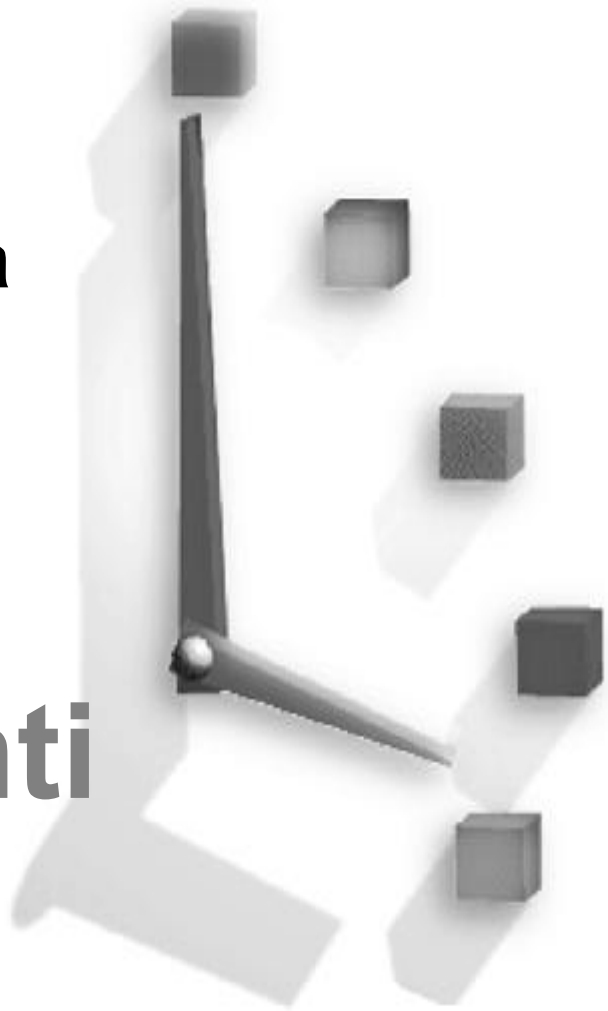
Osa I

Miksi C++ on oikea valinta

Oppitunnit

Osa

I



1. oppitunti

Aloitetaan

Tervetuloa tutustumaan C++ Trainer -teokseen! Tämän ensimmäisen tunnin aikana opit seuraavia asioita:

- Miksi C++ on tulossa vahvaksi standardiksi ohjelmistotuotannossa
- C++ -ohjelman kehittämisen vaiheet
- Ensimmäisen C++ -ohjelman kirjoittaminen, kääntäminen ja linkittäminen

Miksi C++ on oikea valinta?

Suurin osa ammattiohjelmoijista valitsee C++ -kielen työkalukseen, koska sillä voidaan kehittää nopeita ja kiinteäkoodisia ohjelmia vakaassa ja siirrettävässä ympäristössä. C++ -työkaluilla voidaan tänä päivänä kehittää monimutkaisia ja tehokkaita kaupallisia sovelluksia hyvinkin suoraviivaisesti, mutta saadaksesi enemmän irti C++ -kielestä on sinun opiskeltava sitä huomattavasti enemmän.

C++ on suhteellisen uusi ohjelmointikieli. Ohjelmointi itsessään on kylläkin jo 40 vuotta vanha tekniikka. Alkuajoista ovat ohjelmointikielet kokeneet dramaattisen muodonmuutoksen. Alkuaikoina ohjelmoijat käyttivät perustason konekielisiä ohjeita, jotka esitettiin ykkösinä ja nollina. Tämän

jälkeen kehitettiin symbolinen konekieli, jonka ihmiselle tutut komennot vastasivat perustason konekielisiä ohjeita. Nuo ohjeet olivat siis luettavia ja paremmin hallittavia (esimerkkeinä olkoot komennot ADD ja MOV).

Uusi käsite Aikaa myöten tulivat korkeamman tason kielet, kuten BASIC ja COBOL. Noiden kielten avulla ohjelmoijat voivat käyttää ihmisille tuttuja sanoja ja lauseita, kuten Let I = 100. Nuo lauseet käännettiin sitten takaisin konekielisiksi tulkkien ja kääntäjien toimesta. Tulkki, kuten BASIC, kääntää ohjelman lukemisen yhteydessä ohjelmoijan antamat ohjeet eli koodin suoraan konekielisiksi toiminnoiksi.

Uusi käsite Uudet kääntäjät kääntävät koodin objektikoodiksi. Tämän muuntamisen ensimmäistä vaihetta kutsutaan kääntämiseksi. Kääntäjä tuottaa objektikooditiedoston. Toisena vaiheena on linkitys. Linkittäjä muuntaa objektitiedoston suoritettavaksi ohjelmaksi. Käyttöjärjestelmä pystyy sitten ajamaan suoritettavan ohjelman.

Koska tulkit lukevat kirjoitetun koodin ja suorittavat sen samalla, on tulkkien kanssa helppo toimia. Kääntäjät lisäävät pari vaihetta ohjelman kehitykseen aiheuttaen samalla lisävaivaa. Toisaalta ohjelmasta tulee kääntäjän ansiosta nopeampi ajettava.

Kauan aikaa oli ohjelmoijien päätavoitteena kirjoittaa lyhyitä koodimoduuleita, jotka voitiin ajaa nopeasti. Ohjelman tuli olla pieni muistin kalleuden takia ja nopea, koska myös tietokoneen käyttö oli kallista. Tietokoneiden pienentyessä, halvetessa ja tehostuessa ja muistin tullessa edullisemmaksi ovat edellä kuvatut periaatteet muuttuneet. Nykyään on ohjelmointiin käytettävä aika suurempi kustannus kuin laitteisto. Hyvin kirjoitetut, helposti ylläpidettävät ohjelmat ovat keskeisessä asemassa. Ylläpidettävyys tarkoittaa sitä, että liiketoiminnan vaatimusten muuttuessa voidaan ohjelmaa laajentaa ja muokata ilman suuria kustannuksia.

Proseduraalinen, rakenteellinen ja oliopohjainen ohjelmointi

Uusi käsite Proseduraalinen ohjelmointi tuottaa ohjelmia, jotka sisältävät sarjan tietojoukkoon kohdistuvia toimintoja. Rakenteellinen ohjelmointi keksittiin tarjoamaan systemaattinen lähestymistapa noiden proseduurien organisointiin ja suurten tietomäärien hallintaan.

Rakenteellisen ohjelmoinnin pääideana on 'hajota-ja-hallitse' -ajatus. Jokainen tehtävä, jonka määrittely on liian monimutkaista, jaetaan joukoksi pienempiä tehtäviä, joita on helpompi hallita ja ymmärtää.

Esimerkiksi yrityksen jokaisen työntekijän keskiarvopalkan laskeminen on melkoisen monimutkainen tehtävä. Tällainen ongelma voidaan kuitenkin jakaa pienempiin osatehtäviin:

- ❏ Selvitä, mitä jokainen henkilö ansaitsee
- ❏ Laske henkilöiden lukumäärä
- ❏ Laske palkat yhteen
- ❏ Jaa palkkojen summa henkilöiden lukumäärällä

Palkkojen yhteenlaskeminen voidaan jakaa osatehtäviin:

- ❏ Ota esille kunkin työtekijän tiedot
- ❏ Ota palkkatiedot
- ❏ Lisää palkka juoksevaan palkkasummaan
- ❏ Ota esille seuraavan työtekijän tiedot

Ja henkilön tietojen esilleottaminen voidaan jakaa vielä pienempiin osatehtäviin:

- ❏ Avaa työntekijöiden tietokanta
- ❏ Siirry oikeaan tietueeseen
- ❏ Lue tieto levyltä

Rakenteellinen ohjelmointi pysyy erittäin menestyksellisenä lähestymistapana monimutkaisten ongelmien ratkaisemisessa.

Ongelmiakin kuitenkin on. Tiedon eristäminen tehtävistä, jotka käsittelevät tietoa, on yhä vaikeampaa tiedon määrän kasvaessa. Mitä enemmän tietoa käsitellään, sitä vaikeampaa ja epäselvempää tiedon eristäminen tulee olemaan.

Proseduraalisen tyylin ohjelmoijat näyttävät jatkuvasti kehittävän uusia ratkaisuja vanhoihin ongelmiin, mitä kutsutaan 'pyörän uudelleen keksimiseksi'. Tuollainen menettelytapa sotii uudelleen käytettävyyttä vastaan. Uudelleen käytettävyyden ideana on kehittää komponentteja, joilla on tunnetut ominaisuudet ja sijoittaa niitä ohjelmiin aina tarpeen vaatiessa. Idea tulee laitemaailmasta: kun insinööri tarvitsee uuden transistorin, hän ei ala keksiä sitä, vaan hakee varastosta sopivan. Tarvittaessa hän voi vielä muokata lähellä sopivaa olevaa komponenttia. Ennen oliopohjaista ohjelmointia eivät ohjelmoijat voineet käyttää kyseistä vaihtoehtoa.

Uusi käsite Oliopohjaisen ohjelmoinnin ydin on sisällyttää tieto ja tietoa käsittelevät proseduurit yhteen olioon - yksikköön, jolla on oma identiteettinsä ja tietyt ominaisuutensa.

C++ ja oliopohjainen ohjelmointi

C++ tukee täysin oliopohjaista ohjelmointia mukaan lukien oliopohjaisen ohjelmoinnin peruspilarit: kapselointi, tiedon kätkeminen, periytyvyys ja polymorfia.

Kapselointi ja tiedon kätkeminen

Kun insinööri luo uutta laitetta, hän langoittaa yhteen komponentteja. Hänellä on ehkä vastus, kondensaattori ja transistori. Transistorilla on tietyt ominaisuudet ja se toimii tietyllä tavalla. Insinööri voi käyttää transistoria ymmärtämättä yksityiskohtia siitä, kuinka se toimii, kunhan hän vain tietää, mitä se tekee.

Uusi käsite Jotta transistoria voidaan käyttää, on transistorin oltava kokonaisuus. Sen on tehtävä yksi hyvin määritelty tehtävä ja tehtävä se täydellisesti. Tuota yhden tehtävän kokonaisvaltaista suorittamista sanotaan kapseloinniksi.

Kaikki transistorin ominaisuudet on kapseloitu transistori-objektiin; niitä ei ole sijoiteltu eri puolille piiriä. Transistorin käyttämiseksi ei ole tarpeen tietää sen sisäistä toimintaa.

C++ tukee kapseloinnin ja tiedon kätkemisen piirteitä käyttäjän määrittelemien tietotyyppien, luokkien, avulla. Hyvin määritelty luokka toimii täysin kapseloituneena kokonaisuutena; sitä voidaan käyttää kokonaisena yksikkönä. Luokan sisäinen työskentelytapa tulee kätkeä; hyvin määriteltyjen luokkien käyttäjien ei tarvitse tietää, kuinka luokka toimii; heidän on tiedettävä vain se, kuinka luokkaa käytetään. Luokkien luomista käsitellään luvussa "Perusluokat".

Periytyvyys ja uudelleen käyttäminen

1980-luvun lopulla työskentelin Citibank-organisaatiossa rakentamassa laitetta kotien pankkitoimintoihin. Emme halunneet aloittaa tyhjästä, vaan halusimme saada tuotteen markkinoille pian. Siksi aloitimme puhelimesta ja laajensimme sen toimintaa. Uusi laajennettu puhelimemme oli muutoin kuin normaali puhelin, me vain lisäsimme siihen uusia piirteitä. Näin saatoimme käyttää uudelleen vanhan puhelimen ominaisuuksia ja saada siitä hyödyllisemmän lisäämällä siihen uusia ominaisuuksia.

Uusi käsite C++ tukee uudelleen käyttämisen ideaa periytyvyyden kautta. On mahdollista määritellä uusi tyyppi, joka on laajennus olemassa olevasta tyypistä. Tämä uusi aliluokka johdetaan olemassa olevasta tyypistä, ja sitä kutsutaankin usein johdetuksi tyypiksi. Laajennettu puhelin on

johdettu vanhasta puhelimesta ja täten se perii kaikki sen ominaisuudet, mutta laajennettuun puhelimeen voidaan kuitenkin lisätä uusia ominaisuuksia tarpeen mukaan. Periytyvyyttä ja sen sisältymistä C++ -kieleen käsitellään luvussa "Periytyvyys".

Polymorfia (monimuotoisuus)

Laajennettu puhelin käyttäytyy eri tavalla kun vastaanotat puhelun. Kellon soittamisen sijaan näyttö liukuu ylös ja ääni sanoo "Sinulle on puhelu". Puhelinyhtiö ei kuitenkaan tiedä sitä. Se ei lähetä erikoissignaaleita jokaiselle eri puhelimelle. Signaalit ovat samanlaiset kaikkialle, vain puhelimissa on eroja: tavallinen puhelin soi, elektroniset puhelimet antavat erilaisia äänisignaaleita ja laajennettu puhelin antaa ääniviestin. Kukin puhelin toimii oikein sen mukaan kuin se ymmärtää puhelinyhtiön antaman viestin.

Uusi käsite C++ tukee ajattelua, jossa eri oliot suorittavat oikeita toimintoja. Ajattelu sisältyy käsitteisiin funktion monimuotoisuus (polymorfia) ja luokan monimuotoisuus (polymorfia). Sanassa polymorfia tarkoittaa poly monta ja morfia taas muotoisuutta. Niinpä sana polymorfia kätkee sisäänsä ajatuksen siitä, että sama nimi saa useita eri muotoja. Monimuotoisuutta käsitellään luvuissa "Monimuotoisuus ja johdetut luokat" sekä "Kehittynyt monimuotoisuus".

Kuinka C++ kehittyi

Kun oliopohjainen analyysi, suunnittelu ja ohjelmointi saivat jalansijaa, alkoi Bjarne Stroupstrup laajentaa maailman suosituinta ohjelmointikieltä, C-kieltä, ja kehitti siihen uusia piirteitä, jotka mahdollistivat oliopohjaisen ohjelmoinnin. Hän loi C++ -kielen. Aluksi vain kourallinen AT&T-yhtiön ohjelmoijia käytti sitä, mutta yhden vuosikymmenen aikana se on levinnyt arviolta yli miljoonan ohjelmoijan työkaluksi eri puolilla maailmaa.

C++ ei ole vain parempi C-kieli

On totta, että C++ on C-kielen perusjoukko ja näennäisesti jokainen laillinen C-ohjelma on myös laillinen C++ -ohjelma, mutta älä anna sen huijata sinua. H.L. Mecken sanoi kerran, että Wagnerin musiikki on "parempaa kuin se kuulostaa". C- ja C++ -kielten välinen kuilu on suurempi kuin se näyttää.

C++ hyötyi suhteestaan C-kieleen monen vuoden ajan, kun C-ohjelmoijat saattoivat helposti siirtyä C++ -kieleen. Kuitenkin C++ -kielestä saadaan kaikki teho irti vasta, kun päästään C-kielen rajoitteista ja opitaan uusi tapa käsitteellistää ja ratkaista ohjelmointiongelmia.

Pitäisikö opetella ensin C-kieli?

Tämä on ikuinen kysymys: koska C++ on C-kielen perusjoukko, pitäisikö ensin opetella C-kieli? Stroupstrup ja useimmat muut C++ -ohjelmoijat ovat samaa mieltä: ei ole pelkästään turhaa opiskella ensin C-kieltä, se on myös huono ajatus. Tämä kirja olettaa, ettet ole C-ohjelmoija. Jos osaat kuitenkin C-ohjelmoinnin, ei siitä ole haittaa. Lue ensimmäiset viisi lukua kevyesti ja pysy sitten tiukasti mukana.

Ohjelmoimaan valmistautuminen

C++ vaatii ehkä muita ohjelmointikieliä enemmän suunnittelemaan ohjelman ennen koodaamista. Yksinkertaiset ongelmat (kuten ensimmäisten viiden luvun esimerkit) eivät tietenkään vaadi sen kummempaa suunnittelua. Monimutkaisemmat ongelmat, jotka ovat tuttuja kaikille ammattiohjelmoijille kaiken aikaa, vaativat suunnittelua. Mitä syvällisempi suunnittelu on, sitä todennäköisemmin ohjelma tekee ne tehtävät, joita siltä odotettiin ja vieläpä aikataulun ja budjetin mukaisesti valmistuen. Hyvä suunnittelu vähentää myös ohjelman virheitä ja varmistaa ylläpidon. On arvioitu, että 90 % ohjelman aiheuttamista kustannuksista koostuu sen elinaikana virheiden korjauksista ja ylläpidosta. Sen lisäksi, että suunnittelu voi vähentää noita kustannuksia, sillä on merkitys myös projektin peruskustannuksiin.

Ensimmäinen kysymys alettaessa kehittää ohjelmaa, on "Mikä on se ongelma, jonka ohjelman pitäisi ratkaista?" Jokaisella ohjelmalla tulee olla selkeä, hyvin määritelty tavoitteensa ja huomaatkin, että myös kaikkein yksinkertaisimmallakin tämän kirjan ohjelmalla on sellainen.

Toinen hyvän ohjelmoijan asettama kysymys on "Voidaanko tämä ongelma ratkaista kirjoittamatta täysin uutta ohjelmaa?" Vanhan ohjelman uudelleen käyttö, kynän ja paperin käyttäminen tai valmiin ohjelmatuotteen ostaminen voivat usein olla parempia ratkaisuja ongelmaan kuin täysin uuden ohjelman kehittäminen. Ohjelmoijalta, jolla on näkemys noista kaikista vaihtoehdoista, ei koskaan puutu työtä: edullisimpien ratkaisujen hakeminen nykyisiin ongelmiin tuottaa uusia tilaisuuksia myöhemmin.

Olettaen, että tiedät ongelman ja se vaatii uuden ohjelman kirjoittamista, voit aloittaa ohjelman suunnittelun.

C++, Ansi C++, Windows ja muut epäselvät alueet

C++ on kieli. DOS, Windows, UNIX ja MacOS ovat käyttöjärjestelmiä. Kun opit C++ -kielen, opit samalla siirrettävän kielen, jota voit hyödyntää ja ajaa kaikissa käyttöjärjestelmissä.

Uusi käsite

"C++ Trainer" ei tee mitään olettamuksia käyttämästäsi käyttöjärjestelmästä. Tämä teos kattaa ANSI C++ -kielen, joka on siis standardoitu C++. ANSI C++ -kielen mukainen ohjelma on siirrettävissä eri alustoille ja kehitysympäristöihin.

Sen sijaan tässä kirjassa ei puhuta ikkunoista, luetteloruuduista, grafiikasta tms. Kaikki nuo komponentit ovat riippuvia käyttöjärjestelmästä. Ohjelmien tulostukset tapahtuvat standarditulostuksina. Standarditulostuksen saat aikaan luomalla kääntäjälläsi konsolisovelluksen. Jotkut kääntäjät, jotka on tarkoitettu ikkunointiympäristöihin (kuten Windows tai Mac), kutsuvat konsoli-ikkunaa pikaikkunaksi, yksinkertaiseksi ikkunaksi tai ehkäpä konsoli-ikkunaksi.

Kääntäjäsi ja editorisi

Kääntäjässäsi saattaa olla oma sisäänrakennettu editorinsa tai käytät kaupallista tekstieditoria taikka tekstinkäsittelyohjelmaa ohjelmien kirjoittamiseen. Tärkeintä on tietää, että ohjelmateksti tulee tallentaa aina puhtaana tekstinä, jossa ei siis ole mitään muotoiluja tai ohjauskoodeja. Hyviä esimerkkejä editoreista ovat Windows Muistio (Notepad), DOSin Edit-komento, Brief, Epsilon, EMACS, Pico ja vi. Monet kaupalliset tekstieditorit, kuten WordPerfect, Word ja monet muut mahdollistavat myöskin tekstin tallentamisen puhtaana tekstitiedostona.

Editorilla luotua tiedostoa kutsutaan lähdetiedostoksi ja sen tunnuksena on C++:n kohdalla tyypillisesti .CPP, .CP tai .C. Tämän teoksen kaikki esimerkkitiedostot ovat saaneet tarkenteekseen .CPP. Tarkista kääntäjästäsi, mitä tunnusta se käyttää.

Huom!

Useimmat C++ -kääntäjät eivät piittaa siitä, mikä on lähdetiedoston tunnus. Useimpien kääntäjien oletustunnuksena on kuitenkin .CPP.

Tee/Älä tee

Käytä yksinkertaista tekstieditoria lähdekoodin kirjoittamiseen tai hyödynnä kääntäjän mukana tulevaa editoria.
Älä käytä tekstinkäsittelyohjelmaa, joka tallentaa muotoilumerkit tiedostoon. Tallenna tällöin tiedosto ASCII-muodossa.
Käytä tiedoston tunnuksena tunnusta .CPP, .CP tai .C.
Tarkista kääntäjäsi ohjekirjoista tai on-line-ohjeesta, kuinka kääntäjäsi ja linkittäjäsi suhtautuvat tunnuksiin.

Lähdekoodin kääntäminen ja linkittäminen

Vaikka lähdekoodisi näyttää aluksi oudolta eikä kukaan C++ -kieltä taitamaton voi ymmärtää sitä, se on kuitenkin ihmisen ymmärtämässä muodossa. Lähdekooditiedostosi ei ole vielä ohjelma eikä sitä voida ajaa.

Huom! Lähdekoodin muuntamiseksi ohjelmaksi käytetään kääntäjää. Kääntäjän käyttäminen riippuu käytetystä työkalusta, joten tutki tarvittaessa ohjekirjoja.

Jos käännät tiedostosi käyttöjärjestelmän komentoriviltä, ovat yleiset komennot seuraavat:

Borlandin C++ -kääntäjä:
```
bc <tiedostonimi>
```

Borlandin C++ for Windows -kääntäjä
```
bcc <tiedostonimi>
```

Borlandin Turbo C++ -kääntäjä
```
tc <tiedostonimi>
```

Microsoftin kääntäjät
```
cl <tiedostonimi>
```

Kääntäminen integroidussa kehitysympäristössä

Useimmat nykyaikaiset kääntäjät tarjoavat IDE (Integrated Development Environmet) -ympäristön. Tällöin valikoista valitaan tyypillisesti sellaiset komennot kuin Build tai Compile tai käytetään toimintonäppäintä sovelluksen kääntämiseen. Tarkista nuo erityispiirteet oman työkalusi ohjeista.

Ohjelman linkittäminen

Kun lähdekoodi on käännetty, on luotu objektikoodi. Objektitiedoston tunnuksena on yleensä .OBJ. Sekään ei ole vielä ajettava ohjelma. Muuntaaksesi objektitiedoston ajettavaksi tiedostoksi on sinun käytettävä vielä linkittäjää.

C++ -ohjelmat on luotu pääosin linkittämällä yhteen yksi tai useampia OBJ-tiedostoja ja kirjastoja. Kirjasto on kokoelma luomiasi tiedostoja tai kääntäjän mukana tulleita funktioita tai erikseen hankittuja tiedostoja. Kaikkien C++ -kääntäjien mukana tulee joukko hyödyllisiä funktioita (tai proseduureja) sekä luokkia, joita voidaan sisällyttää ohjelmaan. Funktio on koodilohko, joka suorittaa jonkin palvelutoiminnon kuten yhteenlaskun tai

tulostuksen. Luokka on tietojen ja funktioiden yhteenliittymä ja aiheesta kerrotaan runsaasti myöhemmin.

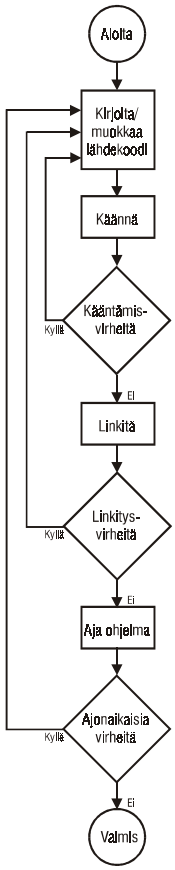
Suoritettava tiedosto luodaan seuraavasti:

1. Luodaan lähdekoodi, jolla on yleensä. CPP-tunnus.
2. Käännetään lähdekoodi. OBJ-tiedostoksi.
3. Linkitetään OBJ-tiedosto tarvittaviin kirjastoihin, jolloin saadaan suoritettava tiedosto.

Kehityskaavio

Jos jokainen ohjelma toimisi heti ensi yrittämällä, olisi koko kehitysjakso vaiheissa: ohjelman kirjoittaminen, lähdekoodin kääntäminen, linkittäminen ja ajaminen. Valitettavasti yksinkertaisimmissakin ohjelmissa on virheitä, bugeja. Jotkut virheet saavat ohjelman kääntämisen kaatumaan, jotkut keskeyttävät linkityksen ja jotkut taas tulevat esille vasta ajon aikana.

Kaikki virheet tulee tietenkin korjata vian tyypistä riippumatta. Tällöin on lähdekoodia muutettava ja suoritettava sen jälkeen uusi kääntäminen ja linkitys sekä koeajo. Koko kehitysjakso nähdään kuvassa 1.1.



Kuva 1.1. C++ -ohjelman kehittämisen vaiheet

HELLO.CPP - Ensimmäinen C++ -ohjelmasi

Klassiset ohjelmointioppaat alkavat esimerkillä, joka kirjoittaa Hello, World! näytölle tai jotain muuta vastaavaa. Tätä perinnettä jatketaan seuraavassakin.

Kirjoita ensimmäinen ohjelmasi suoraan editorillasi juuri siten kuin se on kirjassa esitettynä. Kun koodi on kirjoitettu, käännä se, linkitä ja aja se. Ohjelma tulostaa tekstin Hello, World! näytöllesi. Älä vielä tässä vaiheessa pohdi liikaa sitä, kuinka ohjelma toimii. Esimerkki on tarkoitettu vain kehitysvaiheiden kokeilemiseen. Ohjelman kaikki osat tulevat kyllä käytyä läpi myöhemmin.

Huom! Seuraavassa listauksessa on rivinumerot. Ne ovat vain selitysviittauksia varten eikä sinun tule kirjoittaa niitä. Esimerkiksi rivillä 1 tulee kirjoittaa pelkästään:

```
#include <iostream.h>
```

Listaus 1.1. HELLO.CPP, klassinen Hello World! -ohjelma.

```
1:  #include <iostream.h>
2:
3:  int main()
4:  {
5:    cout << "Hello World!\n";
6:    return 0;
7:  }
```

Varmista, että kirjoitat ohjelman juuri kuten edellä. Kirjoita kaikki merkit kuten kirjassakin. Rivillä 5 olevat <<-merkit ovat uudelleenohjaussymboleita. Muista laittaa myös puolipisteet vaadittujen rivien loppuun!

Ole myös tarkkana, että toimit kääntäjäsi sääntöjen mukaan. Useimmat kääntäjät suorittavat linkitykset automaattisesti, mutta tarkista asia ohjeista. Jos saat virheilmoituksia, tutki koodi tarkasti ja vertaa sitä kirjan koodiin. Jos virheilmoitus liittyy riviin 1 (esimerkiksi cannot find file iostream.h), katso kääntäjäsi ohjeista, kuinka include-polku ja ympäristömuuttujat asetetaan. Jos saat virheen main-prototyypin puuttumisesta, lisää koodi int main(); rivin 3 edelle. Useimmat kääntäjät eivät vaadi tuota ylimääräistä riviä.

Ohjelma on seuraavanlainen:

```
 1:  #include <iostream.h>
 2:
 3:  int main()
 4:  {
 5:    cout << "Hello World!\n";
 6:    return 0;
 7:  }
```

Kun ajat ohjelman HELLO.EXE, se tulostaa näytölle tekstin

Hello World!

Onnitteluni! Olet juuri saanut aikaan ensimmäisen C++ -ohjelmasi. Ohjelma ei näytä kovinkaan edistyneeltä, mutta hyvin monet ammattiohjelmoijat ovat aloittaneet uransa juuri tällä ohjelmalla.

Käännönaikaiset virheet

Kääntämisen aikana ilmenevät virheet voivat johtua monesta eri syystä. Useimmiten ne ovat kirjoitusvirheitä tai johtuvat muista pikkuvirheistä. Hyvä kääntäjä ei pelkästään kerro, että virhe on tapahtunut, vaan osoittaa virheen tarkan sijainnin. Huippukääntäjät voivat antaa jopa korjausehdotuksen!

Voit kokeilla kääntäjääsi laittamalla virheen ohjelmaasi. Jos HELLO.CPP voitiin ajaa moitteettomasti, muokkaa sitä nyt poistamalla lopettava aaltosulku riviltä 7. Ohjelmasi on nyt listauksen 1.2 kaltainen.

Listaus 1.2. Kääntämisvirheen esittely.

```
8:  #include <iostream.h>
9:
10: int main()
11: {
12:   cout << "Hello World!\n";
13:   return 0;
```

Käännä ohjelma uudelleen ja saat seuraavanlaisen virheilmoituksen:

```
Hello.cpp, Line 5; Compound statement missing terminating } in
function main().
```

Virheilmoitus kertoo tiedoston ja rivinumeron sekä ongelman (vaikkakin se on hieman vaikeaselkoinen). Huomaa, että virheilmoitus kertoo virheestä rivillä 5. Kääntäjä ei ollut varma siitä, aioitko laittaa lopettavan aaltosulun ennen vai jälkeen rivin 5 cout-lausetta. Joskus virheilmoitus antaa vain yleiskäsityksen ongelmasta. Jos kääntäjä osaisi identifioida jokaisen ongelman täydellisesti, se voisi korjata koodinkin itse.

Yhteenveto

Luettuasi tämän luvun sinun tulisi tietää C++ -kielen kehittymisestä ja siitä, millaisiin ongelmiin sen haluttiin tuovan ratkaisun. Sinun tulisi nyt olla vakuuttunut siitä, että C++ -kielen opiskeleminen on oikea valinta jokaiselle ohjelmoinnista kiinnostuneelle tulevina vuosinakin. C++ tarjoaa työkalut oliopohjaiseen ohjelmointiin ja samalla järjestelmätason kielen suorituskyvyn.

Tässä aivan ensimmäisessä luvussa opit kirjoittamaan, kääntämään, linkittämään ja ajamaan ensimmäisen C++ -ohjelmasi. Sait samalla kuvan koko kehittämisjaksosta. Sait myös hieman tietoa oliopohjaisesta ohjelmoinnista. Palaamme näihin aiheisiin seuraavissa luvuissa.

Kysymyksiä ja vastauksia

K
Mikä on tekstieditorin ja tekstinkäsittelijän ero?
V
Tekstieditori tuottaa tiedoston, joka sisältää vain puhdasta tekstiä. Tekstissä ei ole mitään muotoilukomentoja tai muita erikoissymboleita, joita tekstinkäsittelyohjelmat käyttävät. Tekstitiedostoissa ei ole automaattisia rivinvaihtoja, lihavointia, kursivointia tms. Tekstinkäsittelyohjelmat voivat tavallisesti tuottaa tekstitiedostoja, kunhan käyttäjä vain valitsee tallennusmuodoksi ascii-muodon tai puhtaan tekstimuodon.

K
Kääntäjässäni on sisäänrakennettu editori; tulisiko sitä käyttää?
V
Melkein kaikki kääntäjät kääntävät millä tahansa editorilla tuotettua koodia. Sisäisen editorin käyttämisen etuna on yleensä työskentelyn nopeutuminen esimerkiksi siksi, että kirjoittamisesta voidaan nopeasti siirtyä kääntämiseen ja päinvastoin. Kehittyneissä kääntäjissä on täysin integroitu kehitysympäristö. Samassa ympäristössä voidaan kirjoittaa koodi, kääntää ja linkittää se sekä ajaa ohjelmaa. Ympäristö tarjoaa myös ohjeet sekä vianhakutyökalut.

K
Voinko ohittaa kääntäjäni antamat varoitusilmoitukset?
V
Ei missään tapauksessa. Käsittele aina ilmoituksia virheinä. C++ käyttää kääntäjää varoittamaan asioista, joita et ehkä aikonut toteuttaa; ota vaarin näistä varoituksista ja toteuta vastaavat parannukset.

K
Mitä tarkoittaa kääntämisenaikainen?
V
Kääntämisenaikaisuus tarkoittaa aikaa, jolloin ajat kääntäjää erotuksena esimerkiksi linkittämiselle (jolloin ajat linkittäjää) tai ajonaikaisuudelle (jolloin ajat ohjelmaa).