuka ini adalah sebagai berikut.[3]

- Penggunaan sumber daya sistem untuk menjalankan perangkat lunak berbasis *command line interface* lebih rendah dibandingkan dengan perangkat lunak berbasis *graphical user interface*.
- Bagi pengguna pemula (atau pengguna awam pada umumnya), perangkat lunak berbasis *command line interface* akan lebih sulit digunakan karena tidak adanya bantuan apapun dalam bentuk visual, sehingga satu-satunya cara untuk tahu bagaimana cara menggunakan fitur-fiturnya adalah melalui dokumentasi perangkat lunak yang ada. Karena alasan yang sama pula, perangkat lunak berbasis *command line interface* lebih sulit untuk dibiasakan penggunaannya.
- Automasi perintah yang bersifat berulang-ulang jauh lebih mudah dilakukan pada perangkat lunak berbasis *command line interface*. Hal ini dikarenakan perangkat lunak berbasis *command line interface* tidak hanya lebih mudah untuk dibuat *script*-nya, tetapi juga lebih efisien untuk digunakan ketika ada banyak sekali perintah yang harus dilakukan pada suatu saat tertentu.

### 2.1.2 *Command Line* di Linux

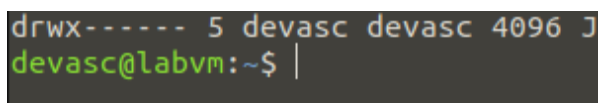
Linux merupakan sebuah sistem operasi yang sangat modular, jadi ada banyak sekali *shell* yang dapat dijalankan dan digunakan di dalamnya. Walaupun begitu, ada satu *shell* yang selalu datang ter-*install* di dalam semua sistem operasi Linux, yaitu "*bash*" (GNU *Bourne Again Shell*).[4]

#### Tampilan

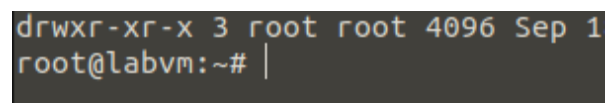
Ketika terminal di Linux dijalankan, akan keluar kotak dialog, beserta sebuah baris. Baris ini biasanya berisi sebuah teks dengan format sebagai berikut.

```
<nama pengguna>@<nama perangkat>:<direktori yang sedang diproses>$
```

Tanda dolar di ujung baris ini menandakan bahwa baris tersebut merupakan baris *shell prompt*, yang merupakan waktu di mana terminal sudah siap menerima masukan dari pengguna untuk diproses. Perlu diingat bahwa di posisi tanda dolar ini, terkadang justru terdapat tanda pagar (#). Tanda pagar di akhir baris *shell prompt* menandakan bahwa terminal tersebut dijalankan dengan tingkat akses *superuser*, yang berarti bahwa entah pengguna masuk ke sistem sebagai user *root*, atau terminal memiliki izin tingkat *superuser/administrator*. [2]



(a) *Shell prompt* terminal dengan tingkat izin normal.



(b) *Shell prompt* terminal dengan tingkat izin *superuser*.

Gambar 2.2: Baris *shell prompt* terminal di sistem operasi Linux.

## 1 Navigasi [2]

2 Sama seperti di Windows, Linux menyimpan file-filenya di sebuah struktur direktori yang bersifat  
3 hierarkial. Hal ini berarti bahwa file-file tersebut disimpan dalam direktori-direktori (atau *folder-*  
4 *folder*) yang tersusun seperti sebuah pohon. dalam arti bahwa satu *folder* bisa jadi berada di dalam  
5 satu *folder* lain, atau berisi beberapa *folder* lainnya.

6 Untuk navigasi, terminal Linux memiliki beberapa perintah utama. Adapun perintah-perintah  
7 tersebut adalah sebagai berikut.

- 8 • **pwd**

9 **pwd** merupakan singkatan dari *print working directory*, yang berarti bahwa perintah ini akan  
10 mengeluarkan *working directory*, atau direktori tempat terminal sekarang sedang bekerja/ber-  
11 jalan, sebagai keluaran dari perintah tersebut. Ketika pengguna pertama kali menjalankan  
12 terminal, *working directory*-nya selalu merupakan direktori *home* dari perangkat.

- 13 • **ls**

14 **ls** digunakan untuk menghasilkan keluaran berupa isi dari folder yang dispesifikasi. Biasanya  
15 digunakan ketika pengguna sudah memasuki folder yang diinginkan, walaupun dengan perintah  
16 ini, pengguna bisa saja mengintip isi dari folder manapun di direktori manapun, dengan  
17 mengikutkan direktori yang diinginkan sebagai parameter dari perintah tersebut. Adapun  
18 Isi dari folder yang diikutkan sebagai parameter tidak hanya berupa folder lain, tetapi juga  
19 seluruh file-file yang ada, walaupun untuk file-file yang disembunyikan (nama file diawali  
20 dengan tanda titik), perlu ditambahkan opsi **-a** agar file-file tersebut muncul pula dalam  
21 keluarannya.

- 22 • **cd**

23 **cd** adalah perintah yang berfungsi untuk mengganti *working directory* dari terminal. Untuk  
24 melakukan hal tersebut, perintah yang perlu dimasukkan adalah sebagai berikut:

```
25 cd <direktori yang diinginkan>
```

26 Direktori yang diinginkan dapat berupa direktori absolut, atau direktori relatif. Perbedaanannya  
27 adalah direktori absolut selalu dimulai dari folder *root*, mengikuti folder-folder apapun yang  
28 ada di antara *root* sampai ke folder yang diinginkan.

29 Sedangkan, direktori relatif selalu dimulai dari *working directory*. Untuk penggunaan direktori  
30 relatif, diperlukan dua buah notasi spesial, yaitu titik (**.**), yang merepresentasikan *working*  
31 *directory* sekarang itu sendiri, dan dua titik (**..**), yang merepresentasikan *parent folder* dari  
32 *working directory*.

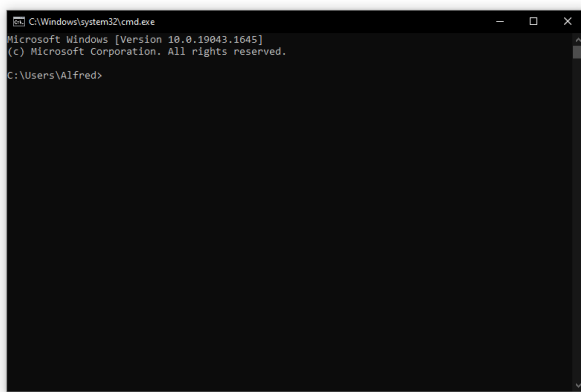
### 33 2.1.3 Command Line di Windows

34 Cara kerja *command line* di Windows serupa dengan cara kerja *command line* di Linux, dalam  
35 arti bahwa untuk bekerja dengan *command line* di Windows, penggunaanya juga akan langsung  
36 berinteraksi dengan utilitas yang disediakan oleh sistem operasi. *Command line* di Windows juga  
37 dapat digunakan untuk hal-hal yang serupa dengan *command line* di Linux, seperti menulis (dan  
38 menjalankan) *script*, menjalankan perintah yang diinginkan pengguna secara otomatis, atau melihat  
39 status dari sistem operasi.[3]

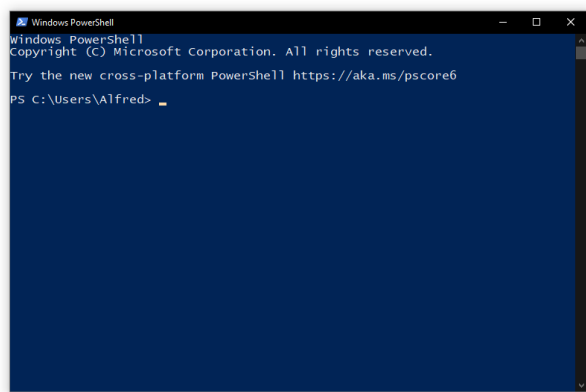
Masih sama dengan Linux, ada banyak sekali command yang bisa digunakan, sehingga susah untuk menghafal seluruh command-command yang ada—termasuk masukan, parameter-parameter yang dibutuhkan, serta keluarannya. Untuk melihat dokumentasi, atau penjelasan detail untuk masukan, keluaran, parameter, serta opsi-opsi dari perintah tertentu, pengguna dapat memasukkan perintah tersebut, diikuti dengan `/?`.<sup>[3]</sup>

## Tampilan

Di sistem operasi Windows, ada dua jenis antarmuka *command line*, yaitu `cmd` (*Command Prompt*) dan *PowerShell*. Keduanya memiliki tampilan yang kurang lebih sama—hanya saja awalnya `cmd` memiliki latar belakang hitam, sedangkan *PowerShell* memiliki latar belakang biru tua, seperti terlihat di gambar 2.3.



(a) Antarmuka Windows *Command Prompt* (`cmd`)



(b) Antarmuka Windows *PowerShell*

Gambar 2.3: Tampilan kedua antarmuka *command line* bawaan di sistem operasi Windows.

## Navigasi

Untuk navigasi di antarmuka *command line* Windows, ada dua perintah penting yang dipakai ketika pengguna sedang berurusan dengan file-file dan navigasi dalam direktori sistem. Kedua perintah tersebut adalah `cd` dan `dir`.

- `cd` (`chdir`)<sup>1</sup>

`cd` merupakan sebuah perintah yang memiliki tiga fungsi utama, yaitu menampilkan *drive* tempat sedang *command line* berada (jika pengguna hanya memasukkan `cd` tanpa parameter apapun), menampilkan direktori tempat *command line* sedang berada (jika pengguna hanya memasukkan *drive* sebagai parameter, atau fungsi yang paling umumnya, untuk mengganti *working directory* dari *command line*.

Adapun format dari perintah `dir` adalah sebagai berikut.

```
cd [/d] [<drive>:][<path>]
```

<sup>1</sup>[cd | Microsoft Docs](#)

Dengan fungsi dari semua opsi dan parameter yang ada sebagai berikut.

– /d

Opsi yang menandakan bahwa pengguna ingin mengganti *drive* (partisi) dan juga *working directory* dari *command line*.

– <drive>:

Kode huruf dari partisi yang akan diproses.

– <path>

Direktori yang akan diproses. Parameter ini harus diikuti dengan kode huruf partisi (tidak dapat berdiri sendiri.)

- **dir**

**dir** merupakan sebuah perintah yang mengeluarkan/menampilkan sebuah daftar berisi file-file yang ada di suatu direktori, termasuk subdirektori. Jika tidak disertai parameter apapun, perintah ini akan menampilkan label volume dan nomor serial *disk*, dilanjutkan dengan daftar direktori dan file di dalamnya. Untuk file, akan ditampilkan nama beserta ukurannya. Perintah ini juga akan menampilkan jumlah direktori dan file yang didaftarkan, ukuran kumulatifnya, dan sisa dari *disk* yang tidak terpakai (dalam *bytes*).<sup>2</sup>

Adapun format dari perintah **dir** adalah sebagai berikut.<sup>[3]</sup>

```
dir [<drive:>][<path>][<filename>] [/A[[:]<attributes>]] [/B]
[/C] [/D] [/L] [/N] [/O[[:]<sortorder>]] [/P] [/Q] [/R] [/S]
[/T[[:]<timefield>]] [/W] [/X] [/4]
```

Untuk perintah ini, seperti terlihat di atas, memiliki banyak sekali opsi dan parameter. Tiap-tiap dari parameter tersebut memiliki fungsi tersendiri, yaitu:

– /A[[:]<attributes>]

Menampilkan file-file dengan atribut tertentu, seperti file yang disembunyikan, file sistem, file *read-only*, dan sebagainya.

– /B

Menghilangkan *heading* dan ringkasan informasi dari keluaran, atau dengan kata lain, hanya menampilkan file-file dan direktori, tanpa informasi tambahan apapun.

– /C

Menggunakan separator koma untuk tiap angka ribuan di ukuran file. Jika opsi yang dimasukkan adalah /-C, separator koma justru akan dihilangkan.

– /D

Menampilkan keluaran dengan format yang lebih lebar. Jika opsi ini diikuti, keluaran akan ditampilkan dengan urutan berdasarkan kolom.

– /L

Seluruh teks dalam keluaran akan menggunakan huruf kecil. Jika opsi ini tidak digunakan, keluaran akan mengandung huruf besar dan huruf kecil *mixed case*.

– /N

Menampilkan daftar dengan format panjang, dengan nama file berada di ujung paling kanan.

<sup>2</sup>[dir | Microsoft Docs](#)

1       – /O[:]<sortorder>]

2       Menampilkan daftar direktori yang terurut berdasarkan urutan tertentu, seperti ber-  
3       dasarkan ekstensi file, berdasarkan tanggal dibuat, berdasarkan nama, dan sebagainya.  
4       Jika tidak diikuti tanda minus (-) sebelum huruf O pada perintah, daftar yang muncul  
5       akan terurut secara menaik.

6       – /P

7       Memberhentikan keluaran selama beberapa waktu singkat (memberi jeda kecil) setelah  
8       setiap halaman informasi.

9       – /Q

10       Menambahkan informasi mengenai pemilik file dalam keluaran.

11       – /R

12       Menampilkan *data stream* alternatif, jika ada.

13       – /S

14       Mendaftarkan seluruh file di direktori dan subdirektori yang diproses. Tiap-tiap direktori  
15       akan memiliki *header* tersendiri dalam keluarannya.

16       – /T[:]<timefield>]

17       Menspesifikasi *time field* mana yang akan tampil dan digunakan sebagai urutan, jika  
18       aturan pengurutan lain tidak ditentukan. *Time field* yang dapat digunakan adalah waktu  
19       pembuatan file, kapan terakhir file diakses, dan kapan file terakhir dimodifikasi. Jika  
20       parameter ini tidak dispesifikasi, *time field* yang digunakan adalah kapan file terakhir  
21       dimodifikasi.

22       – /W

23       Menampilkan keluaran dengan format yang lebih lebar. Opsi ini hampir sama dengan /D,  
24       hanya saja untuk /W, jika opsi ini diikuti, keluaran akan ditampilkan dengan urutan  
25       berdasarkan baris, dan bukan kolom.

26       – /X

27       Menampilkan nama pendek yang dibuat untuk nama-nama file non-8.3. Opsi ini memiliki  
28       format tampilan yang sama seperti opsi /N, hanya saja nama pendeknya ditampilkan di  
29       keluaran sebelum nama panjangnya.

30       – 4

31       Menampilkan angka tahun dengan format angka empat digit.

## 32   2.2   KIRI

33   KIRI merupakan sebuah perangkat lunak berbasis web yang berfungsi untuk menyelesaikan (atau  
34   setidaknya mengurangi) dampak dari masalah-masalah yang dapat diselesaikan oleh transportasi  
35   umum/publik di Indonesia, seperti pemanasan global, kemacetan, atau peningkatan harga bensin.  
36   Selain itu, turis mancanegara juga memilih untuk menaiki transportasi umum, karena jenis sarana  
37   transportasi tersebut tidak hanya jauh lebih murah, tetapi juga memberikan kesempatan yang  
38   mudah kepada mereka untuk melihat seluk-beluk dari kota-kota yang mereka kunjungi. Walaupun  
39   begitu, banyak masyarakat lokal sendiri yang seringkali masih segan untuk menaiki transportasi  
40   publik, umumnya karena transportasi publik dianggap lebih rumit persiapannya dibandingkan  
41   dengan metode-metode transportasi privat, seperti menaiki kendaraan pribadi.<sup>3</sup>

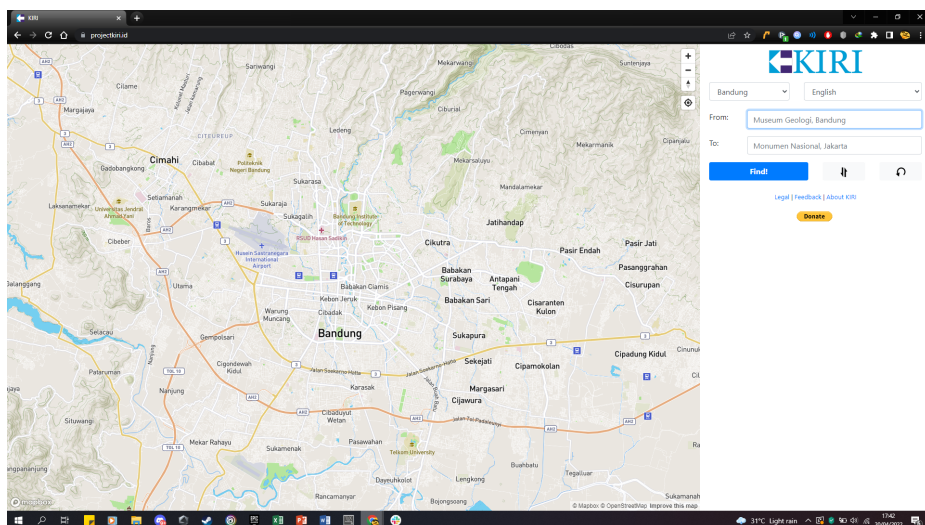
<sup>3</sup><https://projectkiri.github.io/#about-kiri>



Di halaman web KIRI, pengguna dapat memasukkan input berupa lokasi awal dan lokasi tujuan dan KIRI akan menghasilkan seluruh langkah yang harus ditempuh oleh pengguna untuk sampai ke lokasi tujuan, dengan menggunakan angkot. Keluaran ini sudah meliputi kode angkot mana saja yang harus dinaiki, dan juga seberapa jauh pengguna harus berjalan kaki untuk sampai ke lokasi rute angkot berikutnya.

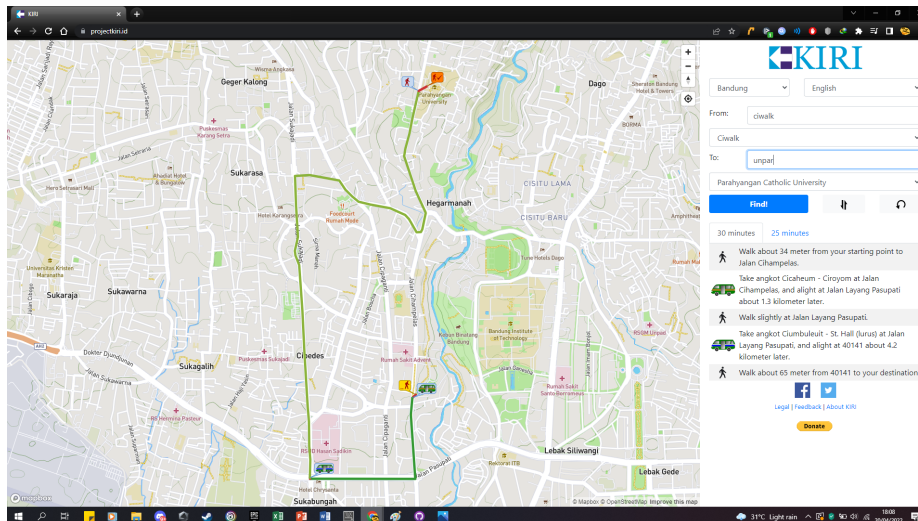
### 2.2.1 Tampilan

Pada saat pertama kali dibuka, hal pertama yang paling mencolok di halaman awal web KIRI adalah sebuah peta besar di sebelah kiri yang dapat diperbesar ataupun diperkecil. Sedangkan, bagian kanan dari halamannya terdiri atas beberapa bagian. Di bagian paling atas terdapat logo KIRI, beserta sepasang menu *dropdown*—yang pertama merupakan pilihan kota tempat pengguna berada (untuk sekarang hanya tersedia pilihan kota Jakarta dan Bandung), dan yang kedua merupakan pilihan bahasa, entah bahasa Indonesia atau Inggris. Di bawahnya merupakan sepasang menu *dropdown* yang merupakan tempat di mana pengguna memasukkan lokasi awal dan tujuan yang akan diproses oleh KIRI. Terakhir, di bawahnya ada sebuah bagian kosong, yang nantinya akan menjadi tempat di mana KIRI akan meletakkan keluaran dari prosesnya. Adapun tampilan awal dari halaman web ini dapat dilihat di gambar 2.4.



Gambar 2.4: Tampilan awal halaman web KIRI.

Ada dua area yang memiliki perbedaan yang signifikan ketika pengguna sudah memasukkan masukan dan menyuruh KIRI untuk memprosesnya. Bagian yang pertama adalah bagian peta, yang setelah pemrosesan masukan, akan memiliki garis-garis berwarna yang menandakan rute angkot maupun tujuan perjalanan kaki yang harus ditempuh oleh pengguna. Bagian kedua adalah bagian keluaran, yang tadinya kosong, sekarang akan berisi langkah-langkah yang harus ditempuh oleh pengguna untuk pergi dari lokasi awal ke lokasi tujuan. Spesifiknya, perbedaan-perbedaan ini dapat dilihat di gambar 2.5.



Gambar 2.5: Tampilan halaman web KIRI setelah pemrosesan masukan dari pengguna selesai.

## 2.2.2 API

KIRI juga memiliki sebuah API yang dapat digunakan untuk keperluan pengembangan perangkat lunak. Seluruh permintaan (*request*) yang dilakukan melalui API KIRI harus dilakukan sebagai permintaan tipe GET ke <https://projectkiri.id/api>, beserta parameter-parameter yang dibutuhkan.

Permintaan tersebut harus memiliki parameter-parameter seperti terlihat di bawah ini.<sup>4</sup>

- **version**

**Kemungkinan nilai:** 2

Parameter ini merupakan tanda bagi API untuk menggunakan protokol versi 2.

- **mode**

**Kemungkinan nilai:** findroute

Parameter ini merupakan mode dari servis/jasa API yang akan digunakan oleh pengguna. Untuk mode **findroute**, jasa yang akan digunakan adalah jasa pencarian rute dengan angkot.

- **locale**

**Kemungkinan nilai:** en atau id

Parameter ini mengatur bahasa apa yang akan digunakan dalam keluaran API nantinya—**en** berarti keluaran akan menggunakan bahasa Inggris, dan **id** berarti keluaran akan menggunakan bahasa Indonesia.

- **start**

**Kemungkinan nilai:** lat, lng; dalam bentuk desimal

Parameter ini merupakan nilai *latitude* dan *longitude* dari titik awal perjalanan pengguna.

- **finish**

**Kemungkinan nilai:** lat, lng; dalam bentuk desimal

Parameter ini merupakan nilai *latitude* dan *longitude* dari titik akhir/tujuan perjalanan pengguna.

<sup>4</sup><https://github.com/projectkiri/Tirtayasa/wiki/KIRI-API-v2>

- **presentation**

**Kemungkinan nilai:** `desktop`

Parameter ini hanya digunakan untuk fitur *backwards compatibility*.

- **apikey**

**Kemungkinan nilai:** angka heksadesimal 16-digit

Parameter ini berisi kunci API pribadi yang harus digenerasi terlebih dahulu sebelum API dapat digunakan.

Sedangkan, respon yang diberikan oleh API berupa sebuah objek JSON yang selalu memiliki setidaknya dua variabel, yaitu:<sup>5</sup>

- **status**

**Kemungkinan nilai:** `ok` atau `error`

Variabel ini manandakan apakah permintaan berhasil diproses atau tidak. Jika permintaan berhasil diproses, variabel ini akan bernilai `ok`, dan jika tidak, variabel ini akan bernilai `error`.

- **message**

Variabel ini bisa berisi dua macam objek. Jika permintaan dari user tidak berhasil diproses, atau dalam kata lain, terjadi sebuah *error*, maka variabel ini akan berisi string yang merupakan pesan *error* serta alasan spesifik mengapa *error* tersebut terjadi. Di lain sisi, jika permintaan dari user berhasil diproses, variabel ini akan mengalami dua perubahan utama. Pertama, nama variabel ini akan berubah menjadi `routingresults`, dan kedua, isi dari variabel ini akan menjadi sebuah *array* JSON yang merupakan respon dari API KIRI berupa keluaran yang akan dilihat oleh pengguna. *Array* JSON ini sendiri terbagi menjadi beberapa variabel lainnya, yang dapat dilihat di daftar di bawah ini.

#### – **steps**

**Tipe:** *array*

Variabel ini merepresentasikan satu buah langkah yang harus ditempuh oleh pengguna. Adapun *array* ini sendiri berisi variabel-variabel berikut:

- \* Tipe transportasi

Tipe sarana transportasi yang harus dipakai oleh pengguna. Jika pengguna harus berjalan kaki, variabel ini akan berisi `walk`. Jika pengguna harus menaiki angkot, variabel ini akan berisi `angkot`.

- \* Kode angkot

Variabel ini menunjukkan angkot mana yang harus dinaiki oleh pengguna di langkah tersebut. Jika penggunaan angkot tidak dimungkinkan pada langkah ini (pengguna harus berjalan kaki), variabel ini akan berisi `walk`.

- \* *Array* `latitude` dan `longitude` lokasi

*Array* nilai-nilai desimal `latitude` dan `longitude` dari berbagai titik lokasi yang terdapat dalam rute.

- \* Deskripsi langkah

Deskripsi langkah yang harus ditempuh, dalam bahasa natural. Bahasa yang digunakan tergantung parameter `locale` yang diatur dalam masukan.

<sup>5</sup><https://github.com/projectkiri/Tirtayasa/wiki/KIRI-API-v2>

1           \* URL untuk mendapatkan tiket kendaraan  
2           Tautan untuk mendapatkan tiket angkutan umum, jika diperlukan. Jika transportasi  
3           pada langkah tersebut tidak memerlukan tiket, variabel ini akan berisi `null`.  
4           \* URL editor rute  
5           Tautan untuk meng-edit rute, jika situasinya memungkinkan. Jika tidak, variabel  
6           ini akan berisi `null`.  
7        – `traveltime`  
8        **Tipe:** string  
9        Variabel ini berisi estimasi jangka waktu yang diperlukan untuk menyelesaikan langkah  
10        tersebut.

## BAB 3

### ANALISIS

#### 3.1 Analisis Aplikasi Sejenis

Untuk pembuatan perangkat lunak dalam skripsi ini, ada dua buah perkakas *command line* yang akan diamati sebagai aplikasi sejenis, yaitu *Chrome Web Store Item Property CLI* dan *iTunes Search API*.

##### 3.1.1 *Chrome Web Store Item Property CLI*<sup>1</sup>

Perkakas *command line* ini merupakan ekstensi dari sebuah aplikasi lain yang memiliki fungsi yang sama, yaitu *Chrome Web Store Item Property*<sup>2</sup>. Perangkat lunak *Chrome Web Store Item Property* ini merupakan perangkat lunak yang akan memanggil fungsi API untuk mendapatkan metadata dari sebuah ekstensi pada *web store* peramban Google Chrome. Perbedaan dari perkakas ini dengan aplikasi dasarnya adalah bahwa perkakas ini dapat digunakan sebagai perkakas *command line*, sedangkan aplikasi dasarnya hanya bisa digunakan dalam perangkat lunak lainnya sebagai pemanggil fungsi API.

##### Penggunaan

Perkakas ini dapat digunakan melalui *command prompt* dengan cara mengetikkan perintah sebagai berikut.

```
chrome-web-store-item-property <identifier>
```

Dengan *identifier* berupa ID dari ekstensi yang diinginkan. Jadi, misalkan pengguna memasukkan `gighmmpiobklfepjocnamgkbiglidom` sebagai ID yang akan digunakan sebagai *identififier*, maka perkakas ini akan mengembalikan metadata dari ekstensi “AdBlock” sebagai keluarannya. Contoh penggunaan perkakas ini dapat dilihat di gambar 3.1.

Sedangkan, keluaran dari perkakas ini merupakan sebuah objek JSON dengan properti-properti sebagai berikut.

- **name**

Nama dari ekstensi yang dicari metadatanya.

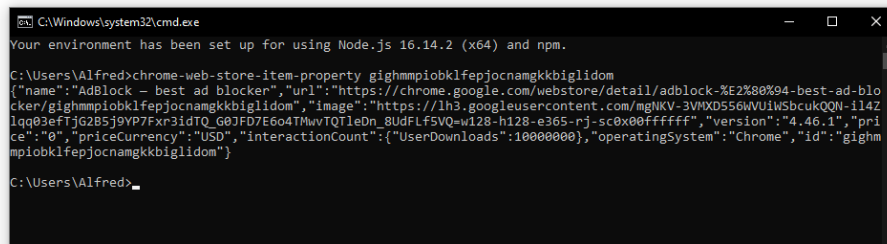
- **url**

URL halaman web dari ekstensi yang dicari di *web store* Google Chrome.

---

<sup>1</sup><https://github.com/pandawing/node-chrome-web-store-item-property-cli>

<sup>2</sup><https://github.com/pandawing/node-chrome-web-store-item-property>



Gambar 3.1: Contoh penggunaan perkakas *Chrome Web Store Item Property CLI*.

- **image**  
Logo (dan ikon *thumbnail*) dari ekstensi yang dicari metadatanya.
- **version**  
Nomor versi dari ekstensi.
- **price**  
Harga dari ekstensi. Jika ekstensi tidak memiliki harga yang perlu dibayarkan (gratis), properti ini akan bernilai 0.
- **priceCurrency**  
Kode mata uang dari harga ekstensi. Jika ekstensi tidak memiliki harga yang perlu dibayarkan, properti ini akan berisi “USD”.
- **interactionCount**  
Properti ini berisi interaksi-interaksi pengguna yang tercatat sebagai data di halaman *web store* ekstensi. Pada saat pembuatan skripsi ini, properti ini hanya memiliki satu buah subproperti, yaitu **userDownloads**, yang menandakan berapa kali ekstensi ini telah diunduh oleh pengguna di manapun.
- **operatingSystems**  
Menandakan di peramban mana ekstensi versi ini dapat diinstal. Karena ekstensi-ekstensinya berada di *web store* Chrome,
- **ratingValue** (tidak digunakan lagi)  
Peringkat yang diberikan oleh para pengguna ekstensi ini. Nilai dari properti ini berupa skala desimal dari 0.00 sampai dengan 5.00. Di versi terbaru dari perkakas ini, properti ini tidak lagi tersedia dalam keluarannya.
- **ratingCount** (tidak digunakan lagi)  
Jumlah pengguna yang telah menilai/memberi peringkat ke ekstensi ini. Di versi terbaru dari perkakas ini, properti ini tidak lagi tersedia dalam keluarannya.
- **id**  
Properti ini mengandung ID dari ekstensi tersebut. Nilai dari properti ini akan sama dengan ID yang digunakan sebagai parameter masukan perkakas.

### 3.1.2 *iTunes Search API*<sup>3</sup>

Perkakas *command line* ini berfungsi untuk melakukan pencarian melalui API iTunes, sehingga seakan-akan pengguna langsung melakukan pencarian di iTunes sendiri. Hasil pencarian yang

<sup>3</sup><https://github.com/awcross/itunes-search-api>

1 dilakukan termasuk judul lagu, nama artis, ataupun nama album, dan pengguna dapat memilih  
2 secara spesifik objek apa yang ingin dicari.

### 3 **Penggunaan**

4 Perkakas ini dapat digunakan melalui *command prompt* dengan cara mengetikkan perintah sebagai  
5 berikut.

```
6             itunes-search-api <input> [<options>]
```

7 Dengan **input** berupa nama dari objek yang dicari. Perkakas ini juga memiliki opsi yang masing-  
8 masing memiliki parameter tersendiri untuk mempersempit hasil pencarian. Adapun opsi-opsi  
9 tersebut dapat dilihat di daftar di bawah ini.

- 10 • **country**

11 **Kemungkinan nilai:** Kode negara dua huruf

12 Opsi ini menerima parameter berupa kode negara asal dari album atau artis yang dicari.

- 13 • **entity**

14 **Kemungkinan nilai:** `song`, `musicArtist`, atau `album`

15 Menandakan jenis objek/entitas yang ingin dicari. Opsi ini dapat bernilai `song` untuk pencarian  
16 berbasis judul lagu, `musicArtist` untuk pencarian nama artis, atau `album` untuk pencarian  
17 nama album. Jika opsi ini tidak dipakai, objek apapun yang memiliki kemiripan dengan  
18 **input** dalam salah satu dari ketiga properti ini akan muncul dalam hasil pencarian.

- 19 • **limit**

20 **Kemungkinan nilai:** Bilangan bulat positif<sup>4</sup>

21 Jumlah hasil pencarian maksimal yang ingin ditampilkan dalam keluaran.

22 Sedangkan, keluaran dari perkakas ini merupakan sebuah objek JSON yang memiliki dua properti  
23 utama, yaitu:

- 24 • **resultCount**

25 Properti ini berisi bilangan bulat yang menandakan berapa buah objek yang terdapat dalam  
26 hasil pencarian.

- 27 • **results**

28 *Array* yang berisi kumpulan objek yang terdapat di dalam hasil pencarian. Objek-objek ini  
29 akan dikembalikan berupa sebuah *array* lain yang berisi seluruh properti dari masing-masing  
30 objek. Apa saja properti yang diikutkan dalam *array* tersebut tergantung tipe dari objek  
31 dalam hasil pencarian.

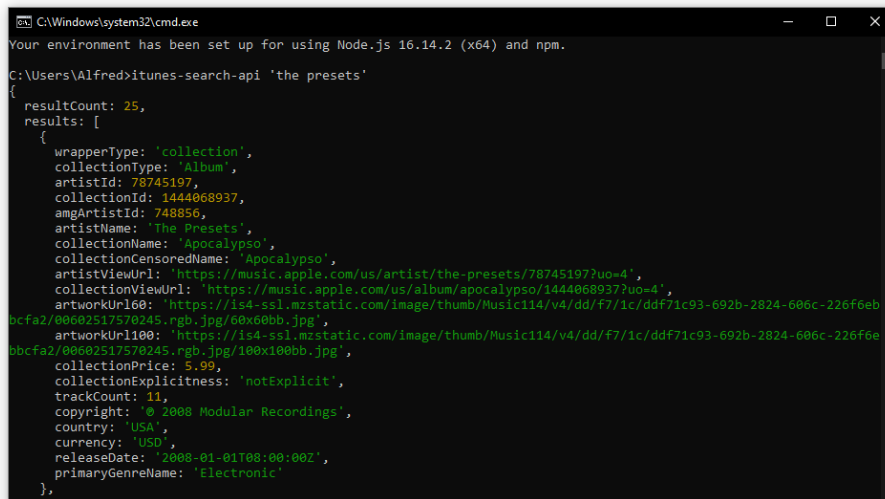
32 Adapun contoh penggunaan dan hasil keluaran perkakas ini dapat dilihat di gambar 3.2.

## 33 **3.2 Analisis Modul dan Fungsi Bahasa C**

34 Di bagian ini akan dilakukan analisis terhadap seluruh modul dan fungsi dalam bahasa pemrograman  
35 C yang akan digunakan dalam pembuatan perkakas ini.

---

<sup>4</sup>Opsi ini juga menerima bilangan bulat negatif, tetapi menggunakan sebuah bilangan bulat negatif akan menghilangkan pengaruh opsi ini terhadap hasil keluaran.



```

C:\Windows\system32\cmd.exe
Your environment has been set up for using Node.js 16.14.2 (x64) and npm.

C:\Users\Alfred>itunes-search-api 'the presets'
{
  resultCount: 25,
  results: [
    {
      wrapperType: 'collection',
      collectionType: 'Album',
      artistId: 78745197,
      collectionId: 1444068937,
      amgArtistId: 748856,
      artistName: 'The Presets',
      collectionName: 'Apocalypse',
      collectionCensoredName: 'Apocalypso',
      artistViewUrl: 'https://music.apple.com/us/artist/the-presets/78745197?uo=4',
      collectionViewUrl: 'https://music.apple.com/us/album/apocalypso/1444068937?uo=4',
      artworkUrl160: 'https://is4-ssl.mzstatic.com/image/thumb/Music114/v4/dd/f7/1c/ddf71c93-692b-2824-606c-226f6ebbcfa2/00602517570245.rgb.jpg/60x60bb.jpg',
      artworkUrl100: 'https://is4-ssl.mzstatic.com/image/thumb/Music114/v4/dd/f7/1c/ddf71c93-692b-2824-606c-226f6ebbcfa2/00602517570245.rgb.jpg/100x100bb.jpg',
      collectionPrice: 5.99,
      collectionExplicitness: 'notExplicit',
      trackCount: 11,
      copyright: '© 2008 Modular Recordings',
      country: 'USA',
      currency: 'USD',
      releaseDate: '2008-01-01T08:00:00Z',
      primaryGenreName: 'Electronic'
    }
  ]
}

```

Gambar 3.2: Contoh penggunaan perkakas *iTunes Search API*. Gambar hanya memuat satu objek untuk menghemat tempat.

### 3.2.1 getopt

**getopt** merupakan sebuah fungsi yang dapat mengautomasi pekerjaan-pekerjaan yang berhubungan dengan penerimaan opsi-opsi untuk *command line* berbasis UNIX.<sup>5</sup>

Fungsi **getopt** dapat dipanggil dengan format sebagai berikut.

```
getopt (argc, argv, <options>)
```

Seluruh kode ini dapat dimasukkan ke suatu variabel berupa sebuah karakter yang merepresentasikan opsi yang ingin digunakan. **argc** merupakan jumlah argumen yang terdapat dalam masukan, sedangkan **argv** merupakan sebuah *array* yang berisi argumen-argumen tersebut. Selain itu, penggunaan **getopt** juga memerlukan penggunaan variabel-variabel tertentu, yang dapat dilihat di daftar berikut.<sup>6</sup>

- **opterr**

Isi dari variabel ini akan memberi signal ke perangkat lunak/perkakas yang menentukan apakah **getopt** akan mengirim pesan ke *error stream* atau tidak. Jika variabel ini bukan bernilai 0, maka pesan *error* akan dikirim. Sebaliknya, jika variabel ini bernilai 0, **getopt** tidak akan mengirim pesan *error* apapun, tetapi tetap akan mengembalikan sebuah karakter tanda tanya (?) sebagai tanda bahwa sebuah *error* telah terjadi.

- **optopt**

Ketika **getopt** menemukan sebuah karakter yang tidak didefinisikan dalam kumpulan opsi, atau sebuah opsi yang tidak disertai argumen yang diperlukan, karakter tersebut akan disimpan di variabel ini.

- **optind**

Variabel ini digunakan oleh **getopt** sebagai indeks untuk *array* **argv**. Jika seluruh argumen

<sup>5</sup>[https://www.gnu.org/software/libc/manual/html\\_node/Getopt.html](https://www.gnu.org/software/libc/manual/html_node/Getopt.html)

<sup>6</sup>[https://www.gnu.org/software/libc/manual/html\\_node/Using-Getopt.html](https://www.gnu.org/software/libc/manual/html_node/Using-Getopt.html)



sudah diproses, nilai variabel ini dapat digunakan untuk menentukan argumen mana yang merupakan argumen tambahan yang tidak terpakai. Nilai dari variabel ini dimulai dari 1.

- **optarg**

Jika opsi yang sedang diproses memerlukan argumen, variabel ini adalah tempat dimana argumen tersebut akan disimpan.

- **<options>**

Variabel ini berupa *string* yang menandakan karakter-karakter apa saja yang menjadi opsi yang mungkin dalam perkakas tersebut, beserta tipenya. Jika karakter opsi:

- Diikuti dengan titik dua (:), maka opsi tersebut memiliki argumen yang bersifat wajib.
- Diikuti dengan titik dua ganda (::), maka opsi tersebut memiliki argumen yang bersifat opsional.
- Tidak diikuti apa-apa, maka opsi tersebut merupakan opsi tidak berargumen.

### **getopt-long**

Ada pula versi **getopt** yang memungkinkan perangkat lunak untuk menerima dua jenis opsi—opsi versi pendek berupa sebuah karakter singular, seperti pada **getopt** biasa, dan/atau opsi panjang bergaya GNU, berupa sebuah kata.

**getopt-long** juga memiliki seluruh variabel-variabel yang dimiliki oleh **getopt**, hanya saja **getopt-long** memiliki sebuah variabel tambahan berupa struktur, yaitu **long\_options**. Variabel ini merupakan sebuah struktur berupa *array* yang berisi beberapa *array* lainnya, di mana *array-array* lain ini merupakan masing-masing opsi dari fungsi **getopt-long** tersebut. Tiap-tiap *array* tersebut memiliki variabel-variabel berikut:

- **name**

Variabel ini merupakan nama panjang dari opsi.

- **has\_arg**

Variabel ini merupakan penanda apakah opsi memerlukan argumen atau tidak. Nilai yang mungkin dalam variabel ini adalah **no\_argument**, **required\_argument**, atau **optional\_argument**.

- **flag & val**

Kedua variabel ini menandakan bagaimana sebuah opsi akan diberlakukan ketika diterima oleh **getopt-long**. Variabel **flag** dapat diisi dengan penunjuk ke suatu variabel lain yang akan diisi dengan isi dari variabel **val** untuk menandakan bahwa **getopt-long** telah berhasil memroses opsi tersebut. Di lain sisi, jika variabel ini berisi *null pointer*, maka fungsi **getopt-long** akan mengembalikan isi dari variabel **val**.

Struktur ini harus diakhiri dengan sebuah *array* tambahan yang seluruh variabelnya bernilai 0.

## **3.3 Analisis API KIRI**



## DAFTAR REFERENSI

- [1] Marsh, N. (2010) *Introduction to the Command Line: The Fat-Free Guide to Unix and Linux Commands*, 2nd edition. CreateSpace, South Carolina.
- [2] Shotts Jr., W. E. (2019) *The Linux Command Line*, 5th internet edition. <https://www.linuxcommand.org/tlcl.php>.
- [3] Mueller, J. P. (2007) *Windows® Administration at the Command Line for Windows Vista™, Windows® 2003, Windows® XP, and Windows® 2000*, 1st edition. Wiley Publishing, Inc., Indiana.
- [4] Neil Matthew, R. S. (2007) *Beginning Linux® Programming*, 4th edition. Wiley Publishing, Inc., Indiana.



# LAMPIRAN A

## KODE PROGRAM

Kode A.1: MyCode.c

```

1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4 #include<stdio.h>
5
6 void myFunction( int input, float* output ) {
7     switch ( array[i] ) {
8         case 1: // This is silly code
9             if ( a >= 0 || b <= 3 && c != x )
10                 *output += 0.005 + 20050;
11             char = 'g';
12             b = 2^n + ~right_size - leftSize * MAX_SIZE;
13             c = (--aaa + &daa) / (bbb++ - ccc % 2 );
14             strcpy(a,"hello_$@?");
15         }
16         count = ~mask | 0x00FF00AA;
17     }
18 }
19
20 // Fonts for Displaying Program Code in LATEX
21 // Adrian P. Robson, nepsweb.co.uk
22 // 8 October 2012
23 // http://nepsweb.co.uk/docs/progfonts.pdf

```

Kode A.2: MyCode.java

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id; //id of the set
8     protected MyEdge FurthestEdge; //the furthest edge
9     protected HashSet<MyVertex> set; //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID; //store the ID of all vertices
12    protected ArrayList<Double> closeDist; //store the distance of all vertices
13    protected int totaltrj; //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35
36 }

```



## LAMPIRAN B

### HASIL EKSPERIMEN

Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4