Opdrachten week 7 les 1 – Datastructuren

Bij het practicum van EMS20 wordt wel eens een uitwerking van een practicumopdracht gekopieerd. Bij EMS30 komt dit gelukkig niet meer voor ;-). Vaak is een student die een programma van een andere student kopieert wel zo slim om de namen van variabelen te wijzigen en de opmaak van het programma te veranderen. Een docent wil daarom een programma maken dat een aantal kenmerken van een (ander) C++-programma vaststelt zodat verschillende door studenten ingeleverde programma's op deze kenmerken met elkaar kunnen worden vergeleken. De docent is van mening dat het aantal maal dat elk C++-keyword in een C++-programma voorkomt een belangrijk kenmerk van een programma is om de originaliteit van een programma vast te stellen.

Je leert deze les hoe je:

- een aantal datastructuren uit de standaard C++ library kunt gebruiken;
- een aantal algoritmen uit de standaard C++ library kunt gebruiken.

Lees nu hoofdstuk 10 tot en 13 van het dictaat.

7.1.1 Schrijf een C++-programma dat in een ander C++-programma telt hoe vaak elk C++-keyword daarin voorkomt. Maak gebruik van de file `keywords.txt` waarin alle C++-keywords (voor C++17) staan. Deze file moet aangepast of uitgebreid kunnen worden[1]. Maak zoveel mogelijk gebruik van datastructuren en algoritmen uit de standaard C++ library. Na afloop moet het programma een lijst produceren met per regel een C++-keyword met daarbij het aantal maal dat dit keyword voorkomt.

Lees de hierna volgende tips als je zelf geen plan van aanpak kunt bedenken.

- Tip 1:
  Het is nog niet zo eenvoudig om alleen de identifiers van een C++-programma in te lezen. Denk eraan dat commentaar en string literals niet meegenomen moeten worden. Gelukkig heeft de docent zelf al een **`class`** genaamd `Identifier_extractor` geschreven die dit mogelijk maakt. Je mag gebruik maken van deze **`class`** die je kunt vinden in `plagiaat_scanner.cpp`.
- Tip 2:
  Als datastructuur kan een `map` met per keyword een teller gebruikt worden. Het programma uit paragraaf 11.4.4 van het dictaat kan prima ter inspiratie gebruikt

[1] In C++20 worden de volgende keywords toegevoegd: `char8_t`, `concept`, `consteval`, `constinit`, `co_await`, `co_return`, `co_yield`, `import`, `module` en `requires`.

worden. Natuurlijk moet je er dan wel voor zorgen dat alleen de keywords geteld worden. Dit kan op twee manieren:

- Maak een `set` aan met daarin alle keywords. Met de memberfunctie `count` van set kan eenvoudig bepaald worden of een ingelezen woord een keyword is (en geteld moet worden).
- Vul de `map` voordat je gaat beginnen met tellen eerst met alle keywords (alle tellers op 0). Je moet dan alleen woorden tellen die al in de `map` voorkomen. Met de functie `count` van `map` kan eenvoudig bepaald worden of een ingelezen woord al in de `map` voorkomt.

- Tip 3:
  Als je er voor hebt gekozen om een `set` te gebruiken, dan kun je de inhoud van de file `keywords.txt` naar de `set` kopiëren met het `copy`-algoritme door gebruik te maken van `istream_iterator`'s.

Het programma uit opdracht 7.1.1 is niet zo gebruiksvriendelijk. Als de docent twee programma's wil vergelijken, dan moet hij het programma twee keer uitvoeren en de uitvoer handmatig met elkaar vergelijken. De docent wil dit vergelijken automatiseren.

7.1.2 Schrijf een C++-programma dat in twee andere C++-programma's telt hoe vaak elk C++-keyword daarin voorkomt. Maak gebruik van de file `keywords.txt` waarin alle te tellen C++-keywords staan. Deze file moet aangepast of uitgebreid kunnen worden. Na afloop van dit programma moet één van de volgende meldingen worden gegeven:

1 Deze programma's zijn hoogstwaarschijnlijk niet origineel.

2 Deze programma's zijn misschien niet origineel.

3 Deze programma's zijn hoogstwaarschijnlijk wel origineel.

Melding 1 moet worden gegeven als alle C++-keywords in beide programma's exact even vaak voorkomen. Melding 2 moet gegeven worden als het aantal maal dat een keyword gebruikt wordt slechts bij hoogstens 3 keywords afwijkt. Bij deze afwijkingen mag het betreffende keyword hooguit 1 keer vaker of 1 keer minder voorkomen. In alle andere gevallen moet de derde melding gegeven worden. Maak zoveel mogelijk gebruik van datastructuren en algoritmen uit de standaard C++ library.

Lees de hierna volgende tip als je zelf geen plan van aanpak kunt bedenken.

- Tip:
  Je kan natuurlijk voor elke file een aparte `map` gebruiken, maar het is eenvoudiger om één `map` te gebruiken en bij het inlezen van de ene file de bij de keywords behorende tellers te verhogen en bij het inlezen van de andere file de bij de keywords behorende tellers te verlagen. Voor alle keywords die even vaak in beide files gebruikt worden zullen de tellers na afloop op 0 staan. Met een `count_if` kun je vervolgens tellen hoeveel tellers ongelijk aan 0 zijn. Vervolgens kun je met een `any_of` bepalen of er een afwijking van meer dan 1 is (hou er wel rekening mee dat het keyword 1 keer meer of 1 keer minder gebruikt kan zijn).