Heavy Object-Oriented Programming in C++ 2

edwin.vanouwerkerkmoria@hku.nl

Over deze lessen

Flink veel programmeer werk

Opfrissen van je kennis? Lynda!

bv “Learning C++” van Peggy Fisher

Module is 2 EC = **56 uur**

8 (werk)colleges

*ong. **4-5** uur/week eigen werktijd*

Over deze lessen: voorkennis

Basiskennis C++

Classes, methods,

'The Big Three': constructor, copy-constructor, assignment operator

stack vs. heap, pointers, references, pass-by-value

Basisprincipes OO: inheritance, polymorphism, overloading, abstract classes

IO streams, stream operators

Over deze lessen: inhoud

Meer memory management
    Pointers vs. references

OO principes: Design patterns

Const-correctness

Moderne (C++ 11/14/17) features

    threading
    ...

Module-repository op GitHub

https://github.com/edovino/HEOBP

Uitgewerkte voorbeelden uit de bijeenkomsten

Evt. notities

Opdrachten

Github: je eigen repository

Code inleveren doe je via GitHub

- maak een repository aan op github.com
  maak een github account als je die nog niet hebt

op het *Create New Repository* scherm:

- selecteer 'Public' repository
- selecteer een .gitignore file (*kies C++ uit de dropdown)*

Na aanmaken, kopiëer de inhoud van de .gitignore uit de klas repo, en plak die in je .gitignore in je eigen repo.
Mail mij de link naar je repository!

Codingstyle

Code schrijf je in eerste instantie voor jezelf, in tweede voor je collega(s), en pas op de laatste plaats voor je compiler.

- Schrijf *leesbaar*: duidelijke variabele- en methode namen
- Formatteer consistent (je IDE helpt)
- Voeg commentaar toe

Haakjes boeien me niet. Wel een paar basic C++ conventies (*geen C# stijl dus!)*

- Classes beginnen met een **HOOFDLETTER**
- methods beginnen met een **kleine letter**
- variabelen beginnen met een **kleine letter**

Opdrachten per week

Geen aparte eindopdracht

Je beoordeling gaat op basis van de opdrachten die je per sessie krijgt

Opdrachten de ochtend vóór volgende keer bij mij inleveren voor beoordeling!
Dus: sessie op donderdagmiddag? Inleveren voor woensdag 12:00

Feedback vragen mag natuurlijk altijd!

We bespreken de oplossingen/problemen gezamenlijk.

Vandaag

Herhaling

- simpele datastructuren
  - *array, vector*

Iterators

- updates via iterators

Datastructuren: array

```
Students[50] students;


for (int i=0; i<50; i++) {
    int ec = student[i].calculateECs();
    if (ec>60) {

        //...
    }

}
```

Datastructuren: vector

```
vector<Student> studenten;

for (int i=0; i<50; i++) {

    Student nieuweStudent = Student("Pietje Puk");

    nieuweStudent.voegModulesToe();

    studenten.push_back(nieuweStudent);

}
```


JURASSIC PARK

Iterators

Iterators 'lopen' over een verzameling objecten heen-en-weer

```
vector<Student> studenten;

vector<Student>::iterator i = studenten.begin();

while (i != studenten.end()) {
    i->updateECs();
    i++;
}
```


JURASSIC PARK

Iterators (2)

Voordeel van iterators: 'begrijpen' de onderliggende datastructuur - ook als je dingen verwijdert:

```
vector<Student>::iterator it = studenten.begin();

while (it != studenten.end()) {
    if (it->isAfgestudeerd()) {
        it = studenten.erase(it);
    } else {
        it++;
    }
}
```

Geheugen in C++

Twee soorten geheugen

- Stack: *automatic allocation*

  beperkte ruimte
  meestal: variabelen binnen een methode, parameters(!)

- Heap: *dynamic allocation*

  max. geheugen in computer
  grote, of grote aantallen objecten (videobeelden, particle clouds)

Geheugen in C++: stack

- Stack (*automatic allocation)*

```
void updateEC() {
    Student student;
    student.updateEC();
}
```

Bestaat alleen binnen huidige scope - automatisch opgeruimd!

Geheugen in C++: Heap

- expliciet aanmaken met ***new*** operator

```
Student* student = new Student();

student->wijsModulesToeVoorJaar(4);
```

pointers: **adres** van een variabele

- *Zelf opruimen!*

```
delete student;
```

Iterators over vectors met *objecten*

```
vector<Student>::iterator i = studenten.begin();
while (i != studenten.end()) {
    i->updateEC();
    i++
}
```

Iterators over vector met *pointers*

```
vector<Student*>::iterator i = studenten.begin()
while (i != studenten.end()) {
    (*i)->updateEC();
    i++;
}
```

*dereference operator **

dereferencing betekent 'zoek object bij deze pointer'

```
Student* particlePointer = new Student("Pietje");
```

Aanroepen methodes op object via pointer:

```
student->updateEC();
```

Aanroepen methodes door eerst object te *dereferencen*

```
(*student).updateEC();
```

Opdracht volgende keer

Ga uit van 3 classes: Module, Student, Docent
module: *naam, ec*, Student: *naam*, Docent: *naam*

- maak 3 Modules, 10 studenten, 3 docenten (*op de heap!)*
- wijs docent toe aan module
- wijs studenten toe aan 1-3 modules (verschillend!)
- print een lijst van modules, met docent & studenten
- toon totaal EC per student

1) Wijzig EC van 1 module: toon totaal EC per student
2) Verwijder student uit module, & toon lijst opnieuw