

Aliohjelmien toteutus

Aliohjelmatyypit

Parametrien tyypit

Aktivointitietue

Aktivointitietuepino

Rekursio

Käyttöjärjestelmäpalvelut

Copyright Teemu Kerola 2004

Tässä luennossa tutustutaan aliohjelmien toteutukseen konekielitasolla. Esittelemme ensin erilaiset aliohjelmatyypit ja aliohjelmien parametrityypit. Käymme sitten läpi aliohjelmien toteutusmekanismin aktivointitietueen avulla ja näytämme, kuinka aliohjelmien kutsu/paluu -hierarkia on toteutettu aliohjelmapinon avulla. Käymme myös läpi rekursiion vaatimukset aliohjelmien toteutukseen ja miten käyttöjärjestelmäpalvelujen käyttö eroaa tavallisten aliohjelmien käytöstä ja mitä yhteistä niillä on.

Aliohjelmatyypit

Korkean tason ohjelmointikielen käsitteet

aliohjelma, proseduuri

- parametrit (sisäänmenoparametrit, ulostuloparametrit)

```
procedure P(x : integer);
```

```
Q (int x; char s[ ])
```

funktio

- parametrit, paluuarvo

```
int F (int x; float y)
```

```
function Comp ( x: real, var y: int) : int ;
```

metodi

- parametrit, ehkä paluuarvo

```
private static void F(int x)
```

Konekielitasen vastaavat käsitteet

aliohjelma, proseduuri

- parametrit ja paluuarvo (paluuarvot)

```
int F (float y)
```

```
void Fun (int z, char c, char [ ] s)
```

Copyright Teemu Kerola 2004

Korkean tason kielissä aliohjelmia on useampaa eri tyyppiä, vaikka ne perusidealtaan ovatkin hyvin samankaltaisia. Aliohjelmahan tarkoittaa kerran määriteltä, mutta mahdollisesti useasta eri kohtaa kutsuttavaa rutiinia, jonka toimintaa voidaan ohjata parametreilla ja joka voi myös palauttaa arvoja ulostuloparametrien avulla. Aliohjelmaan voi sisältyä paikallisia muuttujia, mutta siinä voi myös viitata globaaleihin ja ehkä myös muiden kutsupolulla olevien aliohjelmien tietorakenteisiin. Suurimmat erot eri ohjelmointikielten aliohjelmissä on juuri aliohjelmien omien tietorakenteiden näkyvyysalueet.

Aliohjelmatyypit

Korkean tason ohjelmointikielen käsitteet

aliohjelma, proseduuuri

- parametrit (sisäänmenoparametrit, ulostuloparametrit)

```
procedure P(x : integer);
```

```
Q (int x; char s[ ])
```

funktio

- parametrit, paluuarvo

```
int F (int x; float y)
```

```
function Comp ( x: real, var y: int) : int ;
```

metodi

- parametrit, ehkä paluuarvo

```
private static void F(int x)
```

Konekielitasen vastaavat käsitteet

aliohjelma, proseduuuri

- parametrit ja paluuarvo (paluuarvot)

```
int F (float y)
```

```
void Fun (int z, char c, char [ ] s)
```

Copyright Teemu Kerola 2004

Otaksumme nyt jatkossa, että aliohjelmassa voi viitata ainoastaan sen omiin paikallisiin tietorakenteisiin tai globaaleihin, pääohjelmatasolla määriteltyihin tietorakenteisiin. Monimutkaisemmat tunnusten näkyvyysalueet käsitellään sitten myöhemmillä ohjelmointikielten kursseilla. Perustapaus aliohjelmasta sisältää sen nimen ja parametrit. Joissakin kielissä (osa) parametreista voi olla ulostuloparametreja eli niiden arvo voi muuttua aliohjelman suorituksen aikana, mutta esimerkiksi C:ssä ja Javassa parametrina annettu muuttuja ei voi vaihtaa arvoaan aliohjelman suorituksen aikana.

Aliohjelmatyypit

Korkean tason ohjelmointikielen käsitteet

aliohjelma, proseduuri

- parametrit (sisäänmenoparametrit, ulostuloparametrit)

```
procedure P(x : integer);
```

```
Q (int x; char s[ ])
```

funktio

- parametrit, paluuarvo

```
int F (int x; float y)
```

```
function Comp ( x: real, var y: int) : int ;
```

metodi

- parametrit, ehkä paluuarvo

```
private static void F(int x)
```

Konekielitason vastaavat käsitteet

aliohjelma, proseduuri

- parametrit ja paluuarvo (paluuarvot)

```
int F (float y)
```

```
void Fun (int z, char c, char [ ] s)
```

Copyright Teemu Kerola 2004

Funktio on erikoismuoto geneerisestä aliohjelmasta. Se eroaa aliohjelmasta siinä, että se palauttaa aina jonkin arvon. Yleensä tämä arvo voi olla vain yksinkertaista yhteen sanaan mahtuvaa tietotyyppiä (esim. kokonaisluku tai liukuluku), mutta jotkut ohjelmointikielet sallivat myös monimutkaisten tietotyyppien (esim. tietue) arvon palauttamisen. Voidaan myös ajatella, että aliohjelma on erikoistapaus funktiosta, jossa paluuarvoa ei ole. Joissakin ohjelmointikielissä on vain funktioita, ja aliohjelmat ovat tällöin funktioita, joiden paluuarvo puuttuu eli on 'void'.

Aliohjelmatyypit

Korkean tason ohjelmointikielen käsitteet

aliohjelma, proseduuri

- parametrit (sisäänmenoparametrit, ulostuloparametrit)

```
procedure P(x : integer);
```

```
Q (int x; char s[ ])
```

funktio

- parametrit, paluuarvo

```
int F (int x; float y)
```

```
function Comp ( x: real, var y: int) : int ;
```

metodi

- parametrit, ehkä paluuarvo

```
private static void F(int x)
```

Konekielitason vastaavat käsitteet

aliohjelma, proseduuri

- parametrit ja paluuarvo (paluuarvot)

```
int F (float y)
```

```
void Fun (int z, char c, char [ ] s)
```

Copyright Teemu Kerola 2004

Olioperustaisissa kielissä aliohjelman korvaa metodi. Metodille määritellään näkyvyysalue, mutta muuten se on hyvin samankaltainen aliohjelmien ja funktioiden kanssa. On myös 'tavallisia' lohkorakenteisia ohjelmointikieliä, joissa voidaan määritellä aliohjelmien ja funktioiden näkyvyysalueita.

Aliohjelmatyypit

Korkean tason ohjelmointikielen käsitteet

aliohjelma, proseduuri

- parametrit (sisäänmenoparametrit, ulostuloparametrit)

```
procedure P(x : integer);
```

```
Q (int x; char s[ ])
```

funktio

- parametrit, paluuarvo

```
int F (int x; float y)
```

```
function Comp ( x: real, var y: int) : int ;
```

metodi

- parametrit, ehkä paluuarvo

```
private static void F(int x)
```

Konekielitason vastaavat käsitteet

aliohjelma, proseduuri

- parametrit ja paluuarvo (paluuarvot)

```
int F (float y)
```

```
void Fun (int z, char c, char [ ] s)
```

Copyright Teemu Kerola 2004

Konekielen tasolla nämä kaikki korkean tason kielen aliohjelmatyypit ovat hyvin samankaltaisia. Meillä on vain aliohjelma (proseduuri), jolla voi olla useita parametreja ja joka voi palauttaa jonkin arvon. Parametrien ja paluuarvon tyyppi voi konekielen tasolla olla mitä vain, mutta kääntäjät tietenkin rajaavat ne kyseisen korkean tason ohjelmointikielen syntaksiin ja semantiikkaan sopiviksi. Jatkossa puhumme vain aliohjelmista, oli vastaava korkean tason kielen käsite sitten aliohjelma, funktio tai metodi.

Muodolliset ja todelliset parametrit

Muodolliset parametrit ja paluuarvo

- määritelty aliohjelmassa ohjelmointihetkellä
- ohjelmoijan valitsemat nimet, järjestys ja tyyppi
- paluuarvon nimi on sama kuin funktion nimi, sillä on annettu tyyppi

Tulosta (int x, y)

Laske (int x): int

Todelliset parametrit ja paluuarvo

- todelliset parametrit sijoitetaan muodollisten parametrien paikalle suoritusaikana kutsuhetkellä
- sijoitus muodollisten parametrien paikalle usein järjestyksen perusteella, joskus myös annetun muodollisen parametrin nimen perusteella
- paluuarvo saadaan suoritusaikana paluuhetkellä ja sitä käytetään kuin mitä tahansa arvoa

Tulosta (5, apu)

Tulosta (y=>apu, x=>5)

x = Laske (y+234)

Copyright Teemu Kerola 2004

Aliohjelmien parametrit määritellään ohjelmointihetkellä, jolloin ohjelmoija antaa ne jossakin järjestyksessä ja antaa parametreille aliohjelman toteutuskoodissa käytettävän nimen ja tyypin. Muodolliset parametrit ovat aliohjelman koodissa hyvin samanlaisessa asemassa kuin paikalliset muuttujatkin. Merkittävä ero on kuitenkin, että niiden alkuarvo määräytyy vasta kutsuhetkellä.

Muodolliset ja todelliset parametrit

Muodolliset parametrit ja paluuarvo

- määritelty aliohjelmassa ohjelmointihetkellä
- ohjelmoijan valitsemat nimet, järjestys ja tyyppi
- paluuarvon nimi on sama kuin funktion nimi, sillä on annettu tyyppi

Tulosta (int x, y)

Laske (int x): int

Todelliset parametrit ja paluuarvo

- todelliset parametrit sijoitetaan muodollisten parametrien paikalle suoritusaikana kutsuhetkellä
- sijoitus muodollisten parametrien paikalle usein järjestyksen perusteella, joskus myös annetun muodollisen parametrin nimen perusteella
- paluuarvo saadaan suoritusaikana paluuhetkellä ja sitä käytetään kuin mitä tahansa arvoa

Tulosta (5, apu)

Tulosta (y=>apu, x=>5)

x = Laske (y+234)

Copyright Teemu Kerola 2004

Todelliset parametrit ovat tietenkin ne parametrit, jotka aliohjelmalle annetaan sitä kutsuttaessa. Useimmiten todelliset parametrit annetaan samassa järjestyksessä, kuin mitä muodollisetkin parametrit annettiin, mutta esimerkiksi Ada-ohjelmointikielessä todelliset parametrit voidaan antaa myös mielivaltaisessa järjestyksessä, mutta tällöin niiden vastaavuus muodollisiin parametreihin pitää sitten antaa eksplisiittisesti. Funktion paluuarvo määräytyy vasta aliohjelmasta paluuhetkellä, mutta senkin tulee tietenkin olla funktion määrittelyssä sovittua tyyppiä.

Parametrityypit

Arvoparametri

- välitetään (todellisen) parametrin arvo kutsuhetkellä
- todellisen parametrin arvoa ei voi muuttaa, koska vain arvo (kopio) välitettiin

lausekkeella on arvo

Laske (5, y, x+43);

Viiteparametri

- välitetään (todellisen) parametrin osoite
- välitettävällä parametrilla pitää olla osoite
- arvoa voidaan muuttaa, voi olla ulostuloparametri

Päivitä (X, 7, Summa);

Päivitä (Y+2, K, Osasumma);

Nimiparametri

- välitetään (todellisen) parametrin nimi, muodollinen parametri on merkkijono
- mikä tahansa merkkijono kelpaa todelliseksi parametriksi
- käsitteellisesti monimutkainen, vaatii aliohjelman (uudelleen)käännöksen tai tulkitsemisen

Swap(i, k);

Swap(5.3, T[i]);

Swap(i, Y := R+6);

Copyright Teemu Kerola 2004

Parametreja on kolmea eri perustyyppiä. Yleisin käytetty on arvoparametri, jossa muodollinen parametri kutsun yhteydessä alustetaan todellisen parametrin arvolla. Todellinen parametri on turvassa muutoksilta, koska aliohjelmalle välitetään ainoastaan sen arvon kopio. Arvoksi sopii mikä tahansa, joten todellinen parametri voi olla myös aritmeettisen lausekkeen arvo.

Parametrityypit

Arvoparametri

- välitetään (todellisen) parametrin arvo kutsuhetkellä
- todellisen parametrin arvoa ei voi muuttaa, koska vain arvo (kopio) välitettiin

Laske (5, y, x+43);

Viiteparametri

- välitetään (todellisen) parametrin osoite
- välitettävällä parametrilla pitää olla osoite
- arvoa voidaan muuttaa, voi olla ulostuloparametri

Päivitä (X, 7, Summa);

Päivitä (Y+2, K, Osasumma);

ei saa muuttaa!

ei osoitetta!

Nimiparametri

- välitetään (todellisen) parametrin nimi, muodollinen parametri on merkkijono
- mikä tahansa merkkijono kelpaa todelliseksi parametriksi
- käsitteellisesti monimutkainen, vaatii aliohjelman (uudelleen)käännöksen tai tulkitsemisen

Swap(i, k);

Swap(5.3, T[i]);

Swap(i, Y := R+6);

Copyright Teemu Kerola 2004

Toinen yleisesti käytetty parametrityyppi on viiteparametri, jossa välitetään todellisen parametrin osoite. Jos todellinen parametri on esimerkiksi muuttuja Summa, niin aliohjelma Päivitä voi muuttaa muuttujan Summa arvoa, koska sillä on käytössään muuttujan Summa osoite. Olisi vaarallista välittää esimerkiksi vakion 7 osoite, koska emme halua antaa aliohjelmalle mahdollisuutta vaihtaa vakion 7 arvoa! Todellinen parametri täytyy kuitenkin olla sellainen olio, jolla on osoite. Esimerkiksi aritmeettinen lauseke ei kelpaa tässä kohtaa yleensä, koska siihen liittyy käsitteellisesti ainoastaan arvo, mutta ei osoitetta.

Parametrityypit

Arvoparametri

- välitetään (todellisen) parametrin arvo kutsuhetkellä `Laske (5, y, x+43);`
- todellisen parametrin arvoa ei voi muuttaa, koska vain arvo (kopio) välitettiin

Viiteparametri

- välitetään (todellisen) parametrin osoite `Päivitä (X, 7, Summa);`
- välitettävällä parametrilla pitää olla osoite `Päivitä (Y+2, K, Osasumma);`
- arvoa voidaan muuttaa, voi olla ulostuloparametri

Nimiparametri

- välitetään (todellisen) parametrin nimi, muodollinen parametri on merkkijono
- mikä tahansa merkkijono kelpaa todelliseksi parametriksi
- käsitteellisesti monimutkainen, vaatii aliohjelman (uudelleen)käännöksen tai tulkitsemisen

`Swap( i, k );`

kokonaisluku

`Swap( 5.3, T[i] );`

liukuluku taulukko

`Swap( i, Y := R+6 );`

lause

Copyright Teemu Kerola 2004

Kolmas parametrityyppi on nimiparametri, jossa parametrin arvon tai osoitteen asemesta välitetään sen nimi. Tämä on oikeastaan aika monimutkaista, koska edes parametrin arvotyyppiä ei välttämättä tarvitse ilmaista, vaikka näin usein tehdäänkin. Todellinen parametri on nyt mikä tahansa merkkijono, joka sijoitetaan aliohjelman koodiin muodollisen parametrin joka esiintymiskohtaan. Tämä voi muuttaa koodin rakennetta, joten koko koodi täytyy kääntää tai tulkita uudelleen joka kutsukerralla. Tämä tekee nimiparametrin käytännön toteutuksen aika monimutkaiseksi.

Arvoparametri

Välitetään todellisen parametrin arvo (eli arvon kopio)

Tulosta (A+3, B)

- muuttuja, vakio, lauseke, olioviite (olion osoite), data- tai koodiosoitin

Aliohjelma ei voi mitenkään muuttaa todellisena parametrina käytettyä muuttujaa tai arvoa

- muutetaan alkuperäisen parametrin arvon kopiota
- jos todellinen parametri on osoitin (pointteri), niin todellista parametria (osoitinta) ei voi muuttaa, mutta sen osoittamaa tietoa voi tietenkin muuttaa

88
laske (R, ptrLuku);

```
laske (int y,  
       int *ptrX)  
{  
    ...  
    y = 5  
    *ptrX = 10;  
    ptrX = ptrP;  
    ...  
}
```



Javassa ja C'ssä vain arvoparametreja

- parametrina voi silti olla osoittimen tai osoitinmuuttujan arvo

Copyright Teemu Kerola 2004

Arvoparametri on helpointa ymmärtää. Yleisesti ottaen muodollinen parametri vastaa jonkin tyyppin muuttujaa, joka sitten alustetaan kutsuhetkellä todellisen parametrin arvolla. Parametri itse voi tietenkin olla mitä tahansa arvotyyppiä, esimerkiksi kokonaisluku, merkkijono, liukuluku tai osoitin mihin tahansa tietoon. Arvoparametrin käyttö osoittimien välitykseen antaa niille huomattavan määrää voimaa, mutta se on myös jonkin verran vaarallista, koska kutsuvan ohjelman täytyy nyt luottaa siihen, että aliohjelma ei käytä väärin samaansa osoitetta.

Arvoparametri

Välitetään todellisen parametrin arvo (eli arvon kopio)

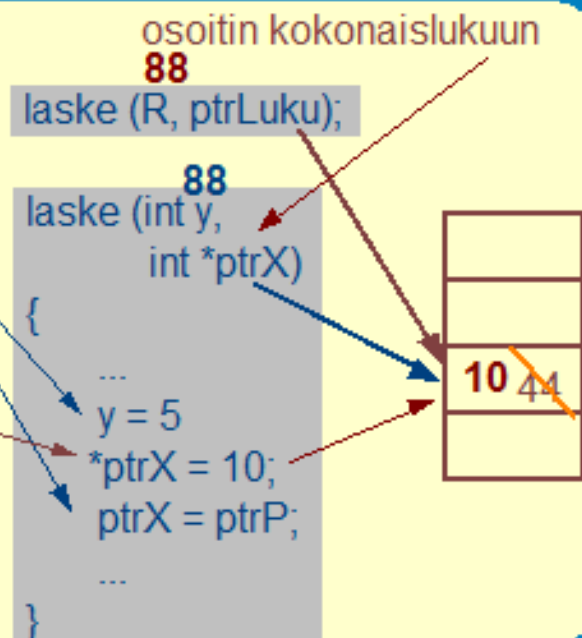
Tulosta (A+3, B)

- muuttuja, vakio, lauseke, olioviite (olion osoite), data- tai koodiosoitin

Aliohjelma ei voi mitenkään muuttaa todellisena parametrina käytettyä muuttujaa tai arvoa

- muutetaan alkuperäisen parametrin arvon kopiota
- jos todellinen parametri on osoitin (pointteri), niin todellista parametria (osoitinta) ei voi muuttaa, mutta sen osoittamaa tietoa voi tietenkin muuttaa

```
*ptrX = 10; /* muuta ptrX'n osoittamaa arvoa */
```



Javassa ja C'ssä vain arvoparametreja

- parametrina voi silti olla osoittimen tai osoitinmuuttujan arvo

Copyright Teemu Kerola 2004

Koska arvoparametri on alustettu todellisen parametrin arvon kopiolla, todellinen parametri on täysin turvassa muutoksilta. Esimerkissä aliohjelma 'laske' ei voi muuttaa todellisen parametrin 'R' arvoa 88, mutta voi muuttaa sillä alustetun kopion 'y' arvoa. Osoitinparametri 'ptrX' osoittaa alkuaan samaan muuttujaan kuin osoitinmuuttuja 'ptrLuku', joten sen kautta voidaan muuttaa osoitinmuuttujan 'ptrLuku' osoittamaa arvoa, mutta itse osoitinmuuttuja 'ptrLuku' on turvassa muutoksilta. Se osoittaa aliohjelmasta 'laske' paluun jälkeen samaan muistipaikkaan, vaikka kyseisen muistipaikan sisältö onkin muuttunut.

Arvoparametri

Välitetään todellisen parametrin arvo (eli arvon kopio)

Tulosta (A+3, B)

- muuttuja, vakio, lauseke, olioviite (olion osoite), data- tai koodiosoitin

Aliohjelma ei voi mitenkään muuttaa todellisena parametrina käytettyä muuttujaa tai arvoa

- muutetaan alkuperäisen parametrin arvon kopiota
- jos todellinen parametri on osoitin (pointteri), niin todellista parametria (osoitinta) ei voi muuttaa, mutta sen osoittamaa tietoa voi tietenkin muuttaa

88
laske (R, ptrLuku);

```
laske (int y,  
       int *ptrX)  
{  
    ...  
    y = 5  
    *ptrX = 10;  
    ptrX = ptrP;  
    ...  
}
```



Javassa ja C'ssä vain arvoparametreja

- parametrina voi silti olla osoittimen tai osoitinmuuttujan arvo

Copyright Teemu Kerola 2004

C- ja Java-kielissä on vain arvoparametreja ja kaikkien parametrin tulee olla yksinkertaista tietotyyppiä. Esimerkiksi 10-alkioinen taulukko ei voi olla parametri, mutta sen osoite voi olla parametri. Osoitteethan ovat aina yksinkertaista tietotyyppiä ja ne mahtuvat aina yhteen tai kahteen sanaan. Tämä yksinkertaistaa näitä kieliä huomattavasti, mutta rajaa myös pois mahdollisuuden, että parametrina annettua osoitetta voisi käyttää vain tiedon lukemiseen. Javassa ja C'ssä ei ole erikseen määrettä sisäänmeno- ja ulostuloparametreille, kuten esimerkiksi Pascal- tai Ada-kielissä.

Viiteparametri

muuttujan sum osoite, ei arvo

Psummaa (54, sum)

Välitetään todellisen parametrin osoite

- muuttujan osoite, osoitinmuuttujan osoite, tietueen osoite, aliohjelman osoite, ...
- ero selvemmin näkyvissä korkean tason kielen tasolla
- konekielen tasolla eri viiteparametrin ja arvoparametrina välitetyn osoitteen välillä on aika pieni

Aliohjelma voi muuttaa parametrina annetun muuttujan arvoa

C-kielen arvoparametrina osoite

```
Csummaa (int x, int *cum_sum)
{
    ...
    *cum_sum = *cumsum + x
    ...
}
```

Csummaa (6, &kok_lkm);

Pascal-kielen var-parametri

```
Psummaa (x: int; var cum_sum: int)
{
    ...
    cum_sum := cum_sum + x;
    ...
}
```

Psummaa (6, kok_lkm);

Copyright Teemu Kerola 2004

Viiteparametrissa välitetään nimetyn muuttujan arvon asemesta sen osoite. Todellisen parametrin täytyy nyt olla jokin rakenne, jolla on osoite. Konekielen tasolla tämä on hyvin samankaltainen tilanne, kuin jos arvoparametrina välitetään jonkun tiedon osoite, mutta korkean tason kielen tasolla ero on suuri.

Viiteparametri

muuttujan sum osoite, ei arvo

Psummaa (54, sum)

Välitetään todellisen parametrin osoite

- muuttujan osoite, osoitinmuuttujan osoite, tietueen osoite, aliohjelman osoite, ...
- ero selvemmin näkyvissä korkean tason kielen tasolla
- konekielen tasolla eri viiteparametrin ja arvoparametrina välitetyn osoitteen välillä on aika pieni

Aliohjelma voi muuttaa parametrina annetun muuttujan arvoa

C-kielen arvoparametrina osoite

```
Csummaa (int x, int *cum_sum)
{
    ...
    *cum_sum = *cumsum + x
    ...
}
```

Csummaa (6, &kok_lkm);

Pascal-kielen var-parametri

```
Psummaa (x: int; var cum_sum: int)
{
    ...
    cum_sum := cum_sum + x;
    ...
}
```

Psummaa (6, kok_lkm);

Copyright Teemu Kerola 2004

Viiteparametria käytettäessä kutsuvasta rutiinista annetaan aliohjelmalle aina pointteri kutsuvan rutiinin käytössä olevaan tietorakenteeseen, jota kutsuttu aliohjelma voi mielin määrin muuttaa. Tämä vaatii hyvin paljon luottamusta kutsuttavaan aliohjelmaan, joka voi sijaita jossakin ulkopuolisessa kirjastomodulissa. Tämän vuoksi useissa ohjelmointikielissä ei ole toteutettu viiteparametreja ja osoittimen käyttö pitää aina koodata ekplisiittisesti näkyville. Toisaalta, ei tämäkään tapa ole yhtään sen turvallisempi, mutta voi itse asiassa olla vielä virhealttiimpi.

Viiteparametri

muuttujan sum osoite, ei arvo

Psummaa (54, sum)

Välitetään todellisen parametrin osoite

- muuttujan osoite, osoitinmuuttujan osoite, tietueen osoite, aliohjelman osoite, ...
- ero selvemmin näkyvissä korkean tason kielen tasolla
- konekielen tasolla eri viiteparametrin ja arvoparametrina välitetyn osoitteen välillä on aika pieni

Aliohjelma voi muuttaa parametrina annetun muuttujan arvoa

C-kielen arvoparametrina osoite

```
Csummaa (int x, int *cum_sum)
{
    ...
    *cum_sum = *cumsum + x
    ...
}
```

Csummaa (6, &kok_lkm);

Pascal-kielen *var*-parametri

```
Psummaa (x: int; var cum_sum: int)
{
    ...
    cum_sum := cum_sum + x;
    ...
}
```

Psummaa (6, kok_lkm);

Copyright Teemu Kerola 2004

Pascal-kielessä on viiteparametreja ja ne merkitään var-etuliitteellä aliohjelman määrittelyssä. Aliohjelmaa käytettäessä ei kuitenkaan ole mitenkään näkyvissä, että kyseinen parametri onkin viite- eikä arvoparametri. Se pitää vain ohjelmoijan muistaa. Toisaalta on kätevää, kun esimerkissä annettu viiteparametri 'kok_lkm' voidaan antaa ihan samalla tavalla kuin samassa kutsussa oleva arvoparametri 6.

Viiteparametri

muuttujan sum osoite, ei arvo

Psummaa (54, sum)

Välitetään todellisen parametrin osoite

- muuttujan osoite, osoitinmuuttujan osoite, tietueen osoite, aliohjelman osoite, ...
- ero selvemmin näkyvissä korkean tason kielen tasolla
- konekielen tasolla eri viiteparametrin ja arvoparametrina välitetyn osoitteen välillä on aika pieni

Aliohjelma voi muuttaa parametrina annetun muuttujan arvoa

C-kielen arvoparametrina osoite

```
Csummaa (int x, int *cum_sum)
{
    ...
    *cum_sum = *cumsum + x
    ...
}
```

Csummaa (6, &kok_lkm);

Pascal-kielen *var*-parametri

```
Psummaa (x: int; var cum_sum: int)
{
    ...
    cum_sum := cum_sum + x;
    ...
}
```

Psummaa (6, kok_lkm);

Copyright Teemu Kerola 2004

Ero C-kieleen on suuri. C:ssä pitää aliohjelman määrittelyssä kertoa, että parametri cum_sum on osoitin. Tämä vastaa hyvin paljon viiteparametrin määrittelyä, mutta eroaa siinä kohtaa, että parametri cum_sum on ihan tavallinen osoitin-tyyppinen arvoparametri. C-kielessä pitää joka kerta cum_sum:iin viitattaessa erikseen koodata näkyville, että viittaus tapahtuu osoitinmuuttujan osoittamaan tietoon, kun Pascal-kielessä tämä on täysin automaattista var-määrittelyn ansiosta. C- ja Pascal-kielissä on siis merkittäviä semanttisia eroja.

Nimiparametri

Välitetään todellisen parametrin nimi

- merkkijono, jolla korvataan nimiparametri jokaisessa viittauskohdassa
- Algol 60
- yleensä nykyisin makroissa
- 'sivuvaikutuksia'

```
void swap (name int x, y)
{
    int t;
    t := x; x := y; y := t;
}
```

vaihda i, j ?
swap (i, j)

```
{
    int t;
    t := i; i := j; j := t;
}
```

Copyright Teemu Kerola 2004

Kolmas parametrityyppi on siis nimiparametri. Nimen välittäminen parametrina on hyvin erilainen tilanne kuin arvon tai osoitteen välittäminen. Aliohjelman koodi voi muuttua hyvinkin paljon nimiparametrin yhteydessä, joten se pitää oikeastaan kääntää uudelleen joka kutsukerralla. Tämä monimutkaistaa korkean tason kieltä yleensä liikaa, joten nimiparametreja ei niissä juurikaan käytetä. Ohjelmointikielissä on kuitenkin ohjelmointia helpottavia makroja, jotka käsitellään ennen varsinaista kääntämistä ja näissä makrokielissä parametrit ovat tyypillisesti juuri nimiparametreja.

Nimiparametri

Välitetään todellisen parametrin nimi

- merkkijono, jolla korvataan nimiparametri jokaisessa viittauskohdassa
- Algol 60
- yleensä nykyisin makroissa
- 'sivuvaikutuksia'

```
void swap (name int x, y)
{
    int t;
    t := x; x := y; y := t;
}
```

vaihda i, j ?
swap (i, j)

```
{
    int t;
    t := i; i := j; j := t;
}
```

Copyright Teemu Kerola 2004

Tässä esimerkissä meillä on rutiini swap(), joka vaihtaa kahden parametrina annetun alkion arvot keskenään, käyttäen apuna paikallista muuttujaa t. Normaalitapauksessa swap()ia kutsutaan antamalla vaihdettavien muuttujien nimet (tässä i ja j), jolloin swap()in rungossa muodolliset parametri x ja y korvataan i'llä ja j'llä niiden kaikissa esiintymiskohdissaan. Siltä varalta, että i tai j olisi jotenkin väärämuotoinen (esim. i olisi merkkijono 'abc'), swap'in runko tulee tarkistaa kääntämällä se uudelleen.

Nimiparametri

Välitetään todellisen parametrin nimi

- merkkijono, jolla korvataan nimiparametri jokaisessa viittauskohdassa
- Algol 60
- yleensä nykyisin makroissa
- 'sivuvaikutuksia'

```
void swap (name int x, y)
{
    int t;
    t := x; x := y; y := t;
}
```

vaihda i, j ?

swap (i, j)

```
{
    int t;
    t := i; i := j; j := t;
}
```

vaihda n ja A[n] ?

swap (n, A[n])

```
{
    int t;
    t := n; n := A[n]; A[n] := t;
}
```

"väärä" n

Copyright Teemu Kerola 2004

Nimiparametrien varsinainen voima ja vaarallisuus tulee siitä, että nimiparametrina voi itse asiassa välittää minkä tahansa merkkijonon, joka sitten sijoitetaan muodollisen parametrin paikalle ennen lauseiden evaluointia. Esimerkiksi tälle swap-rutiinille voidaan välittää taulukon indeksi ja taulukon alkio parametreina. Tässä esimerkissä swap-rutiinin sisällä jälkimmäinen viittaus taulukon alkioon A[n] kohdistuu eri alkioon kuin ensimmäisellä viittauksella, koska muuttujan n arvo on muuttunut ja A[n] evaluoidaan vasta muutoksen jälkeen. Emme käsittele nimiparametreja enää jatkossa, mutta niiden olemassaolo on hyvä tietää.

Aliohjelman komponentit

Paluuosoite

- kutsukohtaa seuraavan käskyn osoite

Parametrien välitys

Paluuarvon välitys

Paikalliset muuttujat

Rekistereiden allokointi

- kutsuvalla ohjelman osalla voi olla käytössään rekistereitä, joiden arvo halutaan säilyttää
- kuka tallettaa ne ja palauttaa kutsun jälkeen ennalleen?
 - kutsuva rutiini? talleta aina kaikki rekisterit varmuuden vuoksi!
 - kutsuttu rutiini? talleta vain itse käyttämät rekisterit
- kutsuttu rutiini tallettaa käyttämänsä rekistereiden alkuperäiset arvot muistiin ja palauttaa ne ennalleen ennen paluuta

Aliohjelman runko

- varsinainen työ tehdään täällä

Copyright Teemu Kerola 2004

Aliohjelmat ovat siis rutiineja, joita voidaan kutsua liki mistä päin vain ohjelmaa, niillä voi olla parametreja ja paikallisia muuttujia. Lisäksi aliohjelman suorituksen aikana siinä voidaan viitata myös globaaleihin tietorakenteisiin ja joissakin ohjelmointikielissä myös joihinkin muissa aliohjelmissa määriteltyihin tietorakenteisiin. Nämä kaikki aliohjelmiin liittyvät käsitteet pitää nyt sitten toteuttaa yksi kerrallaan. Otaksumme jatkossa, että aliohjelmassa on viitattavissa vain sen omat ja globaalit tietorakenteet.

Aliohjelman komponentit

Paluuosoite

- kutsukohtaa seuraavan käskyn osoite

Parametrien välitys

Paluarvon välitys

Paikalliset muuttujat

Rekistereiden allokointi

- kutsuvalla ohjelman osalla voi olla käytössään rekistereitä, joiden arvo halutaan säilyttää
- kuka tallettaa ne ja palauttaa kutsun jälkeen ennalleen?
 - kutsuva rutiini? talleta aina kaikki rekisterit varmuuden vuoksi!
 - kutsuttu rutiini? talleta vain itse käyttämät rekisterit
- kutsuttu rutiini tallettaa käyttämänsä rekistereiden alkuperäiset arvot muistiin ja palauttaa ne ennalleen ennen paluuta

Aliohjelman runko

- varsinainen työ tehdään täällä

Copyright Teemu Kerola 2004

Aliohjelmien toteutukseen sisältyy myös puhdasta käytännöllistä hallintoa. Haluamme, että aliohjelma ei sotke kutsuvan rutiinin käytössä olevia rekistereitä. Esimerkiksi, pääohjelma pitää rekisterissä R2 koko ajan ohjelmassa tarvittavaa muuttujan HyvinTärkeä arvoa, ja halutaan, että aliohjelman Apu käytön jälkeen rekisterin R2 arvon pitää olla sama kuin se oli ennen aliohjelman Apu kutsua. Aliohjelman toteutuksen yksi perusvaatimus on siten se, että kaikkien työrekistereiden arvot pitää olla samat aliohjelmasta paluun jälkeen kuin mitä ne olivat juuri ennen aliohjelman kutsua.

Aliohjelman komponentit

Paluuosoite

- kutsukohtaa seuraavan käskyn osoite

Parametrien välitys

Paluuarvon välitys

Paikalliset muuttujat

Rekistereiden allokointi

- kutsuvalla ohjelman osalla voi olla käytössään rekistereitä, joiden arvo halutaan säilyttää
- kuka tallettaa ne ja palauttaa kutsun jälkeen ennalleen?
 - kutsuva rutiini? talleta aina kaikki rekisterit varmuuden vuoksi!
 - kutsuttu rutiini? talleta vain itse käyttämät rekisterit
- kutsuttu rutiini tallettaa käyttämänsä rekistereiden alkuperäiset arvot muistiin ja palauttaa ne ennalleen ennen paluuta

Aliohjelman runko

- varsinainen työ tehdään täällä

Copyright Teemu Kerola 2004

Totta kai aliohjelmiin sisältyy myös niiden runko, eli se osa aliohjelmää, joka tekee varsinaisen työn. Yksinkertaisissa esimerkeissämme ohjelman runko-osa on usein hyvin pieni ja näyttää jopa suhteettoman pieneltä verrattuna aliohjelmien toteutuksen muihin osiin, mutta todellisissa sovelluksissa aliohjelmien koot ovat tietenkin vähän tai aika paljonkin suurempia. Keskitymme nyt kuitenkin itse aliohjelma-käsitteen toteutukseen järjestelmässä ja tässä yhteydessä on epärelevanttia, mitä aliohjelmat oikeastaan tekevät.

Aktivointitietue (activation record, activation frame)

int funcA (int x, y)

Aliohjelman toteutustietue

- jokaisella järjestelmällä omansa

Ttk-91 järjestelmän aktivointitietue (funktiole funcA)

- Funktion paluuarvo (paluuarvot?)
- Parametrien arvot (sisääntulo- ja ulostuloparametrit)
- Paluuosoite
- Kutsukohdan aktivointitietue
- Paikalliset muuttujat ja tietorakenteet
- Käytettyjen rekistereiden alkuperäiset arvot

pinossa, muistissa

paluuarvo
parametri x
parametri y
vanha PC
vanha FP
paik. muutt. i1
paik. muutt. i2
alkup. R1
alkup. R2

FP:

Copyright Teemu Kerola 2004

Järjestelmä toteuttaa aliohjelman käsitteen yhden tietueen avulla. Tämä on todella nerokas toteutustapa niinkin monimutkaiselle ja voimakkaalle käsitteelle kuin aliohjelma. Aktivointitietueen rakenne vaihtelee järjestelmän ja ohjelmointikielen mukaan, mutta on myös mahdollista käyttää samaa aktivointitietueen rakennetta useassa eri ohjelmointikielessä. Keskitymme nyt esimerkikoneessa ttk-91:ssä käytettävään aktivointitietueeseen. Muistakaa, että tietueen sisäinen rakenne ei ole niinkään tärkeä, vaan se, mitä kenttiä siellä yleensä on.

Aktivointitietue (activation record, activation frame)

int funcA (int x, y)

Aliohjelman toteutustietue

- jokaisella järjestelmällä omansa

Ttk-91 järjestelmän aktivointitietue (funktiole funcA)

- Funktion paluuarvo (paluuarvot?)
- Parametrien arvot (sisääntulo- ja ulostuloparametrit)
- Paluuosoite
- Kutsukohdan aktivointitietue
- Paikalliset muuttujat ja tietorakenteet
- Käytettyjen rekistereiden alkuperäiset arvot

pinossa, muistissa

paluuarvo
parametri x
parametri y
vanha PC
vanha FP
paik. muutt. i1
paik. muutt. i2
alkup. R1
alkup. R2

FP:

Copyright Teemu Kerola 2004

Ttk-91 järjestelmän aktivointitietueessa on 6 erilaista kenttää ja joissakin kentissä on monta sanaa. Ne on tietueen sisällä tietyssä järjestyksessä, minkä merkitys selviää ihan kohta. Käydään nyt ensin läpi nämä kuusi kenttää ja niiden sisällöt. Ensimmäisenä kenttänä on funktion paluuarvo. Paluuarvon tietotyyppi on tietenkin kokonaisluku, koska ttk-91:ssä ei ole muita tietotyypppejä. Jos aliohjelma ei palauta mitään arvoa, niin tämä kenttä tietenkin puuttuu. Toisaalta, mikään ei konekielen tasolla estä toteuttamasta funktiota, jolla olisi monta paluuarvoa. Jatkossa rajoitamme paluuarvojen lukumäärän kuitenkin yhteen.

Aktivointitietue (activation record, activation frame)

int funcA (int x, y)

Aliohjelman toteutustietue

- jokaisella järjestelmällä omansa

Ttk-91 järjestelmän aktivointitietue (funktiole funcA)

- Funktion paluuarvo (paluuarvot?)
- Parametrien arvot (sisääntulo- ja ulostuloparametrit)
- Paluuosoite
- Kutsukohdan aktivointitietue
- Paikalliset muuttujat ja tietorakenteet
- Käytettyjen rekistereiden alkuperäiset arvot

pinossa, muistissa

paluuarvo
parametri x
parametri y
vanha PC
vanha FP
paik. muutt. i1
paik. muutt. i2
alkup. R1
alkup. R2

FP:

Copyright Teemu Kerola 2004

Seuraavana aktivointitietueessa on sen kaikki parametrit. Niille kaikille varataan tilaa yksi sana, mikä riittää juuri sopivasti sekä arvoparametrien kokonaisluvulle että viiteparametrien muistiosoitteelle. Nimiparametrejahan ei ttk-91:ssä ole. Parametrit vievät saman verran tilaa, olivatpa ne sisäänmeno- tai ulostuloparametreja. Erot sisäänmeno- ja ulostuloparametrien välillä ovat vain siinä, kuinka niitä käytetään aliohjelman rungossa.

Aktivointitietue (activation record, activation frame)

int funcA (int x, y)

Aliohjelman toteutustietue

- jokaisella järjestelmällä omansa

Ttk-91 järjestelmän aktivointitietue (funktiole funcA)

- Funktion paluuarvo (paluuarvot?)
- Parametrien arvot (sisääntulo- ja ulostuloparametrit)
- Paluuosoite
- Kutsukohdan aktivointitietue
- Paikalliset muuttujat ja tietorakenteet
- Käytettyjen rekistereiden alkuperäiset arvot

pinossa, muistissa

paluuarvo
parametri x
parametri y
vanha PC
vanha FP
paik. muutt. i1
paik. muutt. i2
alkup. R1
alkup. R2

FP:

Copyright Teemu Kerola 2004

Aliohjelmien perusidea on, että niitä voidaan kutsua mistä päin tahansa koodia, ja, että kutsun jälkeen kontrolli palaa kutsukohtaa seuraavaan konekäskyyn. Tämä paluuosoite pitää tietenkin tallettaa kutsuhetkellä jonnekin, ja ttk-91 aktivointitietueessa se paikka on tässä kohtaa. Paluuosoite on muistiosoite ja sekin mahtuu juuri sopivasti 32-bitin sanaan ttk-91:ssä.

Aktivointitietue (activation record, activation frame)

int funcA (int x, y)

Aliohjelman toteutustietue

- jokaisella järjestelmällä omansa

Ttk-91 järjestelmän aktivointitietue (funktiolle funcA)

- Funktion paluuarvo (paluuarvot?)
- Parametrien arvot (sisääntulo- ja ulostuloparametrit)
- Paluuosoite
- Kutsukohdan aktivointitietue
- Paikalliset muuttujat ja tietorakenteet
- Käytettyjen rekistereiden alkuperäiset arvot

pinossa, muistissa

paluuarvo
parametri x
parametri y
vanha PC
vanha FP
paik. muutt. i1
paik. muutt. i2
alkup. R1
alkup. R2

FP:

Copyright Teemu Kerola 2004

Toinen aliohjelmien perusidea on tunnusten viiteympäristö, millä tarkoitetaan sitä asiaa, että mitkä tunnukset ovat milloinkin käytettävissä ja miten niihin viitataan. Tämäkin käsite toteutetaan aktivointitietueen avulla. Rekisteri R7 eli FP (frame pointer) osoittaa aina tällä hetkellä aktiivisena olevaan aktivointitietueeseen ja aliohjelman suorituksen aikana sen viiteympäristö määräytyy täysin FP:n avulla. Aliohjelmasta paluun yhteydessä halutaan kontrollin lisäksi myös viiteympäristön palaavan kutsukohtaan, ja sitä varten meillä tallessa kutsukohdan FP.

Aktivointitietue (activation record, activation frame)

int funcA (int x, y)

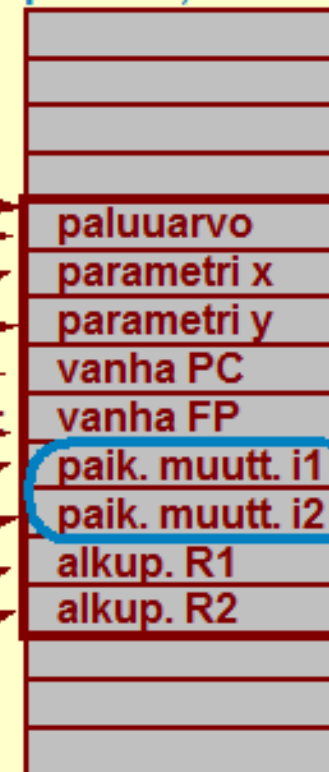
Aliohjelman toteutustietue

- jokaisella järjestelmällä omansa

Ttk-91 järjestelmän aktivointitietue (funktiole funcA)

- Funktion paluuarvo (paluuarvot?)
- Parametrien arvot (sisääntulo- ja ulostuloparametrit)
- Paluuosoite
- Kutsukohdan aktivointitietue
- Paikalliset muuttujat ja tietorakenteet
- Käytettyjen rekistereiden alkuperäiset arvot

pinossa, muistissa



Copyright Teemu Kerola 2004

Kaikki aliohjelman paikalliset muuttujat varataan myös aktivointitietueesta. Yleensä aktivointitietueen koko on rajoitettu, joten useat ohjelmointikielet estävät suurten tietorakenteiden määrittelyn paikallisina muuttujina. Suuret tietorakenteet onkin usein parempi allokoita keosta. Aktivointitietue taas talletetaan pinoon. Käsitlemme tätä tarkemmin ihan piakkoin.

Aktivointitietue (activation record, activation frame)

int funcA (int x, y)

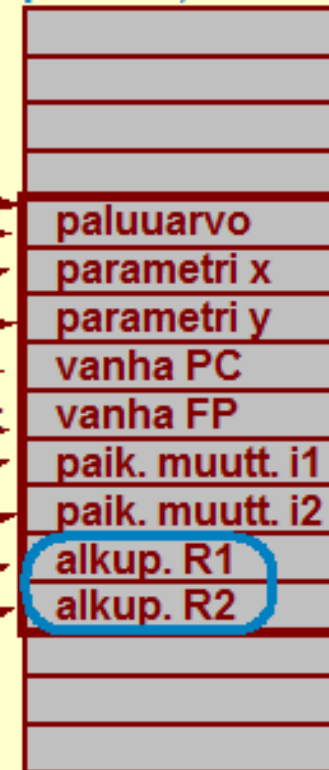
Aliohjelman toteutustietue

- jokaisella järjestelmällä omansa

Ttk-91 järjestelmän aktivointitietue (funktiole funcA)

- Funktion paluuarvo (paluuarvot?)
- Parametrien arvot (sisääntulo- ja ulostuloparametrit)
- Paluuosoite
- Kutsukohdan aktivointitietue
- Paikalliset muuttujat ja tietorakenteet
- Käytettyjen rekistereiden alkuperäiset arvot

pinossa, muistissa



Copyright Teemu Kerola 2004

Funktiossa funcA käytetään ainoastaan työrekestereitä R1 ja R2, joten ainoastaan niiden alkuperäiset arvot laitetaan talteen aktivointitietueeseen. Jos aliohjelman koodia myöhemmin muutetaan ja siellä otetaan käyttöön esimerkiksi rekisteri R5, niin R5 pitää myös muistaa lisätä tähän joukkoon. Nämä kuusi kenttää muodostavat nyt siis aktivointitietueen, mikä on aliohjelmien toteutuksen perusrakenne. Jokaista aliohjelman käyttökertaa varten sen käyttökerran aktivointitietue rakennetaan kokonaisuudessaan, ja aliohjelmasta poistuttaessa tämä aktivointitietue tuhoetaan.

Aktivointitietue (activation record, activation frame)

int funcA (int x, y)

Aliohjelman toteutustietue

- jokaisella järjestelmällä omansa

Ttk-91 järjestelmän aktivointitietue (funktiole funcA)

- Funktion paluuarvo (paluuarvot?)
- Parametrien arvot (sisääntulo- ja ulostuloparametrit)
- Paluuosoite
- Kutsukohdan aktivointitietue
- Paikalliset muuttujat ja tietorakenteet
- Käytettyjen rekistereiden alkuperäiset arvot

pinossa, muistissa



-4
-3
-2
+1
+2

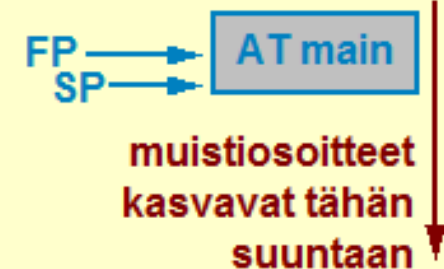
Copyright Teemu Kerola 2004

Aktivointitietueen osoite on normaalista monisanaisesta tiedosta poiketen sen keskellä olevan vanhan FP talletuskohdan osoite ja rekisteri FP osoittaa juuri tähän kohtaan aliohjelman suorituksen aikana. Me emme tiedä tarkalleen missä päin muistia aktivointitietue on, mutta kaikki sen tiedot ovat viitattavissa FP:n kautta. Parametrien osoitteet ovat FP-2, FP-3 jne. Paikallisten muuttujien osoitteet ovat FP+1, FP+2 jne. Parametreihin ja paikallisiin muuttujiin viittaminen on nyt vähän hankalampaa, koska niihin ei voi viitata minkään symbolin arvon avulla, vaan viittaminen tapahtuu aina niiden osoitteen perusteella, FP:n avustuksella.

Aktivointitietuepino

Aktivointitietueet (AT) varataan dynaamisesti suoritusaikana kutsuhetkellä pinosta ja niiden tila vapautetaan takaisin pinoon paluun yhteydessä

- R6 eli SP osoittaa aina pinon pinnalle



Aktivointitietuepino

- R7 eli FP osoittaa nyt suorituksessa olevan aliohjelman (tai pääohjelman) aktivointitietueen sovittuun kohtaan (ttk-91:ssä vanhan FP talletuspaikka)

Aktivointitietueen rakentaminen ja purku

- PUSH, POP
- PUSHR, POPR
- CALL, EXIT
- SVC, IRET

Copyright Teemu Kerola 2004

Aktivointitietueet allokoidaan pinosta. Kutsun yhteydessä muodostetaan aina uusi aktivointitietue ja aliohjelmasta paluun yhteydessä se puretaan. Jos ja kun aliohjelma kutsuu toista aliohjelmaa, niin tämän toisen aliohjelman aktivointitietue rakennetaan edellisen 'päälle' ja tällä tavoin pinossa on aina kutsuhierarkian mukaisesti kaikki kutsupolulla olevien aliohjelmakutsujen aktivointitietueet. Pinorekisteri SP pitää kirjaa pinon käytöstä ja osoittaa aina pinon pinnalle. Kehysrekisteri FP taas osoittaa tällä hetkellä käytössä olevaan aktivointitietueeseen ja määrittelee siten ohjelman suoritussympäristön.

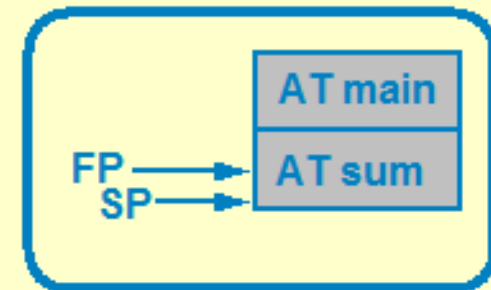
Aktivointitietuepino

Aktivointitietueet (AT) varataan dynaamisesti suoritusaikana kutsuhetkellä pinosta ja niiden tila vapautetaan takaisin pinoon paluun yhteydessä

- R6 eli SP osoittaa aina pinon pinnalle

Aktivointitietuepino

- R7 eli FP osoittaa nyt suorituksessa olevan aliohjelman (tai pääohjelman) aktivointitietueen sovittuun kohtaan (ttk-91:ssä vanhan FP talletuspaikka)



Aktivointitietueen rakentaminen ja purku

- PUSH, POP
- PUSHR, POPR
- CALL, EXIT
- SVC, IRET

Copyright Teemu Kerola 2004

Esimerkissä pääohjelma main kutsui aliohjelmaa sum, joten aliohjelman sum käyttökerran aktivointitietue on rakennettu pääohjelman aktivointitietueen päälle. SP osoittaa edelleenkin pinon pinnalle ja FP osoittaa nyt käytössä olevaan aktivointitietueeseen.

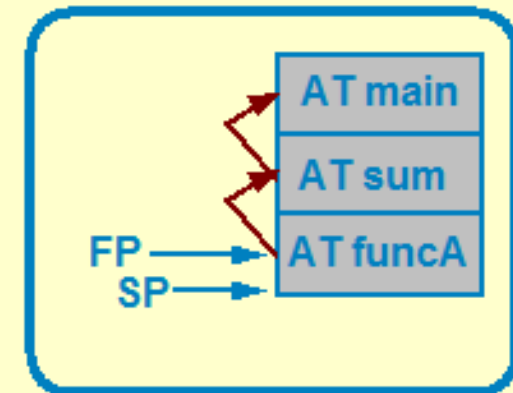
Aktivointitietuepino

Aktivointitietueet (AT) varataan dynaamisesti suoritusaikana kutsuhetkellä pinosta ja niiden tila vapautetaan takaisin pinoon paluun yhteydessä

- R6 eli SP osoittaa aina pinon pinnalle

Aktivointitietuepino

- R7 eli FP osoittaa nyt suorituksessa olevan aliohjelman (tai pääohjelman) aktivointitietueen sovittuun kohtaan (ttk-91:ssä vanhan FP talletuspaikka)



Aktivointitietueen rakentaminen ja purku

- PUSH, POP
- PUSHR, POPR
- CALL, EXIT
- SVC, IRET

Copyright Teemu Kerola 2004

Aliohjelma sum kutsui funktiota funcA, joten tätä kutsukertaa vastaava aktivointitietue rakennettiin aikaisempien päälle. AT-pinossa on nyt kolme alkiota. SP osoittaa jälleen pinon pinnalle ja FP nyt aktiivisena olevaan funcA'n kutsukerran aktivointitietueeseen. Funktion funcA'n aikana viitattavissa on ainakin kaikki funcA'n paikalliset muuttujat ja pääohjelmatasolla määritellyt globaalit muuttujat. Ohjelmointikielestä riippuen käytössä nyt voisi olla myös sum'in paikalliset muuttujat, joihin päästään käsiksi seuraamalla FP-ketjua yhden pykälän.

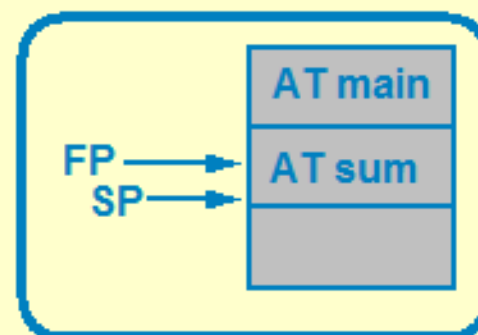
Aktivointitietuepino

Aktivointitietueet (AT) varataan dynaamisesti suoritusaikana kutsuhetkellä pinosta ja niiden tila vapautetaan takaisin pinoon paluun yhteydessä

- R6 eli SP osoittaa aina pinon pinnalle

Aktivointitietuepino

- R7 eli FP osoittaa nyt suorituksessa olevan aliohjelman (tai pääohjelman) aktivointitietueen sovittuun kohtaan (ttk-91:ssä vanhan FP talletuspaikka)



Aktivointitietueen rakentaminen ja purku

- PUSH, POP
- PUSHR, POPR
- CALL, EXIT
- SVC, IRET

Copyright Teemu Kerola 2004

Funktiosta funcA palattiin sitä kutsuneeseen rutiiniin ja funktion funcA aktivointitietue poistettiin pinosta. Pinossa on nyt vain kaksi aktivointitietuetta. SP osoittaa jälleen pinon pinnalle ja FP nyt taas aktiivisena olevan aliohjelman sum aktivointitietueeseen. Aktivointitietueita käsitellään siis kokonaisuuksina pinon pinnalla ja ainoastaan pinossa päällimmäisenä eli viimeistä aliohjelmakutsua vastaava aktivointitietue voidaan poistaa pinosta. Vaikka funcA'n aktivointitietue poistettiin pinosta, niin data ei tietenkään pinosta mihinkään hävinnyt. Ainoastaan SP'n arvo päivittyi.

Aktivointitietuepino

Aktivointitietueet (AT) varataan dynaamisesti suoritusaikana kutsuhetkellä pinosta ja niiden tila vapautetaan takaisin pinoon paluun yhteydessä

- R6 eli SP osoittaa aina pinon pinnalle

Aktivointitietuepino

- R7 eli FP osoittaa nyt suorituksessa olevan aliohjelman (tai pääohjelman) aktivointitietueen sovittuun kohtaan (ttk-91:ssä vanhan FP talletuspaikka)

Aktivointitietueen rakentaminen ja purku

- PUSH, POP
- PUSHF, POPF
- CALL, EXIT
- SVC, IRET

```
PUSH SP, X  
PUSH SP, R3  
PUSH SP, =44
```

SP:

33
11
44

```
POP SP, R1
```

SP:

33
11

Copyright Teemu Kerola 2004

Aktivointitietueita rakennetaan ja puretaan kokonainen tietue kerrallaan, mutta ttk-91:ssä ei ole mitään tietueiden rakentamis- ja purkamiskäskyjä. Sen sijaan tietueet rakennetaan ja puretaan paloittain tavallisilla konekäskyillä. PUSH-käskyllä kasvatetaan pinorekisteriä yhdellä ja viedään jälkimmäisen operandin arvo pinoon, joten SP edelleenkin osoittaa pinon pinnalle eli pinon päällimmäisenä olevaan alkioon. POP-käsky taas aina kopioi pinon pinnalla olevan alkion johonkin rekisteriin, ja samalla vähentää SP'n arvoa yhdellä, joten se edelleenkin osoittaa pinon pinnalle. Muistakaa, että pinohan on vain osa keskusmuistia.

Aktivointitietuepino

Aktivointitietueet (AT) varataan dynaamisesti suoritusaikana kutsuhetkellä pinosta ja niiden tila vapautetaan takaisin pinoon paluun yhteydessä

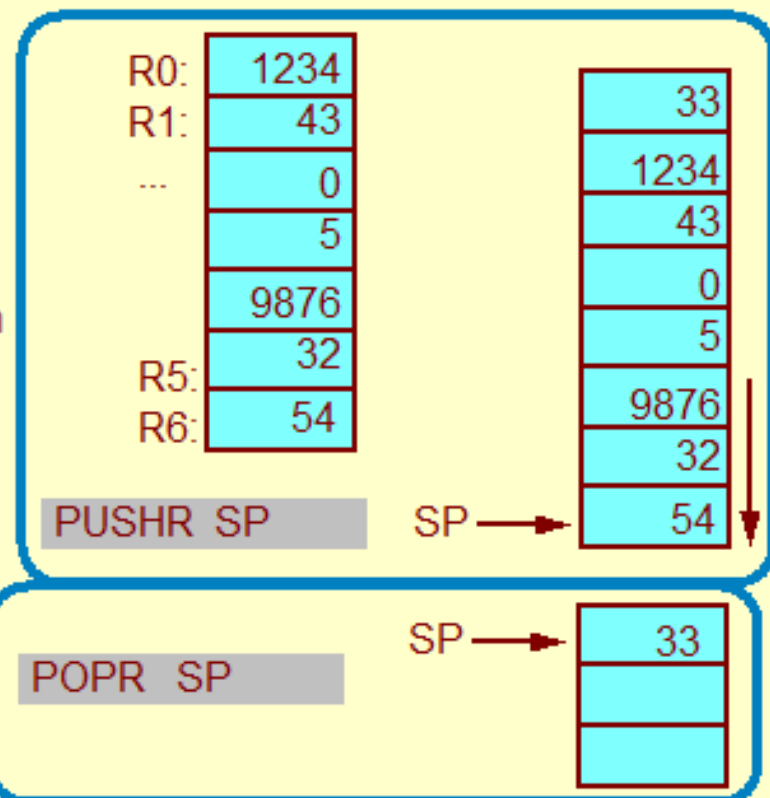
- R6 eli SP osoittaa aina pinon pinnalle

Aktivointitietuepino

- R7 eli FP osoittaa nyt suorituksessa olevan aliohjelman (tai pääohjelman) aktivointitietueen sovittuun kohtaan (ttk-91:ssä vanhan FP talletuspaikka)

Aktivointitietueen rakentaminen ja purku

- PUSH, POP
- **PUSHR, POPR**
- CALL, EXIT
- SVC, IRET



Copyright Teemu Kerola 2004

Ttk-91:ssä on myös vähän erikoiset rekistereiden talletus- ja palautuskäskyt PUSHR ja POPR. PUSHR'llä voi yhdellä käskyllä tallettaa kaikki työrekisterit pinoon. Se tallettaa myös rekisterin R6 arvon pinoon mikä on vähän sekavaa, mutta selittyy sillä että ttk-91:ssä SP'n ei ole pakko olla rekisteri R6, vaan se oikeastaan voi olla mikä tahansa rekisteri. Me käytämme kuitenkin vain R6'sta SP'nä. Vastaavasti POPR-käskyllä saadaan kaikkien työrekistereiden arvot palautettua yhdellä kertaa pinosta. Ihan tällaisia käskyjä ei oikeissa koneissa ole, mutta kylläkin muita erikoisia rakenteita aliohjelmakutsujen nopeuttamiseksi.

Aktivointitietuepino

Aktivointitietueet (AT) varataan dynaamisesti suoritusaikana kutsuhetkellä pinosta ja niiden tila vapautetaan takaisin pinoon paluun yhteydessä

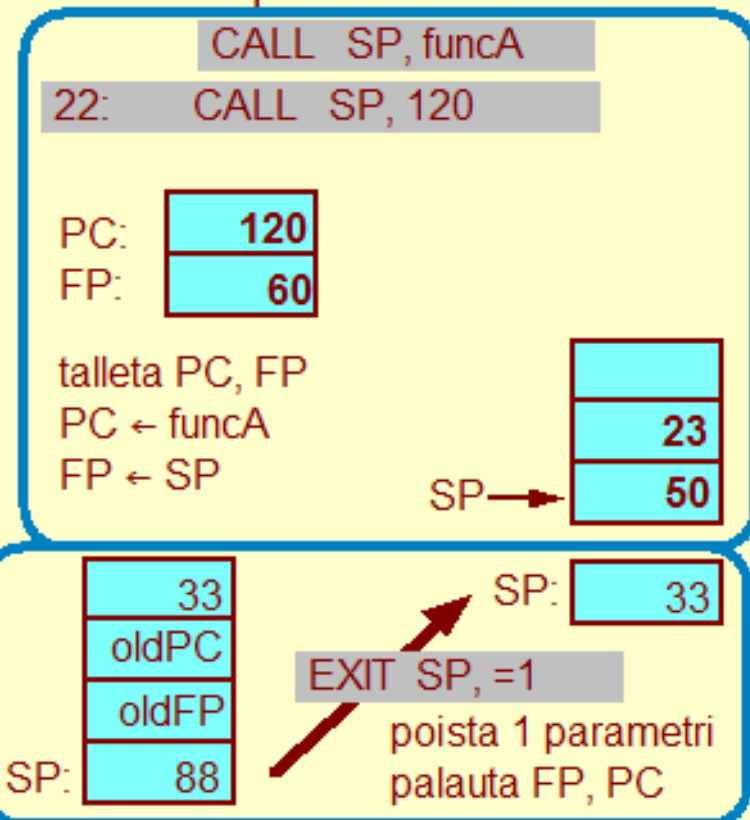
- R6 eli SP osoittaa aina pinon pinnalle

Aktivointitietuepino

- R7 eli FP osoittaa nyt suorituksessa olevan aliohjelman (tai pääohjelman) aktivointitietueen sovittuun kohtaan (ttk-91:ssä vanhan FP talletuspaikka)

Aktivointitietueen rakentaminen ja purku

- PUSH, POP
- PUSHR, POPR
- CALL, EXIT
- SVC, IRET



Copyright Teemu Kerola 2004

CALL käsky tekee monta asiaa. Ensinnäkin se tallettaa tätä CALL käskyä seuraavan käskyn osoitteen (aliohjelmasta paluusoitteen) ja nykyisen FP'n arvon pinoon. Sitten PC'n arvoksi asetetaan kutsutun aliohjelman osoite ja FP'n arvoksi nykyinen SP'n arvo. Tällä tavoin sekä kontrolli että viiteympäristö vaihtuu yhdellä kertaa kutsuttavaan aliohjelmään. Aliohjelmasta paluun tekevä EXIT-käsky poistaa käskyn vakio-osassa annetun määrän parametreja pinosta ja palauttaa pinosta alkuperäiset arvot FP'lle ja PC'lle. Kontrolli ja suoritussympäristö muuttuu yhdessä hujauksessa takaisin alkuperäiseen kutsun tehneeseen rutiiniin.

Aktivointitietuepino

Aktivointitietueet (AT) varataan dynaamisesti suoritusaikana kutsuhetkellä pinosta ja niiden tila vapautetaan takaisin pinoon paluun yhteydessä

- R6 eli SP osoittaa aina pinon pinnalle

Aktivointitietuepino

- R7 eli FP osoittaa nyt suorituksessa olevan aliohjelman (tai pääohjelman) aktivointitietueen sovittuun kohtaan (ttk-91:ssä vanhan FP talletuspaikka)

Aktivointitietueen rakentaminen ja purku

- PUSH, POP
- PUSHR, POPR
- CALL, EXIT
- SVC, IRET

```
SVC SP, =ReadFile
```

```
ReadFile ....
```

```
IRET SP, =3
```

Copyright Teemu Kerola 2004

Käyttöjärjestelmän palvelupyyntökäsky SVC on hyvin samankaltainen aliohjelman kutsukäskyn kanssa. Sen toimintaa esimerkkikoneessa ei ole oikein hyvin speksattu, mutta voimme kuvitella sen toimivan hyvin analogisesti CALL-käskyn kanssa. Merkittävä ero SVC:llä ja CALL-käskyllä on kuitenkin siinä, että jokainen SVC:llä kutsuttu rutiini aina tarkistaa ensin, oliko kutsu laillinen. Toinen ero on siinä, että KJ-rutiinit suorittavat koodia etuoikeutetussa tilassa, joten SVC-kutsun yhteydessä suoritustila vaihtuu etuoikeutetuksi. IRET-käsky toimii kuten EXIT, mutta palauttaa myös suoritustilan ennalleen.

Aktivointitietueen käsittely

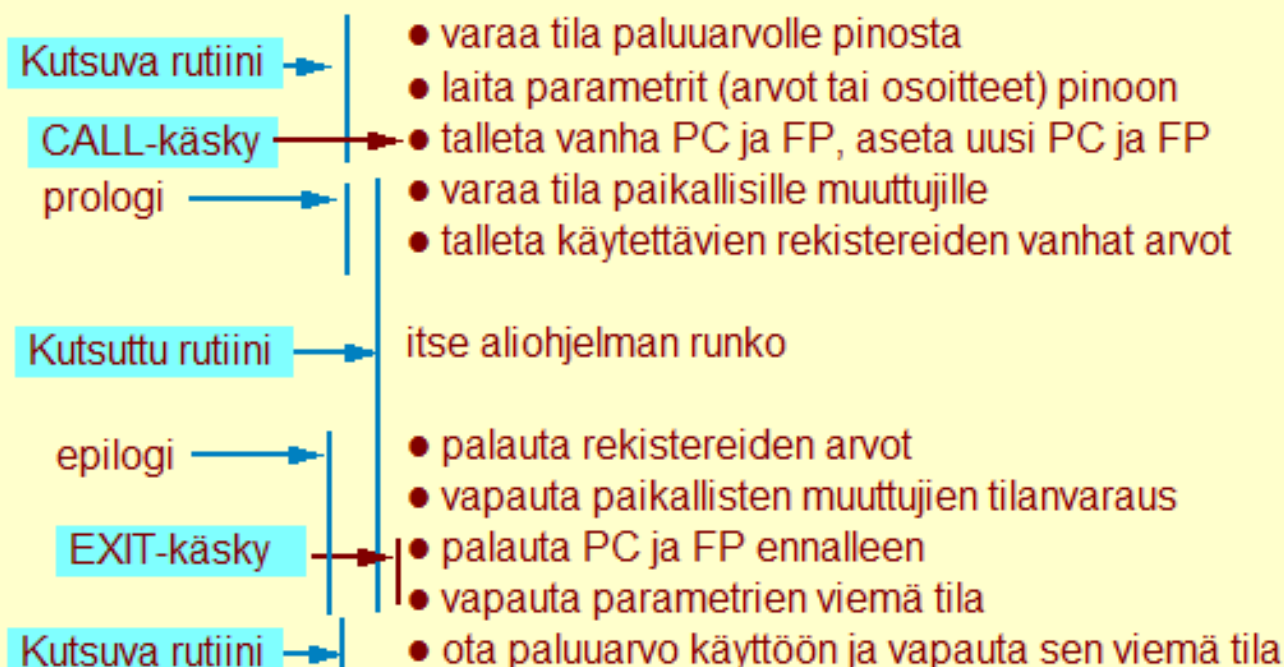
```
function Aliohj (p: int, q: int): int ;  
{ int i, j; ..... }
```

T = Aliohj (X, Y);

AT'n rakentaminen ja purkaminen jaettu eri yksiköille

- kutsuva rutiini (CALL) + kutsuttu rutiini (EXIT)

Toteutuksen osat



Copyright Teemu Kerola 2004

Aktivointitietue siis kootaan ja puretaan edellämainituilla konekäskyillä PUSH, PUSHR, CALL, POP, POPR ja EXIT. Työ on jaettu käytännöllisyyden vaatimusten mukaisesti kutsuvan ja kutsutun rutiinin välillä. Tottakai sekä CALL ja EXIT-käskyt on nimenomaan suunniteltu sitä varten, että aliohjelmien toteutus olisi helppoa niiden avulla. Kontrollin siirto kutsuvan ja kutsutun rutiinin välillä tapahtuu aina silloin, kun PC'n arvo vaihtuu kyseiseen rutiiniin. CALL- ja EXIT-käskyt ovat siis yhdenmuotoisia hyppykäskyjä, jotka myös vaihtavat ohjelman suoritussympäristön.

Aktivointitietueen käsittely

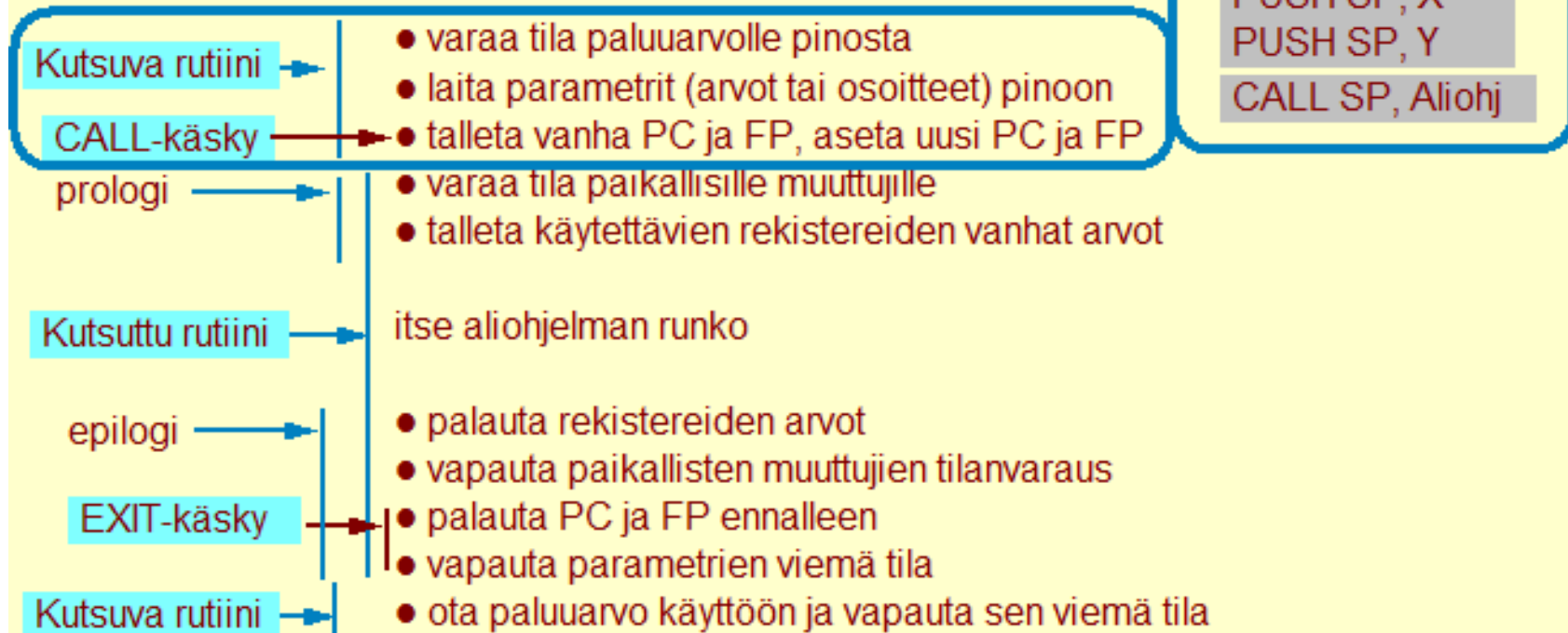
```
function Aliohj (p: int, q: int): int ;  
{ int i, j; ..... }
```

```
T = Aliohj (X, Y);
```

AT'n rakentaminen ja purkaminen jaettu eri yksiköille

- kutsuva rutiini (CALL) + kutsuttu rutiini (EXIT)

Toteutuksen osat



Copyright Teemu Kerola 2004

Oletetaan tässä, että haluamme tässä kohtaa koodia kutsua funktio Aliohj. Ohjelmoija (ja kääntäjä) tietää rutiinin Aliohj määrittelystä, että se on funktio ja että sillä on kaksi kokonaislukuarvoista parametria. Rupeamme nyt rakentamaan uutta aktivointitietuetta tätä kutsukertaa varten. Ensinnäkin varaamme pinosta tilaa funktion paluuarvolle ja sitten sijoitamme molemmat parametrit pinoon. Tämä riittää kutsuvalle rutiinille, joten se voi nyt siirtää kontrollin ja suoritussympäristön rutiinille Aliohj CALL-käskyllä.

Aktivointitietueen käsittely

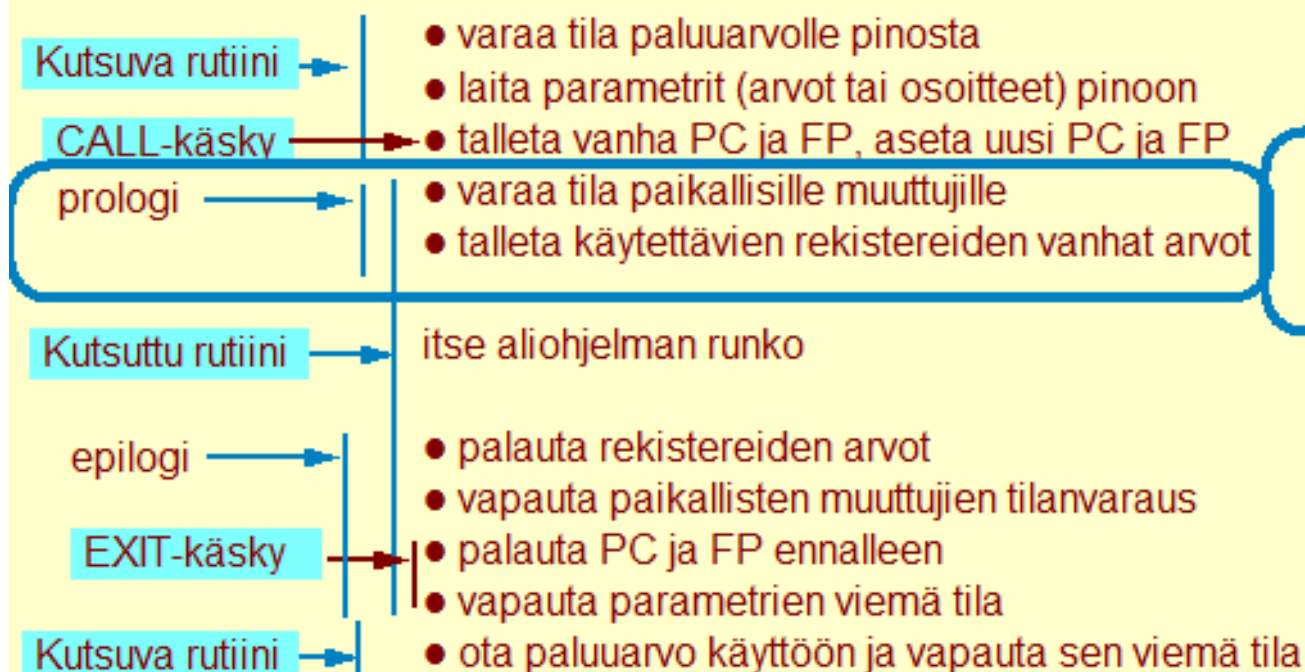
```
function Aliohj (p: int, q: int): int ;  
{ int i, j; ..... }
```

```
T = Aliohj (X, Y);
```

AT'n rakentaminen ja purkaminen jaettu eri yksiköille

- kutsuva rutiini (CALL) + kutsuttu rutiini (EXIT)

Toteutuksen osat



```
PUSH SP, =0
```

```
PUSH SP, X
```

```
PUSH SP, Y
```

```
CALL SP, Aliohj
```

```
PUSH SP, =0 ; iLoc
```

```
PUSH SP, =0 ; jLoc
```

```
PUSHR SP
```

Copyright Teemu Kerola 2004

Tässä vaiheessa kontrolli on siis siirtynyt kutsuttuun rutiiniin, ja seuraavaksi suoritettava koodi on siis aivan kutsutun aliohjelman alussa. Tämän kutsukerran aktivointitietue on jo osittain rakennettu, mutta se ei ole vielä valmis. Aliohjelman alussa oleva koodinpätkä, prologi viimeistelee aktivointitietueen. Prologissa ensin varataan pinosta tilaa paikallisille muuttujille, joko vain SP:tä kasvattamalla tai viemällä sinne nollia tarpeellinen määrä. Sitten pinoon talletetaan kaikkien käytettävien rekistereiden vanhat arvot, jotta ne voidaan aliohjelmasta paluun yhteydessä saattaa ennalleen.

Aktivointitietueen käsittely

```
function Aliohj (p: int, q: int): int ;  
{ int i, j; ..... }
```

```
T = Aliohj (X, Y);
```

AT'n rakentaminen ja purkaminen jaettu eri yksiköille

- kutsuva rutiini (CALL) + kutsuttu rutiini (EXIT)

Toteutuksen osat

Kutsuva rutiini

CALL-käskey

prologi

- varaa tila paluuarvolle pinosta
- laita parametrit (arvot tai osoitteet) pinoon
- talleta vanha PC ja FP, aseta uusi PC ja FP
- varaa tila paikallisille muuttujille
- talleta käytettävien rekistereiden vanhat arvot

```
PUSH SP, =0
```

```
PUSH SP, X
```

```
PUSH SP, Y
```

```
CALL SP, Aliohj
```

```
PUSH SP, =0 ; iLoc
```

```
PUSH SP, =0 ; jLoc
```

```
PUSHR SP
```

Kutsuttu rutiini

itse aliohjelman runko

```
... ; varsinainen työ
```

epilogi

EXIT-käskey

Kutsuva rutiini

- palauta rekistereiden arvot
- vapauta paikallisten muuttujien tilanvaraus
- palauta PC ja FP ennalleen
- vapauta parametrien viemä tila
- ota paluuarvo käyttöön ja vapauta sen viemä tila

Copyright Teemu Kerola 2004

Aktivointitietue on nyt vihdoon valmis ja itse aliohjelman runko alkaa tästä. Kaikki aliohjelmassa olevat viittaukset pinossa oleviin parametreihin ja paikallisiin muuttujiin tulee tehdä FP'n kautta kuten aikaisemmin käsittelimme ja mitä vielä piakkoin käytännön esimerkin avulla näytämme. Tässä vaiheessa meille riittää, että aliohjelman työ tehdään tässä ja seuraavaksi haluamme toteuttaa aliohjelmasta paluun ja tämän aktivointitietueen purkamisen. Oletamme, että aliohjelman palauttama arvo on jo talletettu omalle paikalleen aktivointitietueen ensimmäiseen sanaan.

Aktivointitietueen käsittely

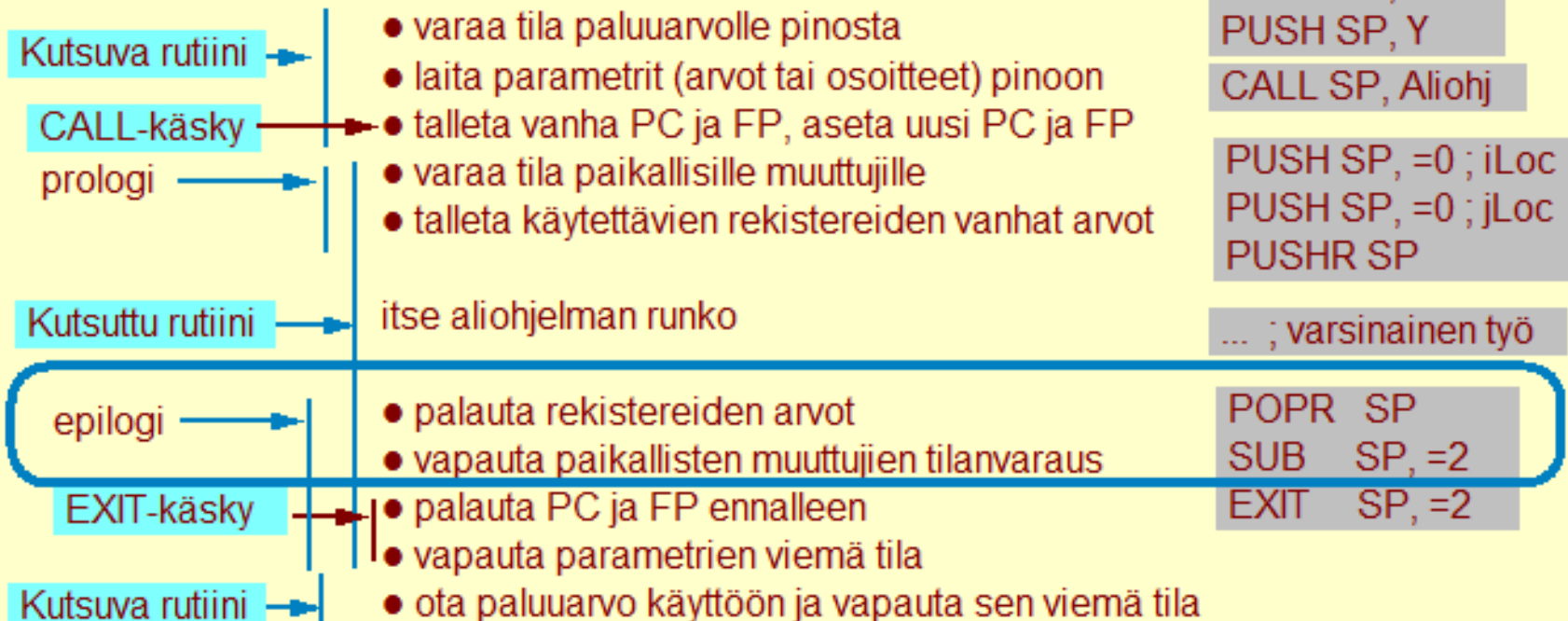
```
function Aliohj (p: int, q: int): int ;  
{ int i, j, ..... }
```

```
T = Aliohj (X, Y);
```

AT'n rakentaminen ja purkaminen jaettu eri yksiköille

- kutsuva rutiini (CALL) + kutsuttu rutiini (EXIT)

Toteutuksen osat



Copyright Teemu Kerola 2004

Aliohjelman päättymistoimenpiteitä kutsutaan epilogiksi. Aktivointitietue puretaan pinosta käänteisessä järjestyksessä kuin mitä se koottiin. Ensimmäisenä palautamme siis kaikkien työrekistereiden arvot paikalleen ja sitten vapautamme paikallisten muuttujien tilanvarauksen. Ohjelmoija (tai kääntäjä) tieteenkin tietää, kuinka monta paikallista muuttujaa aliohjelmalla on, ja se voi poistaa ne pinosta yksinkertaisesti vähentämällä SP'n arvoa sopivasti. POP-käskyä ei tässä voi käyttää, koska se tuhoaisi jonkun alkuperäisen rekisteriarvon, jonka me juuri saatoimme ennalleen.

Aktivointitietueen käsittely

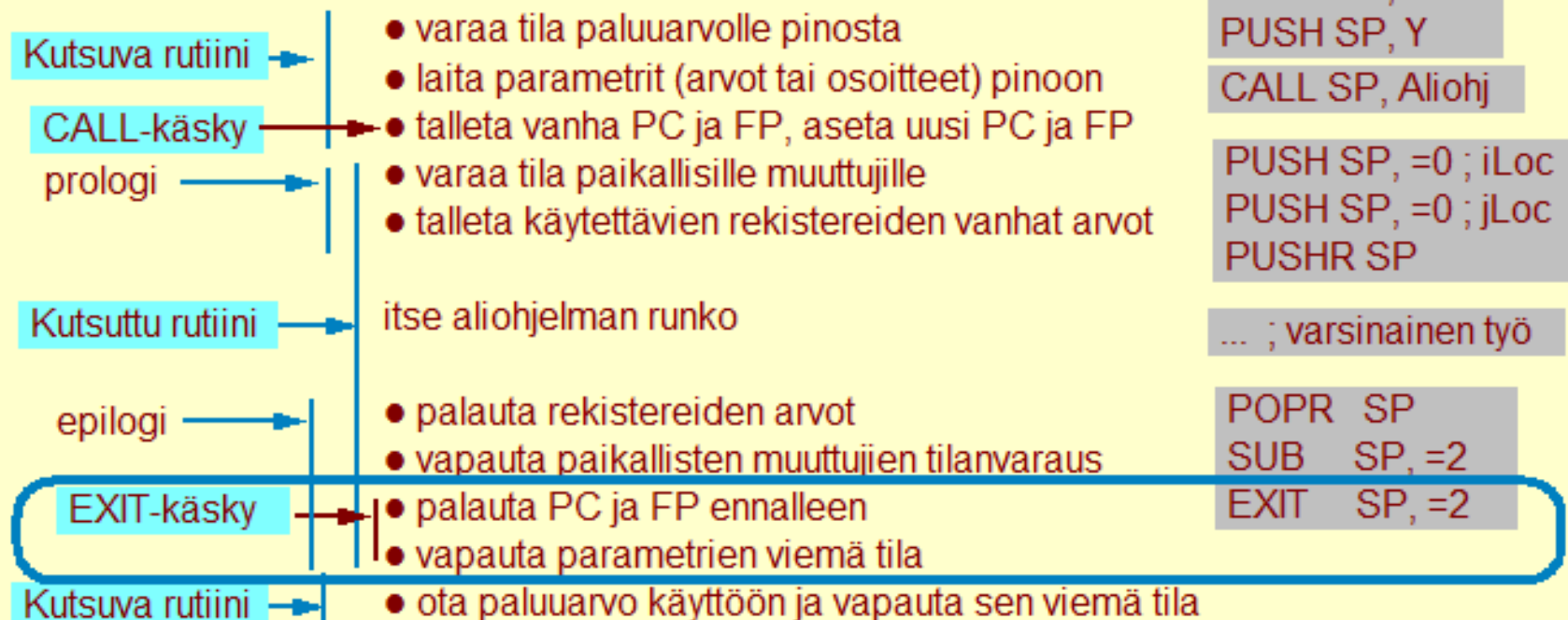
```
function Aliohj (p: int, q: int): int ;  
{ int i, j; ..... }
```

```
T = Aliohj (X, Y);
```

AT'n rakentaminen ja purkaminen jaettu eri yksiköille

- kutsuva rutiini (CALL) + kutsuttu rutiini (EXIT)

Toteutuksen osat



Copyright Teemu Kerola 2004

Epilogin lopussa kontrolli ja suoritusympäristö palautetaan EXIT-käskyllä kutsuvaan rutiiniin. Ttk-91:ssä EXIT-käskey myös vapauttaa aliohjelman parametrien viemän tilan pinosta. Sitä varten EXIT-käskyssä on toisena operandina aliohjelman parametrien lukumäärä.

Aktivointitietueen käsittely

```
function Aliohj (p: int, q: int): int ;  
{ int i, j; ..... }
```

```
T = Aliohj (X, Y);
```

AT'n rakentaminen ja purkaminen jaettu eri yksiköille

- kutsuva rutiini (CALL) + kutsuttu rutiini (EXIT)

Toteutuksen osat

Kutsuva rutiini

CALL-käsky

prologi

Kutsuttu rutiini

epilogi

EXIT-käsky

Kutsuva rutiini

- varaa tila paluuarvolle pinosta
- laita parametrit (arvot tai osoitteet) pinoon
- talleta vanha PC ja FP, aseta uusi PC ja FP
- varaa tila paikallisille muuttujille
- talleta käytettävien rekistereiden vanhat arvot

itse aliohjelman runko

- palauta rekistereiden arvot
- vapauta paikallisten muuttujien tilanvaraus
- palauta PC ja FP ennalleen
- vapauta parametrien viemä tila

- ota paluuarvo käyttöön ja vapauta sen viemä tila

```
PUSH SP, =0
```

```
PUSH SP, X
```

```
PUSH SP, Y
```

```
CALL SP, Aliohj
```

```
PUSH SP, =0 ; iLoc
```

```
PUSH SP, =0 ; jLoc
```

```
PUSHR SP
```

```
... ; varsinainen työ
```

```
POPR SP
```

```
SUB SP, =2
```

```
EXIT SP, =2
```

```
POP SP, R1
```

```
STORE R1, T
```

Copyright Teemu Kerola 2004

Kontrolli on nyt takaisin alkuperäisellä kutsuvalla rutiinilla ja se on suorittamassa CALL-käskyä seuraavaa konekäskyä. Jos kutsuttu rutiini oli funktio, niin kutsua varten luodusta aktivointitietueesta on enää funktion paluuarvo jäljellä pinossa ja se pitää popata sieltä pois ja ottaa käyttöön. Jos kyseessä oli pelkkä aliohjelman kutsu, niin tätä ei tietenkään tarvitse tehdä. Aliohjelmakutsun aktivointitietueen rakentaminen ja purkaminen vaatii siis aika paljon standardoitua yhteistyötä kutsuvalta ja kutsuttavalta rutiinilta, mutta kääntäjät ovat oikein hyviä tällaisten tylsien hallinnollisten tehtävien toteuttamisessa.

Esimerkki: aliohjelman kutsu

Funktio ja sen kutsu

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
....
t = fA(200, r);
```

Copyright Teemu Kerola 2004

Tässä esimerkissä meillä on yksinkertainen kokonaislukuarvoinen funktio fA, jolla on kaksi (kokonaislukuarvoista) arvoparametria x ja y. Funktiossa on yksi paikallinen muuttuja z, jolla on alkuarvo 5. Funktion rungossa z'lle lasketaan yksinkertaisen lausekkeen arvo ja funktio palauttaa arvonaan z'lle lasketun arvon. Myöhemmin ohjelmassa funktiota kutsutaan todellisilla parametreilla 200 ja r, joka on ohjelmassa määritelty globaali muuttuja. Funktion arvo talletetaan muuttujaan t.

Esimerkki: aliohjelman kutsu

Funktio ja sen kutsu

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
....
t = fA(200, r);
```

Kutsun toteutus

```
R    DC  24
T    DC   0
...
      PUSH  SP, =0 ; tila paluuarvolle
      PUSH  SP, =200 ; parametrit
      PUSH  SP, R

      CALL  SP, fA  ; kutsu ja paluu

      POP   SP, R1
      STORE R1, T
```

Copyright Teemu Kerola 2004

Tarkastelemme nyt ensin funktion kutsun toteuttamista. Funktion kutsu konekielellä toteutettuna ei ole sen monimutkaisempi kuin korkean tason kielelläkään. Sen kirjoittamiseen vain kuluu enemmän merkkejä. Lähtökohtaisesti tiedämme siis, että kyseinen funktio palauttaa kokonaisluvun ja että sillä on kaksi kokonaislukuarvoista arvoparametria, muodollisilta nimiltään x ja y. Kutsu tapahtuu yksinkertaisesti laittamalla ensin pinoon tila funktion paluuarvolle ja sitten molempien parametrien arvot, jonka jälkeen kutsumme itse funktiota. Funktiosta paluun jälkeen poppaamme paluuarvon pinosta ja sijoitamme sen t'hen.

Esimerkki: aliohjelman kutsu

Funktio ja sen kutsu

```
int fA (int x, y)
{
    int z = 5;

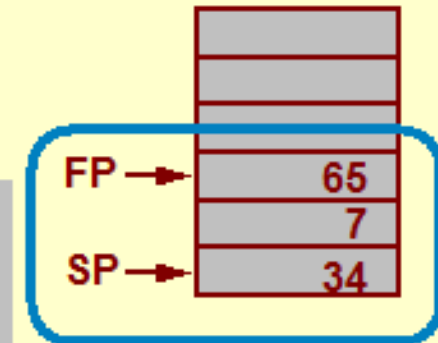
    z = x * z + y;
    return (z);
}
....
t = fA(200, r);
```

Kutsun toteutus

```
R    DC    24
T    DC    0
...
      PUSH  SP, =0 ; tila paluuarvolle
      PUSH  SP, =200 ; parametrit
      PUSH  SP, R

      CALL  SP, fA ; kutsu ja paluu

      POP   SP, R1
      STORE R1, T
```



Copyright Teemu Kerola 2004

Tarkastellaan kutsun toteutusta vielä aktivointitietueen tasolla. Oletetaan, että tällä hetkellä pinossa on jokin aktivointitietue, jonka vanhana FP-arvona on 65 ja sen jälkeen siinä on pinossa vielä yhden paikallisen muuttujan arvo 7 ja käytössä olevan jonkin rekisterin vanha arvo 34.

Esimerkki: aliohjelman kutsu

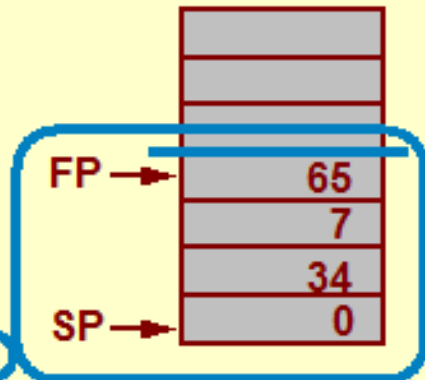
Funktio ja sen kutsu

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
....
t = fA(200, r);
```

Kutsun toteutus

```
R    DC    24
T    DC    0
...
    PUSH   SP, =0 ; tila paluuarvolle
    PUSH   SP, =200 ; parametrit
    PUSH   SP, R
    CALL   SP, fA ; kutsu ja paluu
    POP    SP, R1
    STORE  R1, T
```



Copyright Teemu Kerola 2004

Uuden aktivointitietueen rakentaminen aloitetaan varaamalla sen alusta yksi sana funktion paluuarvoa varten. Funktion suorituksen aikana sen arvo sitten talletetaan tähän kohtaan.

Esimerkki: aliohjelman kutsu

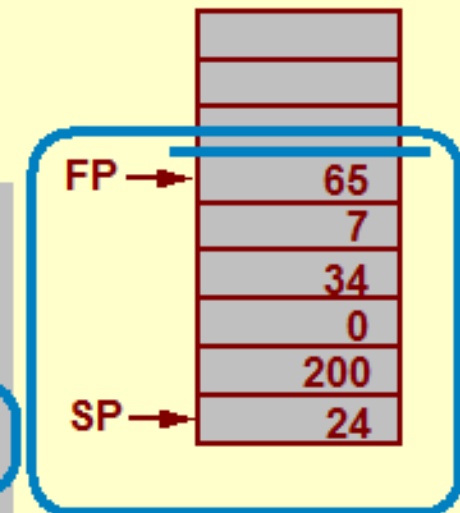
Funktio ja sen kutsu

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
....
t = fA(200, r);
```

Kutsun toteutus

```
R    DC    24
T    DC    0
...
    PUSH   SP, =0 ; tila paluuarvolle
    PUSH   SP, =200 ; parametrit
    PUSH   SP, R
    CALL   SP, fA ; kutsu ja paluu
    POP    SP, R1
    STORE  R1, T
```



Copyright Teemu Kerola 2004

Seuraavaksi aktivointitietueeseen pusketaan ensin vakioarvo 200 ensimmäisenä parametrina ja sitten muuttujan R arvo toisena parametrina. Huomaa, että PUSH-käskyn toiminta tässä jälkimmäisessä tapauksessa on aika monimutkainen, koska se kopioi muuttujan R arvon muistista pinon huipulle, mikä myöskin sijaitsee muistissa. Käsky siis suorittaa kaksi muistiviitettä käskyn suoritusaikana.

Esimerkki: aliohjelman kutsu

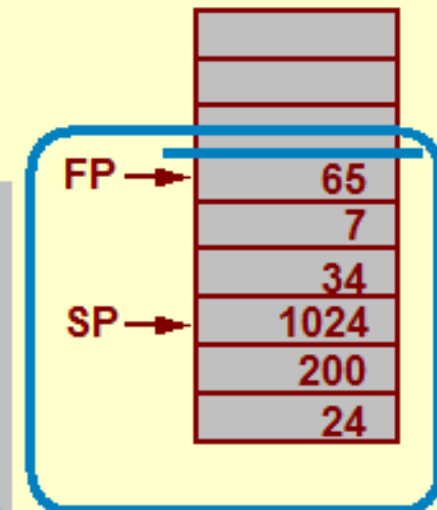
Funktio ja sen kutsu

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
....
t = fA(200, r);
```

Kutsun toteutus

```
R    DC    24
T    DC    0
...
    PUSH   SP, =0 ; tila paluuarvolle
    PUSH   SP, =200 ; parametrit
    PUSH   SP, R
    CALL   SP, fA ; kutsu ja paluu
    POP    SP, R1
    STORE  R1, T
```



Copyright Teemu Kerola 2004

Varsinaisen aliohjelman suorituksen aikana fA ensin täydentää aktivointitietueen loppuun ja sitten tekee varsinaisen työnsä, tallettaen funktion arvon sille varattuun paikkaan aktivointitietueessa. Lopuksi fA purkaa lähes koko aktivointitietueen - ainoastaan paluuarvo jää vielä paikalleen pinon pinnalle.

Esimerkki: aliohjelman kutsu

Funktio ja sen kutsu

```
int fA (int x, y)
{
    int z = 5;

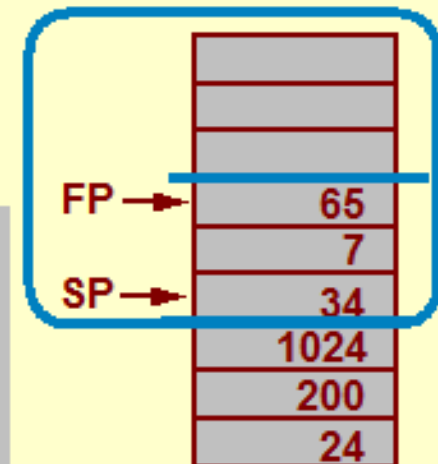
    z = x * z + y;
    return (z);
}
....
t = fA(200, r);
```

Kutsun toteutus

```
R    DC    24
T    DC    0
...
    PUSH    SP, =0 ; tila paluuarvolle
    PUSH    SP, =200 ; parametrit
    PUSH    SP, R

    CALL    SP, fA ; kutsu ja paluu

    POP     SP, R1
    STORE   R1, T
```



Copyright Teemu Kerola 2004

Lopuksi kutsuva rutiini poppaa paluuarvon pois pinosta ja nyt koko aktivointitietue on poistettu ja AT-pino on täsmälleen samanlainen kuin mitä se oli ennen kuin tätä kutsua toteuttavaa aktivointitietuetta ruvettiin rakentamaan. Sillä, että pinossa SP'n 'yläpuolella' olevat arvot ovat muuttuneet, ei ole toiminnan kanssa mitään merkitystä, koska nämä arvot eivät kuulu enää pinoon.

Esimerkki: aliohjelman toteutus

Funktio ja sen kutsu

```
int fA (int x, y)
```

```
{
```

```
    int z = 5;
```

```
    z = x * z + y;
```

```
    return (z);
```

```
}
```

```
....
```

```
t = fA(200, r);
```

Kutsun toteutus

```
40: PUSH    SP, =0
```

```
41: PUSH    SP, =200
```

```
42: PUSH    SP, R
```

```
43: CALL    SP, fA
```

```
44: POP     SP, R1
```

```
45: STORE   R1, T
```

Aliohjelman/funktion toteutus

```
retfA EQU   -4 ; paluuarvo osoite
```

```
parX EQU    -3 ; parametrit
```

```
parY EQU    -2
```

```
locZ EQU    1 ; paikallinen muuttuja
```

```
fA      PUSH    SP, =5 ; paik. muutt. Z
```

```
        PUSH    SP, R1 ; talleta rekisterit
```

```
        LOAD    R1, parX (FP)
```

```
        MUL     R1, locZ (FP)
```

```
        ADD     R1, parY (FP)
```

```
        STORE   R1, locZ (FP)
```

```
        STORE   R1, retfA (FP)
```

```
        POP     SP, R1 ; palauta R1
```

```
        SUB     SP, =1 ; vapauta Z
```

```
        EXIT    SP, =2 ; vapauta param
```

77:

65

7

34

0

200

24

44

77

FP →

SP →

Esimerkki: aliohjelman toteutus

Funktio ja sen kutsu

```
int fA (int x, y)
```

```
{
```

```
    int z = 5;
```

```
    z = x * z + y;
```

```
    return (z);
```

```
}
```

```
....
```

```
t = fA(200, r);
```

Kutsun toteutus

```
40: PUSH  SP, =0
```

```
41: PUSH  SP, =200
```

```
42: PUSH  SP, R
```

```
43: CALL  SP, fA
```

```
44: POP   SP, R1
```

```
45: STORE R1, T
```

Aliohjelman/funktion toteutus

```
retfA EQU  -4 ; paluuarvo osoite
```

```
parX EQU  -3 ; parametrit
```

```
parY EQU  -2
```

```
locZ EQU  1 ; paikallinen muuttuja
```

```
fA  PUSH  SP, =5 ; paik. muutt. Z  
    PUSH  SP, R1 ; talleta rekisterit
```

```
    LOAD  R1, parX (FP)
```

```
    MUL   R1, locZ (FP)
```

```
    ADD   R1, parY (FP)
```

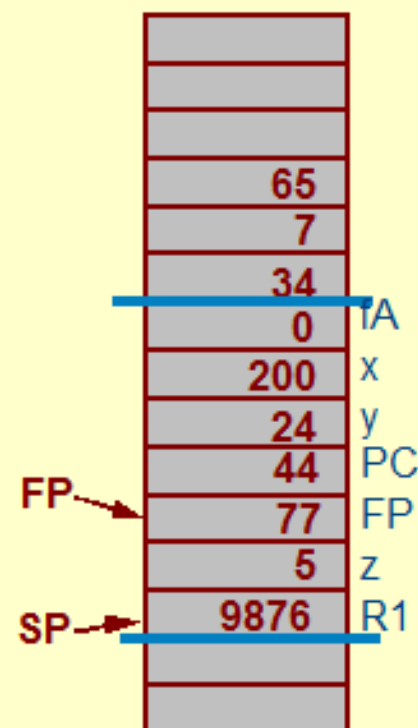
```
    STORE R1, locZ (FP)
```

```
    STORE R1, retfA (FP)
```

```
    POP   SP, R1 ; palauta R1
```

```
    SUB   SP, =1 ; vapauta Z
```

```
    EXIT  SP, =2 ; vapauta param
```



Copyright Teemu Kerola 2004

Funktion fA alussa varataan tilaa paikalliselle muuttujalle Z ja samalla alustetaan se arvoon 5. Seuraavaksi talletetaan rekisterin R1 alkuperäinen arvo (9876) pinoon. Nyt tämän kutsukerran aktivointitietue on valmis ja itse työ voi alkaa. Tässä tapauksessa osa työstäkin on jo vähän tehty, koska paikallinen muuttuja Z on jo alustettu.

Esimerkki: aliohjelman toteutus

Funktio ja sen kutsu

```
int fA (int x, y)
```

```
{
```

```
    int z = 5;
```

```
    z = x * z + y;
```

```
    return (z);
```

```
}
```

```
....
```

```
t = fA(200, r);
```

Kutsun toteutus

```
40: PUSH    SP, =0
```

```
41: PUSH    SP, =200
```

```
42: PUSH    SP, R
```

```
43: CALL    SP, fA
```

```
44: POP     SP, R1
```

```
45: STORE   R1, T
```

Aliohjelman/funktion toteutus

```
retfA EQU   -4 ; paluuarvo osoite
```

```
parX EQU    -3 ; parametrit
```

```
parY EQU    -2
```

```
locZ EQU    1 ; paikallinen muuttuja
```

```
fA    PUSH   SP, =5 ; paik. muutt. Z
```

```
      PUSH   SP, R1 ; talleta rekisterit
```

```
      LOAD   R1, parX (FP)
```

```
      MUL    R1, locZ (FP)
```

```
      ADD    R1, parY (FP)
```

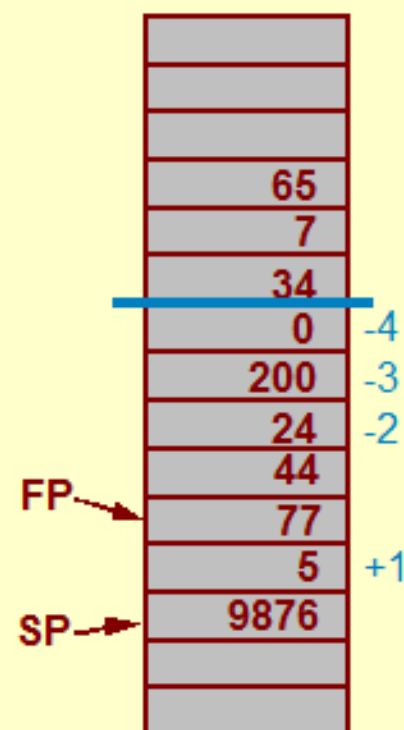
```
      STORE  R1, locZ (FP)
```

```
      STORE  R1, retfA (FP)
```

```
      POP    SP, R1 ; palauta R1
```

```
      SUB    SP, =1 ; vapauta Z
```

```
      EXIT   SP, =2 ; vapauta param
```



Copyright Teemu Kerola 2004

Parametrien ja paikallisten muuttujien tarkka sijainti muistissa ratkeaa vasta suoritusaikana, ja osoitteet voivat olla vielä erilaisia eri kutsukerroilla. Sijainti on kuitenkin aktivointitietueen sisällä aina vakio ja sitä varten määrittelemme funktion sisällä paluuarvolle, parametreille ja paikalliselle muuttujalle EQU-määreellä symbolit, jotka vastaavat kyseisten alkioden suhteellista sijaintia aktivointitietueessa. Aktivointitietueen osoitehan on FP:ssä. Esimerkiksi funktion paluuarvo retfA on nyt osoitteessa FP-4 ja paikallinen muuttuja locZ on osoitteessa FP+1.

Esimerkki: aliohjelman toteutus

Funktio ja sen kutsu

```
int fA (int x, y)
```

```
{
```

```
    int z = 5;
```

```
    z = x * z + y;
```

```
    return (z);
```

```
}
```

```
....
```

```
t = fA(200, r);
```

Kutsun toteutus

```
40: PUSH    SP, =0
```

```
41: PUSH    SP, =200
```

```
42: PUSH    SP, R
```

```
43: CALL    SP, fA
```

```
44: POP     SP, R1
```

```
45: STORE   R1, T
```

Aliohjelman/funktion toteutus

```
retfA EQU   -4 ; paluuarvo osoite
```

```
parX EQU    -3 ; parametrit
```

```
parY EQU    -2
```

```
locZ EQU    1 ; paikallinen muuttuja
```

```
fA      PUSH    SP, =5 ; paik. muutt. Z
```

```
        PUSH    SP, R1 ; talleta rekisterit
```

```
        LOAD    R1, parX (FP)
```

```
        MUL     R1, locZ (FP)
```

```
        ADD     R1, parY (FP)
```

```
        STORE   R1, locZ (FP)
```

```
        STORE   R1, retfA (FP)
```

```
        POP     SP, R1 ; palauta R1
```

```
        SUB     SP, =1 ; vapauta Z
```

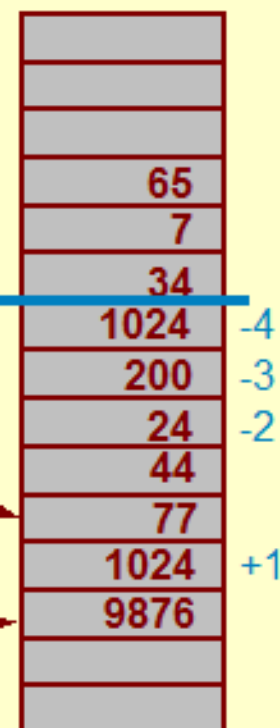
```
        EXIT    SP, =2 ; vapauta param
```

prologi

FP →

SP →

epilogi



Copyright Teemu Kerola 2004

Funktion varsinaisessa koodissa kaikki viittaukset paluuarvoon, parametreihin tai paikallisiin muuttujiin tehdään nyt käyttäen näiden tietojen suhteellisia osoitteita, indeksoiden suhteellista osoitetta FP:llä. Muuttujan Z uusi arvo lasketaan ja talletetaan muistiin ja sama arvo talletetaan myös funktion paluuarvoksi. Tämän jälkeen olemmekin valmiita tekemään funktiosta paluun.

Esimerkki: aliohjelman toteutus

Funktio ja sen kutsu

```
int fA (int x, y)
```

```
{
```

```
    int z = 5;
```

```
    z = x * z + y;
```

```
    return (z);
```

```
}
```

```
....
```

```
t = fA(200, r);
```

Kutsun toteutus

```
40: PUSH  SP, =0
```

```
41: PUSH  SP, =200
```

```
42: PUSH  SP, R
```

```
43: CALL  SP, fA
```

```
44: POP   SP, R1
```

```
45: STORE R1, T
```

Aliohjelman/funktion toteutus

```
retfA EQU  -4 ; paluuarvo osoite
```

```
parX EQU  -3 ; parametrit
```

```
parY EQU  -2
```

```
locZ EQU  1 ; paikallinen muuttuja
```

```
fA  PUSH  SP, =5 ; paik. muutt. Z
```

```
    PUSH  SP, R1 ; talleta rekisterit
```

```
    LOAD  R1, parX (FP)
```

```
    MUL   R1, locZ (FP)
```

```
    ADD   R1, parY (FP)
```

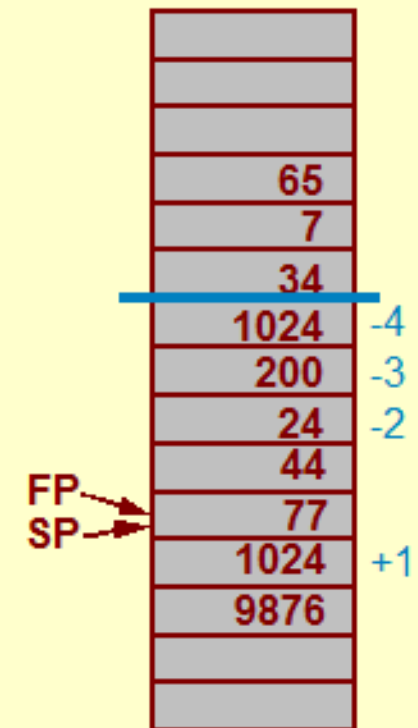
```
    STORE R1, locZ (FP)
```

```
    STORE R1, retfA (FP)
```

```
    POP   SP, R1 ; palauta R1
```

```
    SUB   SP, =1 ; vapauta Z
```

```
    EXIT  SP, =2 ; vapauta param
```



R1: 9876

Copyright Teemu Kerola 2004

Aktivointitietueen purkaminen alkaa palauttamalla R1'n arvo ennalleen. Tämän jälkeen minkään rekisterin arvoa ei saa muuttaa ennen EXIT-käskyä. Seuraavaksi vapautetaan paikallisen muuttujan Z tila vähentämällä 1 SP'n arvosta.

Esimerkki: aliohjelman toteutus

Funktio ja sen kutsu

```
int fA (int x, y)
{
    int z = 5;

    z = x * z + y;
    return (z);
}
....
t = fA(200, r);
```

Kutsun toteutus

```
40: PUSH  SP, =0
41: PUSH  SP, =200

42: PUSH  SP, R
43: CALL  SP, fA
44: POP   SP, R1
45: STORE R1, T
```

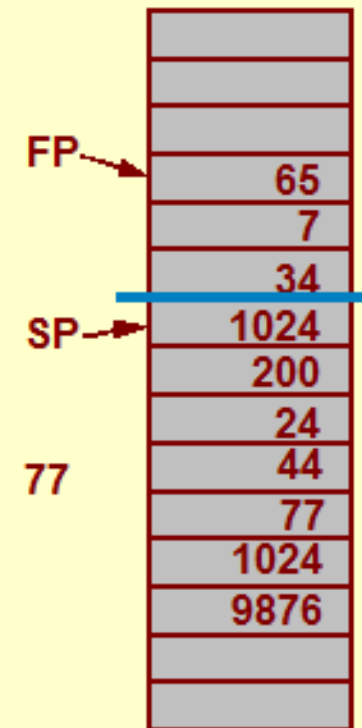
Aliohjelman/funktion toteutus

```
retfA EQU  -4 ; paluuarvo osoite
parX EQU  -3 ; parametrit
parY EQU  -2
locZ EQU   1 ; paikallinen muuttuja

fA  PUSH  SP, =5 ; paik. muutt. Z
    PUSH  SP, R1 ; talleta rekisterit

    LOAD  R1, parX (FP)
    MUL   R1, locZ (FP)
    ADD   R1, parY (FP)
    STORE R1, locZ (FP)
    STORE R1, retfA (FP)

    POP   SP, R1 ; palauta R1
    SUB   SP, =1 ; vapauta Z
    EXIT  SP, =2 ; vapauta param
```



PC: 44

Copyright Teemu Kerola 2004

EXIT-käskey tekee monta asiaa samalla kertaa. Se palauttaa pinosta FP'n arvon ennalleen, ja poppaa PC'n arvoksi pinon toiseksi päällimmäisen alkion. Sitten se vielä poistaa pinosta EXIT-käskyn osoiteosassa mainitut kaksi parametria. Pinoon jää nyt ainoastaan funktion arvo, kun kontrolli seuraavalla käskyllä palaa takaisin kutsuvaan rutiiniin.

Viiteparametri

Pascal aliohjelma

```
procB (x, y: int, var pZ: int)  
{  
  pZ = x * 5 + y;  
  return;  
}  
....  
procB( 200, r, t);
```

pZ'n osoittaman
muistipaikan arvo
muuttuu,
pZ'n arvo ei muutu

C aliohjelma

```
procB (int x, y, int * pZ)  
{  
  *pZ = x * 5 + y;  
  return;  
}  
....  
procB( 200, r, &t);
```

Välitetään t:n osoite,
ei t:n arvoa

Copyright Teemu Kerola 2004

Viiteparametreja käytetään eri ohjelmointikielissä eri tavoin. Pascalissa ne ovat ihan omana parametrityyppinä, jolloin riittää aliohjelman määrittelyn yhteydessä esitellä parametri viiteparametrina. Tällöin aliohjelman rungossa kaikki viitteet tehdään automaattisesti parametrin osoitteen perusteella ja aliohjelmaa kutsuttaessa myös parametrin osoite välitetään automaattisesti sen arvon asemesta. C-kielessä ei ole viiteparametreja, mutta sama asia voidaan kuvata osoitearvoisten arvoparametrien avulla. Tällöin kuitenkin joka viittauskohdassa täytyy muistaa eksplisiittisesti ilmaista, että kyseinen (arvo)parametri sisältää muuttujan osoitteen eikä sen arvoa.

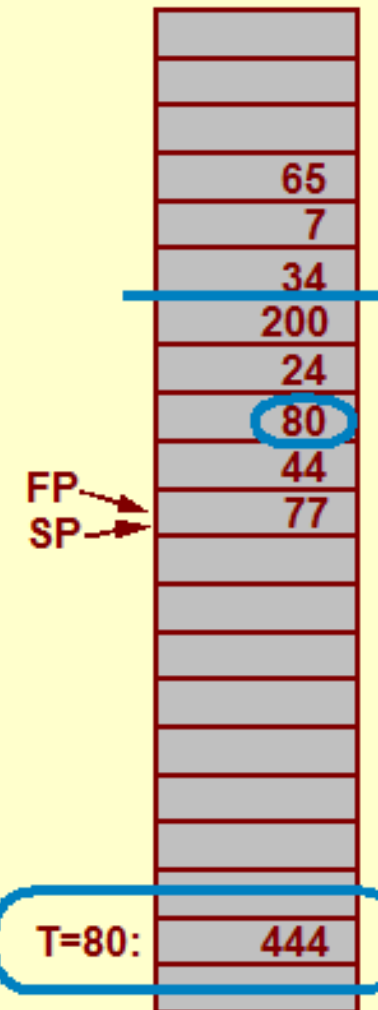
Viiteparametri

Pascal aliohjelma

```
procB (x, y: int, var pZ:int)
{
    pZ = x * 5 + y;
    return;
}
....
procB( 200, r, t);
```

Kutsun toteutus

```
40: PUSH    SP, =200
41: PUSH    SP, R
42: PUSH    SP, =T ; tn osoite
43: CALL    SP, procB
44: ...     ; tllä on nyt uusi arvo
```



Copyright Teemu Kerola 2004

Aliohjelman kutsu tapahtuu nyt muuten ihan samalla tavalla kuin aikaisemminkin, mutta kolmannen parametrin T osalta välitetään Tn osoite 80 eikä sen tämänhetkistä arvoa.

Viiteparametri

Pascal aliohjelma

```
procB (x, y: int, var pZ:int)
{
    pZ = x * 5 + y;
    return;
}
....
procB( 200, r, t);
```

Kutsu

```
40: PUSH    SP, =200
41: PUSH    SP, R
42: PUSH    SP, =T ; t:n osoite
43: CALL    SP, procB
44: ...      ; t:llä on nyt uusi arvo
```

Aliohjelman/funktion toteutus

```
parX EQU -4 ; parametril  
parY EQU -3  
parpZ EQU -2 ; viiteparametri
```

```
procB PUSH SP, R1 ; talleta rekisteri
```

LOAD R1, parX (FP)

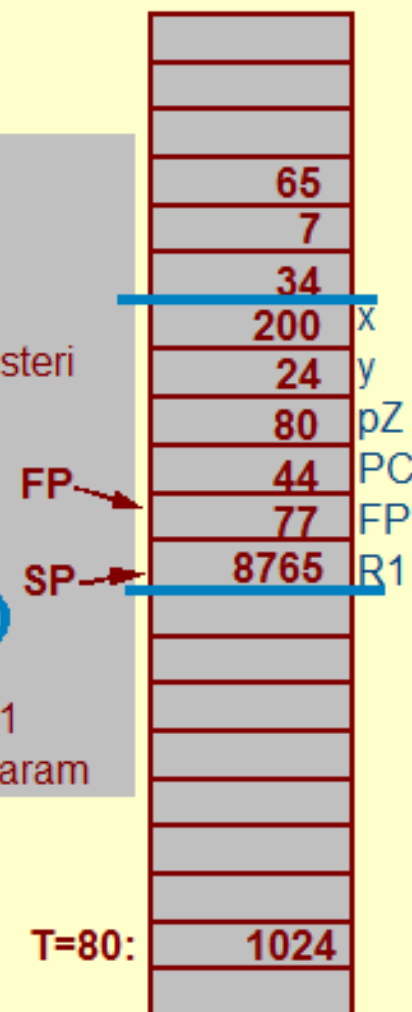
MUL R1, #5

ADD R1, parY (FP)

STORE R1, @parpZ (FP)

POP SP, R1 ; palauta R1

```
EXIT    SP, =3 ; vapauta param
```



Copyright Teemu Kerola 2004

Aliohjelman määrittelyssä on nyt hyvä jollain tavoin parametrin nimessä ilmaista, että kyseessä on viiteparametri. Tässä muodollisen parametrin `parpZ` nimessä on ylimääräinen `p` (niin kuin `pointer`) asiasta muistuttamassa. Vastaavasti kun viiteparametriin viitataan, niin haluamme siis viitata parametrin osoittamaan tietoon, eikä itse parametriin. Tätä tarkoitusta varten `ttk-91` koneessa on epäsuora tiedonosoitusmoodi. Muistisääntönä voi pitää, että '@-merkki kompensoi ylimääräistä p-kirjainta'. Aliohjelma siis tallettaa nyt uuden arvon 1024 pääohjelman muuttujaan `T`.

Viiteparametri

Pascal aliohjelma

```
procB (x, y: int, var pZ:int)
{
    pZ = x * 5 + y;
    return;
}
....
procB( 200, r, t);
```

Kutsun toteutus

```
40: PUSH    SP, =200
41: PUSH    SP, R
42: PUSH    SP, =T ; t:n osoite
43: CALL    SP, procB
44: ...      ; t:llä on nyt uusi arvo
```

Aliohjelman/funktion toteutus

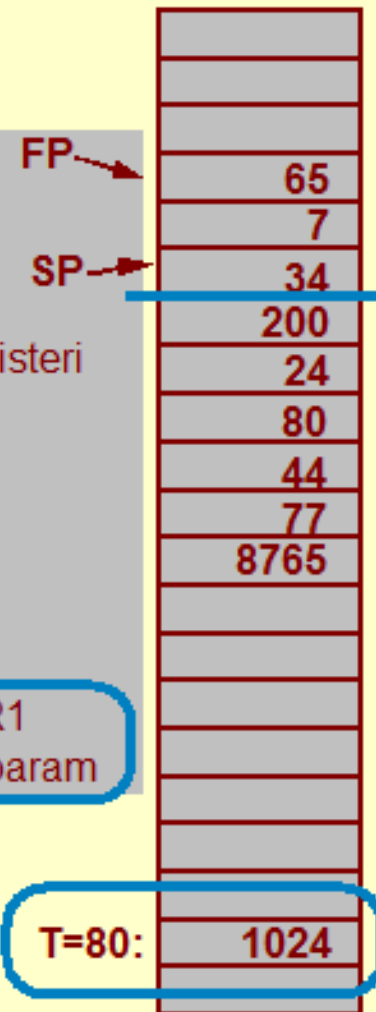
```

parX EQU -4 ; parametrinopeus FP
parY EQU -3
parpZ EQU -2 ; viiteparametri SP

procB PUSH SP, R1 ; talleta rekisterin
      LOAD R1, parX (FP)
      MUL R1, =5
      ADD R1, parY (FP)
      STORE R1, @parpZ (FP)

```

```
POP    SP, R1 ; palauta R1
EXIT   SP, =3 ; vapauta param
```



Aliohjelma kutsuu toista aliohjelmaa

Korkean tason kieli

```
procC (x, y: int, var pZ:int)
{
    pZ = fA( x, y);
    return;
}
....
procC( 200, r, t);
```

Kutsu

```
40: PUSH    SP, =200
41: PUSH    SP, R
42: PUSH    SP, =T
43: CALL     SP, procC
44: ...
```

procC

```
parXc EQU  -4 ; parametrit
parYc EQU  -3
parpZ EQU  -2

procC PUSH  SP, R1 ; tall rek

; @parpZ = fA (parXc, parYc)
      PUSH  SP, =0 ; paluuarv
      PUSH  SP, parXC (FP)
      PUSH  SP, parYc(FP)
      CALL  SP, fA
      POP   SP, R1
      STORE R1, @parpZ (FP)

      POP   SP, R1 ; pal R1
      EXIT  SP, =3 ; vap par
```

fA

```
retfA EQU  -4 ; paluuarvo
parX EQU  -3 ; parametrit
parY EQU  -2
locZ EQU  1 ; paik muut

fA      PUSH  SP, =5 ; paik. Z
      PUSH  SP, R1 ; tall rek

      LOAD   R1, parX (FP)
      MUL    R1, locZ (FP)
      ADD    R1, parY (FP)
      STORE  R1, locZ (FP)
      STORE  R1, retfA (FP)

      POP    SP, R1 ; pal R1
      SUB    SP, =1 ; vap Z
      EXIT   SP, =2 ; vap par
```

Copyright Teemu Kerola 2004

Aliohjelma voi tietenkin kutsua toista aliohjelmaa. Tämähän on normaali tapa toteuttaa mitä tahansa ohjelmaa. Tässä esimerkissä aliohjelma procC on muuten samanlainen kuin edellä esitetty procB, mutta laskenta annetaankin nyt jo valmiiksi muualla määritellylle funktiolle fA. Aliohjelman procC kolmas parametri on edelleen viiteparametri.

Aliohjelma kutsuu toista aliohjelmaa

Korkean tason kieli

```
procC (x, y: int, var pZ:int)
{
    pZ = fA( x, y);
    return;
}
```

```
....
procC( 200, r, t);
```

Kutsu

```
40: PUSH    SP, =200
41: PUSH    SP, R
42: PUSH    SP, =T
43: CALL     SP, procC
44: ...
```

procC

```
parXc EQU  -4 ; parametrit
parYc EQU  -3
parpZ EQU  -2

procC PUSH  SP, R1 ; tall rek

; @parpZ = fA (parXc, parYc)
      PUSH  SP, =0 ; paluuarv
      PUSH  SP, parXc (FP)
      PUSH  SP, parYc(FP)
      CALL  SP, fA
      POP   SP, R1
      STORE R1, @parpZ (FP)

      POP   SP, R1 ; pal R1
      EXIT  SP, =3 ; vap par
```

fA

```
retfA EQU  -4 ; paluuarvo
parX EQU  -3 ; parametrit
parY EQU  -2
locZ EQU  1 ; paik muut

fA   PUSH  SP, =5 ; paik. Z
     PUSH  SP, R1 ; tall rek

     LOAD   R1, parX (FP)
     MUL    R1, locZ (FP)
     ADD    R1, parY (FP)
     STORE  R1, locZ (FP)
     STORE  R1, retfA (FP)

     POP    SP, R1 ; pal R1
     SUB    SP, =1 ; vap Z
     EXIT   SP, =2 ; vap par
```

Copyright Teemu Kerola 2004

Aliohjelman procC kutsu ei mitenkään eroa procB'n kutsusta. Niillä on täysin sama kutsurajapinta, eikä kutsuvalle rutiinille muutenkaan mitenkään kuulu se, millä tavoin kutsuttu aliohjelma procC tehtävänsä suorittaa.

Aliohjelma kutsuu toista aliohjelmaa

Korkean tason kieli

```
procC (x, y: int, var pZ:int)
{
    pZ = fA( x, y);
    return;
}
....
procC( 200, r, t);
```

Kutsu

```
40: PUSH    SP, =200
41: PUSH    SP, R
42: PUSH    SP, =T
43: CALL     SP, procC
44: ...
```

procC

```
parXc EQU -4 ; parametrit
parYc EQU -3
parpZ EQU -2
```

```
procC PUSH    SP, R1 ; tall rek
; @parpZ = fA (parXc, parYc)
      PUSH    SP, =0 ; paluuarv
      PUSH    SP, parXc (FP)
      PUSH    SP, parYc(FP)
      CALL     SP, fA
      POP      SP, R1
      STORE R1, @parpZ (FP)

      POP      SP, R1 ; pal R1
      EXIT     SP, =3 ; vap par
```

fA

```
retfA EQU -4 ; paluuarvo
parX EQU -3 ; parametrit
parY EQU -2
locZ EQU 1 ; paik muut
```

```
fA PUSH    SP, =5 ; paik. Z
   PUSH    SP, R1 ; tall rek
```

```
      LOAD     R1, parX (FP)
      MUL      R1, locZ (FP)
      ADD      R1, parY (FP)
      STORE R1, locZ (FP)
      STORE R1, retfA (FP)
```

```
      POP      SP, R1 ; pal R1
      SUB      SP, =1 ; vap Z
      EXIT     SP, =2 ; vap par
```

Copyright Teemu Kerola 2004

Aliohjelman procC määrittelyssä täytyy muodollisille parametreille X ja Y antaa uniikit nimet parXc ja parYc, koska funktiossa fA oli käytössä symbolit parX ja parY. Symbolisen konekielen tasollahan kaikkien symbolien tulee olla uniikkeja. Aliohjelman procC runko koostuu lähinnä funktion fA kutsusta, huomioiden kuitenkin sen, että parametrina välitettävien procC'n parametreihin tulee viitata normaaliin tapaan käyttäen niiden suhteellisia osoitteita tällä hetkellä aktiivisena olevan aktivointitietueen sisällä. Funktion fA määrittelyhän esitettiin jo aikaisemmin.

Rekursiivinen aliohjelma

Aliohjelma, joka kutsuu itseään

- suoraan (esim. A) tai epäsuoraan (esim. B)

Aktivointitietue hoitaa tilanvarauksen automaattisesti paikallisille muuttujille joka kutsukerralla

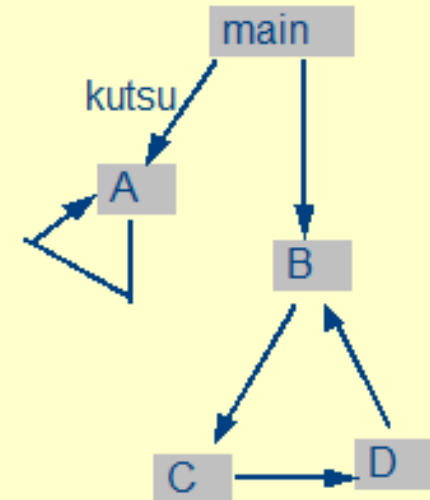
- paikalliset muuttujat allokoidaan pinosta joka kutsukerralle

Rekursio ei onnistu, jos paikallisten muuttujien tilanvaraus on aliohjelman koodin yhteydessä staattisessa paikassa

- jotkut Fortran versiot

Joka kutsukerralla suoritetaan sama koodi, mutta suoritusympäristö (AT) on uniikki sille kutsukerralle

- omat parametrit ja paikalliset muuttujat



Copyright Teemu Kerola 2004

Aliohjelma voi kutsua myös itseään, joko suoraan tai välillisesti jonkun muun aliohjelman kautta. Tällaista kutsua sanotaan rekursiiviseksi aliohjelmakutsuksi. Joka kutsukerralle allokoidaan kaikki paikalliset muuttujat uudelleen aktivointitietuepinosta.

Rekursiivinen aliohjelma

Aliohjelma, joka kutsuu itseään

- suoraan (esim. A) tai epäsuoraan (esim. B)

Aktivointitietue hoitaa tilanvarauksen automaattisesti paikallisille muuttujille joka kutsukerralla

- paikalliset muuttujat allokoidaan pinosta joka kutsukerralle

Rekursio ei onnistu, jos paikallisten muuttujien tilanvaraus on aliohjelman koodin yhteydessä staattisessa paikassa

- jotkut Fortran versiot

```
locX DC 0  
locY DC 0  
fB    PUSH SP, R1  
.....  
EXIT SP, =2
```

Joka kutsukerralla suoritetaan sama koodi, mutta suoritusympäristö (AT) on uniikki sille kutsukerralle

- omat parametrit ja paikalliset muuttujat

Copyright Teemu Kerola 2004

Joissakin Fortran versioissa aliohjelmien paikallisille muuttujille varataankin tilaa ohjelmakoodin yhteydessä. Tällöin tästä aliohjelmasta voi tietenkin olla vain yksi kutsukerta kerrallaan suorituksessa eikä rekursio ole mahdollista.

Rekursiivinen aliohjelma

Aliohjelma, joka kutsuu itseään

- suoraan (esim. A) tai epäsuoraan (esim. B)

Aktivointitietue hoitaa tilanvarauksen automaattisesti paikallisille muuttujille joka kutsukerralla

- paikalliset muuttujat allokoidaan pinosta joka kutsukerralle

Rekursio ei onnistu, jos paikallisten muuttujien tilanvaraus on aliohjelman koodin yhteydessä staattisessa paikassa

- jotkut Fortran versiot

Joka kutsukerralla suoritetaan sama koodi, mutta suoritusympäristö (AT) on uniikki sille kutsukerralle

- omat parametrit ja paikalliset muuttujat

AT main

AT fact

AT fact

AT fact

AT fact

AT fact

AT fact

AT fact

Copyright Teemu Kerola 2004

Rekursiossa suoritamme siis joka kutsukerralla saman koodin eli aliohjelman rungon, mutta joka kutsukerralla meillä on oma suoritusympäristö aktivointitietueessa sisältäen tämän suorituskerran parametrit ja paikalliset muuttujat. Rekursion pitää tietenkin joskus päättyä, koska muuten AT-pino syö nopeasti kaiken käytettävän muistitilan ja ohjelman suoritus päättyy 'stack overflow' virheilmoitukseen. Rekursio päättyy, kun rekursiivinen aliohjelma ei enää kutsukaan itseään rekursiivisesti.

Esimerkki: rekursio

Korkean tason kieli

```
fact (n: int): int
{
  if (n==1)
    return (1)
  else
    return (n * fact(n-1));
}
....
k = fact(4);
```

Kutsu

```
K    DC    0

; k = fact (4)
PUSH SP, =0
PUSH SP, =4
CALL SP, fact
POP  SP, R1
STORE R1, K
```

Toteutus

```
retF EQU -3 ; paluuarvo
parN EQU -2 ; parametri N

fact PUSH SP, R1 ; save R1
      LOAD  R1, parN(FP)
      COMP R1, =1
      JEQU One ; return 1 ?

      ; call fact(N-1)
      SUB   R1, =1
      PUSH SP, =0 ; ret. value
      PUSH SP, R1
      CALL SP, fact
      POP  SP, R1
      MUL  R1, parN(FP) ; fact(n-1)*n

One  STORE R1, retF(FP)
      POP  SP, R1; restore R1
      EXIT SP, =1 ; 1 param.
```

Copyright Teemu Kerola 2004

Tarkastellaan rekursiivista funktiota fact, joka laskee parametrina annetun kokonaisluvun kertoman rekursiivisesti. Muistattehan, että esimerkiksi luvun 5 kertoma on $5 * 4 * 3 * 2 * 1$. Otaksumme tässä naivisti, että parametrin arvo on aina positiivinen. Luvun 1 kertoma on 1 ja suurempien lukujen n kertoma lasketaan siis rekursiivisesti kaavalla $n * \text{luvun } n-1 \text{ kertoma}$. Yleisesti ottaen, rekursio ei ole paras tapa laskea kertomaa, vaan huomattavasti fiksumpaa olisi käyttää iteratiivista lähestymistapaa. Mutta rekursion periaatteen esittämiseen tämä kertoma sopii oikein hyvin!

Esimerkki: rekursio

Korkean tason kieli

```
fact (n: int): int
{
  if (n==1)
    return (1)
  else
    return (n * fact(n-1));
}
....
k = fact(4);
```

Kutsu

```
K    DC    0

      ; k = fact (4)
      PUSH SP, =0
      PUSH SP, =4
      CALL SP, fact
      POP  SP, R1
      STORE R1, K
```

Toteutus

```
retF EQU -3 ; paluuarvo
parN EQU -2 ; parametri N

fact PUSH SP, R1 ; save R1
      LOAD R1, parN(FP)
      COMP R1, =1
      JEQU One ; return 1 ?

      ; call fact(N-1)
      SUB R1, =1
      PUSH SP, =0 ; ret. value
      PUSH SP, R1
      CALL SP, fact
      POP SP, R1
      MUL R1, parN(FP) ; fact(n-1)*n

One STORE R1, retF(FP)
      POP SP, R1 ; restore R1
      EXIT SP, =1 ; 1 param.
```

Copyright Teemu Kerola 2004

Rekursiivista funktiota kutsutaan ihan samalla tavalla kuin muitakin funktioita. Laitamme pinoon tilan paluuarvolle ja parametrin n arvon ennen CALL-käskyä, ja CALL-käskyn jälkeen poppaamme pinosta pois siellä vielä olevan funktion arvon.

Esimerkki: rekursio

Korkean tason kieli

```
fact (n: int): int
{
  if (n==1)
    return (1)
  else
    return (n * fact(n-1));
}
....
k = fact(4);
```

Kutsu

```
K    DC    0

; k = fact (4)
PUSH SP, =0
PUSH SP, =4
CALL SP, fact
POP  SP, R1
STORE R1, K
```

Toteutus

```
retF EQU -3 ; paluuarvo
parN EQU -2 ; parametri N

fact PUSH SP, R1 ; save R1
      LOAD R1, parN(FP)
      COMP R1, =1
      JEQU One ; return 1 ?

      ; call fact(N-1)
      SUB R1, =1
      PUSH SP, =0 ; ret. value
      PUSH SP, R1
      CALL SP, fact
      POP SP, R1
      MUL R1, parN(FP) ; fact(n-1)*n

One STORE R1, retF(FP)
     POP SP, R1 ; restore R1
     EXIT SP, =1 ; 1 param.
```

Copyright Teemu Kerola 2004

Rekursio näkyy varsinaisesti funktion koodissa, jossa tietyn ehdon vallitessa se kutsuu rekursiivisesti itseään. Tietenkään se ei saa joka kutsukerralla kutsua itseään, muutenhan rekursio-kutsuketju olisi päättymätön. Tarkastellaan seuraavaksi, mitä tapahtuu, kun funktiota fact kutsutaan parametrin arvolla 4.

Esimerkki: rekursio

Korkean tason kieli

```
fact (n: int): int
{
  if (n==1)
    return (1)
  else
    return (n * fact(n-1));
}
....
k = fact(4);
```

Kutsu

```
K    DC    0

; k = fact (4)
PUSH SP, =0
PUSH SP, =4
CALL  SP, fact
POP   SP, R1
STORE R1, K
```

Toteutus

```
retF EQU -3 ; paluuarvo
parN EQU -2 ; parametri N

fact PUSH SP, R1 ; save R1
      LOAD  R1, parN(FP)
      COMP R1, =1
      JEQU One ; return 1 ?

      ; call fact(N-1)
      SUB   R1, =1
      PUSH  SP, =0 ; ret. value
      PUSH  SP, R1
      CALL  SP, fact
      POP   SP, R1
      MUL   R1, parN(FP) ; fact(n-1)*n

One   STORE R1, retF(FP)
      POP   SP, R1 ; restore R1
      EXIT  SP, =1 ; 1 param.
```

AT Main

AT fact (4)

AT fact (3)

AT fact (2)

AT fact (1)

Copyright Teemu Kerola 2004

Kun fact'ia kutsutaan parametrin arvolla 4, sitä kutsukertaa varten rakennetaan aktivointitietue pääohjelman aktivointitietueen päälle. Funktion koodissa havaitaan parametrin N arvon olevan erisuuri kuin 1, joten fact'ia kutsutaan uudelleen parametrin arvolla 3, jne. Lopulta, kun fact'ia on kutsuttu parametrin arvolla 1, rekursiivista kutsua ei tehdäkään, ja rekursiota ruvetaan purkamaan.

Esimerkki: rekursio

Korkean tason kieli

```
fact (n: int): int
{
  if (n==1)
    return (1)
  else
    return (n * fact(n-1));
}
....
k = fact(4);
```

Kutsu

```
K    DC    0

; k = fact (4)
PUSH SP, =0
PUSH SP, =4
CALL SP, fact
POP  SP, R1
STORE R1, K
```

Toteutus

```
retF EQU -3 ; paluuarvo
parN EQU -2 ; parametri N

fact PUSH SP, R1 ; save R1
      LOAD R1, parN(FP)
      COMP R1, =1
      JEQU One ; return 1 ?

; call fact(N-1)
SUB R1, =1
PUSH SP, =0 ; ret. value
PUSH SP, R1
CALL SP, fact
POP SP, R1
MUL R1, parN(FP) ; fact(n-1)*n

One STORE R1, retF(FP)
POP SP, R1 ; restore R1
EXIT SP, =1 ; 1 param.
```

AT Main

AT fact (4)

AT fact (3)

AT fact (2)

Copyright Teemu Kerola 2004

Ensimmäinen varsinainen paluu fact:ista tapahtuu, kun parametrin arvon havaitaan olevan 1. Kontrolli siirtyy osoitteeseen One, jossa paluuarvo talletetaan ensin paikalleen sen kutsukerran aktivointitietueeseen ja sitten EXIT-käskyllä palataan sen kertaista kutsua seuraavaan konekäskyyn, jossa lasketaan edellisen rekursiotason lauseke $N * \text{fact}(N-1)$.

Esimerkki: rekursio

Korkean tason kieli

```
fact (n: int): int
{
  if (n==1)
    return (1)
  else
    return (n * fact(n-1));
}
....
k = fact(4);
```

Kutsu

```
K    DC    0

      ; k = fact (4)
      PUSH SP, =0
      PUSH SP, =4
      CALL SP, fact
      POP  SP, R1
      STORE R1, K
```

Toteutus

```
retF EQU -3 ; paluuarvo
parN EQU -2 ; parametri N

fact PUSH SP, R1 ; save R1
      LOAD  R1, parN(FP)
      COMP R1, =1
      JEQU One ; return 1 ?

      ; call fact(N-1)
      SUB   R1, =1
      PUSH SP, =0 ; ret. value
      PUSH SP, R1
      CALL SP, fact
      POP  SP, R1
      MUL  R1, parN(FP) ; fact(n-1)*n

One  STORE R1, retF(FP)
      POP  SP, R1 ; restore R1
      EXIT SP, =1 ; 1 param.
```

AT Main

Copyright Teemu Kerola 2004

Tämän jälkeen kukin rekursiotaso purkautuu samalla tavalla. Ensin siis lasketaan tämän rekursiotason paluuarvo $N * \text{fact}(N-1)$, talletetaan se tämän kutsukerran aktivointitietueeseen paluuarvoksi ja palataan EXIT-käskyllä edelliselle rekursiotasolle, jne. Lopulta kontrolli palaa alkuperäiseen kutsukohtaan ja K'n arvoksi talletetaan 24.

Esimerkki: rekursio

Korkean tason kieli

```
fact (n: int): int
{
  if (n==1)
    return (1)
  else
    return (n * fact(n-1));
}
....
k = fact(4);
```

Kutsu

```
K    DC    0

      ; k = fact (4)
      PUSH SP, =0
      PUSH SP, =4
      CALL SP, fact
      POP  SP, R1
      STORE R1, K
```

Toteutus

```
retF EQU -3 ; paluuarvo
parN EQU -2 ; parametri N

fact PUSH SP, R1 ; save R1
      LOAD R1, parN(FP)
      COMP R1, =1
      JEQU One ; return 1 ?

      ; call fact(N-1)
      SUB R1, =1
      PUSH SP, =0 ; ret. value
      PUSH SP, R1
      CALL SP, fact
      POP SP, R1
      MUL R1, parN(FP) ; fact(n-1)*n

One STORE R1, retF(FP)
      POP SP, R1; restore R1
      EXIT SP, =1 ; 1 param.
```

Copyright Teemu Kerola 2004

Funktio fact, samoin kuin kaikki muutkin tässä luennessa esitetyt aliohjelmat ja funktiot, löytyy kurssin verkkomateriaalista. Voit siis käynnistää ttk-91 simulaattorisi ja suorittaa nämä esimerkkiohjelmat simulaattorissa. Samalla kertaa voit tarkemmin tutkia näiden esimerkkien yksityiskohtaista suoritusta ttk-91 tietokoneessa.

KJ-palvelun kutsu

Samalla tavalla kuin aliohjelmakutsu

- SVC-käsky CALL-käskyn asemesta
- suoritus etuoikeutetussa tilassa

Tilaa paluuarvolle?

Parametrit pinoon vai rekistereiden kautta?

SVC-kutsu

IRET-paluukäsky

Paluuarvo pois pinosta paluun jälkeen

Copyright Teemu Kerola 2004

Käyttöjärjestelmän palvelukutsut toteutetaan hyvin samalla tavalla kuin tavallisetkin aliohjelmakutsut. Erona on kuitenkin se, että käyttöjärjestelmäpalvelut suoritetaan etuoikeutetussa suoritustilassa. SVC-konekäsky toimii muuten hyvin vastaavalla tavalla kuin CALL-käskykin, mutta sillä voidaan kutsua vain käyttöjärjestelmän tunnettuja palveluja. Laitteiston tila vaihtuu kontrollin siirron yhteydessä etuoikeutetuksi ja kutsuttu rutiini heti alkuun tarkistaa kutsun laillisuuden. Tavallinen sovellus ei saa kutsua esimerkiksi tiedostojen hallintaan liittyvää käyttöjärjestelmän sisäistä hallintarutiinia.

KJ-palvelun kutsu

Samalla tavalla kuin aliohjelmakutsu

- SVC-käsky CALL-käskyn asemesta
- suoritus etuoikeutetussa tilassa

Tilaa paluuarvolle?

Parametrit pinoon vai rekistereiden kautta?

SVC-kutsu

IRET-paluukäsky

Paluuarvo pois pinosta paluun jälkeen

Copyright Teemu Kerola 2004

Käyttöjärjestelmän palvelupyynnön muoto voi olla hyvin yhtenevä aliohjelmakutsun kanssa, mutta siinä voi olla myös eroavaisuuksia. Esimerkiksi voi olla, että joillekin KJ-palveluille parametrit välitetään suoraan sovittujen rekistereiden kautta, koska se on nopeampaa kuin pinon kautta tapahtuva parametrien välitys. Palvelusta palataan etuoikeutetulla erikoiskäskyllä IRET, joka kontrollin siirron lisäksi palauttaa myös suorittimen suoritustilan ennalleen.

KJ-palvelun kutsu

Samalla tavalla kuin aliohjelmakutsu

- SVC-käsky CALL-käskyn asemesta
- suoritus etuoikeutetussa tilassa

Tilaa paluuarvolle?

Parametrit pinoon vai rekistereiden kautta?

SVC-kutsu

IRET-paluukäsky

Paluuarvo pois pinosta paluun jälkeen

```
...  
; fOK = ReadBlock (fp, 64)  
PUSH SP, =0 ; paluuarvo fOK  
PUSH SP, =FileBuffer  
PUSH SP, CharCnt  
PUSH SP, FilePtr  
SVC SP, =ReadFile  
POP SP, R1  
JNZER R1, FileTrouble
```

Copyright Teemu Kerola 2004

Esimerkkinä KJ-palvelusta tässä on tiedoston lukemiseen tarkoitettun palvelurutiinin ReadBlock kutsu. ReadBlock'ille annetaan viiteparametrina puskurialue FileBuffer luettaville merkeille, ja arvoparametreina luettavien merkkien lukumäärä CharCnt ja tiedosto-osoitin FilePtr. ReadBlock palauttaa arvonaan nollasta poikkeavan luvun, jos tiedoston luku syystä tai toisesta ei onnistunut.

Aliohjelmien toteutus

Aliohjelmatyypit

Parametrien tyypit

Aktivointitietue

Aktivointitietuepino

Rekursio

Käyttöjärjestelmäpalvelut

Copyright Teemu Kerola 2004

Olemme nyt käyneet läpi aliohjelmien toteutuksen konekielen tasolla. Teillä pitäisi nyt olla hyvä mielikuva siitä, kuinka aliohjelmakutsu toteutetaan aktivointitietueen avulla ja kuinka aliohjelmien kutsuhierarkia toteutetaan aktivointitietuepinon avulla. Kävimme läpi sekä arvo- että viiteparametrien käsitteet ja toteutuksen, ja tutustuimme myös nimiparametreihin. Esitimme, kuinka rekursio toteutetaan luontevasti AT-pinon avulla ja mitä yhteistä ja mitä eroavaisuuksia KJ-palvelujen käytöllä on tavallisten aliohjelmien käytön kanssa.