**TUGAS BESAR 1 IF 2124**

**TEORI BAHASA FORMAL DAN OTOMATA**

***“The Sims Simulator”***

Laporan  
Diajukan untuk Memperoleh Nilai Teori Bahasa Formal dan Otomata

oleh

Samuel 13518041

Kevin Austin Stefano 13518104



TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2019

**PRAKATA**

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa. Atas berkat dan pimpinan-Nya, yang memungkinkan, penulis mampu menyelesaikan penulisan laporan “The Sims Simulator” tepat pada waktunya.

Terima kasih juga kami sampaikan kepada kedua orang tua penulis, ayah dan ibu tercinta. Merekalah yang menginspirasi, membiarkan, dan mendorong kami sebagai penulis untuk menyusun laporan ini.

Ucapan terima kasih teramat khusus penulis sampaikan kepada Ir. Rila Mandala M.Eng.,Ph.D. Sesungguhnya, Bapaklah yang mengajarkan, membimbing, mendorong, dan meneguhkan penulis untuk menulis laporan ini.

Laporan ini penulis susun sebagai syarat untuk memperoleh nilai pada mata kuliah Teori Bahasa Formal dan Otomata. Semoga laporan ini dapat memberikan nilai yang terbaik bagi penulis.

Akhir kata, segala tanggapan (saran, kritik, komentar, dan pernyataan) dari Ibu/Bapak serta kawan-kawan semua, demi pembelajaran yang lebih baik dan benar, penulis terima dengan tulus dan rendah hati karena memang penyempurnaan tidak pernah berhenti dalam dunia ilmu pengetahuan.

Bandung, 27 September 2019

Penulis

**BAB I**

**PENDAHULUAN**

**1.1 Deskripsi Permasalahan**

*The Sims* merupakan permainan yang dibuat oleh *Maxis* dan didistribusikan oleh *Electronic Arts*. *The Sims* memberikan sebuah pengalaman untuk mengatur setiap karakter dalam sebuah kota untuk membangun kota tersebut. Permainan akan dilakukan secara *real time* dan mengharuskan pemainnya untuk benar-benar memperhatikan setiap karakter dalam kota tersebut, dan bisa memilih *goal-goal* yang ingin dicapai. Tugas besar ini adalah simplifikasi dari permainan *The Sims*. Setiap pemain akan mengatur kegiatan seseorang dalam satu hari. Pemain dapat memilih satu diantara banyak aksi yang dapat dilakukan, dimana tiap aksi yang dipilih memiliki konsekuensi baik positif maupun negatif. Seperti layaknya manusia, aktor pada *The Sims* dapat mengalami perubahan kondisi, seperti lapar, bosan, lelah, dan lainnya. Kondisi-kondisi tersebut digambarkan menjadi beberapa atribut. Terdapat beberapa atribut dalam permainan *The Sims* yang dapat mempengaruhi keberjalanan permainan. Atribut tersebut adalah sebagai berikut.

1. *Hygiene*, merupakan tingkat kebersihan dari pemain
2. *Bladder*, merupakan tingkat keinginan pemain untuk membuang air besar atau air kecil.
3. *Hunger*, merupakan tingkat kelaparan pemain.
4. *Energy*, merupakan tingkat energi yang dimiliki pemain.
5. *Social*, merupakan tingkat kesosialan yang dimiliki pemain.
6. *Fun*, merupakan tingkat kesenangan dari pemain.

Untuk mempermudah tugas besar ini, **atribut-atribut tersebut dimodifikasi** sebagai berikut:

1. ***Hygiene***, gabungan dari atribut *Hygiene* dan *Bladder* dalam *The Sims*.
2. ***Energy***, gabungan dari atribut *Energy* dan *Hunger* dalam *The Sims*.
3. ***Fun***, gabungan dari atribut Fun dan Social dalam The Sims.

Setiap atribut memiliki nilai maksimum 15 dan nilai minimum 0. Setiap atribut dapat bertambah (sampai nilai maksimum) atau berkurang (sampai nilai minimum)  jika pemain melakukan aksi tertentu. Untuk mempermudah tugas besar ini, elemen waktu tidak diperhitungkan dalam simulasi ini. **Kondisi awal pemain selalu dalam keadaan sudah bangun tidur dengan atribut *Hygiene* bernilai 0, *Energy* bernilai 10, dan *Fun* bernilai 0. Permainan dinyatakan selesai jika semua atribut bernilai 0 atau semua atribut bernilai 15.**

Berikut ini adalah tabel aksi yang dapat dilakukan, rentang waktu untuk melakukan aksi tersebut, dan konsekuensinya:

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Aksi** | **Jenis** | **Konsekuensi** |
| 1 | Tidur <jenis\_tidur> | Siang | +10 *energy* |
| Malam | +15 *energy* |
| 2 | Makan <jenis\_makanan> | * *Hamburger* * *Pizza* * *Steak and Beans* | *Hamburger*:  +5 *energy*  *Pizza:*  +10 *energy*  *Steak and Beans*:  +15 *energy* |
| 3 | Minum <jenis\_minuman> | * Air * Kopi * Jus | Air:  *-*5 *hygiene*  Kopi:  +5 *energy*  -10 *hygiene*  Jus:  +10 *energy*  -5 *hygiene* |
| 4 | Buang Air <jenis> | Kecil | *+*5 *hygiene* |
| Besar | +10 *hygiene*  -5 *energy* |
| 5 | Bersosialisasi ke Kafe | - | +15 *fun*  *-*10 *energy*  *-*5 *hygiene* |
| 6 | Bermain Media Sosial | - | +10 *fun*  *-*10 *energy* |
| 7 | Bermain komputer | - | +15 *fun*  *-*10 *energy* |
| 8 | Mandi | - | +15 *hygiene*  *-*5 *energy* |
| 9 | Cuci Tangan | - | +5 *hygiene* |
| 10 | Mendengarkan Musik di Radio | - | *+*10 *fun*  *-*5 *energy* |
| 11 | Membaca <jenis\_bacaan> | Jenis Bacaan:   * Koran * Novel | Koran:  *+*5 *fun*  *-*5 *energy*  Novel:  *+*10 *fun*  *-*5 *energy* |

|  |
| --- |
| Hygiene = <Total\_Hygiene>  Energy = <Total\_Energy>  Fun = <Total\_Fun> |

Program yang dibuat tidak memerlukan GUI, sehingga masukan hanya berasal dari *keyboard* di *command line*. Masukan berupa satu aksi. **Jika sebuah aksi menambah atau mengurangi satu atau lebih atribut sehingga bernilai lebih kecil dari 0 atau lebih besar dari 15 merupakan aksi yang tidak valid dan mengeluarkan tulisan “Aksi tidak valid”.** Aksi yang valid akan mengeluarkan tulisan yang berisi nilai dari ketiga atribut dengan *format* sebagai berikut:

Untuk mempermudah pemahaman, diberikan contoh konteks dan dua contoh masukan beserta keluarannya sebagai berikut:

**Contoh konteks**

|  |
| --- |
| Hygiene = 10  Energy = 15  Fun = 15 |

**Contoh Masukan 1**

|  |
| --- |
| Cuci Tangan |

**Keluaran 1**

|  |
| --- |
| Hygiene = 15  Energy = 15  Fun = 15 |

“Cuci Tangan” merupakan aksi yang valid karena menambah *Hygiene* sebanyak 5 sehingga total *Hygiene*-nya 15, seperti yang ditunjukkan pada bagian yang berwarna merah.

**Contoh Masukan 2**

|  |
| --- |
| Makan Hamburger |

**Keluaran 2**

|  |
| --- |
| Aksi tidak valid |

“Makan Hamburger” merupakan aksi yang valid, tetapi aksi ini menambah *Energy* sebanyak 5 sehingga total *Energy*-nya 20. Berarti aksi ini menghasilkan total *Energy* yang berada di luar batas permainan sehingga aksinya tidak valid dan mengeluarkan tulisan, “Aksi tidak valid”.

**BAB II**

**LANDASAN TEORI**

**2.1 Definisi Finite State**

Finite state automata adalah suatu sistem permodelan matematika dengan mesin abstrak yang mampu menerima *input* dan mengeluarkan *output* diskrit serta dapat mengenali bahasa paling sederhana (bahasa reguler) dan dapat diimplementasikan secara nyata.

Finite State Automata (FSA) adalah model yang menerima masukan dan keluaran dalam suatu suatu state atau keadaan dimana state yang satu dapat berpindah ke state yang lainnya dengan fungsi transisi. Finite state automata tidak memiliki tempat penyimpanan/memory, hanya bisa mengingat state terkini.

**2.2 Ciri dan Karakteristik Finite Automata**

Adapun karakteristik dari mesin abstrak Finite Automata adalah

1. Finite Otomata memiliki keadaan atau state dan transisinya
2. Transisi atau perpindahan dari state satu ke state lainnya bersifat deterministik atau non-deterministik.
3. Finite Otomata memiliki satu state awal dan memiliki satu atau beberapa state akhir. State ini menggambarkan keadaan dimana perubahan setiap state akan dilakukan setiap kali pembacaan input karakter.

Sehingga bisa disimpulkan setiap Finite Automata pasti memiliki

1. Kumpulan state dimana state awal dan state akhir merupakan bagian di dalamnya.
2. Simbol masukan
3. Fungsi transisi

Finite State Automata dinyatakan oleh pasangan 5 bagian, yaitu:  
M=(Q , Σ , δ , Q0 , F )  
Q = himpunan state  
Σ = himpunan simbol input  
δ = fungsi transisi δ : Q × Σ  
Q0 = state awal / initial state , Q0 ∈ Q  
F = state akhir, F ⊆ Q

**2.3 Cara Kerja Finite State Automata**

Suatu sistem Finite State Automata bekerja dengan beberapa langkah dalam setiap state yang dilaluinya. Pertama mesin abstrak ini akan membaca masukan input user dari kiri ke kanan dan membacanya satu persatu. Karena pembacaan dari input terus dilakukan, maka prekondisi yang harus dipenuhi oleh user atau penggunanya dan perubahan keadaan atau kondisi *state* akan dilakukan ketika mesin membaca karakter selanjutnya. Bahasa atau *language* bisa dikatakan valid jika pembacaan karakter tepat berhenti pada akhir state atau final state atau dapat dikatakan masukan user diterima oleh Finite Automata.

**2.4 Jenis-Jenis FSA**  
Adapun dua jenis FSA adalah sebagai berikut :

**1. Non-deterministic Finite Automata (NFA)** :

NFA adalah otomata berhingga, namun sifatnya tidak pasti seperti DFA. NFA bisa dikatakan memiliki sifat tidak pasti untuk setiap *state* input dikarenakan masukan dari setiap state bisa berbagai macam. Tidak seperti DFA yang hanya memiliki satu masukan dari user, NFA bisa memiliki pilihan masukan mulai dari 0 (Epsilon) hingga pilihan lebih dari satu untuk tiap statenya. Sehingga bisa dikatakan, dari state yang ada bisa tedapat keluaran epsilon, satu, atau banyak dengan input atau masukan yang sama.

Pada setiap NFA harus dicoba semua kemungkinan yang ada hingga mesin mendapatkan satu yang mencapai state akhir. Hal ini tentunya lebih mmakan waktu yang lebih lama dibandingkan dengan DFA dan memberatkan mesin pengerja.

**2. Deterministic Finite Automata (DFA)** :

DFA merupakan teori komputasi yang mana merupakan *Finite-state Machine* atau mesin dengan *state* terhingga yang menerima atau menolak *string* dari simbol

Suatu otomata bisa kita katakan sebagai suatu DFA (*Deterministic Finite Automata*) jika FSA (*finite state automata*) tersebut memiliki keadaan atau *state* yang menerima tepat satu *state* untuk tiap masukan. DFA bisa digambarkan dengan tiga cara, yaitu Transition diagram, tabel tansisi, dan notasi matematis.

Untuk membuat suatu DFA, langkah yang harus kita lakukan adalah dengan mendefinisikan semua keadaan atau *state* yang akan dilalui oleh sebuah string. Adapun ketentuan yang harus dipenuhi adalah, tidak boleh ada *state* yang tertinggal dan satu keadaan harus tepat mengandung 1 state juga. Tidak boleh ada keadaan atau *state* yang beririsan dengan *state* lainnya.

**BAB III**

**ANALISIS PERSOALAN**

**3.1 Tabel Transisi**

Tabel transisi DFA yang dibuat dilampirkan bersama laporan ini.

**3.2 Langkah Implementasi Pembuatan State ke Tabel Transisi**

Adapun data-data yang berhasil didapat, dilakukan dengan langkah sebagai berikut.

1. Menentukan jumlah *state* DFA yang mungkin terjadi. Ada tiga jenis atribut, yaitu *Hygiene*, *Energy*, dan *Fun*. Setiap atribut memiliki empat jenis nilai yang mungkin, yaitu 0, 5, 10, dan 15. Oleh karena kedua hal tersebut, jumlah *state* yang mungkin adalah , yaitu 64.
2. Melakukan perpindahan *state* untuk setiap jenis masukan aksi yang berbeda berdasarkan hasil akhir dari nilai setiap atribut.
3. Mengeliminasi perpindahan *state* yang tidak valid, yaitu jika hasil akhir salah satu nilai atribut setelah aksi tersebut lebih dari 15 atau kurang dari 0. Perpindahan seperti itu tidak valid sehingga masukan aksi tersebut akan dianulir dan tidak menyebabkan perpindahan *state*.

**3.3 Catatan dalam Pembuatan State**

Dalam pembuatan *state* DFA, ada beberapa catatan yang harus dipatuhi. Adapun *state* yang valid adalah *state* yang memenuhi dua hal berikut. DFA dikatakan memiliki *state* yang valid berdasarkan analisis kami adalah jika tidak ada *state* yang tertinggal dan satu keadaan harus tepat mengandung 1 *state* juga. Hal selanjutnya yang harus diingat berdasarkan analisis kami adalah tidak boleh ada keadaan atau *state* yang beririsan dengan *state* lainnya.

**3.4 Contoh DFA Valid**

Berdasarkan tabel transisi yang telah kami buat, maka kami mendapatkan terdapat dua *final state* dalam DFA kami. Final state yang pertama terjadi ketika Hygiene, Energy,danFunbernilai 0 semua. Keadaan ini berarti karakter mati. Final state yang kedua terjadi ketika Hygiene, Energy,danFun bernilai 15. Hal ini menandakan karakter sudah terpuaskan. Selain itu, saat salah satu atribut sudah berada di 0 ataupun 15, aksi yang mampu menambah atribut tersebut menjadi tidak valid dan pengguna harus memasukkan aksi jenis lain yang valid, yaitu aksi yang mampu mempengaruhi atribut lain yang tidak 0 ataupun 15 atau aksi yang mampu menambah atribut yang 0 tanpa menambah atribut yang bernilai 15 atau aksi yang mampu mengurangi atribut yang bernilai 15 tanpa mengurangi atribut yang bernilai 0. Dengan begitu, simulasi ini bisa berlanjut ke *state* berikutnya.

Adapun contoh *state* valid yang bisa ditempuh dalam program ini adalah --Masukan dengan final state jenis pertama

* 1. (*q8,* Buang Air Besar) = *q36*,
  2. (*q36,* Minum Air) = *q20*,
  3. (*q20,* Minum Air) = *q4*,
  4. (*q4,* Buang Air Besar) = *q32*,
  5. (*q32,* Minum Air) = *q16*,
  6. (*q16,* Minum Air) = *q0* (Final state)

Masukkan dengan final state jenis kedua

* 1. (*q8,* Cuci Tangan) = *q24*,
  2. (*q24,* Cuci Tangan) = *q40*,
  3. (*q40,* Cuci Tangan) = *q56*,
  4. (*q56,* Bermain Komputer) = *q51*,
  5. (*q51,* Makan Hamburger) = *q55*,
  6. (*q55,* Makan Hamburger) = *q59*,
  7. (*q59,* Makan Hamburger) = *q63* (Final state)

Masukan dengan input tidak valid

* 1. (*q8,* Minum Air) = *q8*,
  2. (*q8,* Tidur Malam) = *q8*,
  3. (*q8,* Mandi) = *q52*,
  4. (*q52,* Makan Hamburger) = *q56*,
  5. (*q56,* Makan Hamburger) = *q60*,
  6. (*q60,* Buang Air Besar) = *q60*,
  7. (*q60,* Bermain Komputer) = *q55*
  8. (*q55,* Makan Pizza) = *q63* (Final state)

Berdasarkan contoh-contoh di atas, dapat dilihat bahwa DFA ini dimulai dari *start state*, yaitu *q8*, di mana karakter saat itu baru bangun tidur dan nilai atributnya adalah 0 untuk Hygiene, 10 untuk Energy, dan 0 untuk Fun. Kemudian, program akan membaca input selanjutnya berupa *string* yang berisi aksi yang akan dilakukan. Jika *string* yang dimasukkan berupa aksi yang valid, seperti yang sudah dijelaskan di atas, *state* akan berpindah ke *state* selanjutnya yang menggambarkan nilai atribut setelah perubahan. Namun, jika aksi yang dila-kukan tidak valid, seperti yang sudah dijelaskan di atas juga, tidak ada perpindahan *state* yang terjadi sehingga akan tetap berada di *state* itu sendiri dan tetap meminta masukan dari pengguna hingga *final state* terpenuhi. Jadi, jika ada masukan *string* yang tidak diterima, hal ini tidak mengindikasikan bahwa program akan berhenti bekerja dan bahasa ditolak. Namun, yang terjadi adalah tetap meminta masukan lagi dari pengguna karena cara untuk menghentikan program ialah dengan menyelesaikan simulasi tersebut dengan mencapai *final state* seperti yang sudah dijelaskan di atas. Kemungkinan lain ialah dengan menekan tombol Alt + F4 atau Ctrl + C (Windows) atau Ctrl + Z (Linux) di *command prompt* atau *terminal*. Namun, hal tersebut sudah cukup jelas (*obvious*) sehingga seharusnya tidak perlu dijelaskan lagi.

Representasi DFA menggunakan tabel transisi ini sudah mencakup semua *state* yang mungkin dari simulasi ini dan juga tidak ada irisan antara sebuah *state* dengan *state* lainnya. Dengan begitu, DFA ini sudah dibuat dengan konsep yang benar, bukan dengan prinsip *trial and error* yang sangat tidak baik untuk dilakukan.

**BAB IV**

**IMPLEMENTASI DAN PENGUJIAN**

**4.1 Spesifikasi Teknis Program**

Adapun dalam pembuatan program ini kami mebuat beberapa fungsi untuk dijalankan dalam *main program* kami. Pertama kami mebuat tipe bentukan bernama *state* yang terdiri dari Hygiene, Energy, dan Fun*.*

Type State <

Hygiene : **Integer**

Energy : **Integer**

Fun : **Integer**

>

Kami membuat beberapa fungsi-fungsi yang menunjang *main program* kami. Adapun fungsi fungsi kami adalah sebagai berikut

|  |
| --- |
| **function** isDead (x : State) -> **boolean** |
| /\* Apakah karakter sudah mati, yaitu semua state bernilai 0 \*/ |
|  |
| **function** isFull (x : State) -> **boolean** |
| /\* Apakah kaarakter sudah penuh maksimal, yaitu semua state sudah 15 \*/ |

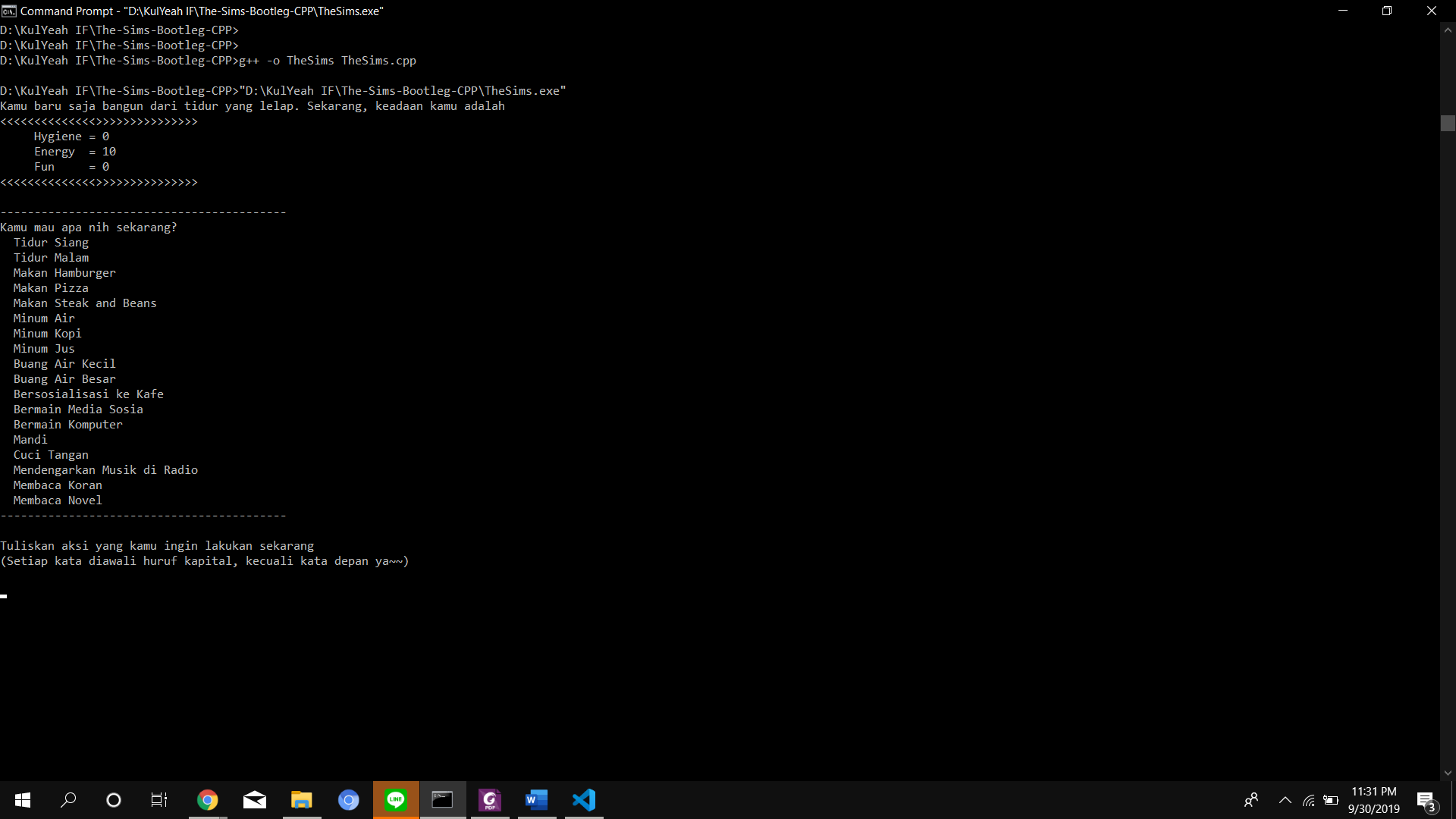
|  |
| --- |
| **function** isStateValid (x : State) -> **boolean** |
| /\* Apakah state karakter saat ini valid \*/ |
|  |
| **procedure** changeState (currentstate : State, statechange : State); |
| /\* Mengubah current state dengan melakukan operasi dengan statechange jika valid. |
| Jika tidak, diberi pesan error \*/ |
|  |
| **procedure** printState (x : State) |
| /\* Mencetak ke layar state saat ini \*/ |
|  |
| **function** adjustState (x : State, y : State) -> **State** |
| /\* Melakukan penjumlahan dua buah state \*/ |
|  |
| **function** Tidur (c : char) -> **State** |
| /\* Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika melakukan tidur. \*/ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |
| **function** Makan (c : char) -> **State** |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika melakukan makan } |

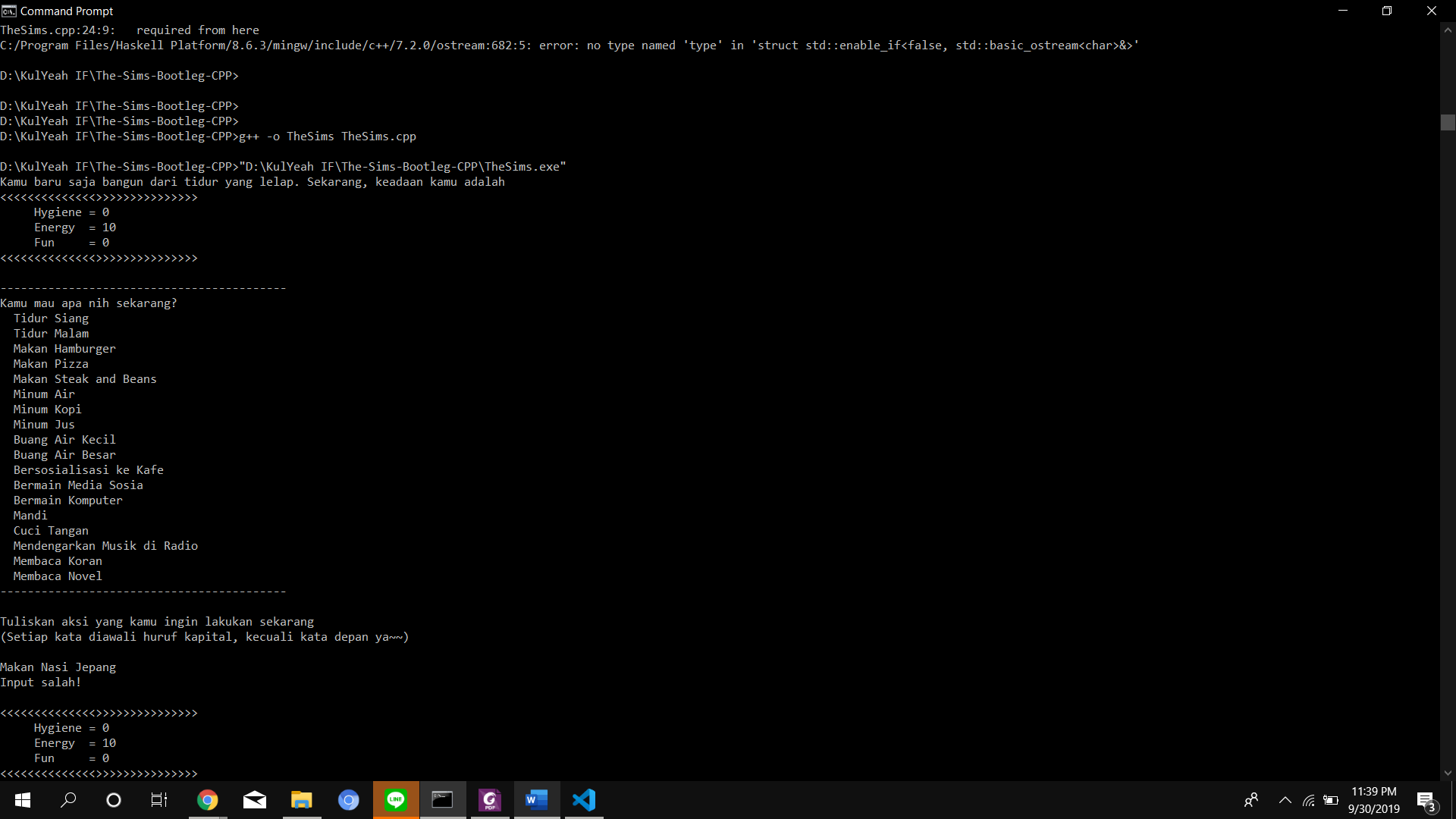
|  |
| --- |
| **function** Minum (c : char) -> **State** |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika melakukan minum } |
|  |
| **function** BuangAir (c : char) -> **State** |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika melakukan buang air } |
|  |
| **function** PergiKeKafe ()->**State** |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika melakukan pergi ke kafe } |
|  |
| **function** MainMedsos ()->**State** |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika melakukan main media sosial } |
|  |
| **function** MainKomputer ()->**State** |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika melakukan main komputer } |
|  |
| **function** Mandi ()-> **State** |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika melakukan mandi } |
|  |
| **function** CuciTangan ()-> **State** |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika mencuci tangan } |
|  |
| **function** DengarRadio ()-> **State** |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika mendengar radio } |
|  |
| **function** Membaca (c : char) -> State |
| { Fungsi untuk menghasilkan sebuah state yang berisi perubahan yang dialami jika melakukan membaca } |
|  |
|  |

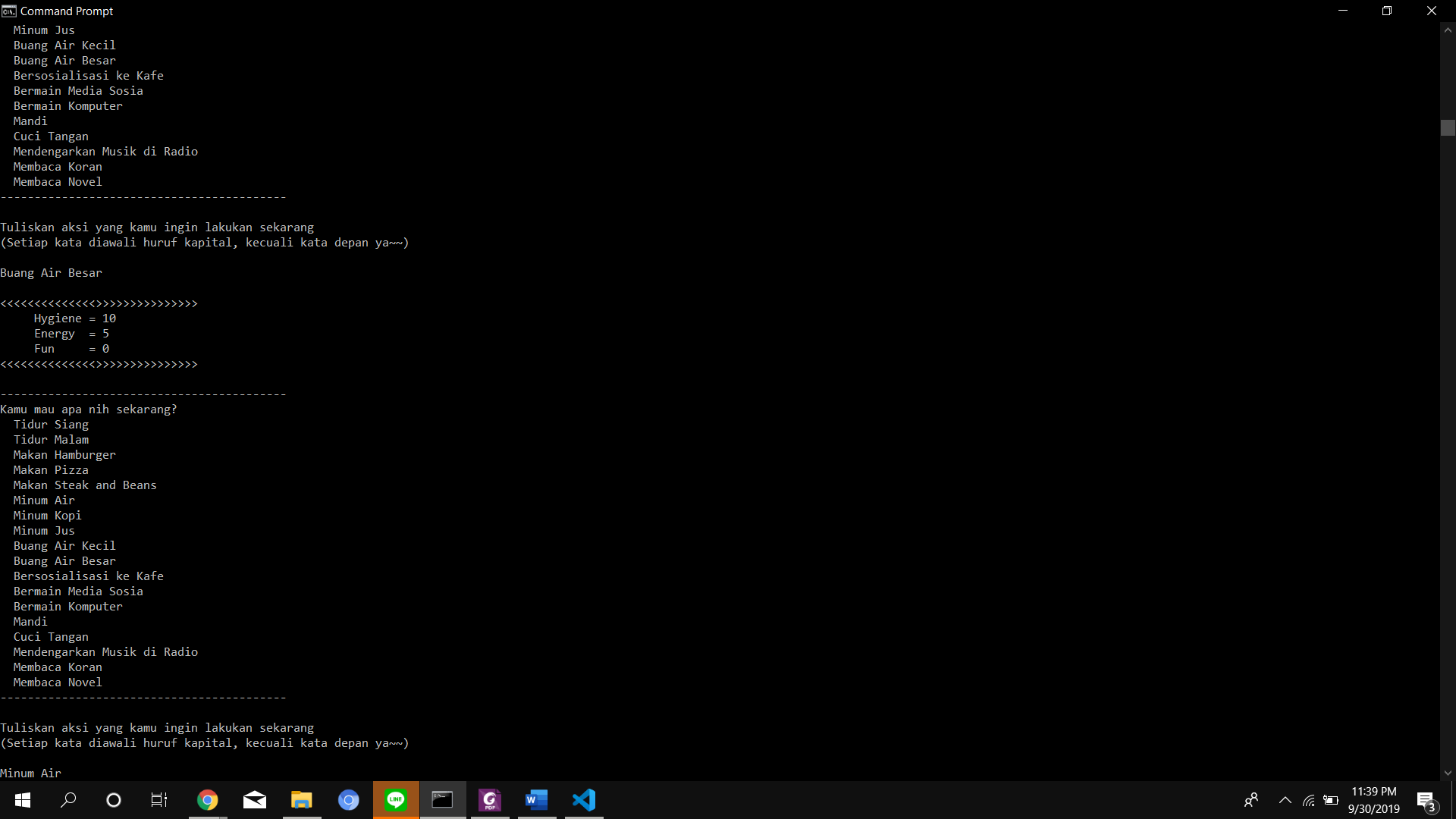
**4.2 Kasus Uji**

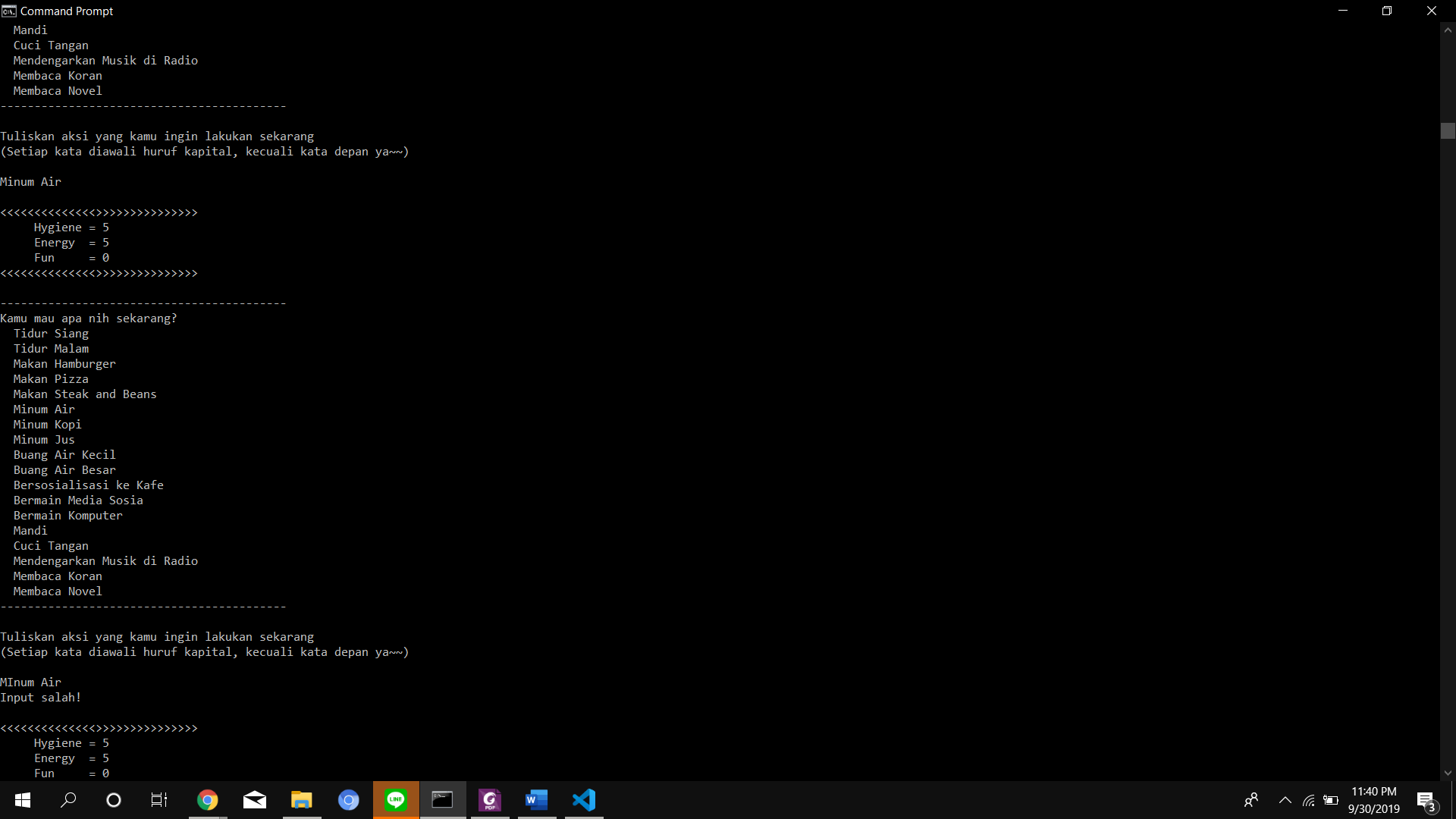
Tampilan awal dari program kami adalah sebagai berikut

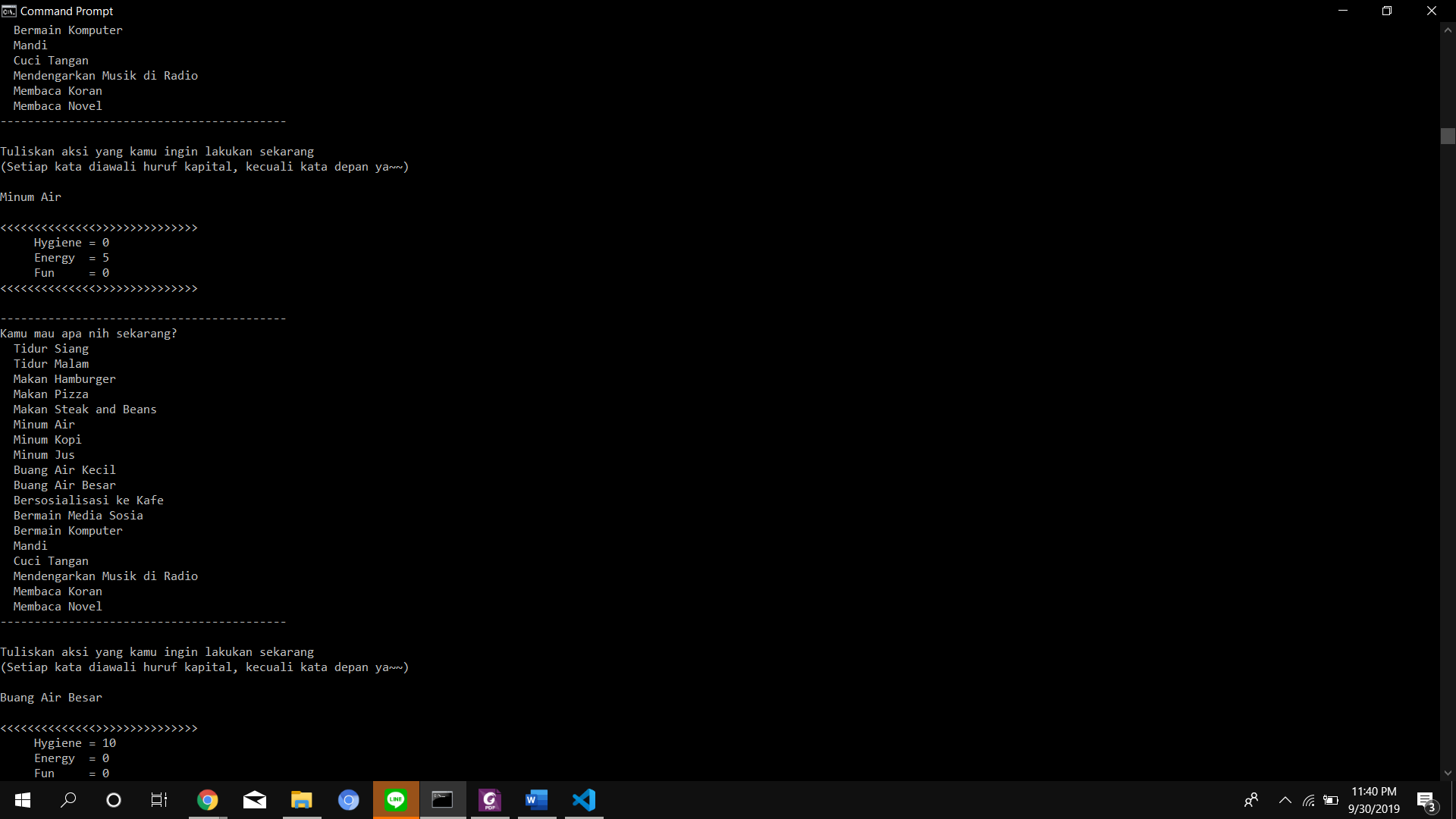


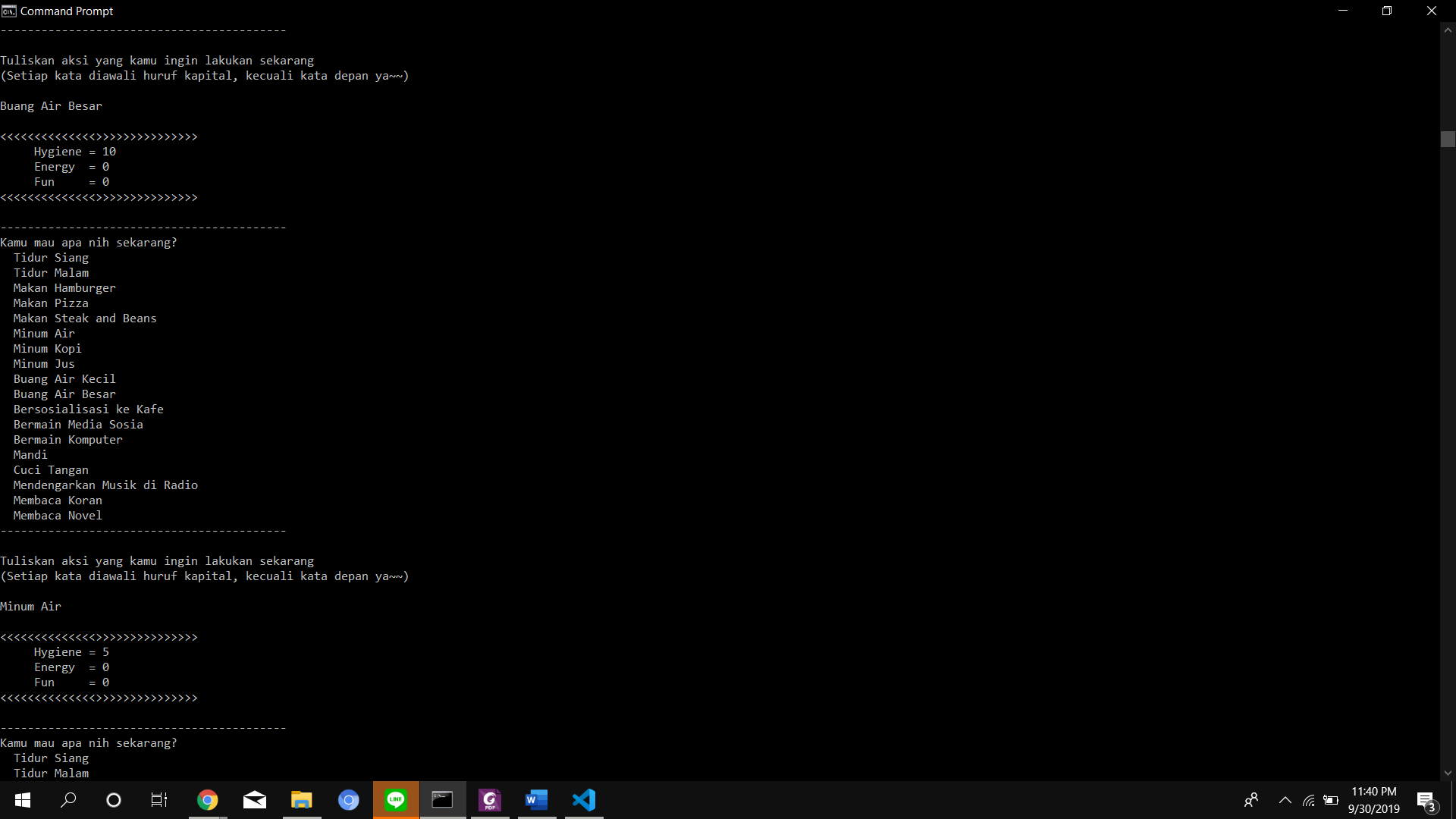
Kasus uji pertama adalah kasus ketika final *state* jenis pertama tercapai, yaitu ketika mati. Hal ini disebabkan ketika Hygiene, Energy, dan Funketiganya bernilai 0.

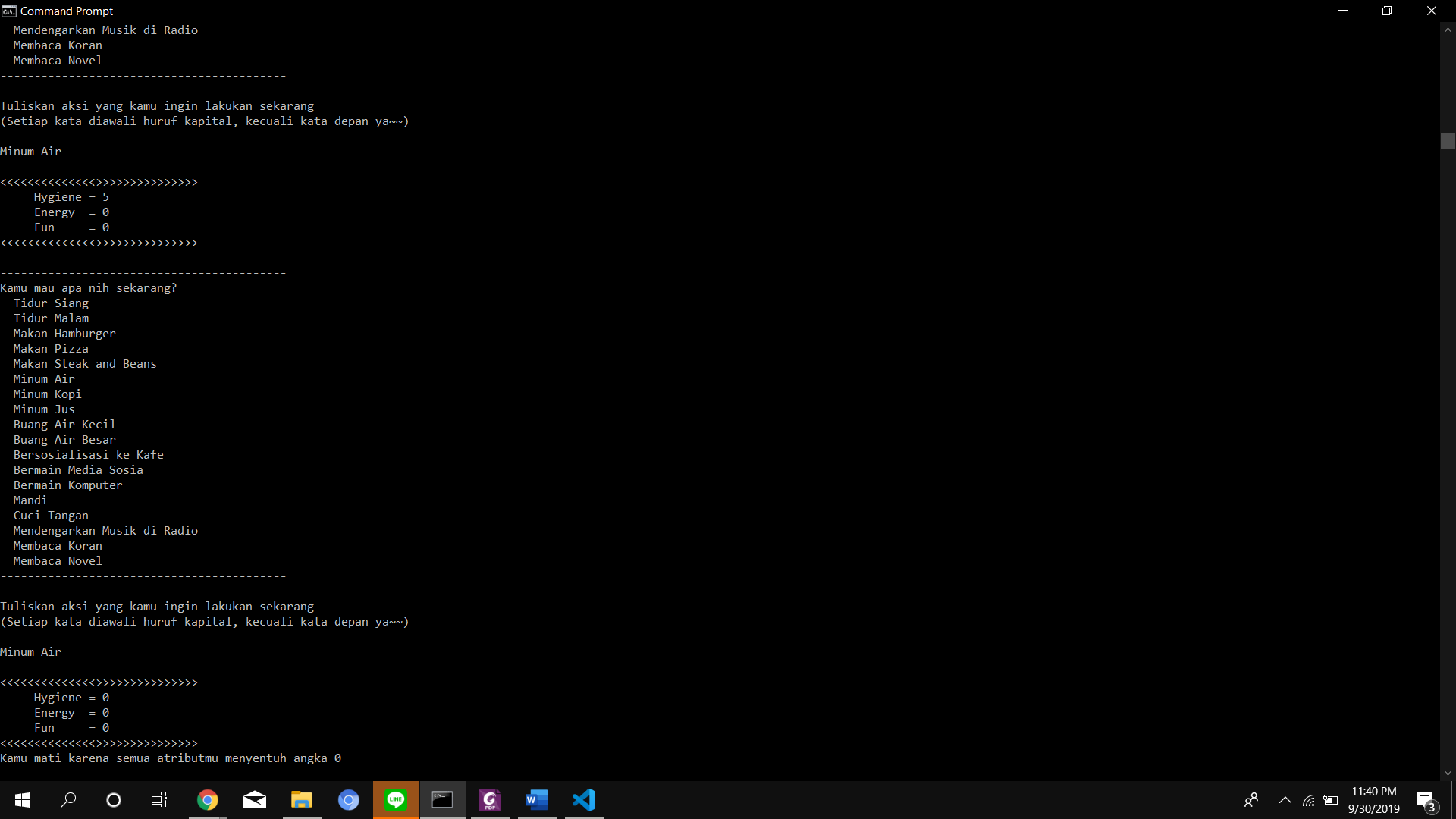


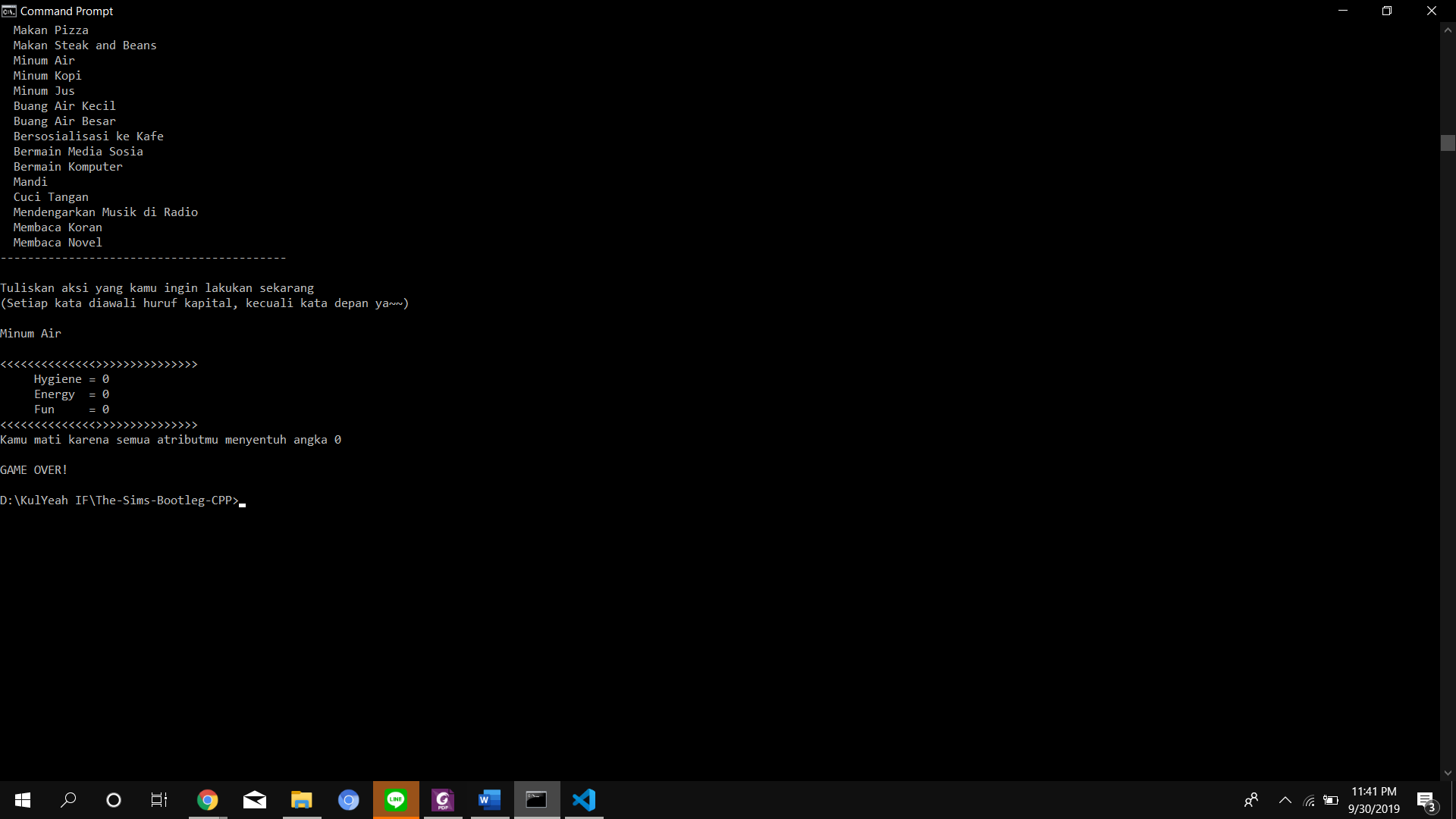






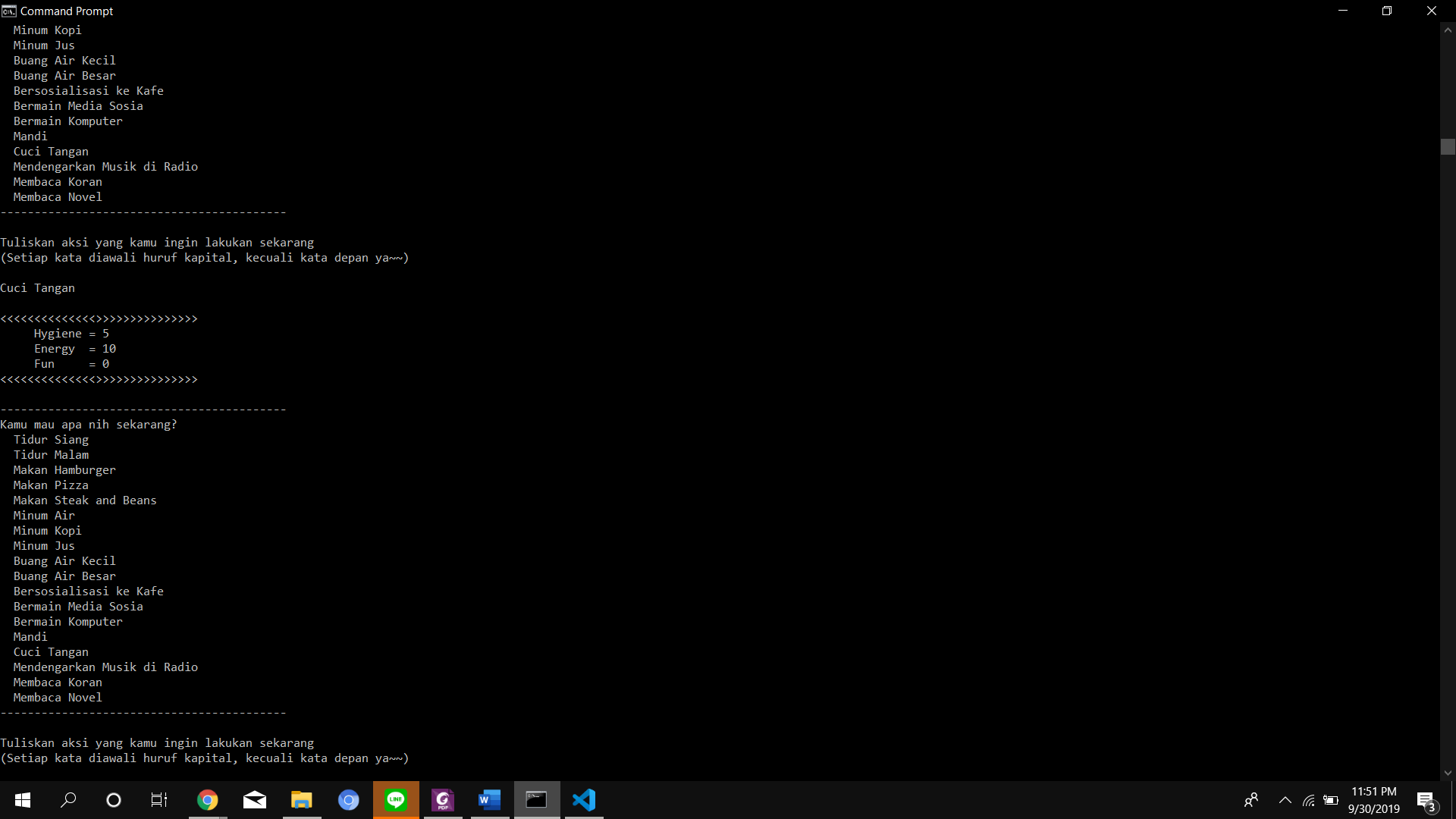
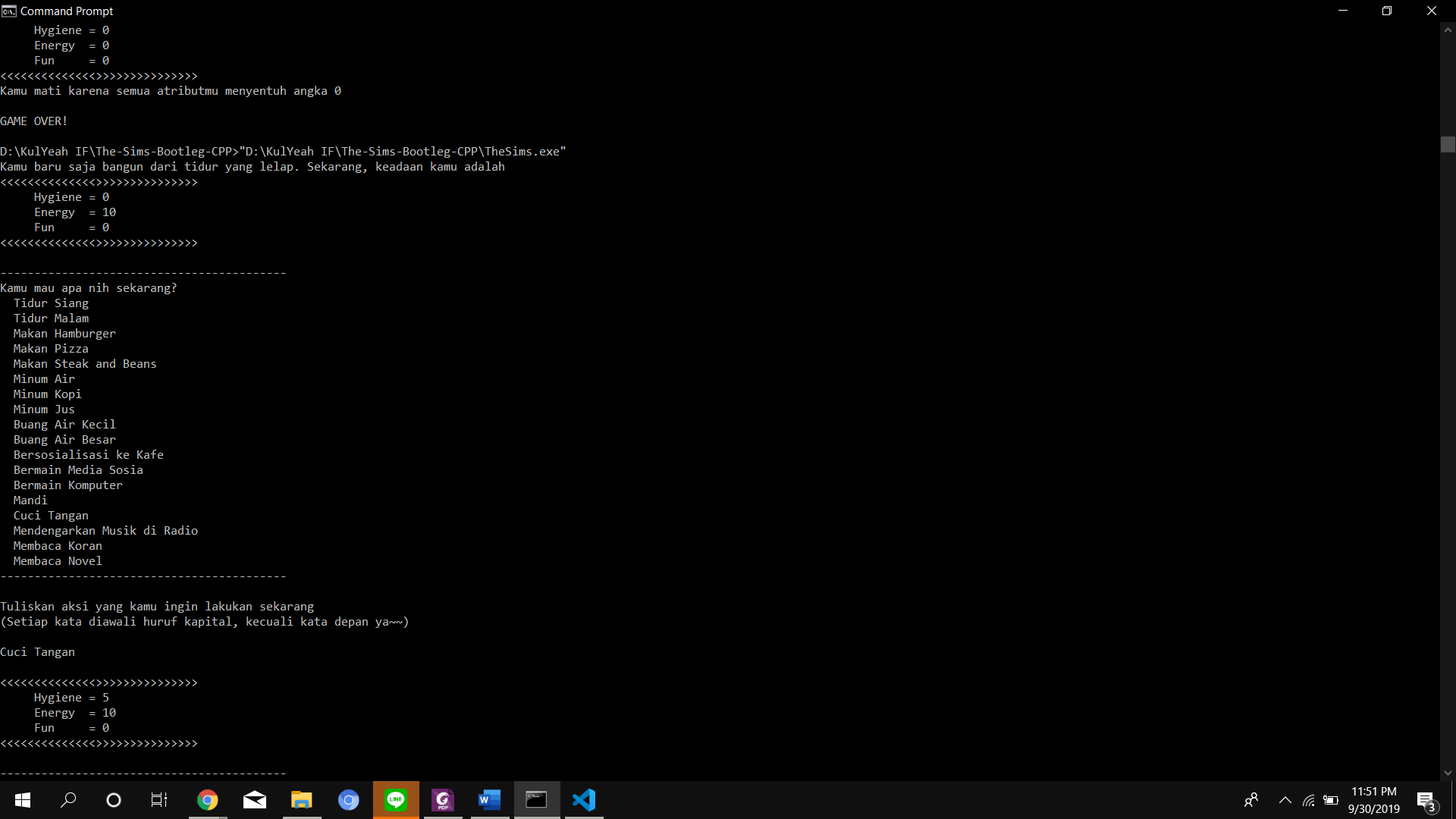


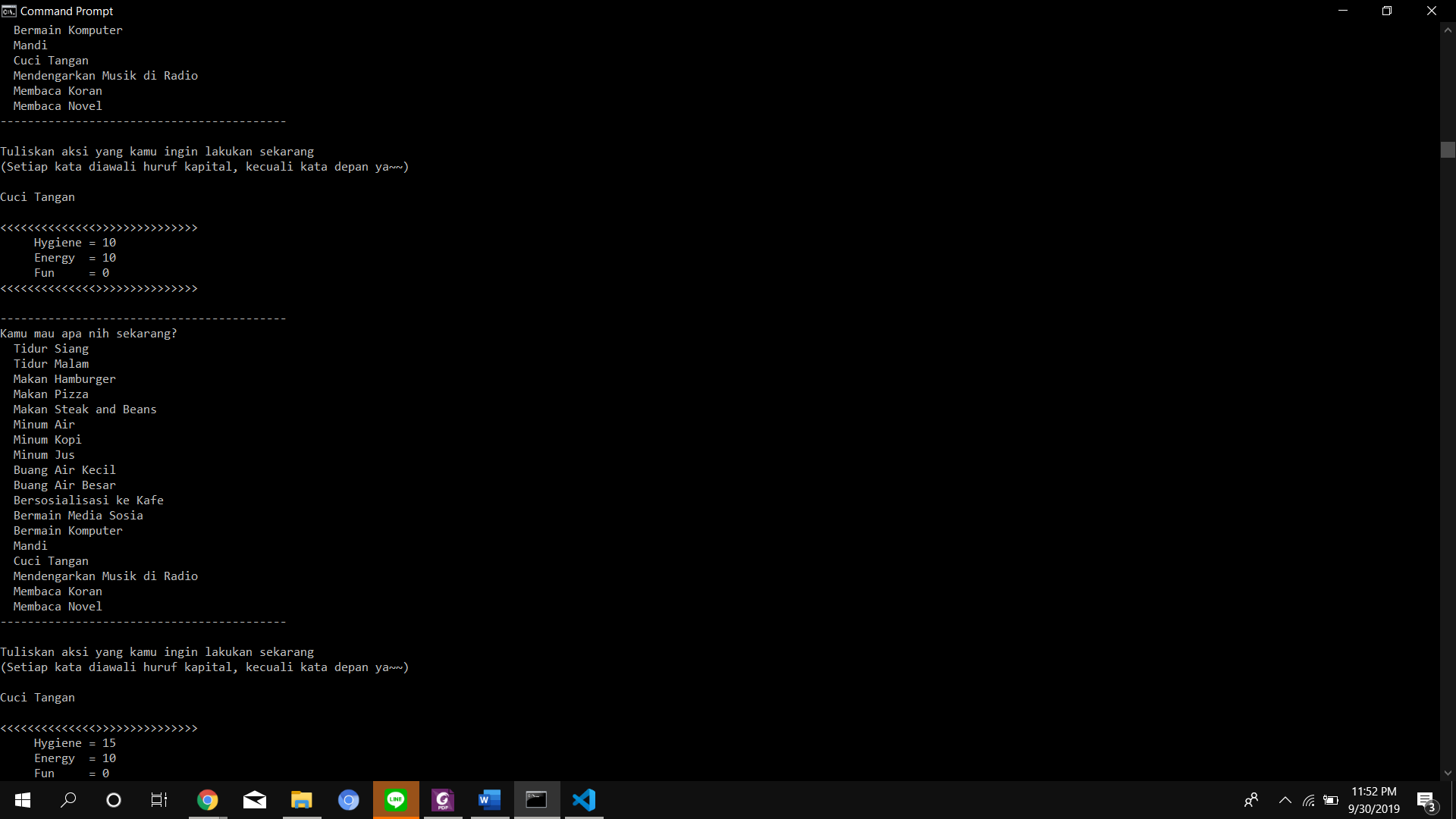


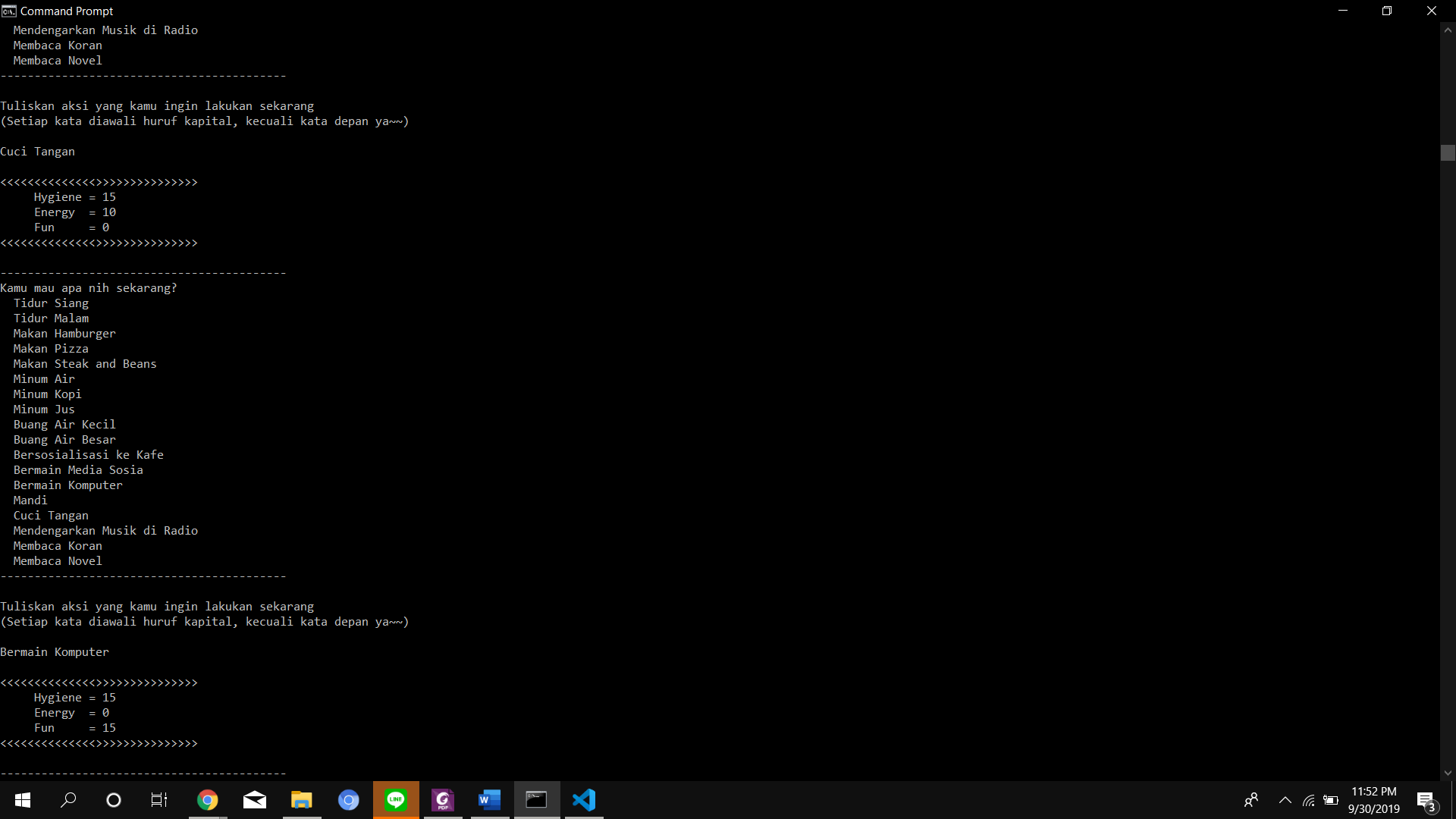


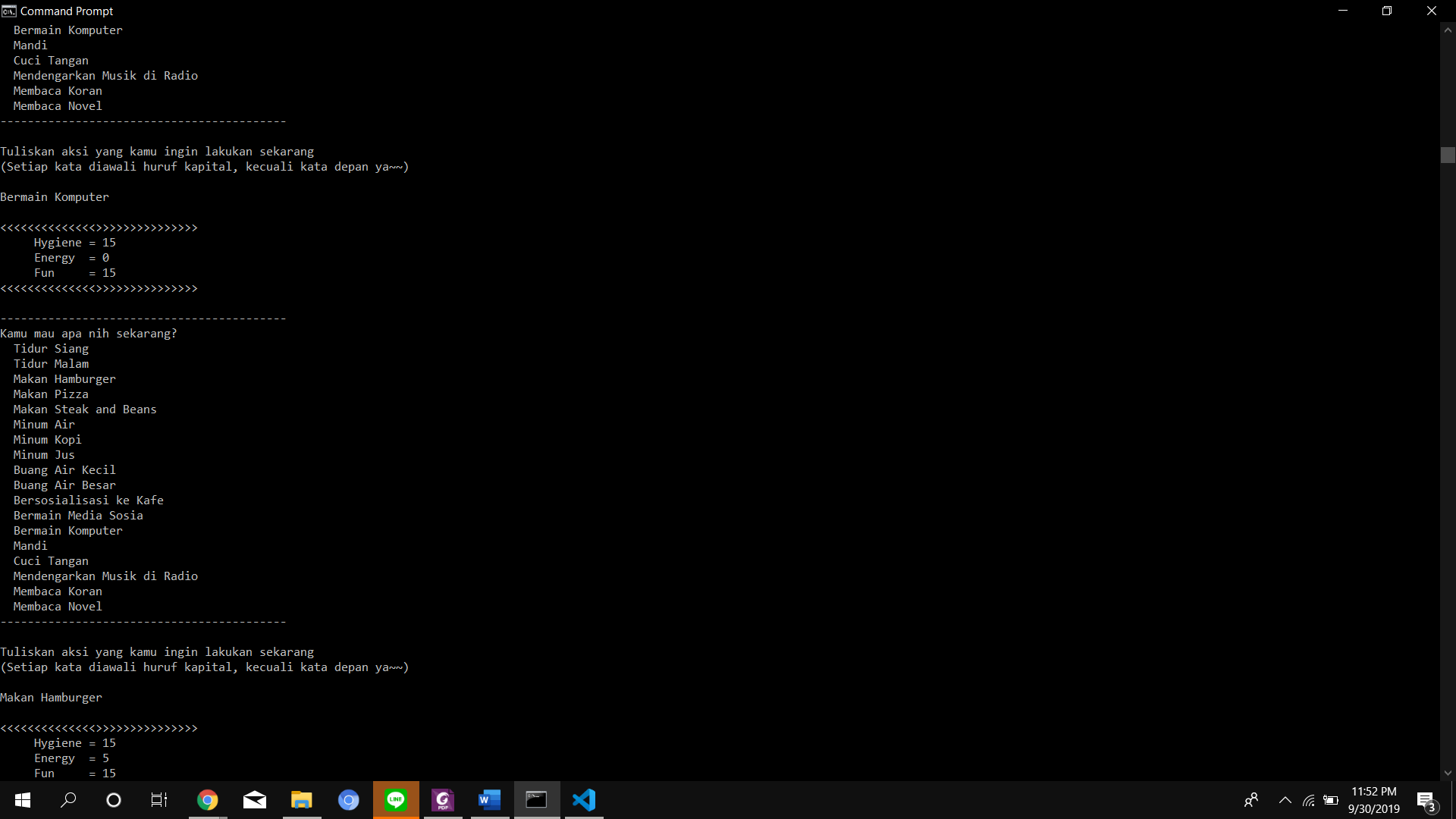
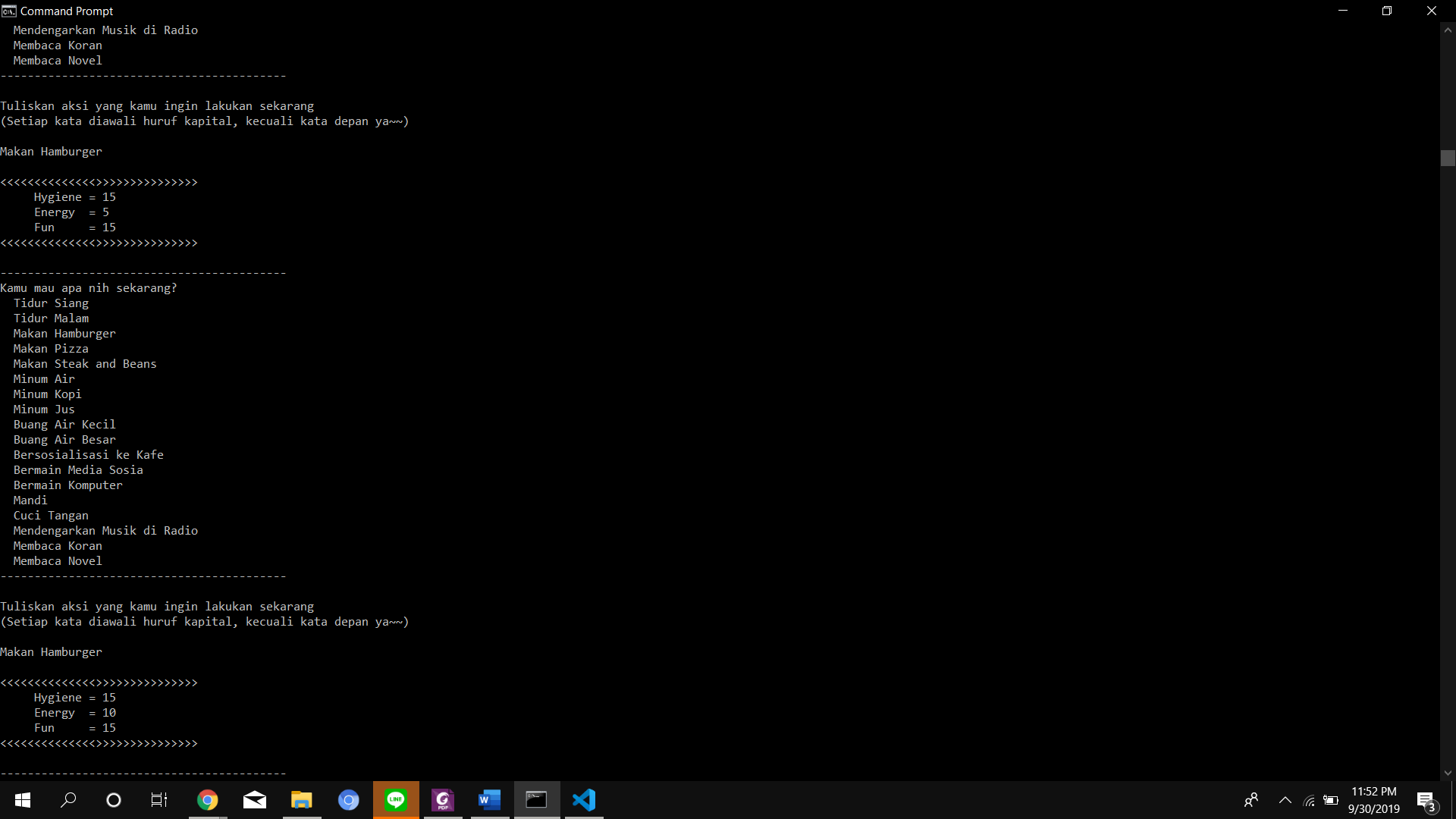
Karena hasil terakhir yang didapat adalah semua statenya nol, program sudah mencapai final *state* jenis pertama. Maka akan ditampilkan keluaran seperti yang tertera pada gambar di atas.

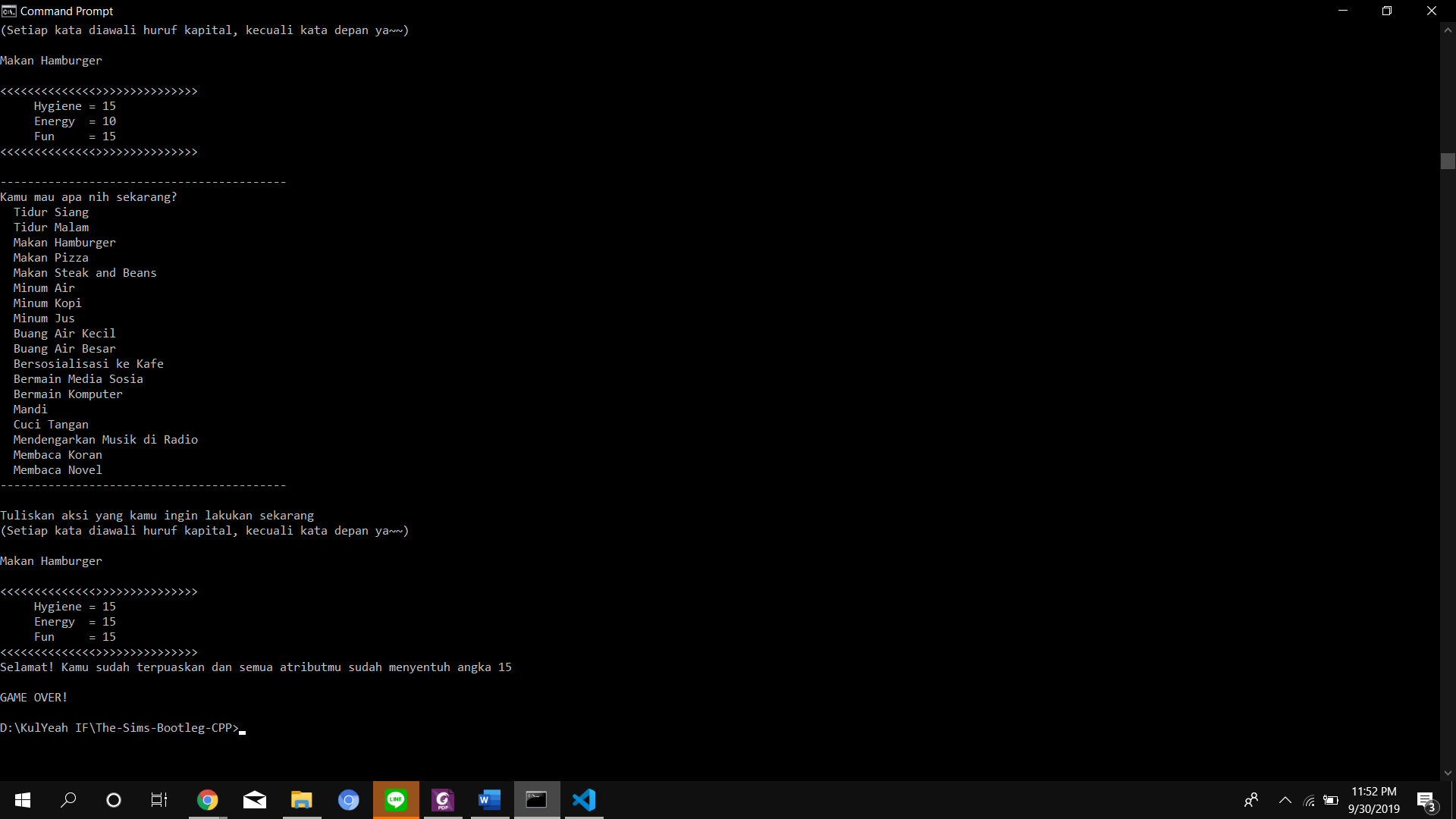
Kasus uji kedua adalah kasus ketika final *state* jenis kedua tercapai, yaitu ketika *win* atau menang. Hal ini disebabkan ketika Hygiene, Energy, dan Funketiganya bernilai 15.



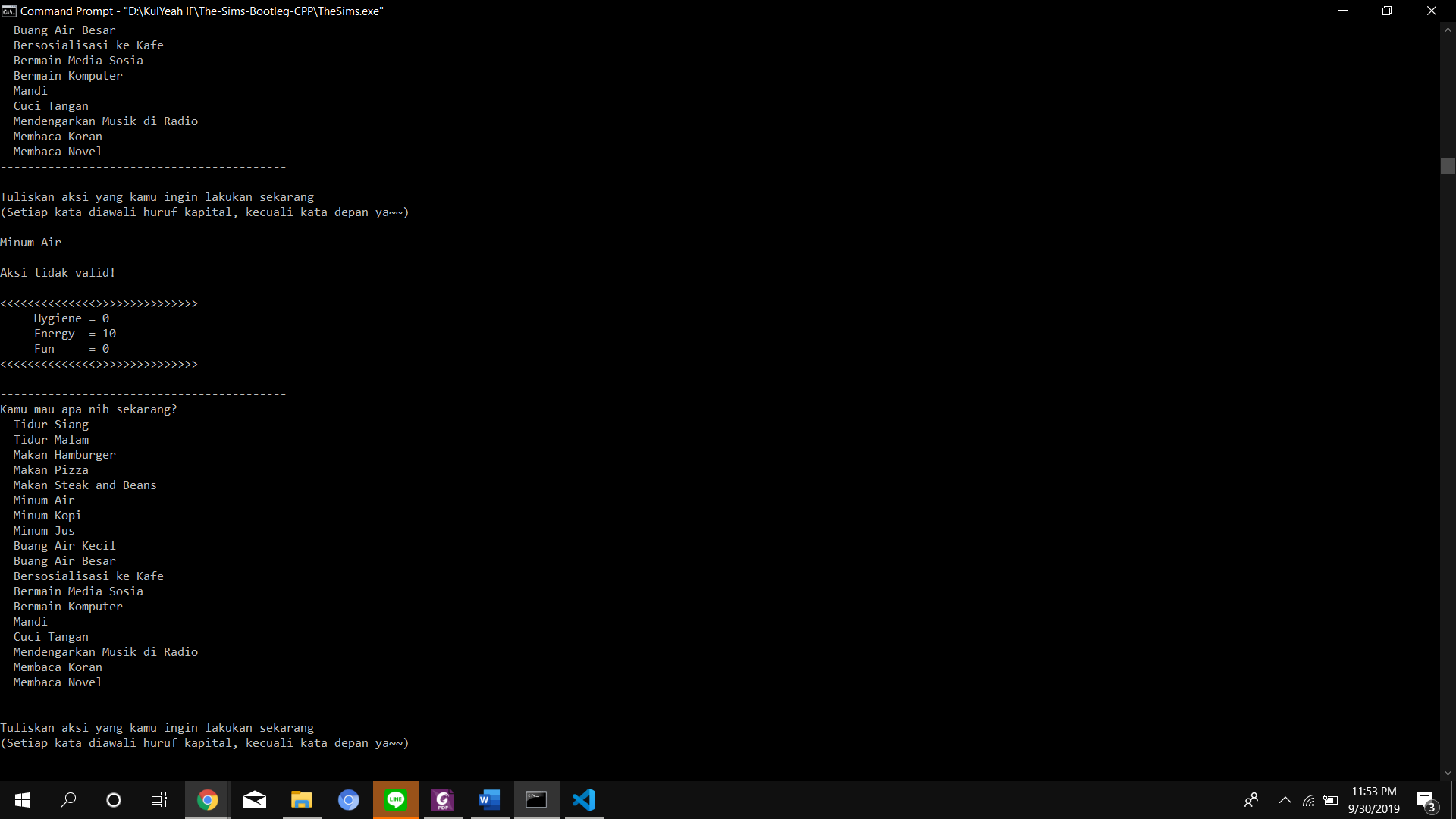




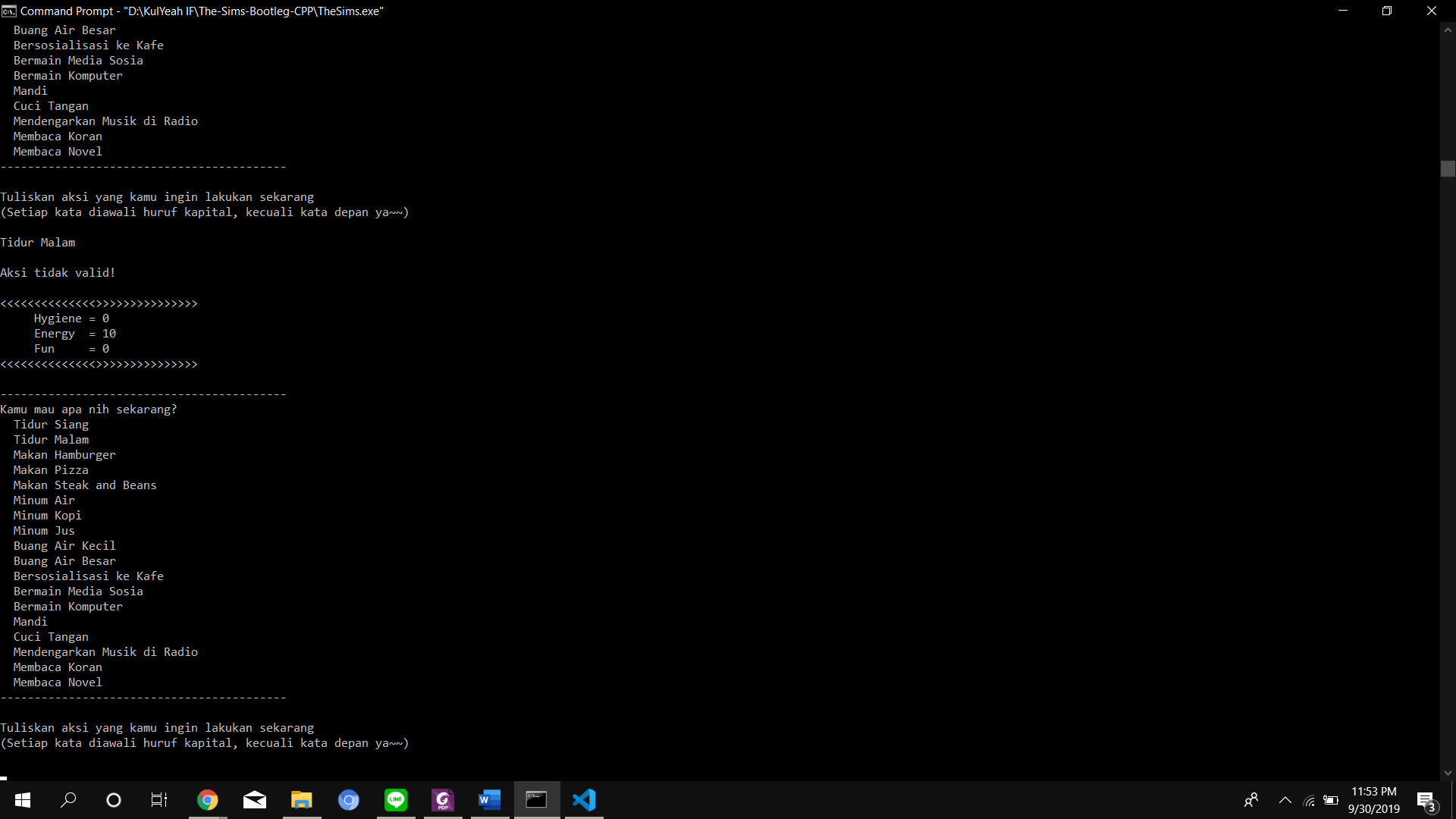


Karena hasil terakhir yang didapat adalah semua statenya 15, maka program sudah mencapai final *state* jenis kedua. Maka akan ditampilkan keluaran seperti yang tertera pada gambar di atas.

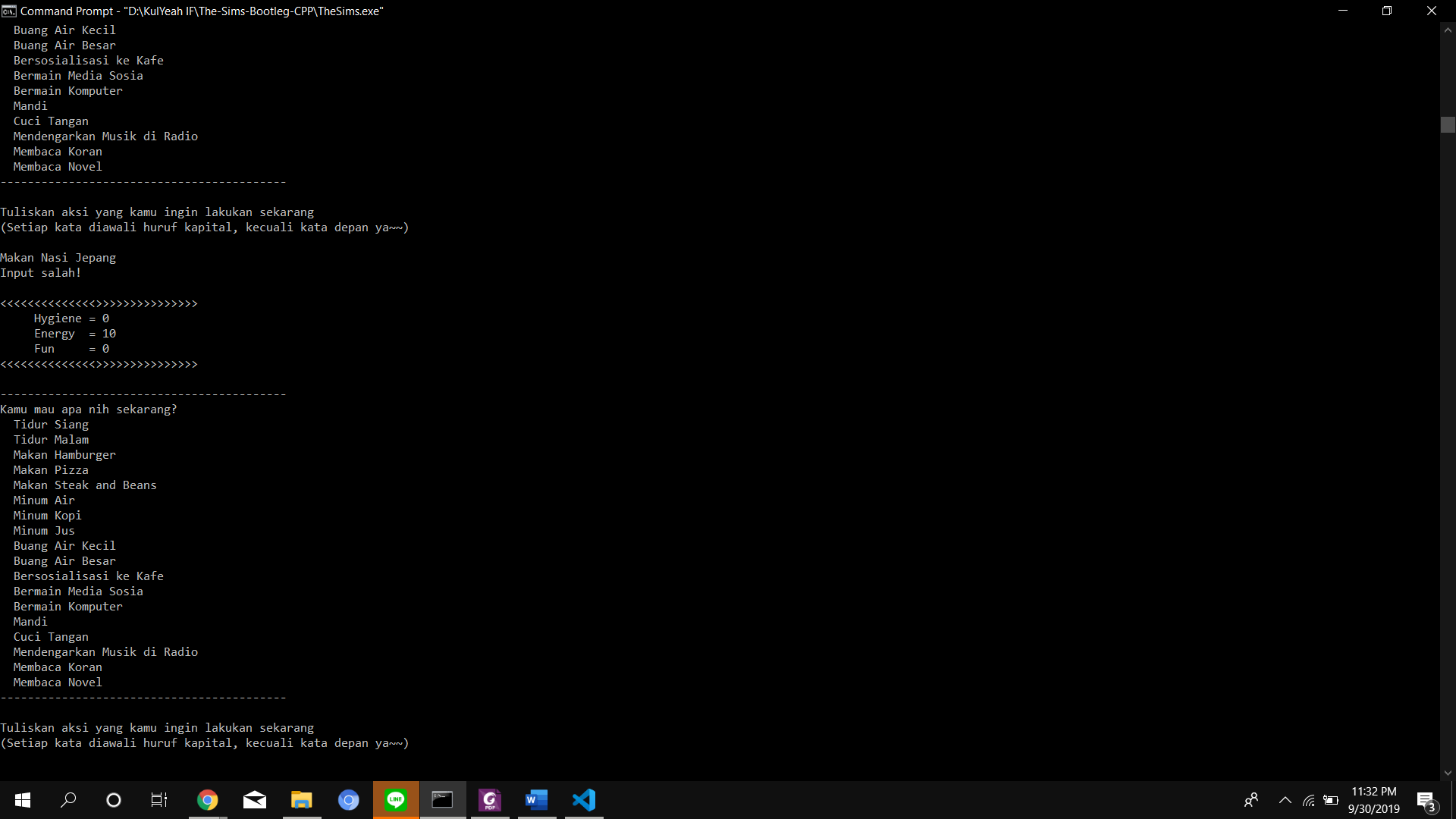
Kasus uji ketiga adalah kasus ketika masukan secara *random* dan akan ditampilkan ketika Hygiene, Energy, atau Funketika bernilai dibawah 0 atau saat *state* tidak valid jenis pertama. Maka akan dikeluarkan pesan aksi tidak valid.



Kasus uji keempat adalah kasus ketika masukan secara *random* dan akan ditampilkan ketika Hygiene, Energy, atau Funketika bernilai di atas 15 atau saat state tidak valid jenis kedua. Maka akan dikeluarkan pesan aksi tidak valid.



Kasus uji terakhir adalah saat input salah. Maka dari itu, program kami akan menampilkan sebagai berikut. Saat input salah maka program kami akan melakukan pengulangan hingga didapatkan masukan yang benar.



LAMPIRAN