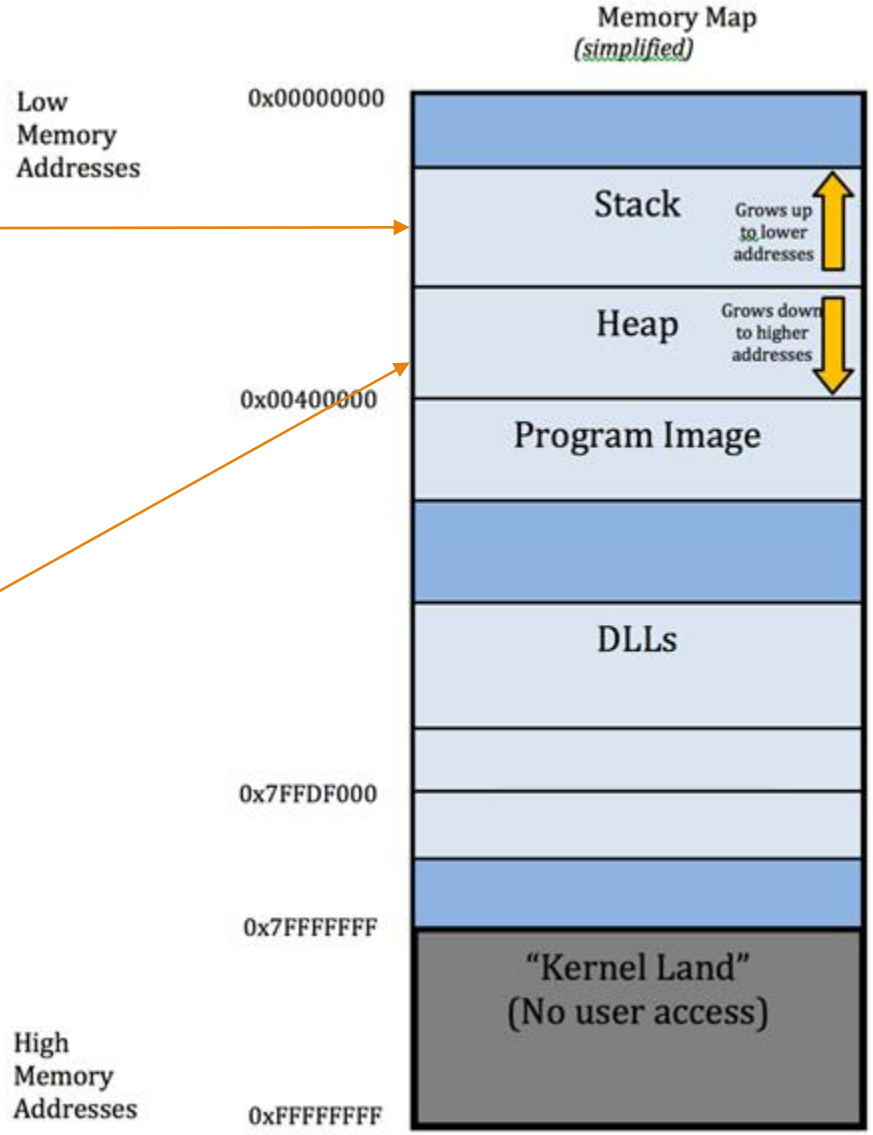
RAM-muisti: Pino ja pinokehys

- Pinomuistin (stack) koko vaihtelee eri järjestelmissä ja ohjelmointiympäristöissä, joten se täytyy aina erikseen selvittää spekseistä.
- MinGW –käännösympäristössä pinomuistin oletuskoko on n. 2 MB. Pinomuisti sijaitsee alkupäässä RAM muistia.
- Pino on ohjelmalle varattu muistialue, jonne talletetaan esimerkiksi funktioiden paikalliset muuttujat, funktion parametrit, ohjelman paluuarvo kun funktio suoritus päättyy jne.
- Ohjelmistosuunnittelun aikana on pinomuistin koko tiedettävä, jotta tiedetään kuinka paljon voidaan funktioissa varata muistia. Jos on vaarana, että pinomuisti ei riitä, niin otetaan käyttöön vapaan muistin alue (Heap, keko).

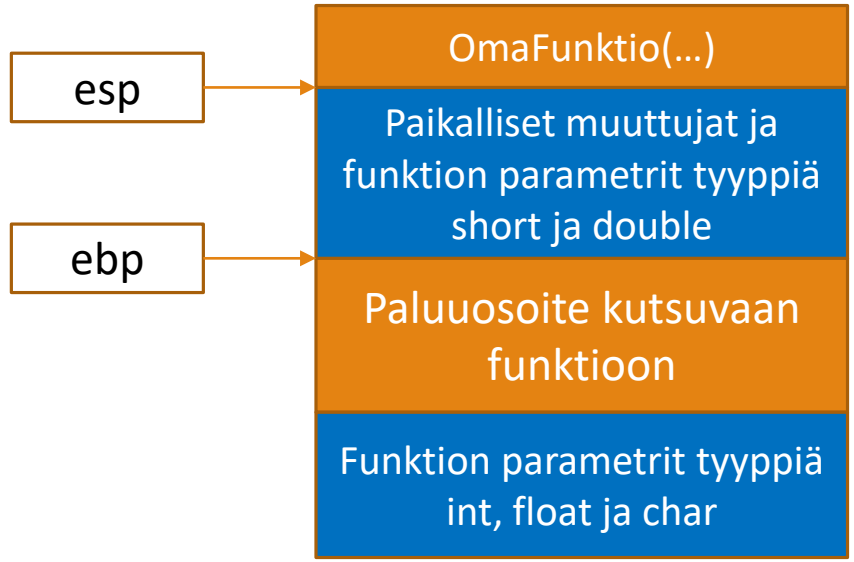


RAM-muisti: Pino ja pinokehys

- Pinomuistin alueelle rakentuu pinokehyksiä (stack frame) sen mukaan miten ohjelmassa suoritetaan funktioita. Jokaiselle funktiolle rakentuu oma pinokehys oikealla olevan kuvan mukaisesti.

- Prosessori pitää tallessa tietoa pinokehyksen senhetkisestä päällimmäisestä sijainnista rekisterissä (ESP, *stack pointer,* pino-soitin).
  HUOM! Pino-osoitin kasvaa aina alaspäin, siis kohti muistiosoitetta 0.

- Prosessori pitää tallessa tietoa pinokehyksen alimmasta sallitusta sijainnista rekisterissä (EBP, *stack base pointer*, pinokehyksen pohjan osoitin)



RAM-muisti: Pino ja pinokehys

- Tee uusi projekti nimeltä **PinoJaPinokehys** (**Non-Qt Project->Plain C++ Application**).
- Kommentoi **main.cpp** tiedostossa oleva koodi kokonaan ja kirjoita alla oleva koodi Qt Creatoriin.

```
#include <iostream>

using namespace std;

int main()
{
   short kokonaislukumuuttuja=0;
   kokonaislukumuuttuja=15;

   cout << "kokonaislukumuuttuja arvo: " << kokonaislukumuuttuja<< endl;
   cout << "kokonaislukumuuttuja ensimmainen muistiosoite: " << &kokonaislukumuuttuja<< endl;
   cout << "kokonaislukumuuttuja koko: " << sizeof(kokonaislukumuuttuja) << endl;

   return 0;
}
```

- Seuraavalla sivulla on ohje miten edetään debuggauksessa

RAM-muisti: Pino ja pinokehys

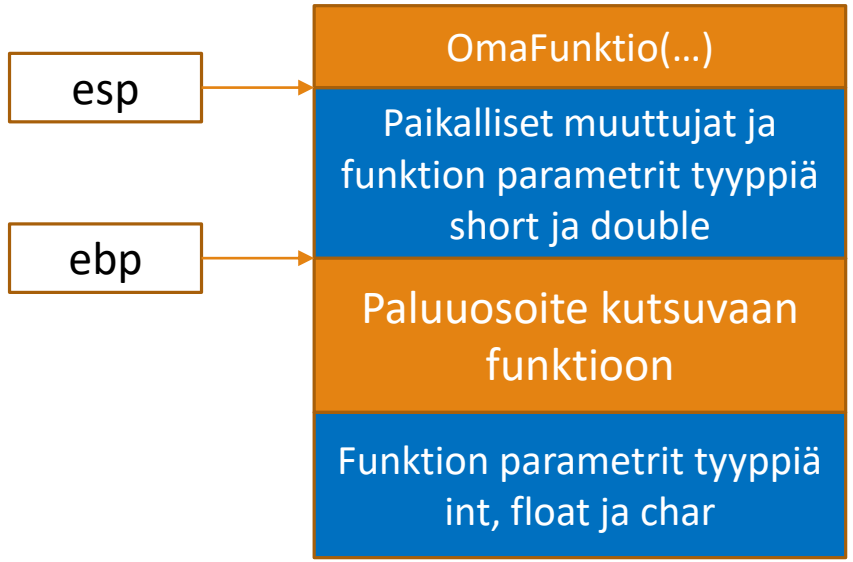
- Aseta keskeytyspiste vain riville **return 0;**
- Aloita Debuggaus
- Aseta rekisterit näkymään päävalikosta Window->Views->Registers
  - Selvitä **Registers** –ikkunasta **main()** funktion pinokehyksen pino-osoittimen (esp rekisteri) ja pinokehyksen pohjan (ebp rekisteri) arvot, ja kirjoita ne ylös.
- Avaa **Open Memory Editor** –ikkuna aikaisemman opitun mukaisesti. Kirjoita ikkunaan muistiosoitteeksi muuttujan **short kokonaislukumuuttuja** muistiosoitteen arvo.
- Etsi muisti-ikkunasta **main()** funktion pinokehyksen pino-osoittimen (esp rekisteri) muistipaikka ja pinokehyksen pohjan (ebp rekisteri) muistipaikka.
- Muuttuja **short kokonaislukumuuttuja** löytyy pinosta rekisterien **esp** ja **ebp** väliseltä alueelta. Etsi muuttujan **kokonaislukumuuttuja** muistipaikka ja arvo muisti-ikkunasta.
- Lopeta debuggaus

RAM-muisti: Pino ja pinokehys, debuggaus

- Kommentoi **main.cpp** tiedossa oleva koodi ja kirjoita oikealla oleva koodi tiedostoon
- Käännä ohjelma
- Seuraavalla sivulla kerrotaan miten debugataan ohjelmaa

```
#include <iostream>

using namespace std;

void omaFunktio(short parametri)
{
   short omaMuuttuja=0;

   omaMuuttuja=15;
   cout << "omaMuuttuja arvo: " << omaMuuttuja << endl;
   cout << "omaMuuttuja ensimmainen muistipaikka: " << &omaMuuttuja << endl;
   cout << "parametri arvo: " << parametri << endl;
   cout << "parametri ensimmainen muistipaikka: " << &parametri << endl;

   cout << "Poistutaan funktiosta omaFunktio()" << endl;
}

int main()
{
   short muuttujaYksi=10;

   cout << ”muuttujaYksi  arvo: " << muuttujaYksi << endl;
   cout << ”muuttujaYksi ensimmainen muistipaikka: " << &muuttujaYksi << endl;

   omaFunktio(10);

   return 0;
}
```

RAM-muisti: Pino ja pinokehys, debuggaus



- Aseta keskeytyspisteet seuraavasti:
  - funktiossa **omaFunktio()** riville
    cout << "Poistutaan funktiosta omaFunktio()" << endl;

  - funktiossa **main()** riville
    omaFunktio(10);

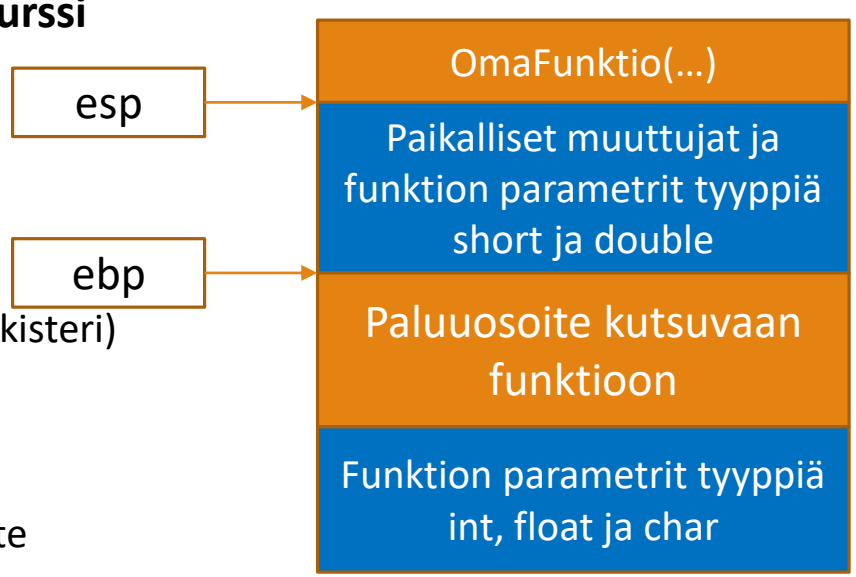
- Aloita Debuggaus

- Aseta **Stack**-ikkuna näkymään valitsemalla päävalikosta **Window->Views->Stack**
  - Stack-ikkunasta näkee missä funktiossa ollaan, millä rivillä ja mikä on koodirivin muistiosoite muistin koodialueella (Address sarake).
  - Jos Address sarake ei näy, siirrä hiiren kursori **Stack** ikkunan alueelle, klikkaa hiiren oikeaa ja valitse valikosta vaihtoehto **Show Address Data in Stack View when Debugging**

- Aseta prosessorin rekisterit näkymään päävalikosta **Window->Views->Registers**
  - Selvitä funktion **main()** pino-osoittimen (esp rekisteri) ja pinon pohjan (ebp rekisteri) muistiosoitteiden arvot ja laita ne ylös.

- Seuraavalla sivulla jatketaan debuggausta

RAM-muisti: Pino ja pinokehys, debuggaus



- Askella ohjelmaa (F10) kunnes ollaan toisen keskeytyspisteen rivillä
- Selvitä funktion **omaFunktio()** pino-osoittimen (esp rekisteri) ja pinon pohjan (ebp rekisteri) muistiosoitteiden arvot.
  - Vertaa lukuja **main()** funktion pino-osoittimen ja pinon pohjan arvoihin.
- **Stack**-ikkunasta näkee missä funktiossa ollaan, millä rivillä ja mikä on rivin muistiosoite
- Avaa **Open Memory View** ja syötä muistiosoitteen arvoksi funktion **omaFunktio()** esp rekisterin arvo
- Etsi muisti-ikkunasta paikallisen muuttujan **short omaMuuttuja** ja parametrimuuttujan **short parametri** muistipaikat
- ***Muuttujat sijaitsevat siis funktion pinokehyksen alueella!***
- Muisti-ikkunassa **ebp** –rekisterin muistiosoitteen jälkeen (ei välttämättä heti seuraava muistipaikka) pitäisi löytyä paluuosoite **main()** funktioon. Arvon pitää olla sama joka on **Stack** ikkunassa **main()** funktion rivillä kohdassa **Address**.
- Etsi paluuosoite muisti-ikkunasta funktioon **main()**.
- Lopeta debuggaus

RAM-muisti, pino ja pinokehys

- Lisää funktioon **omaFunktio** kaksi uutta parametria (**short** tyyppi), debuggaa ohjelmaa ja selvitä muistipaikat, jotta löydät uudet parametrimuuttujat pinomuistista

- Lisää seuraavat ohjelmaan
  - lisää ohjelmaan uusi funktio **short toinenFunktio(short parametri)**
  - lisää funktioon paikallinen muuttuja **short toinenMuuttuja** ja anna sille arvo 15
  - tulosta muuttujien **toinenMuuttuja** ja **parametri** arvot ja ensimmäiset muistipaikat
  - lisää rivi **cout << "Poistutaan funktiosta toinenFunktio()" << endl;**
  - aseta keskeytyspiste edellä lisätylle koodiriville
  - palauta paikallisen muuttujan arvo funktion viimeisellä rivillä **return toinenMuuttuja;**

  - kutsu funktiota **toinenFunktio(10)** funktiosta **omaFunktio** sen viimeisellä rivillä
  - debuggaa ohjelmaa ja selvitä seuraavat asiat muisti-ikkunasta:
    * funktion **toinenFunktio** pinokehyksen pino-osoittimen ja pinon pohjan arvot ja sijainnit muistissa
    * funktion **toinenFunktio** paikallisen muuttujan muistiosoitteet ja sijainti muistissa
    * funktion **toinenFunktio** parametrin muistiosoitteet ja sijainti muistissa
    * etsi paluuosoite funktioon **omaFunktio()**

RAM-muisti: Pinomuisti

- Rekisterien arvoja voidaan ohjelmassa tutkia myös käskyllä **register int** <rekisterinnimi> **asm** (”<rekisterinnimi>”);
- Kommentoi **main.cpp** tiedossa oleva koodi ja kirjoita tiedostoon alla oleva koodi

```
#include <iostream>
using namespace std;

int main()
{
   register int esp asm ("esp");
   register int ebp asm ("ebp");

   cout << "rekisterin esp arvo: " << esp << endl;
   cout << "rekisterin ebp arvo: " << ebp << endl;

   return 0;
}
```

- Aseta keskeytyspiste riville **return 0;** ja debuggaa ohjelmaa
  - Etsi **Registers** ikkunasta rekisterien **esp** ja **ebp** arvot heksalukuina ja selvitä laskimen avulla niiden arvot kokonaislukuna. Tulokseksi pitäisi tulla samat luvut, jotka edellä tehty ohjelma tulostaa ohjelmaikkunaan.
  - Laita rekisterien kokonaislukuarvot ylös – **esp** arvoa tarvitaan myöhemmin
  - Voit lopettaa debuggauksen

RAM-muisti: Pinomuisti ja pinokehyksen koko

- Kirjoita ennen **return 0;** riviä **cout << "pinokehyksen koko: " << ebp - esp << endl;** ja suorita ohjelmaa.
- Kirjoita ylös pinokehyksen koko.

- Kirjoita **register int** muuttujien määrittelyn jälkeen rivi **short kokonaislukuTaulukko[1];** ja suorita ohjelmaa.
- Kirjoita ylös pinokehyksen koko.

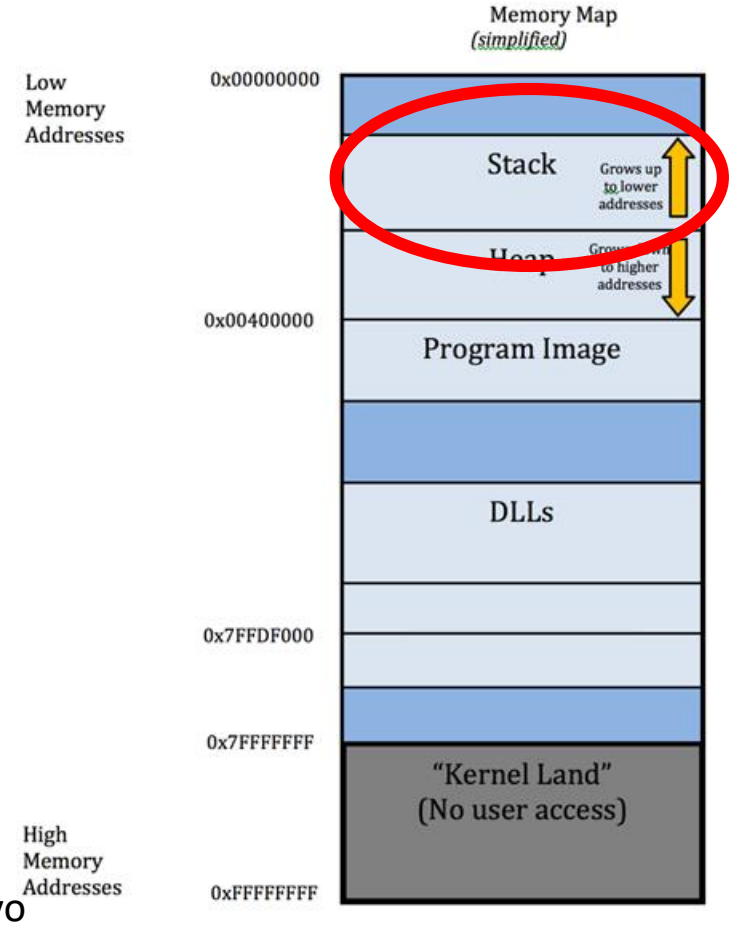
- ***Eli funktion pinokehyksen koko kasvaa kun funktioon lisätään muuttujia!***

- Lisää alkioita taulukkomuuttujaan **kokonaislukuTaulukko[…]**, suorita ohjelmaa ja katso miten pinokehyksen koko kasvaa.

- ***Eli mitä enemmän määritellään ohjelman eri funktioissa muuttujia, niin sitä enemmän käytetään pinomuistia!***

- Koska pinomuistia PC puolella on yleensä 1-4 MB käytössä (riippuen ympäristöstä), on paikallisten muuttujien määrittelyssä oltava huolellinen kun ohjelmaa suunnitellaan ja ohjelmoidaan. Varsinkin kun ollaan tekemisissä pienimuististen laitteiden kanssa, on aina syytä selvittää eri muistialueiden koot ohjelmointiympäristössä. Sulaututeissa järjestelmissä pinomuistia voi olla esim. 128-256KB.

RAM-muisti: Pinomuistin käyttäminen loppuun



- Lisää ohjelmaan alla oleva funktio

```
void omaFunktio()
{
   register int esp asm ("esp");

   cout << "rekisterin esp arvo: " << esp << endl;
   omaFunktio(); //rekursiivinen kutsu
}
```

- Ylläoleva funktio on rekursiivinen koska se kutsuu itseään, tässä esimerkissä loputtomasti.

- Lisää funktioon **main()** ennen riviä **return 0;** rivi **omaFunktio();**

- Kun lähdet suorittamaan ohjelmaan, niin ohjelmaikkunassa alkaa juosta pino-osoittimen **esp** arvo
- *Se pienenee koko ajan, koska ohjelmassa rekursiivisesti kutsutaan aina uutta funktiota **omaFunktio()** joka varaa aina uuden pinokehyksen*
- Jossain vaiheessa ohjelma saavuttaa pinomuistin katon ja kaatuu, koska pinomuistia ei ole enää käytettävissä
- Kirjoita ylös viimeinen pino-osoitin sen jälkeen kun ohjelma on kaatunut.
- Jos nyt vähennät ensimmäisen ylös kirjoittamasi pino-osoittimen arvosta viimeisen pino-osoittimen arvon, pitäisi luku olla yli 2 miljoonaa eli pinomuistin koko on tässä ympäristössä n. 2MB.