EDAF30 – Programmering i C++
Albin Olausson (al6743ol-s)
Ludwig Sinclair (lu7317ha-s)

Inlämningsuppgift
Wordle Solver

EDAF30 – Programmering i C++
Albin Olausson (al6743ol-s)
Ludwig Sinclair (lu7317ha-s)

Introduktion

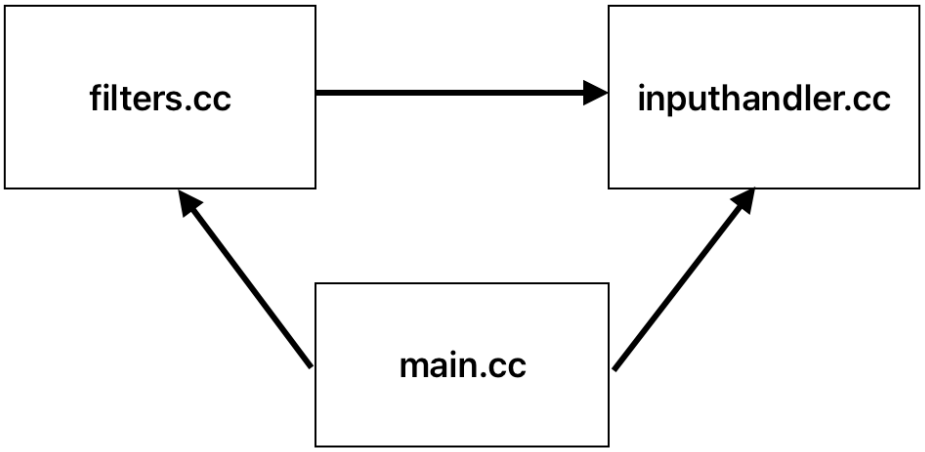
Denna rapport presenterar en möjlig lösning på en uppgift som innebär att utveckla ett program i C++ för att hitta lösningar till bokstavsspelet Wordle. Spelet Wordle går ut på att en användare försöker gissa rätt ord på fem bokstäver inom fem försök. Efter varje gissat ord ges feedback till användaren om hur nära lösningen ordet är:

- Grå bokstav - Ogiltig. Bokstaven finns inte med i ordet.
- Gul bokstav - Felplacerad. Bokstaven finns med i ordet men befinner sig på fel plats.
- Grön bokstav - Korrekt. Bokstaven finns i ordet och befinner sig på rätt plats.

Programmet som utvecklats i denna uppgift kan användas för att hitta möjliga lösningar på ord till Wordle. Efter att en användare gjort en gissning/gissningar kan de använda den feedback de fått på bokstäver från Wordle för att filtrera fram ord i vårt program.

Klasstruktur

Programmet är designat för att filtrera ut ord från en lista av möjliga ordkandidater baserat på användarens input. Nedan beskrivs de olika källkodsfiler och funktioner som används för att uppnå detta och deras relationer till varandra. Funktionerna är beskrivna mer i detalj i källkoden.



Figur 1. Källkodsfiler och dess beroenden mellan varandra. Filen test_wordle_solver.cc har samma beroenden som main.cc i diagrammet ovan, men står inte utskriven.

main.cc

- int main()

filters.cc

- bool contains_any_of(string, string)
- bool contains_at(string, char, size_type)
- bool contains_but_not_at(string, char, size_type)
- map<size_type, string> build_list(string)
- struct wrong_fn(string)
- struct correct_fn(map)
- struct misplaced_fn(map)
- struct exclude_word(string, map, map)

input_handler.cc

- vector<string> getFiveLetterWords(string)
- void toLowerCase(string)
- tuple<string, string, string> prompt()
- vector<string> promptFileName()
- bool anotherIteration()

I filters.cc finns ett antal funktorer (wrong_fn, correct_fn, misplaced_fn och exclude_word_fn). Dessa har sedan överlagrat sin ()-operator till att nyttja contains-funktionerna.

Starta och testa programmet

För att kompilera och köra programmet krävs en linux-miljö. Om du använder en windows-dator behöver du kunna starta WSL, kör du MacOS eller Linux behöver du bara öppna en terminal. Följ därefter följande steg:

1. Extrahera zip-filen till lämplig mapp
2. Öppna terminalen och gå till katalogen du extraherade filerna till.
3. Kör Makefile med kommandot “make”
    a. Starta programmet med kommandot “./main”
        i. (optional) Om du har en egen ordlista du vill använda kan filsökvägen till denna skrivas in i terminalfältet.
        ii. Följ vidare programmets hänvisningar.
        iii. Programmet avslutas antingen när användaren väljer att inte fortsätta eller då det inte finns några förslag att returnera.

    b. Starta testprogrammet med kommande “./test_wordle_solver”
        i. Terminalen skriver ut “true”om testerna går igenom.
        ii. (optional) Om något test inte går igenom skrivs detta ut i terminalen.

EDAF30 – Programmering i C++
Albin Olausson (al6743ol-s)
Ludwig Sinclair (lu7317ha-s)

Brister och kommentarer

Radinläsning

Ett problem vi stötte på under utveckling var att terminalen inte “städades upp” efter att man använt std::cin för användarinput. Detta har lösts på två olika sätt i koden. Första gången problemet uppstod var när en användares input lästes in för de olika bokstäver
na. Detta löstes genom att använda getline() som tar en användares input fram till och med radslut(“\n”).

```
std::string s;
std::getline(std::cin, s);
```

Andra gången problemet uppstod var när en användare skulle uppge om programmet skulle fortsätta eller inte. Först användes getline() vilket inte visade sig fungera eftersom getline() tar med radslutet. Detta tror vi hade kunnat lösas med att trimma bort radslutet från strängen. Istället löstes problemet med att använda en cin.ignore() som städar bort alla tecken från terminalens buffert fram till och med ett radslut. Detta görs för att getline() senare inte ska läsa in radslut från terminalen .

```
std::string s;
std::cin >> s;
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

Funktionell programmering

Vi insåg en bit på väg in i detta projekt att vi aldrig skrev en klass (med undantag för de små structarna som användes som funktorer) utan istället bara hade fria funktioner. Det var lite ovant, men visade sig ha en rad fördelar. Bland annat ville vi i ett avslutande skede refaktorera koden i flera mindre källkodsfiler för att göra koden mer lättläslig och hålla isär filernas ansvarsområde. Detta var oerhört lätt då det bara var att bryta ut och sedan inkludera header-filerna där de behövdes. Diagrammet i Figur 1 ritade vi snabbt upp för att säkerställa att det inte fanns några cirkulära beroenden mellan filerna, och allting fungerade direkt.

I slutändan känner vi oss nöjda med lösningen, men känner också att vi egentligen inte diskuterat det här med funktionell programmering, och vet inte vad konventioner och annat säger. Hade vi gjort lösningen i Java hade vi ju fått tänka helt annorlunda, eftersom allting där *måste* tillhöra en klass. Eftersom C++ är ett nytt språk för oss är det svårt att veta om det vi gjort borde gjorts på ett annat sätt.