
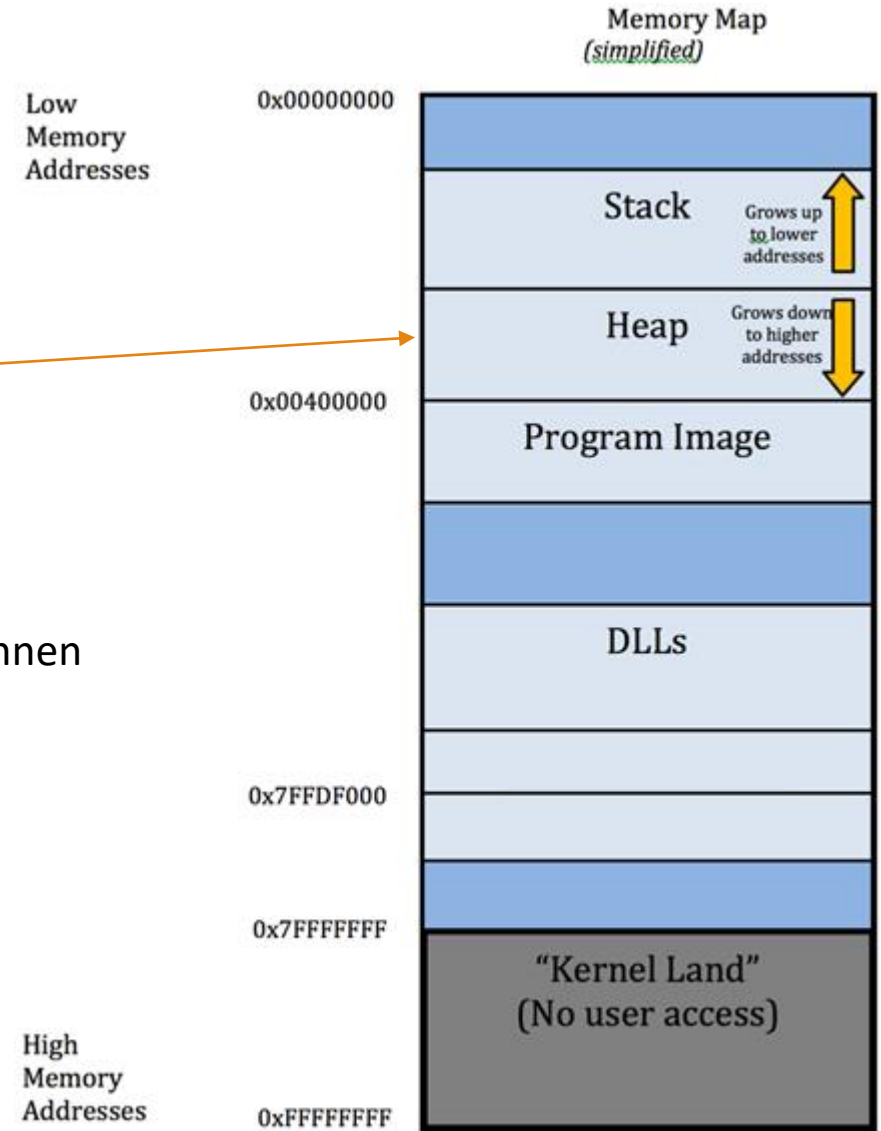


Olio-ohjelmointi: dynaaminen muisti

- Automaattinen olio käyttää pinomuistia ja jos automaattinen olio haluaa kutsua jäsenfunktiota se käyttää pisteoperaattoria esim. ***olio.jasenfunktio();***
- Dynaaminen olio käyttää kekomuistia (Heap, vapaa muisti) 
- Dynaaminen muistinvaraus tehdään C++ -kielessä **new** -komennolla. Se varaa tilaa kekomuistista.
- Koska **new** palauttaa **muistiosoitteen** täytyy dynaaminen olio määritellä ennen muistinvarausta **osoittimeksi** esim. **MyClass *objectMyClass;**
 - Osoittimelle ***objectMyClass** varataan muistia pinosta.
- Sanaa **new** seuraa tyyppinimi, joka kertoo, minkä tyyppistä tietoa varatulle alueelle viedään ja samalla myös sen, kuinka paljon muistia pitää varata.
 - **new short** varaa kaksi tavua muistia.
 - **new long** varaa kahdeksan tavua muistia.
 - **objectMyClass = new MyClass;** varataan tilaa kekomuistista luokan jäsenmuuttujien määrittelyjen mukaisesti.



Olio-ohjelmointi: dynaaminen muisti

- Luo Qt Creatorilla uusi projekti **MyDynamicOOPProject (Non-Qt Project->Plain C++ Application)** ja lisää projektiin luokka **MyDynamicClass**
- Jos luokan rakenteeseen (.h tiedosto) ei tule automaattisesti tuhoajafunktiota **~MyDynamicClass()**, niin lisää se sinne.
- Lisää luokan toteutukseen (.cpp tiedosto) molempien jäsenfunktioiden (muodostin- ja tuhoajafunktio) toteutuksen rungot. Muodostinfunktion runkokoodi voi olla jo valmiina!
- Lisää luokan muodostin- ja tuhoajafunktioon tulostuslauseet **cout** komennolla, jotka tulostavat näytölle tietoa olion luonnista ja tuhoamisesta (muista **endl** niin tulostus näytölle on selkeämpi).
- Muista lisätä **mydynamicclass.h** tiedostoon ennen riviä **class ...** tulostusolion **cout** tarvitsemat koodirivit.
- Suorita build. Jos tulee virheitä, niin ilmoita opettajalle.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: dynaaminen muisti

- Lisää luokan määrittelyn **private** osaan jäsenmuuttuja **short myMemberVariable**;
- Alusta jäsenmuuttujan **myMemberVariable** alkuarvoksi arvo 15 muodostinfunktiossa.
- Tuhoajafunktiossa aseta jäsenmuuttujan **myMemberVariable** arvoksi 0.
- Jäsenmuuttujat on aina syytä alustaa. Tämä on myös hyvä ohjelmointitapa!

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: dynaaminen muisti

- Koska **new** palauttaa **muistiosoitteen** täytyy olio määritellä ennen muistinvarausta **osoittimeksi** esim. **MyClass *objectMyClass;**
- Varsinainen muistinvaraus tehdään käskyllä **objectMyClass = new MyClass;**
- Eli olion luontiin tarvitaan kaksi riviä koodia

```
MyClass *objectMyClass; // osoitin menee pinoon
```

```
// new palauttaa muistiosoitteen ja varaa muistia keosta (Heapista).
```

```
// Muistia varataan luokan jäsenmuuttujien tyyppien mukaan. short 2 tavua, int 4 tavua jne. riippuen kehitysympäristöstä  
objectMyClass = new MyClass;
```

- Toinen mahdollisuus on tehdä yllä oleva yhdellä rivillä: **MyClass *objectMyClass = new MyClass;**
- Jos halutaan välittää muodostinfunktiolle parametrejä kirjoitetaan: **MyClass *objectMyClass = new MyClass (5,10);**
- Kun oliota ei enää tarvita, se tuhotaan **delete** komennolla ja asetetaan olion arvoksi **nullptr**, **NULL** tai **0**.

```
delete objectMyClass;
```

```
objectMyClass = nullptr; // luokan jäsenmuuttujien muistiosoitteet saavat arvon NULL tai 0. Näin pidetään muisti siistinä!
```

- Seuraavilla sivulla lisätään yllä olevat asiat ohjelmaan.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: dynaaminen muisti

- Avaa **main.cpp** tiedosto, poista tiedostossa oleva koodi ja kirjoita tiedostoon alla olevat rivit

```
#include "mydynamicclass.h"

int main()
{
    // Määritellään MyDynamicClass tyyppinen osoitin *objectMyDynamicClass
    // Osoittimelle varataan muistia pinosta

    MyDynamicClass *objectMyDynamicClass;
    cout << "Osoittimien *objectMyDynamicClass muistiosoite pinossa: " << &objectMyDynamicClass << endl;

    // Luodaan olio ja varataan oliolle muistia new operaattorilla.
    // New palauttaa muistiosoitteen ja varaa muistia keosta (Heapista, vapaasta muistista).
    // Muistia varataan luokan jäsenmuuttujien tyyppien mukaan. short 2 tavua, int 4 tavua jne.

    objectMyDynamicClass = new MyDynamicClass;

    // Tuhotaan olio ja vapautetaan olion varaama muistitila.
    // Asetetaan osoittimen arvoksi nullptr (jossain järjestelmissä NULL tai 0), jotta muisti on näin siivottu ohjelmoijan toimesta.
    delete objectMyDynamicClass;
    objectMyDynamicClass = nullptr; // luokan jäsenmuuttujien muistiosoitteet saavat arvon NULL tai 0. Näin pidetään muisti siistinä!

    return 0;
}
```

- Tutustu koodiin huolella, suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa näytölle?

Olio-ohjelmointi: dynaaminen muisti

- Lisää muodostinfunktioon alla oleva rivi ja suorita ohjelmaa. Nyt huomaat jäsenmuuttujan muistiosoitteesta, että muuttujalle ei enää varata tilaa pinosta vaan **kekomuistialueelta**! Tämä siksi, koska luokasta luotu olio on luotu **main()** funktiossa dynaamisesti **new** operaattorilla.

```
cout << "Luokan jäsenmuuttujan myMemberVariable muistiosoite kekomuistissa: "<< & myMemberVariable << endl;
```

- Lisää luokkaan jäsenfunktio **void myDynamicMemberFunction();** ja kirjoita toteutuksen tyhjä runko.
- Lisää jäsenfunktioon tulostuslause, joka kertoo missä funktiossa ollaan.
- Lisää **main()** funktiossa olion luonnin (**new** lause) jälkeen rivi

```
objectMyDynamicClass->myDynamicMemberFunction();
```

- Kun *olio on luotu dynaamisesti* pitää pisteoperaattorin sijaan käyttää merkkejä -> kun olio kutsuu luokan jäsenfunktiota
- Suorita build ja aja ohjelmaa.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: dynaaminen muisti

- Lisää **main()** funktioon juuri ennen **return 0;** riviä koodi **objectMyDynamicClass->myDynamicMemberFunction();**
- Lisää jäsenfunktioon **myDynamicMemberFunction()** koodirivi **cout << &myMemberVariable << endl;**
 - *Nyt jäsenmuuttujan ensimmäinen muistipaikka saa arvon 0, koska olio on tuhottu ja samalla jäsenmuuttujien muistipaikat vapautetaan **delete** käskyllä ja muisti siivotaan asettamalla jäsenmuuttujien arvoksi NULL tai 0.*
- Suorita build ja aja ohjelmaa, niin voit todeta yllä olevan asian jäsenmuuttujan muistipaikan osalta.
 - *Osoitinmuuttuja on periaatteessa näkyvässä pinossa **delete** käskyn jälkeen, mutta olion tietoja ei voi käsitellä, koska jäsenmuuttujien muistipaikat on vapautettu.*
- Lisää jäsenfunktioon **myDynamicMemberFunction();** alla oleva rivi ennen tulostuslausetta

myMemberVariable=10;
- Suorita build ja aja ohjelmaa.
- ***Ohjelma kaatuu, koska olio on tuhottu ja jäsenmuuttujan muistipaikka on vapautettu, joten muuttujia ei voi käyttää! Tyypillinen ohjelmointivirhe!!!***
- Poista/kommentoi koodista tällä sivulla lisätyt koodirivit.

Olio-ohjelmointi: this-osoitin

- Jokainen olio (olipa kysessä automaattinen tai dynaaminen olio) saa yhden osoittimen automaattisesti. Sen osoittimen nimi on **this**. Tämä **this**-osoitin osoittaa olioon itseensä. Koska se on osoitin, niin sen arvo on muistiosoite.
- Ohjelmoija ei esittele **this**-osoitinta, vaan se on oletusarvoisesti luokasta tehtyjen olioiden ja niille kuuluvien jäsenfunktioiden ja –muuttujien käytössä.
- Kutsuttaessa ohjelmassa luokan jäsenfunktioita, voidaan **this**-osoitinta käyttää tai olla käyttämättä koodissa, koska järjestelmä käyttää automaattisesti **this**-osoitinta, vaikka sitä ei olisi koodissa näkyvillä.

```
void Luokka::jasenFunktio()
{
    jokinToinenFunktio();
    this->jokinToinenFunktio(); // sama asia kuin yllä oleva funktiokutsu
}
```

- Kirjoita muodostinfunktioon viimeiseksi riviksi koodi **cout << "Olion ensimmäinen muistipaikka on: " << this << endl;**
- Suorita build ja aja ohjelmaa. Eli nyt huomaat, että olion ensimmäisen muistipaikka on sama kuin olion ensimmäisen jäsenmuuttujan osoite kekomuistissa.

IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: this-osoitin

- Lisää luokkaan jäsenfunktio **void myFunction();** ja kirjoita jäsenfunktioon tulostuslause **cout << "Hello" << endl;**
- Kirjoita jäsenfunktion **myDynamicMemberFunction()** loppuun alla oleva rivi

```
this->myFunction();
```

- Suorita build ja aja ohjelmaa.
- Jos olio tarvitsee välittää **parametrina** jäsenfunktiolle, niin voidaan käyttää **this** –osoitinta.
- Lisää luokkaan jäsenfunktio **void myThisFunction(MyDynamicClass *parameterThis);** ja kirjoita jäsenfunktioon koodirivit

```
cout << "parameterThis muistiosoite: " << parameterThis << endl;  
parameterThis->myFunction();
```

- Kirjoita jäsenfunktioon **myDynamicMemberFunction()** loppuun rivi **this->myThisFunction(this);**
- Suorita build ja aja ohjelmaa.