



TAMPEREEN
AMMATTIKORKEAKOULU

Tietorakenteet ja algoritmit

Harjoitustyö 1 – Symbolien tasapainotusohjelma

Valkoja Iiro



1 Yleistä

1.1 Tavoiteltu pistemäärä

Viittä pistettä oltaisiin hakemassa

1.2 Palauttajan yhteystiedot

Sekä sähköposti että puhelinnumerot ovat saatavissa TAMK:in järjestelmistä, en niitä ala julkaisemaan internetissä.

1.3 Tuntikirjanpito

2017-10-07	17:00-18:00	Tehtävänanto, ajatuksia miten lähteä toteuttamaan, kansiot, tiedostot, alkuperäinen esimerkki toimimaan.
2017-10-08	07:00-08:00	Luokan uudelleennimeäminen, metodien luonnostelua, käyttäjän syötteen tarkastelu, menu mukaan luokkaan.
2017-10-08	08:00-09:00	Tiedostojen lukeminen ja tallentaminen vektoriin riveittäin, vektorien ja stringien iterointi.
2017-10-11	11:30-13:00	Iteraattoreihin perehtymistä & toteuttamista, mahdollisuus kurkata edellinen merkki (kommentteja varten).
2017-10-14	18:00-18:30	Kommentit ja lainausmerkit samaan listaan / structtiin sulkujen kanssa, vaihdetaan luokan nimi kuvaamaan.
2017-10-14	18:30-19:30	Merkkien tunnistus: sekä yksittäisten että //, /* ja */ kanssa edellisen kurkkaaminen.
2017-10-14	19:30-20:30	Tunnistuksen uudelleenjärjestely jotta kommentin, ' ' ja " sisällä oleminen ohittaa muut tarkastelut.
2017-10-14	20:30-21:00	Merkkien lisääminen ja poistaminen listasta omiin metodeihin

1.4 Tuntikirjanpito jatkuu

2017-10-14	23:00-23:30	Virhesanoma ja suorituksen pysäyttäminen virheen löytyessä
2017-10-14	23:30-00:00	Testailua ja sekalaista, mm. ' ja " tilanteessa jossa edessä on \ sekä < vertailu ja stream operaattorina
2017-10-15	00:00-02:00	Jatko edelliseen, missä tilanteissa ”, ', < ja > pitää jättää huomiotta. Sirretty ohitus omaan metodiin. Metsästetty bugissa jossa koodi, jonka pitäisi valvoa että onko lainausmerkki ohitettu ei ohittanut itseään testissä.
2017-10-15	16:00-17:30	Siistimistä, debugrivien poistamista, testailua, kommenttien läpikäyntiä yms viimeistelyä.
2017-10-15	17:30-19:00	Tämä tiedosto, ratkaisun gitlab:iin

2 Ohjelman kääntäminen & ajaminen

Ohjelmaa on käännetty MinGW mukana tulevalla g++ (GCC) 5.3.0 kääntäjällä. Asenna kääntäjä (windows-ympäristössä se kannattaa myös lisätä PATH:iin) ja suorita:

```
g++ -std=c++0x -o .\H1.exe *.cpp
```

Jossa:	
g++	kutsuu kääntäjää
-std=c++0x	kertoo että käytetään C++11 standardia (ISO/IEC 14882:2011)
-o .\H1.exe	käännetyt tiedoston nimi
*.cpp	sisällytetään kaikki cpp tiedostot työkansiota

Tämän jälkeen kansiota pitäisi löytyä H1.exe ajettavaksi.

3 Testitapaukset

3.1 Yhden pisteen työ

Gitlabissa mukana oleva Testi.txt ja jotain vastaavaa käsisyötteenä ”if (testi = joo) {}” tässä kohtaa olennaista oli että ohjelma otti syötteen vastaan.

3.2 Kahden pisteen työ

Samat testitapaukset kuin edellisessä kohdassa.

3.3 Kolmen pisteen työ

Muuten sama Testi.txt kuin edellisissä kohdissa mutta lisätty pari kommenttia. Käsinsyötetty tehtävänannon esimerkkiriviä:

```
( [ /* ( */ ))))” ’[ ]
```

3.4 Neljän pisteen työ

Samat testit kuin aikaisemmissakin kohdissa

3.5 Viiden pisteen työ

Samat testit kuin aikaisemmissakin kohdissa ja ohjelman valmistuessa aloin ajamaan myös symbolMatcher.cpp tiedostoa testinä.

4 Suunnitteluratkaisut lyhyesti

4.1 Yhden pisteen työ

Korjattiin esimerkki (esim koko namespacea ei hyvän tavan mukaan käytetä..) muotoiltiin se ihmisen luettavaksi ja lisättiin pieni valikko. Tiedston lukeminen ifstreamilla rivi kerrallaan stackiin.

4.2 Kahden pisteen työ

Heitettiin esimerkkikoodi menemään, kirjoitettiin koko ratkaisu luokaksi ja vaihdettiin samalla c++ vektoreihin. Toteutus kun on käytännössä sama kuin stackeissa, tarjoavat vaan enemmän liikkumavaraa- Itse ohjelma toimii samalla idealla kuin esimerkki, kun avaava sulkua tulee vastaan se lisätään listaan. Kun sulkeva, tarkistetaan löytyykö listasta sille avaavaa paria. Jos ei, tulostetaan virhe ja lopetetaan. Muutoin mennään ohjelman loppuun, tarkistetaan jäikö listaan sulkua, jos jäi niin annetaan niistä virhe. Tätä ominaisuutta varten sulut on structeja, saadaan näppärästi rivi ja sarake mukaan listaan että paikkaa ei tarvitse tässä kohtaa alkaa kaivamaan.

4.3 Kolmen pisteen työ

Viedään tutkittavat merkit samaan listaan sulkujen kanssa. Kun tutkitaan merkkiä, aloitetaan kurkkamalla onko joku näistä merkeistä viimeisenä listassa, jos on niin ohitetaan kaikki paitsi merkki joka lopettaa kommentin tai merkkijonon. Huomattavaa on että kommentit ovat kaksi merkkiä ja merkeissä & merkkijonoissa kenoviiva toimii escapena. Kommentit ratkaisuun tutkimalla edellistä:

Jos merkki on '*'

Jos edellinen merkki on '/'

Aloita kommentti

Escapet olivat hieman hankalampia, sillä myös kenoviivan voi escapea. Eli jos `\` niin `”` ei oteta huomioon, mutta jos `\\` niin eka kenoviiva nollaa toisen ja `”` käsitelläänkin normaalisti. Tätä varten lisättiin `isEscaped` metodi, joka laskee tarkistettavasta solusta taaksepäin olevat kenoviiva, jos niitä on parillinen määrä niin palautetaan epätosi ja käsitellään merkki normaalisti, jos pariton niin merkki jätetään huomiotta.

4.4 Neljän pisteen työ

Koska merkkien paikkatiedot olivat tässä kohtaa jo valmiiksi structissa mukana, lisättiin vain virheilmoitukseen kuvaus.

4.5 Viiden pisteen työ

Käytetty ratkaisu kolmen pisteen työhön hoiti myös tämän, poislukien hakasulkeet, jotka osoittautuivat varsin ongelmallisiksi. Niitä kun käytetään stream operaattoreina ja vertailussa. Ratkaisussa `<` ja `>` jätetään huomiotta jos rivillä on `”cout”` tai `”cin”` tai jos edellinen sulku on `”(”` (jolloin luultavasti ollaan `if / while / for / tms` sisässä).