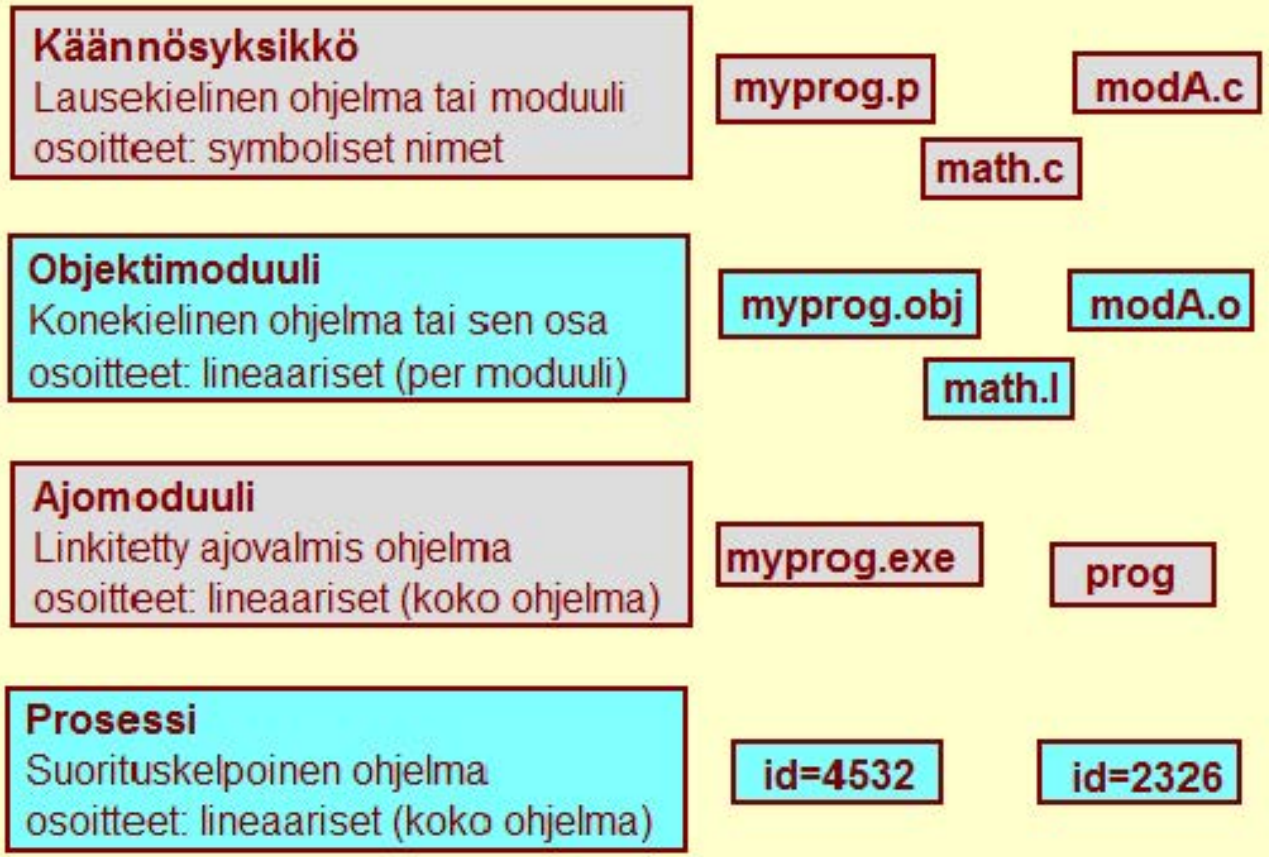
Käännös, linkitys ja lataus

- Käännös
- Linkitys
- Dynaaminen linkitys
- Lataus

Copyright Teemu Kerola 2005

Tässä luvussa tutustutaan kääntämiseen, linkittämiseen ja lataamiseen. Kääntäminen, linkittäminen ja lataaminen ovat käyttöjärjestelmän peruspalikoita, joiden avulla korkean tason kielen ohjelmista saadaan suorituskelpoisia ohjelmia. Valitse valikosta haluamasi aihepiiri ja/tai sivu sen sisällä.

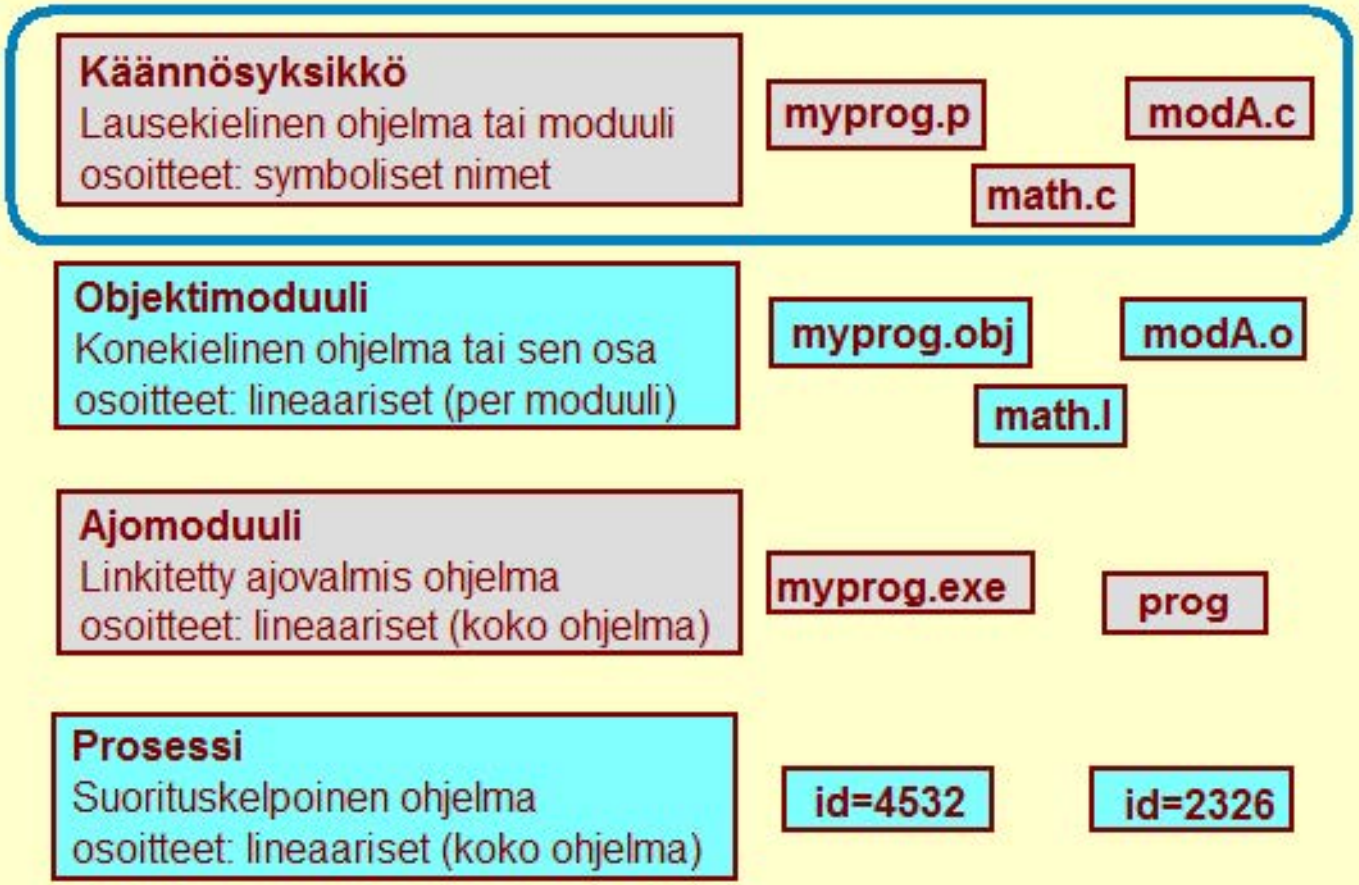
Käännösyksiköstä prosessiin



Copyright Teemu Kerola 2005

Perusongelma on miten kuvata yksinkertaisesti suorituskelpoinen ohjelma. Ohjelmien kuvaaminen on suhteellisen helppoa käyttäen sitä varten suunniteltuja korkean tason kieliä. Nämä korkean tason kielien avulla tehdyt ohjelmien määrittelyt muunnetaan sitten useamman vaiheen avulla suorituskelpoiseksi ohjelmaksi eli prosessiksi. Ohjelmat kuvataan siis yleensä korkean tason kielellä ja tällaista ohjelman tai sen moduulin kuvausta sanotaan käännösyksiköksi.

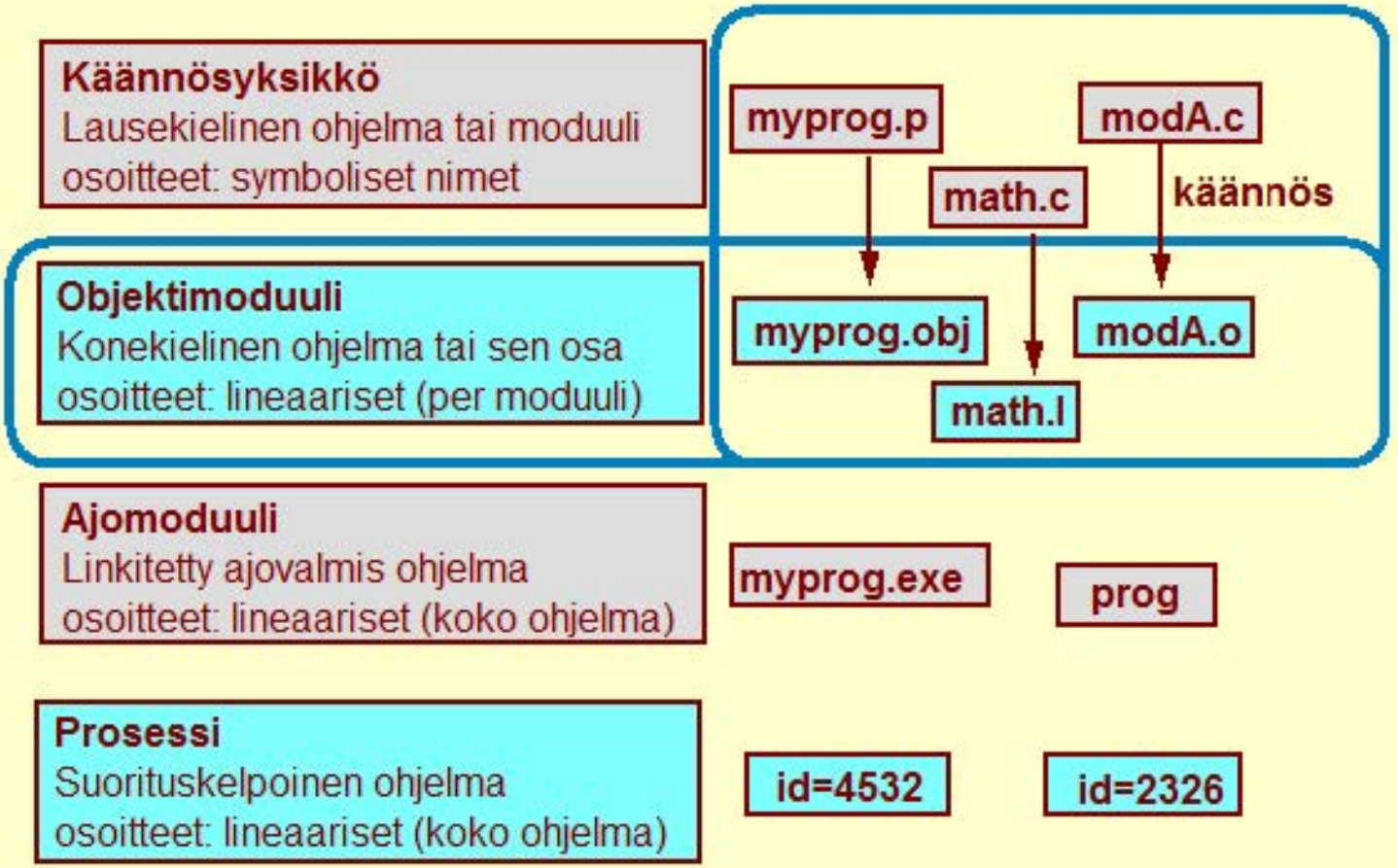
Käännösyksiköstä prosessiin



Copyright Teemu Kerola 2005

Käännösyksikkö on lausekielinen ohjelma tai ohjelman osakokonaisuus, joka aina käännetään yhdellä kertaa. Käännösyksikön tiedostonimen loppuosa ilmentää yleensä käytettyä ohjelmointikieltä. Käännösyksikön osoitteet voidaan ilmaista tavanomaisilla symbolisilla nimillä.

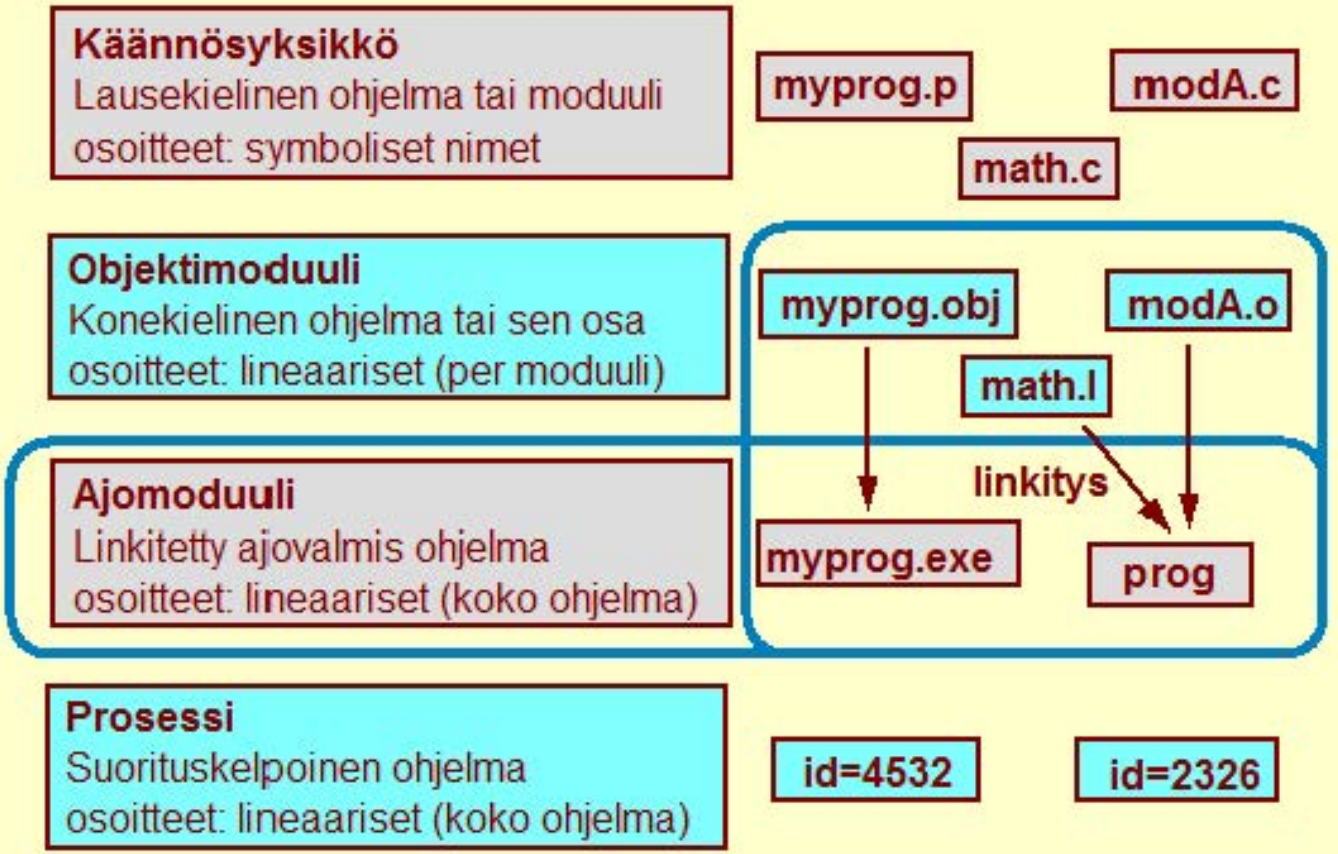
Käännösyksiköstä prosessiin



Copyright Teemu Kerola 2005

Objektimoduuli saadaan kun yksi tai useampi käännösyksikkö käännetään konekielelle. Tavalliset objektimoduulit tunnistetaan käyttöjärjestelmästä riippuen .obj tai .o loppuliitteistä, kun taas yleiskäyttöisemmillä kirjastomoduuleilla on omat loppuliitteensä. Ohjelman kääntäminen tarkoittaa siis yleensä korkean tason lausekielellä kuvatun ohjelman esitysmuodon muuttamista laitteiston ymmärtämään konekieliseen muotoon.

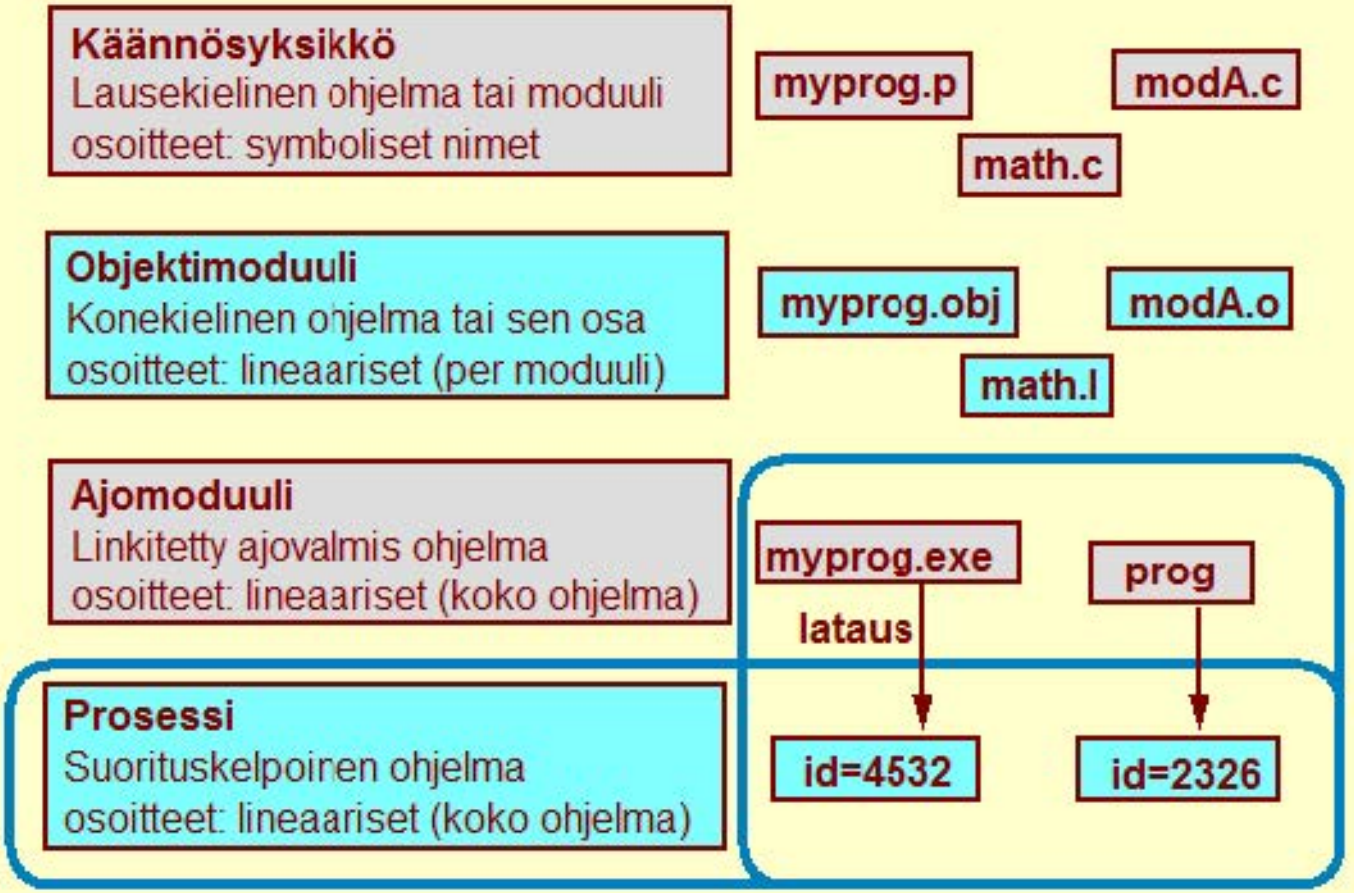
Käännösyksiköstä prosessiin



Copyright Teemu Kerola 2005

Ajomoduuli saadaan linkittämällä yksi tai useampi objektimoduuli sekä ohjelmointikielten että käyttöjärjestelmään kuuluvien kirjastomoduulien kanssa. Linkitys tarkoittaa siis objektimoduulien ja kirjastomoduulien yhdistämistä ajomoduuliksi siten, että sinne ei jää puutteellisia viittauksia muualle.

Käännösyksiköstä prosessiin



Copyright Teemu Kerola 2005

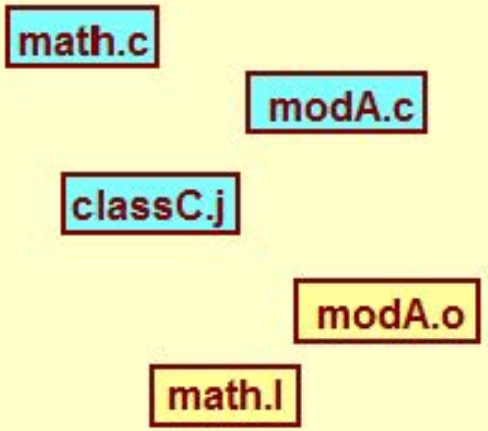
Prosessi on suorituksessa oleva ohjelma. Käyttöjärjestelmä luo prosessin ohjelman ajomoduulista sen lataamisen yhteydessä. Ohjelman lataaminen tarkoittaa siis ohjelman yhdestä suorituskerrasta vastaavan prosessin luomista jo olemassaolevan ajomoduulin perusteella.

Käännösyksikkö

Mikä on käännösyksikkö?

Jollain ohjelmointikielellä kuvattu eheä kokonaisuus, joka halutaan aina kääntää yhdessä.

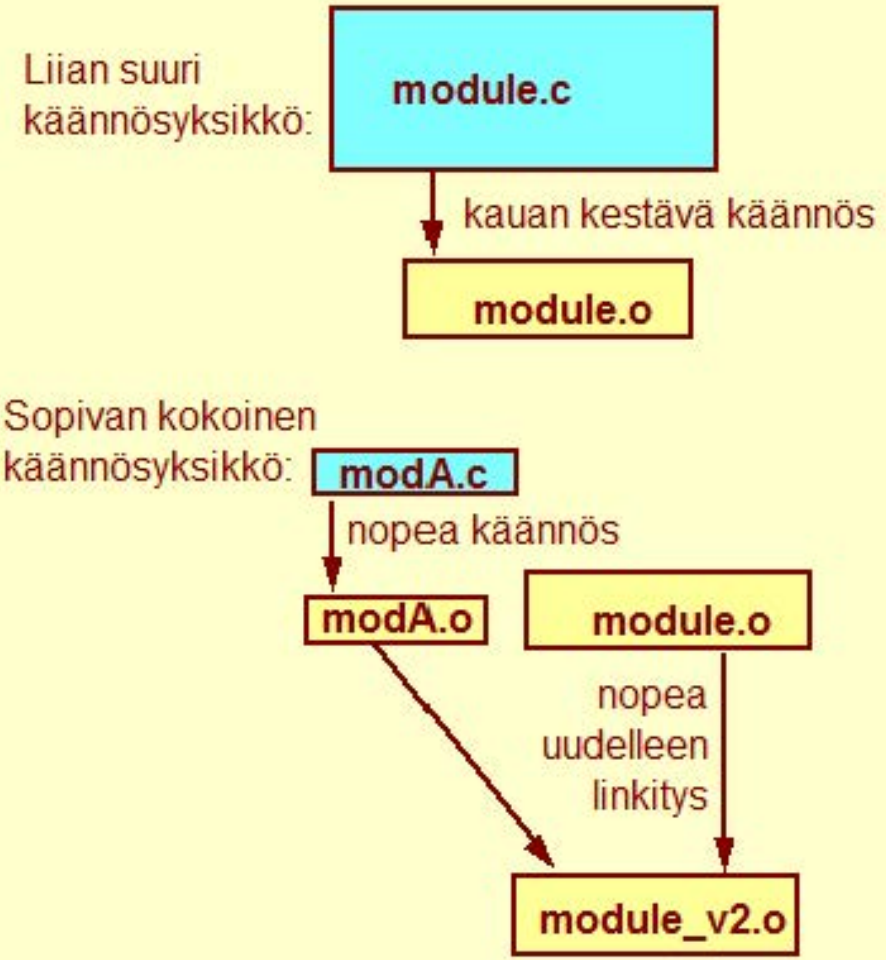
Esimerkiksi, olioperustainen luokka tai jonkun muun perusteella yhteen liittyvät aliohjelmat.



Copyright Teemu Kerola 2005

Käännösyksikkö on jokin ohjelmoijan määrittelemä eheä kokonaisuus, joka aina halutaan kääntää yhdellä kertaa. Se voi olla vaikkapa abstraktin tietorakenteen (esimerkiksi pinon) toteuttava aliohjelmien joukko tai olioperustainen luokka. Osa käännösyksikön aliohjelmista voi olla moduulin ulkopuolelle näkyviä ja osa voi olla moduulin sisäistä käyttöä varten.

Käännösyksikkö



Liian suuri käännösyksikkö?

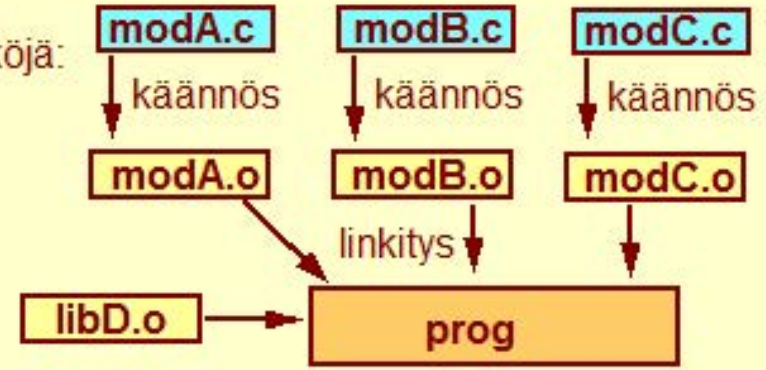
Turhaa aikaa kääntämiseen joka muutoksen jälkeen.

Copyright Teemu Kerola 2005

Jos käännösyksikkö on hyvin suuri, niin aikaa tuhlautuu sen kääntämiseen jokaisen pienen muutoksen jälkeen. Esimerkiksi, 50000 rivin moduulin kääntämiseen voi kulua useita sekunteja tai minuutteja. Olisi paljon nopeampaa kääntää pieni 500 rivin moduuli ja linkittää se uudelleen paikalleen.

Käännösyksikkö

Liian pieniä käännösyksikköjä:

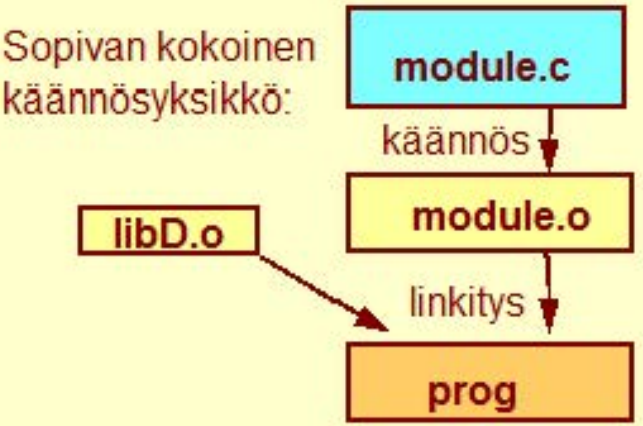


Liian pieni kokonaisuus?

Turhaa suunnittelu- ja koodausaikaa yhteyksien määrittelemiseen

Turhaa koneaikaa linkittämiseen

Sopivan kokoinen käännösyksikkö:



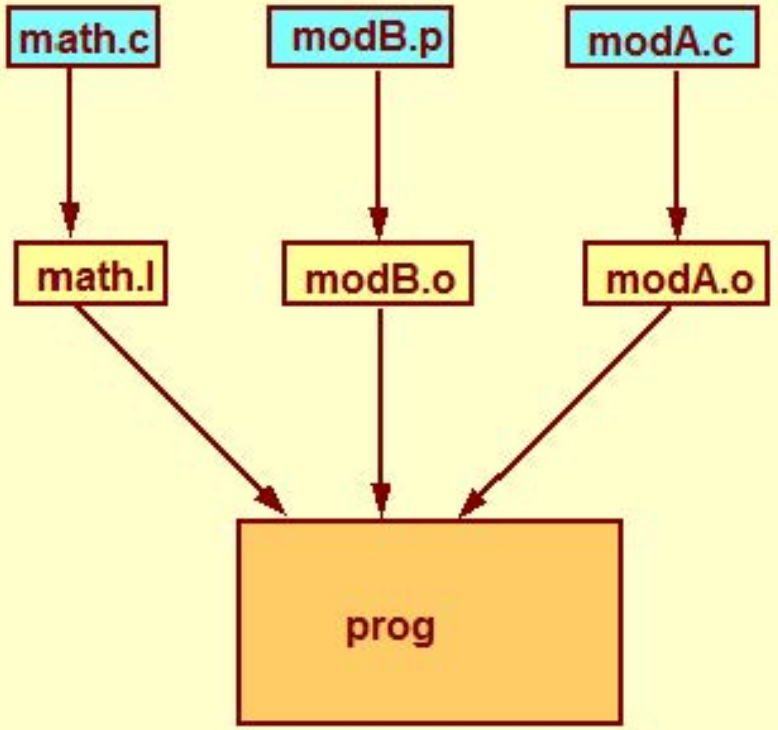
Copyright Teemu Kerola 2005

Jos käännösyksikkö on kovin pieni, niin se kyllä kääntyy nopeasti, mutta sen yhteyksien määritteleminen muihin moduuleihin voi olla monimutkaista. Esimerkiksi, voi olla kätevämpää kääntää ja linkittää yksi 5000 rivin moduuli kuin määritellä 10 kappaletta 500 rivin moduuleita keskinäisine riippuvuuksineen ja sitten kääntää ja linkittää ne keskenään.

Käännösyksikkö

Käännösyksikön ohjelmointikieli ei ole tärkeä

Niiden sitominen yhteen myöhemmin tapahtuu objektimoduulien tasolla

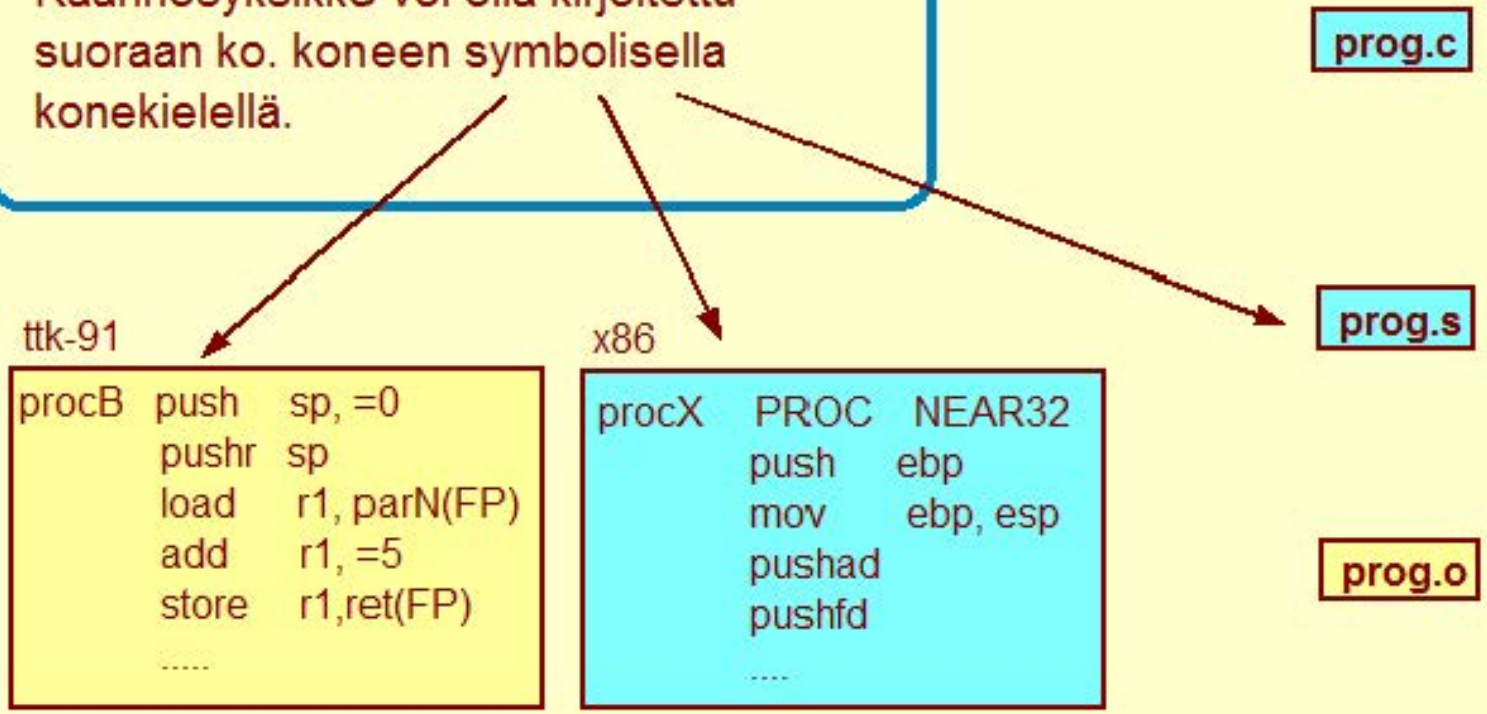


Copyright Teemu Kerola 2005

Moduulin ohjelmointikieli ei ole tärkeä, koska moduulin jatkokäsittely kääntämisen jälkeen tapahtuu ohjelmointikielistä riippumattomien objektimoduulien tasolla.

Assembler-kielinen käännösyksikkö

Käännösyksikkö voi olla kirjoitettu suoraan ko. koneen symbolisella konekielellä.

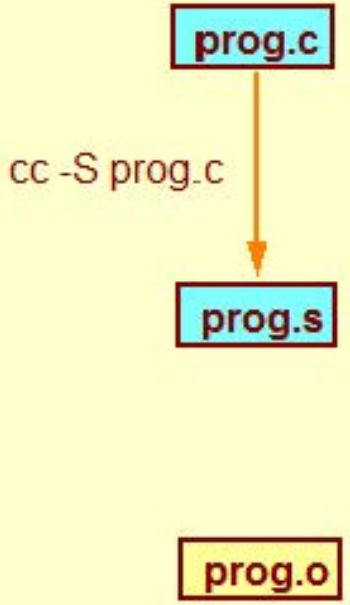


Copyright Teemu Kerola 2005

Käännösyksikkö voi olla myös kirjoitettu käyttäen suorittimen omaa symbolista konekieltä eli sen assembler-kielellä.Tällainen ohjelman osan kuvaus on hyvin täsmällinen, koska suorittimen symbolinen konekieli on niin lähellä todellista konekieltä. Symbolinen konekieli ei ole kuitenkaan helposti siirrettävää, koska se on tarkoitettu vain yhdenlaiseen, juuri tämän suorittimen arkkitehtuuriin.

Assembler-kielinen käännösyksikkö

Assembler käännösyksikkö voidaan generoida suoraan korkean tason kielen kääntäjän avulla, sopivaa optiota käyttäen.

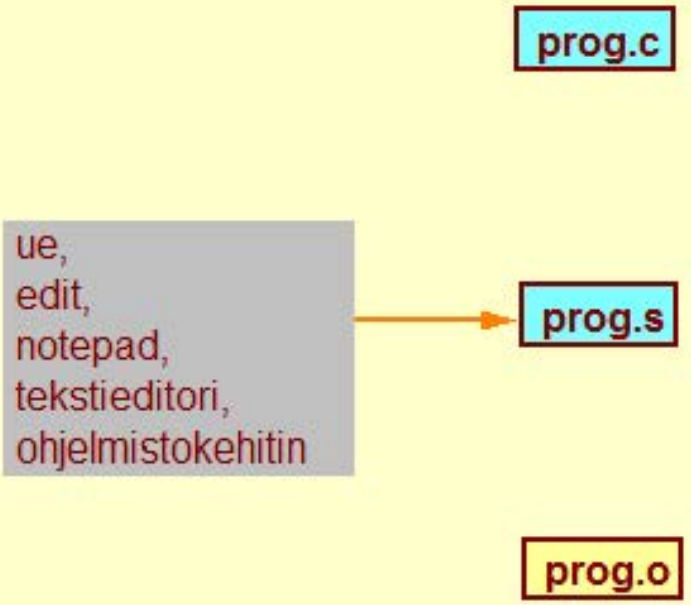


Copyright Teemu Kerola 2005

Assembler-kielinen käännösyksikkö voidaan tuottaa automaattisesti korkean tason kielen ohjelmasta tämän kielen oman kääntäjän avulla. Esimerkiksi C-kääntäjää käyttäen option -S avulla saadaan halutusta ohjelmasta assembler-kielinen versio. Assembler-versiosta nähdään korkean tason kielistä versiota paremmin, mitä todella tapahtuu laitteistossa ohjelmaa suoritettaessa.

Assembler-kielinen käännösyksikkö

Assembler-kielinen käännösyksikkö voidaan myös generoida käsin tavallisella tekstieditorilla, tai kääntäjän generoimaa assembler-kielistä käännösyksikköä voidaan "hienosäätää" käsin editoimalla.



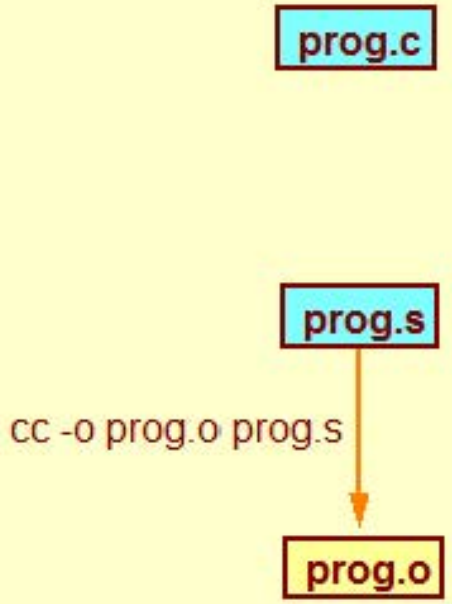
Copyright Teemu Kerola 2005

Assembler-kieliset käännösyksiköt voi myös generoida suoraan tekstieditorilla, vaikka tämä onkin aika työlästä. Useimmiten on helpompaa antaa kääntäjän ensin generoida assembler-kielinen aliohjelman runko ja sitten käsin virittää se optimaaliseksi, esimerkiksi suoritusnopeuden suhteen. Nykyiset kääntäjät ovat tosin niin hyviä, että niiden päihittäminen vaatii todella lahjakkaan ohjelmoijan.

Assembler-kielinen käännösyksikkö

Assembler-kieliset käännösyksiköt käännetään konekielelle assembler-kääntäjän avulla ennen linkitys vaihetta.

Assembler-kääntäjä sisältyy yleensä korkean tason kielen kääntäjiin.



Copyright Teemu Kerola 2005

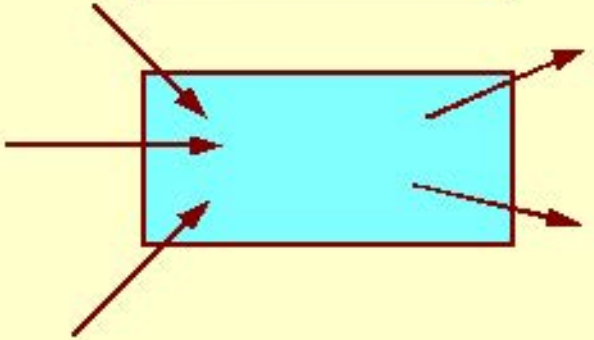
Assembler-kieliset käännösyksiköt käännetään muiden käännösyksiköiden tapaan konekielelle ennen linkitysvaihetta. Käännöksen tekee erityinen assembler-kääntäjä, joka on yleensä yhdistetty korkean tason kielen kääntäjään. Esimerkiksi C-kääntäjälle voi antaa sekä C-kielisiä että assembler-kielisiä käännösyksiköitä käännettäväksi. Esimerkkinä käyttämäämme ttk-91 simulaattoriin sisältyy ttk-91 assembler-kielen assembler-kääntäjä.

Objektimoduuli

Komponentit

Konekielinen koodi

Moduulien linkittämistä varten tarvittavat tiedot

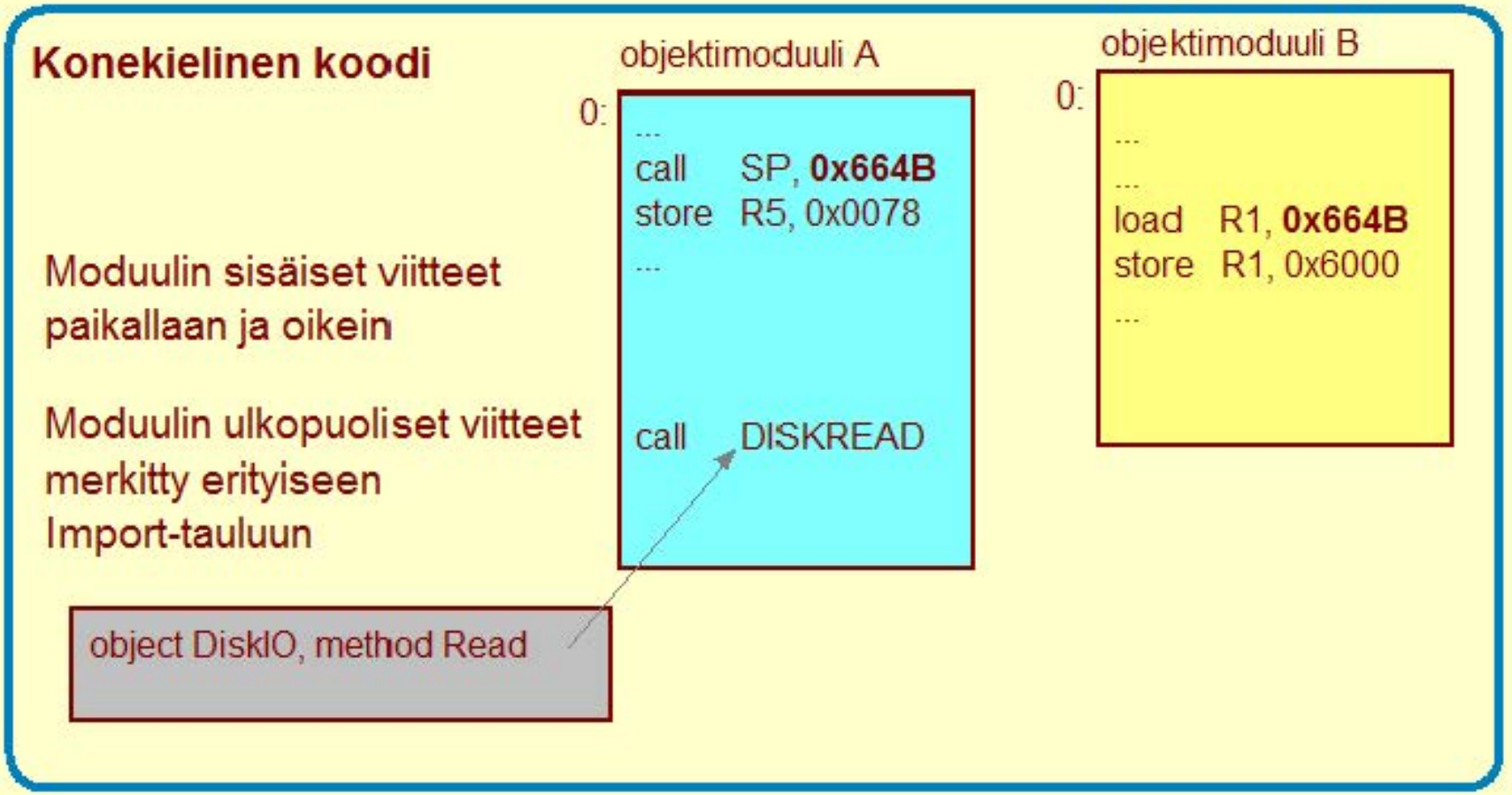


- uudelleensijoitustaulu
- import taulu
- export taulu
- symbolitaulu

Copyright Teemu Kerola 2005

Objektimoduuli on kääntäjän tai assembler-kääntäjän tuottama. Se koostuu konekielisestä koodista ja niistä kaikista tiedoista, joita tarvitaan tämän moduulin linkittämiseksi muihin moduuleihin. Koodi on siis todellista binaarikoodia eikä symbolista konekieltä. Linkitystiedot annetaan erilaisten taulukoiden muodossa.

Objektimoduuli



Copyright Teemu Kerola 2005

Konekielisessä koodissa kaikki moduulin sisäiset viitteet ovat oikein. Ongelmana jatkon kannalta on kuitenkin moduulin oma, nollasta alkava lineaarinen muistiavaruus, mikä on tietenkin samanlainen kaikilla moduuleilla. Esimerkiksi, kaikilla moduuleilla on osoite 0x0000664B käytössä, mutta eri moduuleilla kussakin omaan tarkoitukseensa. Kaikki viittaukset moduulin ulkopuolelle on jollain tavoin merkitty linkitysaikaisia jatkotoimenpiteitä varten.

Objektimoduuli

Linkitystä varten tarvittavat taulut

Uudelleensijoitustaulu (Relocation table)

Tiedot kaikista osoitteista, jotka täytyy päivittää, jos tämän moduulin osoiteavaruus yhdistetään jonkin toisen moduulin kanssa.

Viittaukset tästä muualle -taulu (Import table)

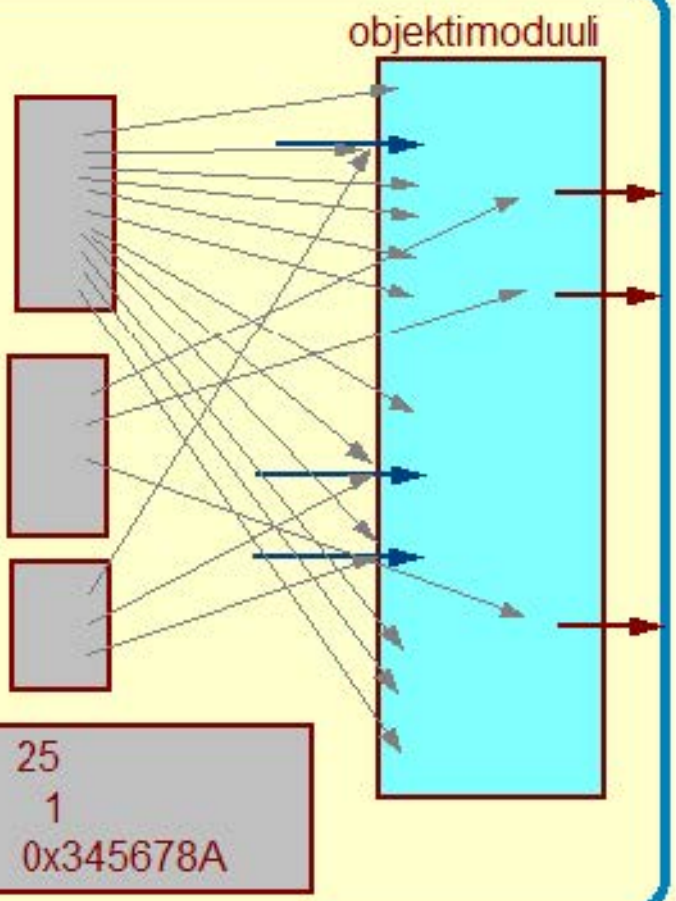
Tiedot kaikista kohdista, joissa viitataan joissakin muissa moduuleissa olevaan dataan tai koodiin.

Viittaukset muualta tähän -taulu (Export table)

Tiedot kaikista data- tai koodiviitteistä, joihin voi viitata muista moduuleista.

Symbolitaulu (Symbol table)

Kaikki tässä moduulissa olevat symbolit.



Copyright Teemu Kerola 2005

Uudelleensijoitustaulussa on kaikk viittaukset tämän moduulin omiin osoitteisiin. Ne ovat moduulin omassa osoiteavaruudessa ja ne täytyy kaikki päivittää osoiteavaruuksia yhdisteltäessä. Import-taulussa on listattu kaikki muihin moduuleihin kohdistuvat viittaukset ja ne pitää kaikki saada kuntoon linkityksessä. Export-taulussa annetaan kaikki muiden moduulien käyttöön annettavat data- ja koodiviitteet - kaikki muut osoitteet ovat vain omaan käyttöön. Symbolitaulussa on eriteltynä kaikki käytetyt symbolit ja niiden arvot.

Symbolitaulu

Joka käännösyksiköllä oma

Kääntäjä generoi

Ylläpidetään linkityksen aikana

Jätetään yleensä pois valmiista ohjelmasta

Pidetään joskus mukana valmiissa suorituskelpoisessa ohjelmassa käyttäjäläheisten virheilmoitusten tekemistä varten

Pidetään joskus mukana dynaamista linkitystä varten

if	keyw	56
then	keyw	57
main	code	0x01CC
Sub1	code	0x04A0
X	data	0x04B6
Apu	data	0x04BA
Five	int	5
Print	code	?

Copyright Teemu Kerola 2005

Symbolitaulu on siis kääntäjän generoima taulukko, jossa on yksi rivi tai entry jokaista käännösyksikön symbolia varten. Osa symboleista on jo etukäteen tunnettuja (kuten esim. symbolit 'if' ja 'then'), mutta suurin osa on ohjelmoijan määrittelemiä. Joka käännösyksiköllä on oma symbolitaulunsa. Linkityksen aikana linkitettävien moduulien symbolitaulut yhdistetään.

Symbolitaulu

Joka käännösyksiköllä oma

Kääntäjä generoi

Ylläpidetään linkityksen aikana

Jätetään yleensä pois valmiista ohjelmasta

Pidetään joskus mukana valmiissa suorituskelpoisessa ohjelmassa käyttäjäläheisten virheilmoitusten tekemistä varten

Pidetään joskus mukana dynaamista linkitystä varten

if	keyw	56
then	keyw	57
main	code	0x01CC
Sub1	code	0x04A0
X	data	0x04B6
Apu	data	0x04BA
Five	int	5
Print	code	?

Copyright Teemu Kerola 2005

Symbolitaulu jätetään yleensä pois valmiista ohjelmasta, jolloin kaikki symbolit ohjelmassa on korvattu niiden arvoilla. Symbolitaulua ei varsinaisesti enää tarvita, joten sen mukana pitäminen vain kasvattaisi suoritettavan ajomoduulin kokoa. Ohjelmiston kehitysaikana symbolitaulu kuitenkin kannattaa vielä pitää mukana, jotta virheilmoitukset olisivat mielekkäämpiä ohjelmoijille. Virheilmoitukseen 'Muuttujan Apu5 ylivuoto' on helpompi reagoida kuin virheilmoitukseen 'muuttujan 0x34A5 ylivuoto moduulissa 0x54'.

Korkean tason lähdekieli vs. konekieli

```
Pascal:    N := I + J;

C:         N = I + J

Java:      N = I + J;
```

ttk-91

```
I       DC  3
J       DC  4
N       DC  0

Doit    LOAD    R1, I
        ADD     R1, J
        STORE   R1, N
```

Intel Pentium II

```
Doit:   MOV     EAX, I
        ADD     EAX, J
        MOV     N, EAX

I       DW      3
J       DW      4
N       DW      0
```

Motorola 680x0

```
Doit    MOVE.L  I, D0
        ADD.L   J, D0
        MOVE.L  D0, N

I       DC.L    3
J       DC.L    4
N       DC.L    0
```

Sun SPARC

```
Doit:   SETHI   %HI(I),%R1
        LD      [%R1+%LO(I)],%R1
        SETHI   %HI(J),%R2
        LD      [%R2+%LO(J)],%R1
        NOP
        ADD     %R1,%R2,%R2
        SETHI   %HI(N),%R1
        ST      %R2,[%R1+%LO(N)]

I:      .WORD 3
J:      .WORD 4
N:      .WORD 0
```

[Tane06, Fig. 7-2]

Copyright Teemu Kerola 2005

Korkean tason lohkorakenteiset kielet ovat hyvin samankaltaisia, vaikka kielissä onkin merkittäviä eroja. Esimerkiksi oheiset Pascal, C ja Java sijoituslausekkeet ovat liki samanlaisia sekä syntaksiltaan (ulkoasu) että semantiikaltaan (merkitys). Niitä vastaavat konekieliset esitysmuodot ovat siten hyvin samanlaisia.

Korkean tason lähdekieli vs. konekieli

```
Pascal:   N := I + J;

C:        N = I + J

Java:     N = I + J;
```

ttk-91

```
I       DC  3
J       DC  4
N       DC  0

Doit    LOAD    R1, I
        ADD     R1, J
        STORE   R1, N
```

Intel Pentium II

```
Doit:   MOV     EAX, I
        ADD     EAX, J
        MOV     N, EAX

I       DW      3
J       DW      4
N       DW      0
```

Motorola 680x0

```
Doit    MOVE.L  I, D0
        ADD.L   J, D0
        MOVE.L  D0, N

I       DC.L    3
J       DC.L    4
N       DC.L    0
```

Sun SPARC

```
Doit:   SETHI   %HI(I),%R1
        LD      [%R1+%LO(I)],%R1
        SETHI   %HI(J),%R2
        LD      [%R2+%LO(J)],%R1
        NOP
        ADD     %R1,%R2,%R2
        SETHI   %HI(N),%R1
        ST      %R2,[%R1+%LO(N)]

I:      .WORD 3
J:      .WORD 4
N:      .WORD 0
```

[Tane06, Fig. 7-2]

Copyright Teemu Kerola 2005

Symbolisten konekielten osalta erot ovat eri kielten välillä suurempia, koska kukin kieli on sovitettu kyseisen koneen arkkitehtuuriin. ttk-91 koneen konekieli on kuitenkin hyvin edustava ja vastaa aika lailla esimerkiksi Intel Pentiumin ja Motorolan konekieliä. Motorolan konekielessä tulosrekisteri on tosin viimeisenä ja näissä molemmissa muuttujien tilanvaraus on tyypitetty.

Korkean tason lähdekieli vs. konekieli

```
Pascal:    N := I + J;

C:         N = I + J

Java:      N = I + J;
```

ttk-91

```
I       DC  3
J       DC  4
N       DC  0

Doit    LOAD    R1, I
        ADD     R1, J
        STORE   R1, N
```

Intel Pentium II

```
Doit:   MOV     EAX, I
        ADD     EAX, J
        MOV     N, EAX

I       DW      3
J       DW      4
N       DW      0
```

Motorola 680x0

```
Doit    MOVE.L  I, D0
        ADD.L   J, D0
        MOVE.L  D0, N

I       DC.L    3
J       DC.L    4
N       DC.L    0
```

Sun SPARC

```
Doit:   SETHI   %HI(I),%R1
        LD      [%R1+%LO(I)],%R1
        SETHI   %HI(J),%R2
        LD      [%R2+%LO(J)],%R1
        NOP
        ADD     %R1,%R2,%R2
        SETHI   %HI(N),%R1
        ST      %R2,[%R1+%LO(N)]

I:      .WORD 3
J:      .WORD 4
N:      .WORD 0
```

[Tane06, Fig. 7-2]

Copyright Teemu Kerola 2005

Sunin SPARC-arkkitehtuuri on hyvin erilainen ja siinä kaikki konekäskyt ovat lyhyitä, mutta niitä tarvitaan enemmän. Esimerkiksi muuttujan lataaminen muistista vaatii kaksi konekäskyä. SETHI-käsky asettaa osoiteosan 16 eniten merkitsevää bittiä rekisteriin R1 ja sitten LD (tai ST) käsky muodostaa 32 bittisen muistiosoitteen ja suorittaa itse muistiviittauksen. Vaikka käskyjä on kaksi kappaletta, ne suoritetaan itse asiassa pääosin samaan aikaan limittäin ja kokonaisaika on hyvin vertailukelpoinen Intelin ja Motorolan arkkitehtuurien kanssa.

Kääntäjän ohjauskäskyt eli pseudokäskyt

Eivät ole varsinaista koodia (niistä ei generoidu konekäskyjä)

Ohjaavat käännöstä

```
Tbl     DS   25
TblLen  DC   25
One     EQU  1
N       EQU 78
        ....
        LOAD  R1, Tbl(R2)
        ....
        COMP  R2, TblLen

        LOAD  R4, =One
        LOAD  R5, =N
```

ttk-91: DC, DS, EQU

Pentium II: SEGMMENT, ENDS, ALIGN, EQU, DB, DD, DW, DQ,
PROC, ENDP, MACRO, ENDM, PUBLIC, EXTERN,
IF, THEN, ELSE
COMMENT, INCLUDE, PAGE, END

Copyright Teemu Kerola 2005

Korkean tason kielen tai assembler-kielen kääntäjän ohjauskäskyt sijoitetaan tavallisen koodin sekaan, vaikka niistä ei generoidukaan suoritettavaa koodia. Ttk-91 koneen ohjauskäskyillä varattiin muistitilaa ja annettiin symboleille tunnettuja arvoja. Todellisten koneiden ohjauskäskyvalikoima on suurempi, mutta yhtä käytännönläheinen.

Kääntäjän ohjauskäskyt eli pseudokäskyt

Eivät ole varsinaista koodia (niistä ei generoidu konekäskyjä)

Ohjaavat käännöstä

```
Tbl     DS   25
TblLen  DC   25
One     EQU  1
N       EQU 78
        ....
        LOAD  R1, Tbl(R2)
        ....
        COMP  R2, TblLen
        ....
        LOAD  R4, =One
        LOAD  R5, =N
```

ttk-91: DC, DS, EQU

Pentium II: SEGMMENT, ENDS, ALIGN, EQU, DB, DD, DW, DQ,
PROC, ENDP, MACRO, ENDM, PUBLIC, EXTERN,
IF, THEN, ELSE
COMMENT, INCLUDE, PAGE, END

Copyright Teemu Kerola 2005

Pentium II:n ohjauskäskyissä SEGMENT ja ENDS -käskyillä määritellään data tai koodisegmentti. ALIGN-käskyllä seuraava data-alkio saadaan tasa-osoitteeseen. Tilanvaraus voidaan tehdä eri kokoisille data-alkioille DB, DD, DW ja DQ -käskyillä. Aliohjelmien alku ja loppu on helppo hoitaa PROC ja ENDP käskyillä. Macrokäskyjen avulla on helppo toteuttaa usein toistuvia konekäskysarjoja. PUBLIC käskyllä julkistetaan tunnuksia muille ja EXTERN käskyllä saadaan muualla määriteltyjä tunnuksia käyttöön. IF-THEN-ELSE käskyillä voidaan generoida eri koodia eri arkkitehtuurin versioille.

Makrot

Helpottavat ohjelmointia

- kääntäjässä valmiiksi määritellyt makrot
- itse määritellyt makrot

```
SaveRegs
CallPrelude
FloatAdd
StringCat
```

Usein toistuva koodisarja voi olla nimetty makro

Makroilla voi olla (nimi)parametreja

Copyright Teemu Kerola 2005

Usein toistuville koodisarjoille voidaan antaa nimi, jolloin niitä kutsutaan makroiksi. Makroissa voi olla myös parametreja, joten niiden sovellusalue on aika laaja. Makrojen parametrit ovat yleisesti nimiparametreja, jolloin parametri korvataan siis tekstuaalisesti todellisen parametrin merkkijonoarvolla. Makrot käsitellään yleisesti ennen varsinaista käännöstä, joten niitä ei enää ole varsinaisessa käännöksessä. Osa makroista on jo valmiiksi määritelty kääntäjässä, mutta yleensä ohjelmoijat voivat määritellä itse omia makroja sitä varten olemassa olevilla kääntäjän ohjauskäskyillä.

Makrot

Helpottavat ohjelmointia

- kääntäjässä valmiiksi määritellyt makrot
- itse määritellyt makrot

Usein toistuva koodisarja voi olla nimetty makro

Makroilla voi olla (nimi)parametreja

Usein toistuva koodi (vaihda P:n ja Q:n arvot)

```
MOV   EAX, P
MOV   EBX, Q
MOV   Q, EAX
MOV   P, EBX
...
MOV   EAX, P
MOV   EBX, Q
MOV   Q, EAX
MOV   P, EBX
```

Sama makroa SWAP_PQ käyttäen

```
SWAP_PQ   MACRO
          MOV   EAX, P
          MOV   EBX, Q
          MOV   Q, EAX
          MOV   P, EBX
          ENDM
          ....
          SWAP_PQ
          ...
          SWAP_PQ
```

Copyright Teemu Kerola 2005

Jos muuttujien P ja Q arvo tulee usein vaihtaa koodissa, niin voi olla kätevää määritellä sitä varten oma makro SWAP_PQ. Makro SWAP_PQ määritellään kääntäjien ohjauskäskyjen MACRO ja ENDM avulla. Ennen varsinaista (assembler) käännöstä makrojen käyttökohdat korvataan makron rungolla, joten varsinaisen käännöksen alkaessa makrosta ei ole enää jälkeäkään. Makrojen kanssa tulee olla huolellinen. Esimerkiksi makroa SWAP_PQ käytettäessä ei näy päälle, että se muuttaa rekistereiden EAX ja EBX arvoja.

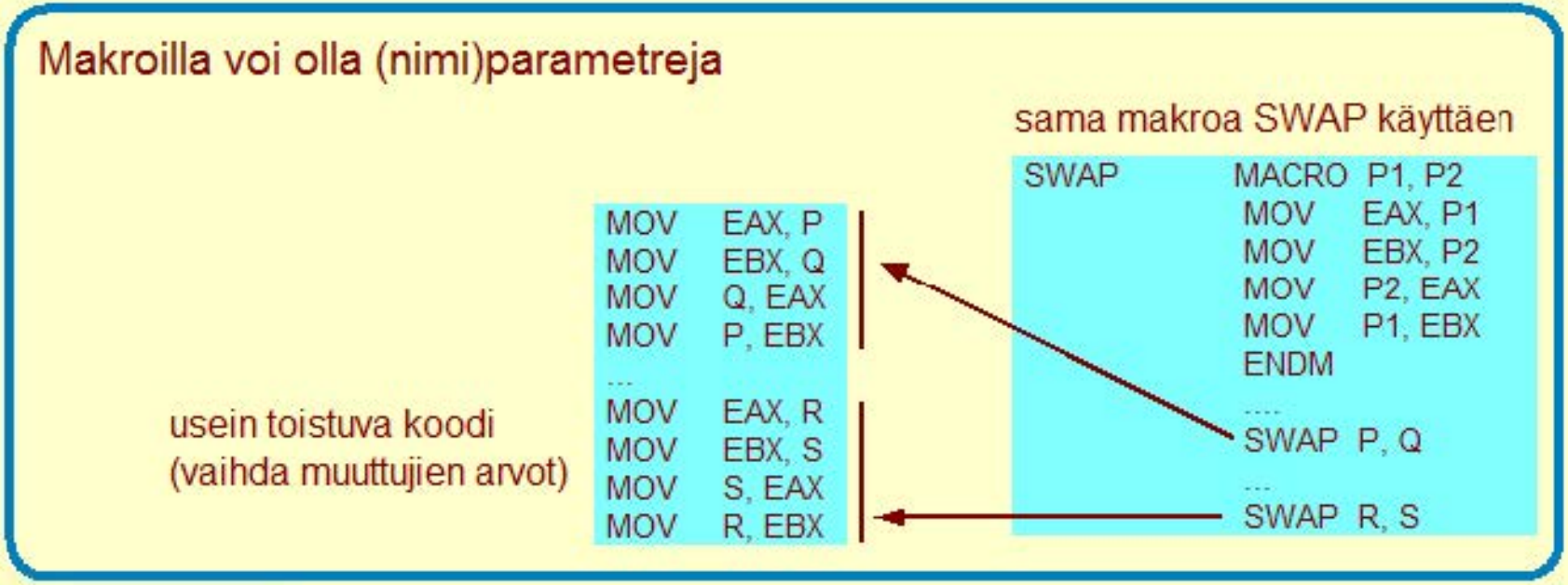
Makrot

Helpottavat ohjelmointia

- kääntäjässä valmiiksi määritellyt makrot
- itse määritellyt makrot

Usein toistuva koodisarja voi olla nimetty makro

Makroilla voi olla (nimi)parametreja



Copyright Teemu Kerola 2005

Makroista tulee paljon käyttökelpoisempia parametrien kanssa. Makrolla SWAP voidaan vaihtaa minkä tahansa kahden muuttujan arvot keskenään, tai jopa kahden rekisterin arvot keskenään, kunhan vain kumpikaan niistä ei ole rekisteri EAX tai EBX.

Makrot

Helpottavat ohjelmointia

- kääntäjässä valmiiksi määritellyt makrot
- itse määritellyt makrot

Usein toistuva koodisarja voi olla nimetty makro

Makroilla voi olla (nimi)parametreja

Makrot vs. aliohjelmat

	Makro	Aliohjelma
Milloin käsitellään?	Käännösaikana	Suoritusaikana
Monistetaanko koodi joka kutsukerralla?	Kyllä	Ei
Tarvitaanko kutsu/paluukäskyä?	Ei	Kyllä
Pienentääkö käyttö koodin kokoa?	Ei	Kyllä
Parametrien yleiset tyypit?	Nimi	Arvo, viite

Copyright Teemu Kerola 2005

Makrojen käyttökohdat on korvattu niiden rungolla käännöksen esivaiheessa, joten makroja ei ole enää olemassa suoritusaikana. Aliohjelmien kutsut sen sijaan käsitellään vasta kutsuhetkellä suoritusaikana. Makrot eivät pienennä koodin kokoa, mutta niiden käyttö nopeuttaa ohjelmointia. Makroissa ei tietenkään voi käyttää arvo- tai viiteparametreja, koska muuttujien arvot ja osoitteet ovat olemassa vain suoritusaikana ja makrot käsitellään jo käännösaikana. Ainoa jäljellä oleva parametrityyppi on siten nimiparametri.

Literaalit

Vakioita

ttk-91: käskyn vakiot 2-tavuisia, arvoalue -32767 ... +32767

```
Pi     DC  3.14159265
One  DC  1  ; vrt. ONE EQU 1
Mega DC 1024576
```

- niin suuria, että eivät mahdu konekäskyn vakio-osaan
- muuten vain halutaan pitää datan yhteydessä eikä koodissa
- niitä ei saa muuttaa

```
Load  R1, One
Add   R1, =1
Store R1, One ; ask for trouble
```

Korkean tason kielissä kaikki isot vakiot literaaleja

```
N := 35000;
```

```
var myStr = "literal";
```

Kääntäjän pitäisi estää niiden muuttaminen

```
FortranX:  5 := 6;
```

```
Load R1, six
Store R1, five
```

Literaalia ei saisi välittää viiteparametrina

aliohjelma voisi muuttaa sen arvoa? Java string?

Literaalien implisiittinen (automaattinen) määrittely

```
LOAD   r14, =f'234'
```

Copyright Teemu Kerola 2005

Literaalit ovat muistissa olevia vakioita. Vakiot voidaan usein tallettaa myös itse konekäskyyn sen vakio-osaan tai laiterekisteriin. Useimmiten ne kuitenkin talletetaan literaaleina erityiselle literaalialueelle, joka on osa ohjelman data-segmenttia. Rekisterissa vakioita pidetaan yleensä vain vahan aikaa, koska rekistereille on muutakin käyttöä. Pienet vakiot mahtuvat myös konekäskyyn, mutta silloin ne joudutaan tallettamaan muistiin koodialueelle joka käyttökertaa varten. Useissa arkkitehtuureissa literaalialue on suojattu kirjoittamiselta.

Literaalit

Vakioita

ttk-91: käskyn vakiot 2-tavuisia, arvoalue -32767 ... +32767

```
Pi     DC  3.14159265
One  DC  1  ; vrt. ONE EQU 1
Mega DC 1024576
```

- niin suuria, että eivät mahdu konekäskyn vakio-osaan
- muuten vain halutaan pitää datan yhteydessä eikä koodissa
- niitä ei saa muuttaa

```
Load  R1, One
Add   R1, =1
Store R1, One ; ask for trouble
```

Korkean tason kielissä kaikki isot vakiot literaaleja

```
N := 35000;
```

```
var myStr = "literal";
```

Kääntäjän pitäisi estää niiden muuttaminen

```
FortranX:  5 := 6;
```

```
Load R1, six
Store R1, five
```

Literaalia ei saisi välittää viiteparametrina

```
aliohjelma voisi muuttaa sen arvoa?   Java string?
```

Literaalien implisiittinen (automaattinen) määrittely

```
LOAD   r14, =f'234'
```

Copyright Teemu Kerola 2005

Useissa korkean tason kielissä joko kaikki tai ainakin kaikki isot vakiot toteutetaan literaaleina suojatulle literaalialueelle. Tällä tavoin niiden suojaaminen muutoksilta on helpompaa. Literaalia ei yleensä saisi välittää viiteparametrina, koska aliohjelmassa ei sitten ole tietoa, että kyseinen parametri onkin suojattu literaali. Useissa assembler-ympäristöissä literaalien määrittely tapahtuu automaattisesti. Tässä esimerkissä =f'234' tarkoittaa sen muistipaikan osoitetta, jonka arvo on 234.

Assembler-käännöksen vaiheet

1. vaihe eli 1. koodin läpikäynti
   - laske käskyjen tilanvaraukset
   - generoi symbolitaulu
   - generoi muut taulut ja käytä niitä

ttk-91 helppoa, koska kaikki käskyt ja data 4 tavua

arvot, pituus tavuina

uudelleensijoitustiedot (omana taulunaan?)

Symbolitaulu

Symboli	Tyyppi	Arvo	Uudell.sij.tieto
A	data	? 32	käskyt 1, 9, 19
B	data	? 33	Käskyt 6, 8, 11, 20
CRT	vakio	0	
Esm2	viite	0	
F	vakio	-3	
HALT	vakio	11	

[Häkk 98, Kuva 6.2]

literaalitaulu
kääntäjän ohjauskäskytaulu
operaatiokooditaulu

2. vaihe
   - generoi objektimoduuli
   - tulosta listaus symbolisella konekielellä
   - anna mahdolliset virheilmoitukset
   - generoi lopulliset taulut linkitystä varten

3. vaihe
   - koodin optimointi
     (generoidaan koodia, joka suoritetaan mahdollisimman nopeasti)
   - optimointia voi tapahtua myös aikaisempien vaiheiden aikana

Copyright Teemu Kerola 2005

Käännös tapahtuu usean vaiheen avulla, missä kussakin vaiheessa koko koodi käydään läpi rivi kerrallaan alusta loppuun. Ensimmäisessä vaiheessa listataan kaikki symbolit ja varataan muuttujille ja koodille sopivan verran tilaa. Vaiheen lopussa kaikilla paikallisilla symboleilla on tunnettu arvo. Tiedot talletetaan useisiin erilaisiin tauluihin. Osa tiedoista luetaan valmiiksi kääntäjässä olevista vakiotauluista, kuten esimerkiksi operaatiokooditaulusta.

Assembler-käännöksen vaiheet

1. vaihe eli 1. koodin läpikäynti
   - laske käskyjen tilanvaraukset
   - generoi symbolitaulu
   - generoi muut taulut ja käytä niitä

2. vaihe
   - generoi objektimoduuli
   - tulosta listaus symbolisella konekielellä
   - anna mahdolliset virheilmoitukset
   - generoi lopulliset taulut linkitystä varten

Objektimoduuli

Moduulin otsake

Export hakemisto

Import hakemisto

Uudelleensijoitushakemisto

Koodi ja alustettu data

Moduulin lopuke

[Häkk 98, Kuva 6.3]

3. vaihe
   - koodin optimointi
     (generoidaan koodia, joka suoritetaan mahdollisimman nopeasti)
   - optimointia voi tapahtua myös aikaisempien vaiheiden aikana

Copyright Teemu Kerola 2005

Assemblerkäännöksen toisessa vaiheessa koodi käydään läpi uudelleen. Nyt tuotetaan varsinainen objektimoduuli linkitystä varten.Ohjelmoijaa varten voidaan haluttaessa tuottaa nykyistä objektimoduulia vastaava symbolisen konekielen listaus, jota tarvitaan ainakin silloin, jos moduulissa on vielä virheitä. Virheettömästä moduulista generoidaan myös kaikki muut lopulliset taulut linkitystä varten.

Assembler-käännöksen vaiheet

1. vaihe eli 1. koodin läpikäynti

- laske käskyjen tilanvaraukset
- generoi symbolitaulu
- generoi muut taulut ja käytä niitä

2. vaihe

- generoi objektimoduuli
- tulosta listaus symbolisella konekielellä
- anna mahdolliset virheilmoitukset
- generoi lopulliset taulut linkitystä varten

```
load  r1, 124 ; x = 124
add   r1, r2    ; y in r2
store r1, 124 ; y gets value x+y
jnzer r3, 65
load  r2, 124 ; store value of x ...
store r2, 200 ;  ... into y in 200
```

optimoi suoritettavien käskyjen lukumäärää ja muistiviitteiden lukumäärää

```
load  r4, 124 ; keep x in r4
add   r4, r2
jnzer r3, 65
store r4, 200
```

3. vaihe

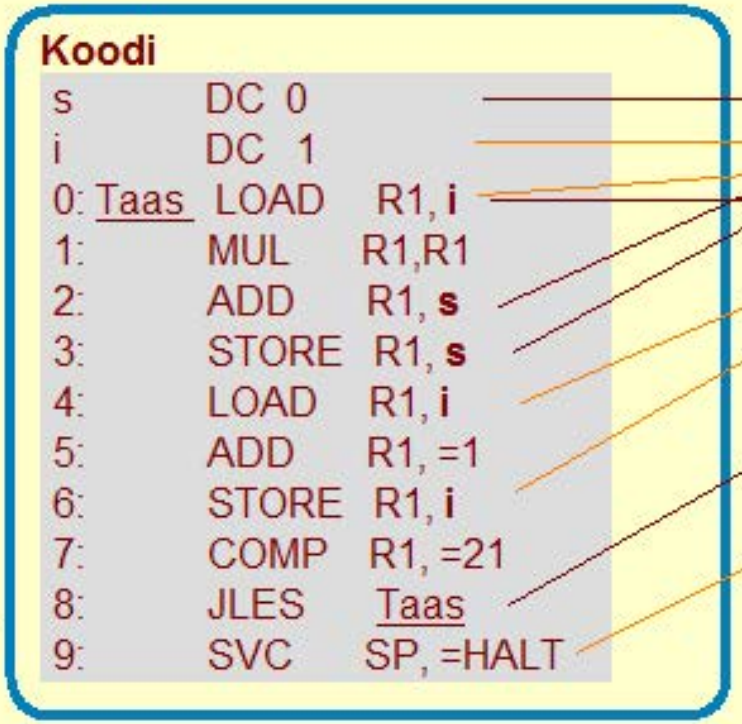
- koodin optimointi
  (generoidaan koodia, joka suoritetaan mahdollisimman nopeasti)
- optimointia voi tapahtua myös aikaisempien vaiheiden aikana

Copyright Teemu Kerola 2005

Jos halutaan, niin objektimoduulia voidaan vielä optimoida siten, että generoitu koodi voitaisiin suorittaa mahdollisimman nopeasti. Tämä toimenpide sisältää tarkkaa analyysiä sekä jo olemassaolevasta koodista että kohdearkkitehtuurista, jossa koodi lopulta tullaan suorittamaan. Koodia voidaan optimoida usein eri tasoisesti riippuen siitä, kuinka paljon aikaa halutaan optimointiin käyttää ja kuinka tärkeätä koodin nopea suoritus itse asiassa on. Optimointiin kuluva aika voi helposti olla yli puolet kaikesta kääntämiseen kuluvasta ajasta.

Assembler-käännös esimerkki ttk-91 koneelle

1. vaiheen aikana



s = ?

i = ?

Taas = 0

HALT = 11 (valmiiksi määritelty)

Symbolitaulu

Symboli	Tyyppi	Arvo	Uudell.sij.tieto
s	data	?	2, 3
i	data	?	0, 4, 6
Taas	viite	0	9
HALT	vakio	11	

Copyright Teemu Kerola 2005

TTK-91 assembler käännöksen ensimmäisessä vaiheessa tehdään tilanvaraukset muuttujille ja rakennetaan symbolitaulu loppuun. Muuttujien s ja i sijaintia ei tiedetä vielä koodin läpikäynnin alkuvaiheessa, koska data sijoitetaan vasta koodin jälkeen, emmekä tiedä koodin määrää vielä. Symbolin Taas arvo saadaan sen ensi käyttökerralla, koska se esiintyy käskyn osoitekentässä. Symbolin HALT arvo taas on etukäteen tunnettu vakio. Muuttujien i ja s osoitteet jäävät vielä auki tällä läpikäynnillä.

Assembler-käännös esimerkki ttk-91 koneelle

1. vaiheen lopussa

Koodi

```
s     DC    0
i     DC    1
0: Taas  LOAD   R1, i
1:       MUL    R1,R1
2:       ADD    R1, s
3:       STORE  R1, s
4:       LOAD   R1, i
5:       ADD    R1, =1
6:       STORE  R1, i
7:       COMP   R1, =21
8:       JLES   Taas
9:       SVC    SP, =HALT
10:      0      ; muuttuja s
11:      1      ; muuttuja i
```

s = 10

i = 11

Taas = 0

HALT = 11 (valmiiksi määritelty)

Symbolitaulu

Symboli	Tyyppi	Arvo	Uudell.sij.tieto
s	data	**10**	2, 3
i	data	**11**	0, 4, 6
Taas	viite	0	9
HALT	vakio	11	

Copyright Teemu Kerola 2005

Ensimmäisen vaiheen lopussa koodi on käyty läpi ja se vie tilaa 10 muistipaikkaa. Data-alue alkaa siis muistipaikasta 10, joten muuttujat s ja i sijoitetaan muistipaikkoihin 10 ja 11. Symbolien s ja i arvoiksi tulee vastaavasti luvut 10 ja 11. Tämä tieto talletetaan symbolitauluun koodin toista läpikäyntiä varten.

Assembler-käännös esimerkki ttk-91 koneelle

2. vaiheen lopussa

Symbolitaulu

Symboli	Tyyppi	Arvo	Uudell.sij.tieto
s	data	10	2, 3
i	data	11	0, 4, 6
Taas	viite	0	9
HALT	vakio	11	

Alkup. koodi

```
0: Taas  LOAD   R1, i
1:       MUL    R1,R1
2:       ADD    R1, s
3:       STORE  R1, s
4:       LOAD   R1, i
5:       ADD    R1, =1
6:       STORE  R1, i
7:       COMP   R1, =21
8:       JLES   Taas
9:       SVC    SP, =HALT
10:      0      ; muuttuja s
11:      1      ; muuttuja i
```

Objektikoodi

```
0:       LOAD   R1, 11
1:       MUL    R1,R1
2:       ADD    R1, 10
3:       STORE  R1, 10
4:       LOAD   R1, 11
5:       ADD    R1, =1
6:       STORE  R1, 11
7:       COMP   R1, =21
8:       JLES   0
9:       SVC    SP, =11
10:      0
11:      1
```

Objektikoodi (numeerinen)

```
0:      2   1 0 0   11
1:     19   1 0 1    0
2:     17   1 1 0   10
3:      1   1 0 0   10
4:      2   1 0 0   11
5:     17   1 0 0    1
6:      1   1 0 0   11
7:     31   1 0 0   21
8:     39   0 0 0    0
9:    112   6 0 0   11
10:                  0
11:                  1
```

Copyright Teemu Kerola 2005

Toisen vaiheen jälkeen objektimoduuli onkin sitten valmis linkitystä varten. (Simulaattorissa tosin sen voi jo suoraan ladata ja sitten suorittaa.) Kaikki symbolit on koodissa korvattu numeroarvoillaaan - ttk-91:hän ei ollut kuin numeroarvoisia symboleja. Objektikoodin voi edelleen tulostaa tekstuaalisessa muodossa, symboleja käyttäen, mutta siitä on olemassa myös puhtaasti numeerinen versio jatkoa varten. Ohessa numeerinenkin versio on vielä esitetty kentittäin, mutta yhtä hyvin kunkin sanan sisällön olisi voinut antaa yhtenä kokonaislukuna, jolloin se olisi ollut vielä vähemmän ihmisen luettavassa muodossa.

Ttk-91 objektimoduuli

Moduulin otsakeosa

- moduulin nimi ja linkittäjän tarvitsemia tietoja

Export-hakemisto

- muihin moduuleihin näkyvät tunnukset
  - rutiinit, aliohjelmat, oliot, metodit, yhteiskäytt. data
- tunnusten osoitteet ja käyttöoikeus

Import-hakemisto

- tunnukset, jotka on määritelty muissa moduuleissa

Koodi ja alustettu data

- koodi sellaisenaan
- viitteet ulkopuolelle Export-hakemistossa tai merkittynä koodiin
- alustettu data paikallaan koodin jalkeen
- alustamatonta dataa ei tarvittaisi vielä, mutta se on ttk-91'ssä paikallaan

Uudelleensijoitushakemisto

- niiden käskyjen osoitteet, joiden osoiteosaa (käskyn vakiokenttää) on muutettava, kun moduulin osoiteavaruus yhdistetään muiden moduulien osoiteavaruuksiin

objektimoduuli
Moduulin otsake
Export hakemisto
Import hakemisto
Uudelleensijoitushakemisto
Koodi ja alustettu data
Moduulin lopuke

objektimoduulin osien pituudet
käännöspäivämäärä
kääntäjän nimi ja versio
ensimm. suor. käskyn osoite
(aikaisemman linkityksen pvm)

Copyright Teemu Kerola 2005

TTK-91 koneen objektimoduulissa on kuusi osioita. Otsakeosa antaa yleisiä hallintotietoja, kuten moduulin nimen ja linkitysaikana tarvittavat yleistiedot. Osien pituuksia tarvitaan tilanvarauksia varten. Käännösajankohtia tarvitaan päättelemään, onko moduuli ajan tasalla. Moduulin lopuke ilmaisee yksinkertaisesti moduulin päättymisen.

Ttk-91 objektimoduuli

Moduulin otsakeosa

- moduulin nimi ja linkittäjän tarvitsemia tietoja

Export-hakemisto

- muihin moduuleihin näkyvät tunnukset
  - rutiinit, aliohjelmat, oliot, metodit, yhteiskäytt. data
- tunnusten osoitteet ja käyttöoikeus

Import-hakemisto

- tunnukset, jotka on määritelty muissa moduuleissa

Koodi ja alustettu data

- koodi sellaisenaan
- viitteet ulkopuolelle Export-hakemistossa tai merkittynä koodiin
- alustettu data paikallaan koodin jälkeen
- alustamatonta dataa ei tarvittaisi vielä, mutta se on ttk-91'ssä paikallaan

Uudelleensijoitushakemisto

- niiden käskyjen osoitteet, joiden osoiteosaa (käskyn vakiokenttää) on muutettava, kun moduulin osoiteavaruus yhdistetään muiden moduulien osoiteavaruuksiin

objektimoduuli
Moduulin otsake
Export hakemisto
Import hakemisto
Uudelleensijoitushakemisto
Koodi ja alustettu data
Moduulin lopuke

r / w / rw / e

Copyright Teemu Kerola 2005

Export hakemistossa on kaikki ne tunnukset, jotka tästä moduulista näkyvät ulospäin. Erityisesti on huomattava, että kaikki muut tunnukset ovat suojattuja ulkopuolisille, eikä niihin pitäisi olla minkäänlaista pääsyä moduulin ulkopuolelta. Ulospäin näkyviä tunnuksia ovat esimerkiksi yhteiskäyttöiset aliohjelmat tai joskus myös jotkut data-alueet. Yleensä on tietenkin parempi kapseloida yhteiskäyttöisen datan käyttö sitä manipuloiviin aliohjelmiin. Javan 'public' määreellä varustetut metodit tullaan listaamaan täällä.

Ttk-91 objektimoduuli

objektimoduuli
- Moduulin otsake
- Export hakemisto
- Import hakemisto
- Uudelleensijoitushakemisto
- Koodi ja alustettu data
- Moduulin lopuke

Moduulin otsakeosa

- moduulin nimi ja linkittäjän tarvitsemia tietoja

Export-hakemisto

- muihin moduuleihin näkyvät tunnukset
  - rutiinit, aliohjelmat, oliot, metodit, yhteiskäytt. data
- tunnusten osoitteet ja käyttöoikeus

Import-hakemisto

tunnus | osoitteet, joissa tunnus esiintyy

- tunnukset, jotka on määritelty muissa moduuleissa

Koodi ja alustettu data

- koodi sellaisenaan
- viitteet ulkopuolelle Export-hakemistossa tai merkittynä koodiin
- alustettu data paikallaan koodin jälkeen
- alustamatonta dataa ei tarvittaisi vielä, mutta se on ttk-91'ssä paikallaan

Uudelleensijoitushakemisto

- niiden käskyjen osoitteet, joiden osoiteosaa (käskyn vakiokenttää) on muutettava, kun moduulin osoiteavaruus yhdistetään muiden moduulien osoiteavaruuksiin

Copyright Teemu Kerola 2005

Import hakemistossa listataan kaikki ne tunnukset, jotka esiintyvät tässä moduulissa, mutta joita ei ole määritelty tässä moduulissa. Tyypillinen esimerkki on tulostusrutiini Print. Kääntäjä tietää omista määrittelyistään, että kyseiset tunnukset ovat muualta ja sallii niiden käytön tässä moduulissa. Kaikki tällaiset viitteet ulkopuolelle täytyy kuitenkin sitten löytää (eli ratkaista) linkitysaikana. Esimerkiksi, kun tulostusrutiinia Print kutsutaan suoritusaikana , niin sen todellisen muistiosoitteen tulee olla selvillä.

Ttk-91 objektimoduuli

Moduulin otsakeosa

- moduulin nimi ja linkittäjän tarvitsemia tietoja

Export-hakemisto

- muihin moduuleihin näkyvät tunnukset
  - rutiinit, aliohjelmat, oliot, metodit, yhteiskäytt. data
- tunnusten osoitteet ja käyttöoikeus

Import-hakemisto

- tunnukset, jotka on määritelty muissa moduuleissa

Koodi ja alustettu data

- koodi sellaisenaan
- viitteet ulkopuolelle Export-hakemistossa tai merkittynä koodiin
- alustettu data paikallaan koodin jälkeen
- alustamatonta dataa ei tarvittaisi vielä, mutta se on ttk-91'ssä paikallaan

Uudelleensijoitushakemisto

- niiden käskyjen osoitteet, joiden osoiteosaa (käskyn vakiokenttää) on muutettava, kun moduulin osoiteavaruus yhdistetään muiden moduulien osoiteavaruuksiin

objektimoduuli
Moduulin otsake
Export hakemisto
Import hakemisto
Uudelleensijoitushakemisto
Koodi ja alustettu data
Moduulin lopuke

1000 alkion taulukolle ei tarvitse varata tilaa. Tilanvaraus voitaisiin tehdä vasta latausaikana.

Copyright Teemu Kerola 2005

Koodi talletetaan sellaisenaan binäärimuotoisena. Viitteitä muihin moduuleihin ei ttk-91:ssä tarvitse tallettaa koodiin, koska tiedot niistä löytyvät Export taulusta ja muiden moduulien viitteet sijoitetaan aina samaan paikkaan ttk-91 koneen konekäskyssä, sen vakio kenttään. Kaikki DC valekäskyillä varattu alustettu data sijoitetaan heti koodisegmentin jälkeen. DS valekäskyillä varattua alustamatonta dataa ei tarvitsisi vielä oikeastaan tallettaa, mutta tässä järjestelmässä sekin varataan jo nyt alustetun datan yhteyteen.

Ttk-91 objektimoduuli

Moduulin otsakeosa

- moduulin nimi ja linkittäjän tarvitsemia tietoja

Export-hakemisto

- muihin moduuleihin näkyvät tunnukset
  - rutiinit, aliohjelmat, oliot, metodit, yhteiskäytt. data
- tunnusten osoitteet ja käyttöoikeus

Import-hakemisto

- tunnukset, jotka on määritelty muissa moduuleissa

Koodi ja alustettu data

- koodi sellaisenaan
- viitteet ulkopuolelle Export-hakemistossa tai merkittynä koodiin
- alustettu data paikallaan koodin jälkeen
- alustamatonta dataa ei tarvittaisi vielä, mutta se on ttk-91'ssä paikallaan

Uudelleensijoitushakemisto

uudelleensijoitusvakio tai -vakiot

- niiden käskyjen osoitteet, joiden osoiteosaa (käskyn vakiokenttää) on muutettava, kun moduulin osoiteavaruus yhdistetään muiden moduulien osoiteavaruuksiin

koodi- ja dataviitteet erillään?

objektimoduuli
Moduulin otsake
Export hakemisto
Import hakemisto
Uudelleensijoitushakemisto
Koodi ja alustettu data
Moduulin lopuke

suoraviivainen lisäys joka käskyyn ei toimi, koska osoitteen asemesta siellä voi olla vakio!

Copyright Teemu Kerola 2005

Uudelleensijoitushakemistossa on kaikkien niiden käskyjen osoitteet, jotka linkityksen yhteydessä voivat muuttua. Normaalistihan moduulin kaikki osoitteet alkavat nollasta. Jos nyt esimerkiksi moduuli sijoitetaan alkamaan muistiosoitteesta 150, niin kaikkiin moduulin käyttämiin paikallisiin osoitteisiin tulee lisätä 150. Ja jos moduulin koodi- ja data-alueet yhdistetään erikseen, niin tämä lisäysmuutos (eli uudelleensijoitusvakio) on erilainen koodi- ja dataviitteille.

Korkean tason kielten käännös

Enemmän vaiheita (kuin assembler-käännöksessä)

Syntaktisten alkioiden etsintä

BEGIN

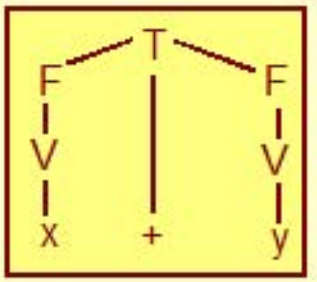
(

}

IF

234.56

Lauseiden tunnistaminen syntaksipuun avulla



Välikielen (välikoodin) generointi

Välikieliesitys ja symbolitaulu

Pascalin P-code
Javan ByteCode?

Koodin generointi ja optimointi

Copyright Teemu Kerola 2005

Korkean tason kielen käännös on monimutkaisempi kuin assembler-kielen käännös, koska sen syntaksi ja semantiikka (kielioppi ja merkitys) ovat paljon monimutkaisempia. Käännöksessä on usein neljä tai viisi erillistä vaihetta. Ensimmäisessä vaiheessa paikallistetaan syntaktiset alkiot, kuten varatut sanat ja erityyppiset lukuarvot. Toisessa vaiheessa näitä alkioita yhdistellään lauseiksi kieliopin sääntöjen mukaan. Kolmannessa vaiheessa generoidaan laiteriippumaton kielen semantiikan toteuttava yleinen välikieli, josta sitten lopuksi generoidaan laitesidonnainen varsirainen objektimoduuli.

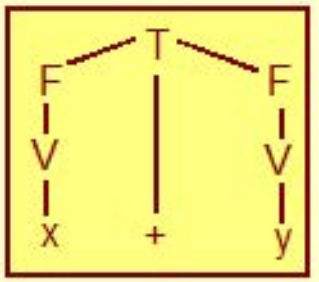
Korkean tason kielten käännös

Enemmän vaiheita (kuin assembler-käännöksessä)

Syntaktisten alkioiden etsintä

BEGIN (} 234.56

Lauseiden tunnistaminen syntaksipuun avulla



Välikielen (välikoodin) generointi

- Välikieliesitys ja symbolitaulu
- Pascalin P-code
  Javan ByteCode?

Koodin generointi ja optimointi

front-end

back-end

Copyright Teemu Kerola 2005

Kääntäjän alkuosaa lauseiden tunnistamiseen asti sanotaan front-end'iksi. Front-end'iin sisältyy oikeastaan kaikki ohjelmointikielisidonnaiset osat kääntäjää. Kääntäjän loppuosa on geneeristä eikä siten suoraan sidoksissa mihinkään tiettyyn ohjelmointikieleen. Kääntäjän loppuosa on nimeltään back-end, ja se on kääntäjän ainoa laitesidonnainen osa. Jos kääntäjä halutaan siirtää (portata) uudelle laitteistolle, niin riittää uuden back-end'in toteuttaminen tälle arkkitehtuurille.

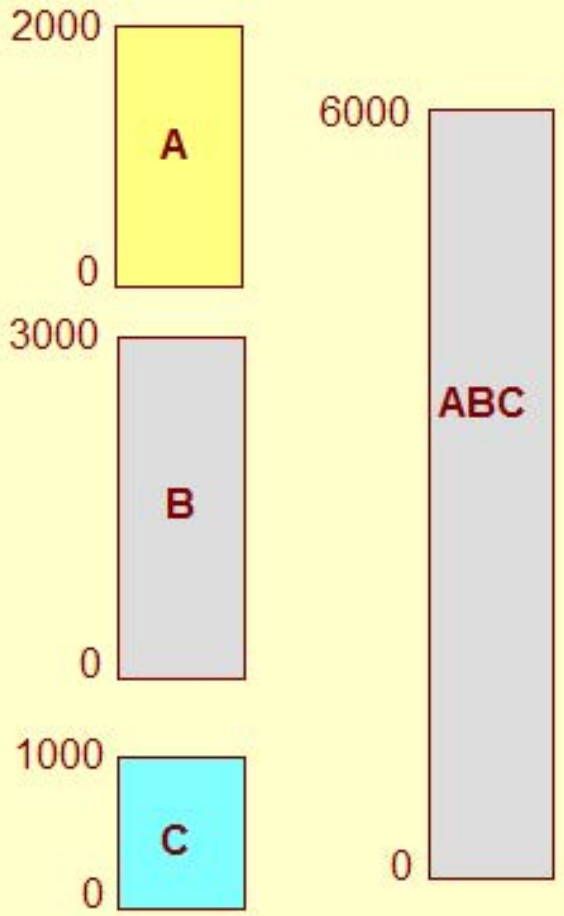
Uudelleensijoitusongelma (relocation problem)

Jokaisen objektimoduulin osoitteet alkavat nollasta

Tulosmoduulissa kaikkien tulee olla yhdessä yhtenäisessä osoiteavaruudessa

Useimpien moduulien lähes kaikkia osoitteita muutettava

käskyjen osoitteet
datan osoitteet



Copyright Teemu Kerola 2005

Uudelleensijoitusongelma on linkityksen perusongelma. Kun moduuleita linkitetään yhteen, niin melkein kaikkien moduulien kantaosoite muuttuu, ja näiden moduulien kaikkia muistiosoitteita pitää sen vuoksi päivittää. Linkityksen tulosmoduulin ulkoasun tulee olla saman tyyppinen kuin sisäänmenomoduulien ulkoasu, joten tulosmoduulin kantaosoite on nolla ja siellä on yhtenäinen data-alue. Linkityksen tulosmoduulia on voitava käyttää myöhempien linkityksien sisäänmenomoduulina.

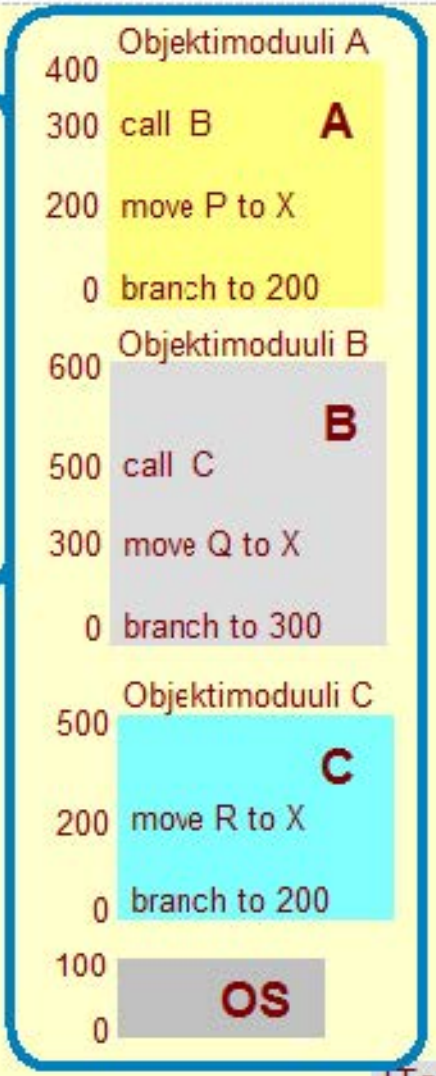
Linkitysesimerkki

Kolme moduulia: A, B ja C

- moduulien koot 400, 600 ja 500 sanaa
- kunkin moduulin entry-point osoitteessa 0, jossa haarautuminen moduulin pääohjelmaan
- kussakin moduulissa oma X
- A kutsuu B:tä, B kutsuu C:tä
- Käyttöjärjestelmäkoodi vaatii 100 sanaa tilaa alusta (osoitteesta 0)

Uudelleensijoitusvakio

- sijoita kaikki moduulit päällekkäin yhteiseen osoiteavaruuteen
- moduulin uud.sij. vakio on moduulin alkuosoite
- kunkin moduulin kutsukohta on sen alkuosoite
- lisää oma uud.sij.vakio paikallisiin osoitteisiin



[Tane 06, Figs 7-14, 7-15]

Copyright Teemu Kerola 2005

Tämä linkitysesimerkki on yksinkertaistettu siten, että data- ja koodiviitteitä ei ole eritelty. Mukana on kolme moduulia (A, B ja C), ne ovat eri kokoisia ja kutsuvat toisiaan. Kaikilla on omia paikallisia tunnuksia. Esimerkiksi, kaikilla on oma muuttujansa X. Moduulit A, B ja C pitäisi nyt linkittää yhteen ja samaan osoiteavaruuteen. Linkitettävien moduulien lisäksi mukaan otetaan käyttöjärjestelmämoduuli, jonka otaksutaan vievän tilaa 100 sanaa.

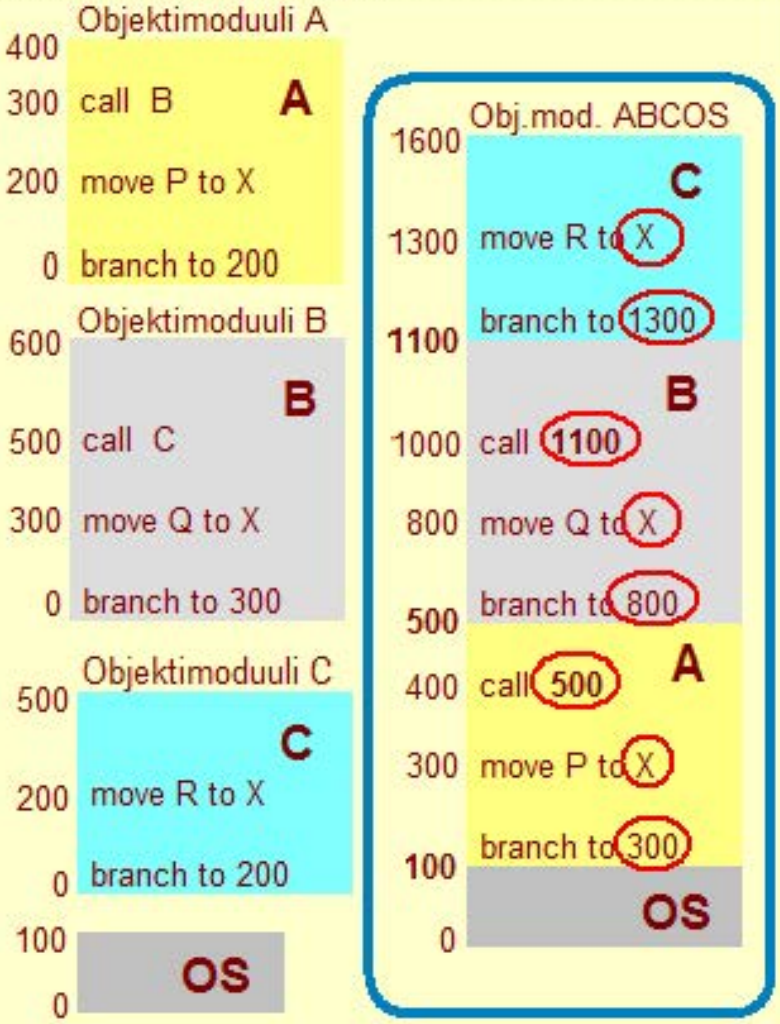
Linkitysesimerkki

Kolme moduulia: A, B ja C

- moduulien koot 400, 600 ja 500 sanaa
- kunkin moduulin entry-point osoitteessa 0, jossa haarautuminen moduulin pääohjelmaan
- kussakin moduulissa oma X
- A kutsuu B:tä, B kutsuu C:tä
- Käyttöjärjestelmäkoodi vaatii 100 sanaa tilaa alusta (osoitteesta 0)

Uudelleensijoitusvakio

- sijoita kaikki moduulit päällekkäin yhteiseen osoiteavaruuteen
- moduulin uud.sij. vakio on moduulin alkuosoite A: 100, B:500, C: 1100
- kunkin moduulin kutsukohta on sen alkuosoite
- lisää oma uud.sij.vakio paikallisiin osoitteisiin lisää 100 A:n osoitteeseen X



[Tane 06, Figs 7-14, 7-15]

Copyright Teemu Kerola 2005

Moduulit kasataan käyttöjärjestelmämoduulin päälle haluttuun järjestykseen ja jokaiselle moduulille lasketaan sen uusi alkuosoite, joka siis on tämän moduulin uudelleensijoitusvakio. Muiden moduulien kutsukohdaksi laitetaan kutsutun moduulin alkuosoite. Kaikkia paikallisia osoitteita päivitetään lisäämällä niihin kyseisen moduulin uudelleensijoitusvakio. Todellisuudessa tilanne on vielä vähän monimutkaisempi, koska data- ja koodiviitteet käsitellään erikseen ja linkityskertoja voi olla useita.

Muuttujaan X kohdistuneiden viittausten päivitys

Miten pitää kirjaa kaikista kohdista, joissa muuttujaan X viitataan?

Vastaus 1: uudelleensijoitustaulukossa, jossa listattuna kaikki kohdat

- taulukosta voi tulla hyvin iso, jos joka muuttujan kohdalla pitää varautua maksimilukumäärään

Vastaus 2: linkitetään kaikki viittauskohdat objektimoduulissa keskenään → esimerkki seuraavassa näkymässä

- uudelleensijoitustauluun vain ensimmäisen viitteen kohta
- objektimoduulissa on alkuaan X-viitteen kohdalla seuraavan X-viitteen esiintymispaikka

viitteet uudelleensijoitustaulussa

Symbolitaulu

Symb	Arv	viitteet
...		
X	700	23, 34, 555
...		

ABC:n lähdekoodi:

```
      ...
23:   Load   R1, X
      ...
34:   Store  R3, X(R1)
      ...
555:  Add    R4, X


700:  DC   0  ; X
```

Copyright Teemu Kerola 2005

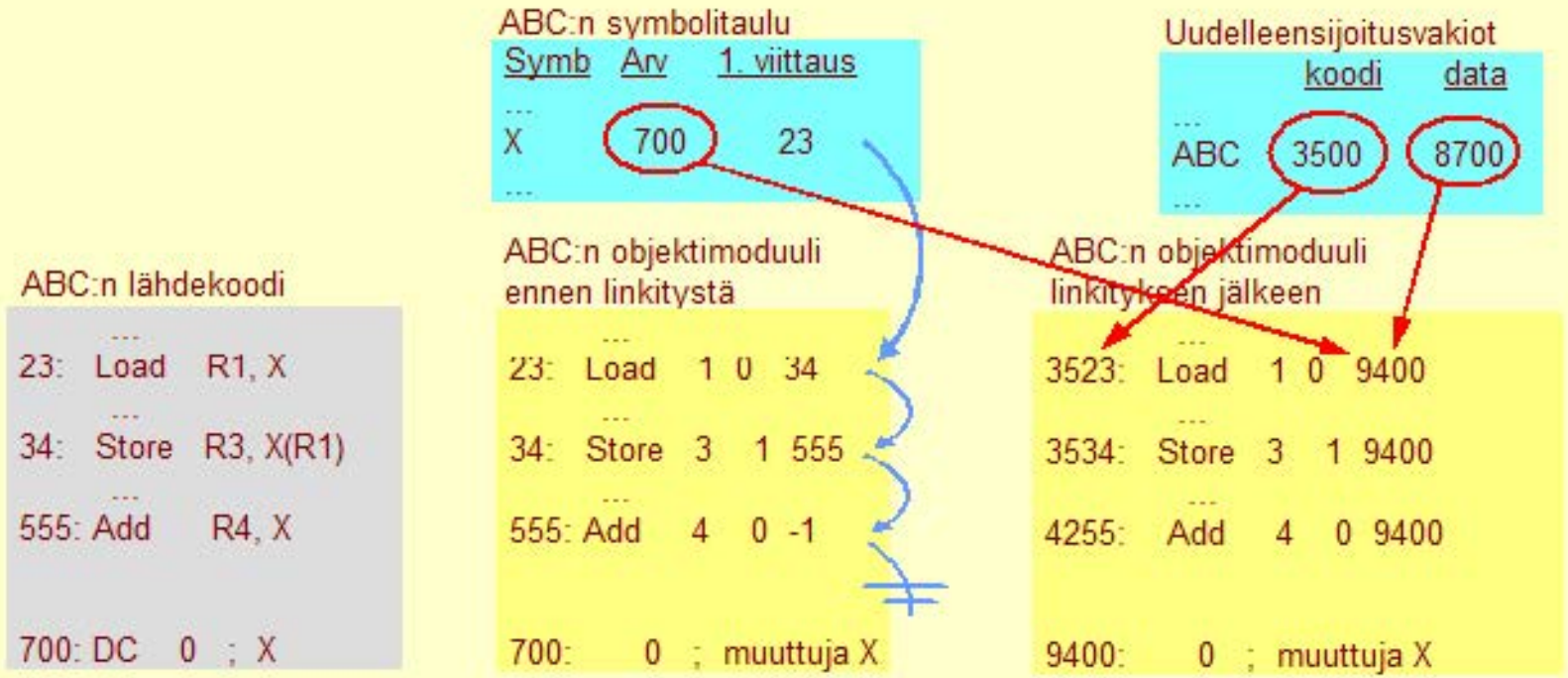
Uudelleensijoitustaulusta voi tulla hyvin suuri, jos jokaiselle symbolille varataan tilaa esimerkiksi 200 viittauskohtaa varten. Tämän vuoksi yleensä turvaudutaan johonkin tilaa säästävään menetelmään. Yksi mahdollisuus on linkittää kaikki muuttujan viittauskohdat keskenään ja laittaa ainoastaan tämän listan alkuosoite uudelleensijoitustauluun. Linkitysaikana lista sitten käydään läpi ja jokaiseen viittauskohtaan laitetaan symbolin lopullinen arvo. Tässä menetelmässä on tietenkin se huono puoli, että linkitetty lista tuhoutuu sitä läpikäydessä ja se voidaan siten käydä läpi vain kerran.

Muuttujaan X kohdistuneiden viittausten päivitys

Miten pitää kirjaa kaikista kohdista, joissa muuttujaan X viitataan?

Vastaus 2: linkitetään kaikki viittauskohdat objektimoduulissa keskenään

- uudelleensijoitustauluun vain ensimmäisen viitteen kohta
- objektimoduulissa on alkuaan X-viitteen kohdalla seuraavan X-viitteen esiintymispaikka



Copyright Teemu Kerola 2005

Tässä esimerkissä muuttuja X on paikallisessa osoitteessa 700 ja siihen on kolme viittausta koodissa. Uudelleensijoitustauluun (joka tässä on symbolitaulun yhteydessä) laitetaan ainoastaan ensimmäinen X:n viittauskohta eli 23. Objektimoduulissa käskyn 23 osoiteosassa onkin sitten seuraavan X:n viittauskohdan osoite eli 34. Viimeisessä viittauskohdassa (käskyssä 555) osoiteosassa on linkkiketjun päättymismerkkinä luku -1. Listaa läpikäytäessä ABC:n datan uudelleensijoitusvakioon 8700 lisätään symbolin paikallinen osoite 700 ja tämä summa 9400 sitten talletetaan kaikkiin symbolin X viittauskohtiin linkitetyn listan avulla.

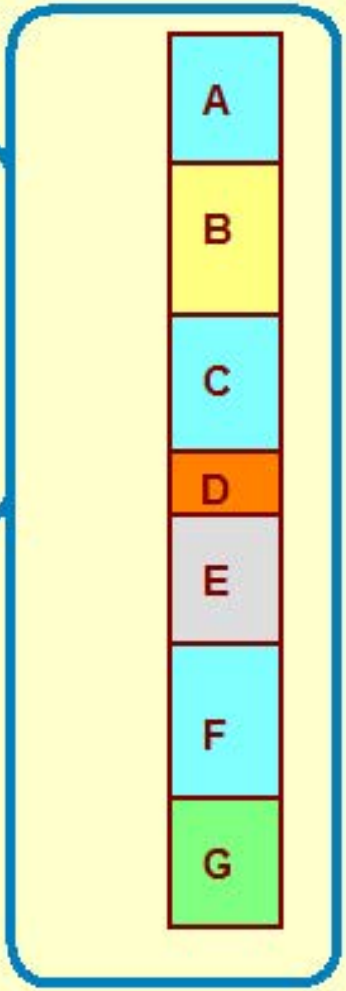
Staattinen ja dynaaminen linkitys

Staattinen linkitys

- kaikki mahdollisesti viitattavat moduulit ja kirjastorutiinit linkitetään paikalleen ennen suoritusta
- ajomoduulista voi tulla hyvin iso
  - mukana myös moduuleja, joihin ei yhdellä suorituskerralla tule lainkaan viittauksia

Dynaaminen linkitys

- ei linkitetä kaikkia moduuleja paikalleen
- puuttuvista moduuleista jätetään kutsukohdat auki
- jos puuttuvaan moduuliin tulee viite, niin puuttuva moduuli linkitetään paikalleen sillä hetkellä
  - suoritus keskeytyy ja puuttuva moduuli linkitetään paikalleen
- pienempi ajomoduuli, nopeampi lataus
- suoritus voi hidastua yllättäen milloin vain
  - dynaamisen linkityksen ajaksi



Copyright Teemu Kerola 2005

Tähän asti esitetty linkitysmenetelmä on ns. staattinen linkitys. Siinä kaikki yhdellä suorituskerralla mahdollisesti tarvittavat ohjelmamoduulit ja kirjastomoduulit on linkitetty valmiiksi. Olemme siis varautuneet kaikkeen. Tällaisesta ajomoduulista voi kuitenkin tulla tarpeettoman suuri, koska mukana on paljon moduuleja, joita ei ohjelman yhdellä suorituskerralla koskaan tarvita. Ohjelma vie siten tarpeettoman paljon muistitilaa ja sen suuren koon vuoksi sen käynnistyminen voi viedä paljon aikaa. Useissa sovellustilanteissa nopea käynnistyminen on välttämätöntä tai ainakin toivottavaa - esimerkkinä vaikkapa kännykkä.

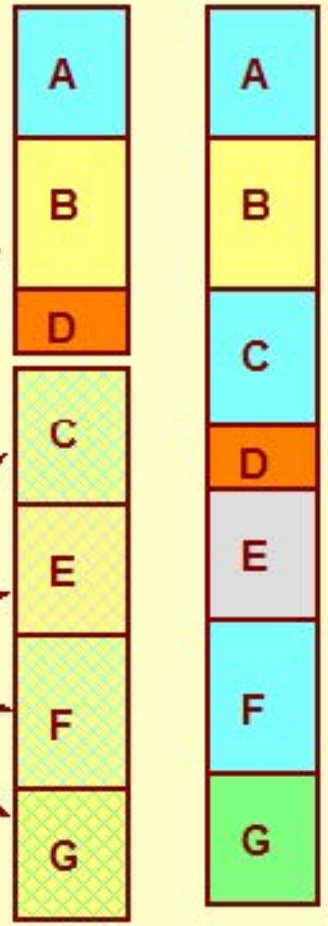
Staattinen ja dynaaminen linkitys

Staattinen linkitys

- kaikki mahdollisesti viitattavat moduulit ja kirjastorutiinit linkitetään paikalleen ennen suoritusta
- ajomoduulista voi tulla hyvin iso
  - mukana myös moduuleja, joihin ei yhdellä suorituskerralla tule lainkaan viittauksia

Dynaaminen linkitys

- ei linkitetä kaikkia moduuleja paikalleen
- puuttuvista moduuleista jätetään kutsukohdat auki
- jos puuttuvaan moduuliin tulee viite, niin puuttuva moduuli linkitetään paikalleen sillä hetkellä
  - suoritus keskeytyy ja puuttuva moduuli linkitetään paikalleen
- pienempi ajomoduuli, nopeampi lataus
- suoritus voi hidastua yllättäen milloin vain
  - dynaamisen linkityksen ajaksi



Copyright Teemu Kerola 2005

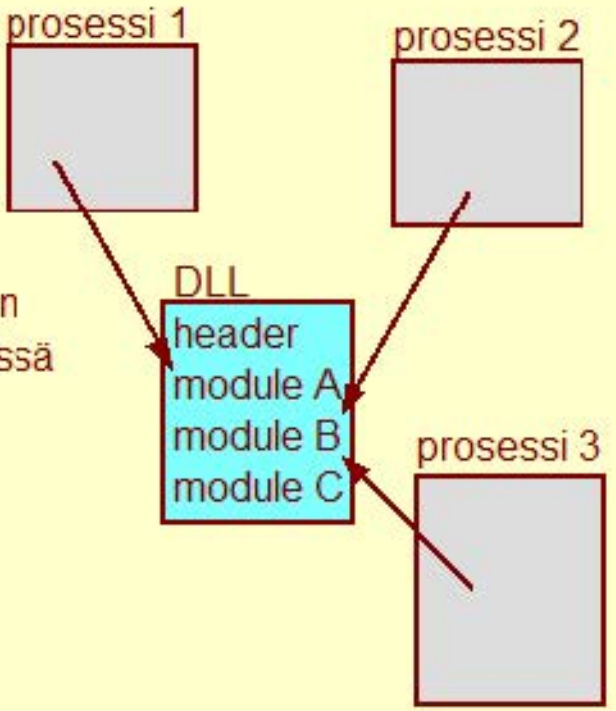
Dynaamisessa linkityksessä optimoidaan ohjelman kokoa ja käynnistymisen nopeutta suoritusajan kustannuksella. Ajomoduuliin otetaan mukaan vain osa kaikista moduuleista ja muut linkitetään lennossa sitä mukaa kuin niitä tarvitaan. Linkitysaika on tietenkin jonkin verran aikaa vievä, joten tämä menetelmä ei sovi kaikkialle - esimerkkinä vaikkapa lentokoneen hallintajärjestelmä. Staattisen ja dynaamisen linkityksen välimaastossa on vielä useita eri menetelmiä, joilla yritetään yhdistellä molempien menetelmien hyviä puolia samalla minimoiden niiden heikkouksia.

Windows DLL (dynamically linked library)

dll yleinen geneerinen kirjasto
drv driver, laiteajuri
fon fontti, kirjasintyyli

Koodia, dataa tai molempia

- säästää muistitilaa myös yhteiskäytön vuoksi
- helppo vaihtaa versiota
  - vaihda vanha DLL uuteen
  - seur. suorituskerralla uusi versio käyttöön
- ajomoduuli kootaan kuten tavallinen ajomoduuli
  - DLL moduuliviitteet merkitty lipukkeilla



Windows'in epäsuora dynaaminen linkitys

- kaikki viitatut moduulit ladataan lopulta (virtuaali)muistiin
- lataus aloitetaan heti ohjelman käynnistymisen yhteydessä
- DLL'ään viitataan staattisesti linkitetyn liitospalikan (import library) avulla

Windows'in suora dynaaminen linkitys

- DLL ladataan muistiin vain jos sitä käytetään
- koodiin generoidaan viittauspaikalle liitospalikan kutsu, jonka avulla linkitys käynnistyy vasta ensimmäisen viittauksen yhteydessä

Copyright Teemu Kerola 2005

Windowsissa dynaamisesti linkitet:äviä kirjastomoduuleja kutsutaan DLL:ksi. DLL:t voivat sisältää koodia, dataa tai molempia. Tiedostojärjestelmä tunnistaa useita erilaisia DLL-moduuleja. Loppuliite .dll tarkoittaa geneeristä moduulia, .drv viittaa laiteajuriin ja .fon kirjasintyyliin. DLL:t ovat hyvin käteviä kirjastomoduulien versioiden päivittämiseen. Riittää, kun uusi DLL laitetaan paikalleen ja heti seuraavalla suorituskerralla se otetaan automaattisesti käyttöön. Haittana tietenkin on DLL:n linkittämistyö joka suorituskerralla sen sijaan, että se tehtäisiin yhdellä kertaa kunnolla lcppuun ennen ohjelman suorittamista.

Windows DLL (dynamically linked library)

Koodia, dataa tai molempia

- säästää muistitilaa myös yhteiskäytön vuoksi
- helppo vaihtaa versiota
  - vaihda vanha DLL uuteen
  - seur. suorituskerralla uusi versio käyttöön
- ajomoduuli kootaan kuten tavallinen ajomoduuli
  - DLL moduuliviitteet merkitty lipukkeilla

```
        ...
        Call StubS
        ...
StubS   "wait until load done"
        Call S
        return
```

Windows'in epäsuora dynaaminen linkitys (implicit linking)

- kaikki viitatut moduulit ladataan lopulta (virtuaali)muistiin
- lataus aloitetaan heti ohjelman käynnistymisen yhteydessä
- DLL'ään viitataan staattisesti linkitetyn liitospalikan (import library) avulla

Windows'in suora dynaaminen linkitys

- DLL ladataan muistiin vain jos sitä käytetään
- koodiin generoidaan viittauspaikalle liitospalikan kutsu, jonka avulla linkitys käynnistyy vasta ensimmäisen viittauksen yhteydessä

Copyright Teemu Kerola 2005

Windowsissa on kaksi tapaa toteuttaa dynaaminen linkitys. Epäsuorassa dynaamisessa linkityksessä itse latausmoduuli on pieni, joten ohjelmat käynnistyvät nopeasti. Kaikki DLL:t kuitenkin ladataan muistiin, mutta niiden lataus tapahtuu samanaikaisesti kun ohjelmaa suoritetaan. DLL:ään viitattaessa voidaan sitten joutua odottamaan, että juuri sen DLL:n lataus on saatu päätökseen. Latausmoduulissa on itse moduulin asemesta vain pieni tynkä tai liitospalikka, jossa oleva koodi tarkistaa moduulin lataustilan ennen sen kutsua.

Windows DLL (dynamically linked library)

Koodia, dataa tai molempia

- säästää muistitilaa myös yhteiskäytön vuoksi
- helppo vaihtaa versiota
  - vaihda vanha DLL uuteen
  - seur. suorituskerralla uusi versio käyttöön
- ajomoduuli kootaan kuten tavallinen ajomoduuli
  - DLL moduuliviitteet merkitty lipukkeilla

```
        ...
        Call StubS
        ...
StubS   if  "ensimmäinen viittaus" then
                "link S"
                "load S"
        Call S
        return
```

Windows'in epäsuora dynaaminen linkitys

- kaikki viitatut moduulit ladataan lopulta (virtuaali)muistiin
- lataus aloitetaan heti ohjelman käynnistymisen yhteydessä
- DLL'ään viitataan staattisesti linkitetyn liitospalikan (import library) avulla

Windows'in suora dynaaminen linkitys (explicit linking)

- DLL ladataan muistiin vain jos sitä käytetään
- koodiin generoidaan viittauspaikalle liitospalikan kutsu, jonka avulla linkitys käynnistyy vasta ensimmäisen viittauksen yhteydessä

Copyright Teemu Kerola 2005

Windows'in suora dynaaminen linktys on puhdas dynaamisen linkitykser toteutus. DLL-moduulia ei ladata muistiin lainkaan, jos siihen ei tule viittausta ja viitattujenkin DLL-moduulien lataus aloitetaan vasta ensimmäisen viittauksen sattuessa. Odotusajat tällaisessa tapauksessa voivat muodostua pitkiksi, jopa useaksi sekunniksi. Tämä on kuitenkin hyvin käyttökelpoinen menetelmä esimerkiksi tietokonepelien eri tasojen tai harvinaisten kirjasintyylien suhteen.Yhteisenä etuna molemmissa tapauksissa on kirjastomoduulien päivitysmahdollisuus siten, että niitä käyttäviin ohjelmiin ei tarvitse lainkaan koskea.

Sijainnista riippumaton koodi (position independent code)

Miten tehdä koodi sellaiseksi, että siirrettäessä se toiseen paikkaan mitään osoitetta ei tarvitse päivittää?

Ei viittauksia suorien tai fyysisten osoitteiden avulla, jotka ovat riippuvaisia koodin sijainnista muistissa

Kaikki muistiviittaukset ovat

- absoluuttisia (esim. keskeytyskäsittelijän osoite),
- suhteessa PC:hen tai
- pinossa

```
Call   SP, @0x00000003
Jump   -16(PC)
Add  R1, -4(FP)
```

Copyright Teemu Kerola 2005

Olisi toivottavaa, että mahdollisimman moni moduuli olisi sijainnista riippumatonta, jossa moduuli voidaan sijoittaa mihin päin tahansa muistiavaruutta ilman, että sitä tarvitsee muuttaa lainkaan. Toisin sanoen, sen uudelleensijoitushakemisto on tyhjä! Tähän voidaan päästä, jos kaikki moduulin koodi- ja dataviitteet ovat suhteellisia johonkin laiterekisteriin, esimerkiksi koodi- tai datasegmenttirekisteriin, aktivointitietueen kantarekisteriin tai käskynlaskuriin.

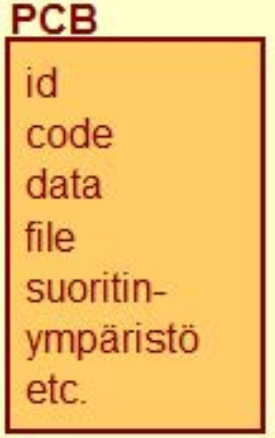
Lataus

Linkityksen tuottamasta ajomoduulista luodaan suorituskelpoinen prosessi.

- rakenna PCB ja sen viitteeet kuntoon

Prosessin koodialueet ja tarvittava data-alue ladataan muistiin, prosessi siirretään R-to-R jonoon odottamaan suoritusvuoroa

Suoritusvuoron alussa MMU ja laiterekisterit ladataan PCB:stä tämän prosessin tiedoilla



Copyright Teemu Kerola 2005

Ohjelman latauksen yhteydessä käyttöjärjestelmä luo ensin tyhjän prosessin ja sitten muokkaa sen ajomoduulin tietojen perusteella tämän ohjelman mukaiseksi. Prosessin tiedot talletetaan PCB:hen. Uuden prosessin tarvittavat koodi- ja data-alueet kopioidaan muistiin ja prosessi laitetaan R-to-R jonoon odottamaan vuoroaan suorittimelle. Sitten myöhemmin vuoron tultua prosessin suoritinympäristötiedot ladataan PCB:stä laiterekistereihin ja prosessin suoritus voi alkaa.

Nimien sidonta (name binding)

Milloin symbolin (L) suoritusaikainen todellinen muistiosoite tai muu lopullinen arvo sidotaan?

Olisi joustavaa tehdä sitominen mahdollisimman myöhään

Olisi ajankäytön kannalta optimaalista tehdä sitominen mahdollisimman aikaisin - ainakin ennen suoritusta

Muuttujan sijaintipaikkaa ei voi siirtää sitomisen jälkeen ilman lisätyötä tai laitteistotukea

symbolin sitomisaika

ohjelman kirjoitusaikana
- esim. vakiot

käännösaikana
- esim. käskykoodi

linkityksessä
- esim. kutsukohta

latauksessa
- esim. data-alueen osoite

prosessin vaihdossa
- esim. kantarekisteri

konekäskyn suoritusaikana
- esim. virtuaaliosoite

Copyright Teemu Kerola 2005

Ennen ohjelman suoritusta jokaiselle käytetylle symbolille on täytynyt löytyä jokin arvo. Tätä kutsutaan nimen sidonnaksi. Joustavuuden takia nimen sidonta olisi hyvä tehdä mahdollisimman myöhään (kuten esim DLL:n käytettäessä), mutta suoritustehon kannalta se taas olisi hyvä tehdä mahdollisimman aikaisin. Kun nimen sidonta jollekin tunnukselle on tehty, niin tätä tunnusta ei tarvitse enää käsitellä jatkossa. Kun lopullinen arvo on löytynyt, niin sen voi sijoittaa kaikkialle symbolin itsensä asemesta.

Käännös, linkitys ja lataus

Käännös
Linkitys
Dynaaminen linkitys
Lataus

Käännösyksikkö
Lausekielinen ohjelma tai moduuli
osoitteet: symboliset nimet

Objektimoduuli
Konekielinen ohjelma tai sen osa
osoitteet: lineaariset (per moduuli)

Ajomoduuli
Linkitetty ajovalmis ohjelma
osoitteet: lineaariset (koko ohjelma)

Prosessi
Suorituskelpoinen ohjelma
osoitteet: lineaariset (koko ohjelma)

Copyright Teemu Kerola 2005

Kävimme juuri läpi koko prosessin, jonka avulla lähdekielisestä ohjelmasta saadaan lopulta aikaiseksi järjestelmässä suorituskelpoinen prosessi. Tutustuimme käännökseen pääasiassa assembler-kielen osalta, ja jätimme korkean tason kielten käännösten detaljit seuraaville kursseille. Esittelimme linkityksen perusidean ja kävimme läpi staattisen ja dynaamisen linkityksen erot. Dynaaminen linktys esiteltiin lähinnä Windows-esimerkin avulla. Objektimoduulien lataus prosesseiksi käsiteltiin pääpiirteittäin.