Introductie in C++

Jan van Rijn

September 2013

Inhoud

Classes

Voordelen van classes:

- Modelleren
- Modulariteit
- Informatie afschermen
- Makkelijk(er) te debuggen

Classes

- Classes bestaan uit methods (members) en variabelen
  - `public`: toegankelijk vanuit het gehele programma
  - `protected`: zie verder
  - `private`: toegankelijk alleen vanuit de klasse
- Variabelen zijn normaal gesproken `private` of `protected`
  - Initialiseren in constructor
  - Ophalen m.b.v. speciale getter functie
  - Aanpassen m.b.v. setter functie

Classes

```
class Box {
  public:
    Box( int l, int w, int h );
    int getVolume( );

    int getLength( );
    int getWidth( );
    int getHeight( );

    void setLength( );
    void setWidth( );
    void setHeight( );
  private:
    int length, width, height;
};
```

Classes

```
Box::Box( int l, int w, int h ) {
  length = l;
  width = w;
  height = h;
}

void Box::setLength( int l ) {
  length = l;
}

int Box::getLength( ) {
  return length;
}
```

Classes

- Werking van een programma kan opgesplitst worden in classes
- Elke class hoort een declaratie in een header file te hebben en een uitwerking in een source file
- Elke (enigszins grote) class in aparte file
- Header file extensies: `.h` / `.hpp` / `.hh`
- Source file extensies: `.cpp` / `.cc`
- Source files worden apart gecompiled; header files worden alleen met includes bijgevoegd

Classes

- Met goed gebruik van classes zijn geen globale variabelen nodig
- Functies bevatten bij voorkeur hooguit 10–15 regels code
- Classes horen te werken als een black box; het gebruik laat niets zien over de interne werking
- De header file is een uitstekende plek voor (veel) commentaar (Doxygen)
  - Pre- en Postcondities, parameters, return variabele

Classes

- Constructor
  - Wordt automatisch aangeroepen bij het aanmaken.
  - Prima manier om variabelen te initialiseren
  - Probeer het aantal bewerkingen te beperken!
  - `public Box( )`
- Destructor
  - Wordt automatisch aangeroepen bij vernietiging
  - Essentieel om geen geheugen geblokkeerd te houden
  - `public ~Box( )`

Classes

Preprocessor operators

- Commando's worden uitgevoerd voor het daadwerkelijke compileren
- Al eerder gebruikt: `#include <iostream>`
- Wordt veelal gebruikt om te voorkomen dat functies dubbel worden gedefinieerd
- Header files:

```
#ifndef Bestandsnaam_h
#define Bestandsnaam_h

// Hier je reguliere C++ code

#endif
```

Overerving

Overerving of “inheritance” wordt gebruikt om

- Functionaliteit te hergebruiken
- Eenheid te creëren tussen classes
- Kinderen die een ouder uitbreiden nemen alle members en variabelen van de ouder over, behalve `private` members en variabelen
- Kinderen kunnen members overschrijven

Overerving

- Naast `public` en `private`, nu ook `protected` variabelen en members
- `protected`: Variabele / functie niet bereikbaar van buiten af, maar is wel beschikbaar in child classes

Overerving

- Verschillende vormen van overerving:
  - `public` overerving
  - `protected` overerving
  - `private` overerving
- Wordt gebruikt om verdere restricties op te leggen voor toegang tot members en variabelen van de child class.
- Bij het vak Datastructuren gebruiken we meestal `public` overerving

```
class Box {
  public:
    Box( int l, int w, int h );
    int getVolume( );

    int getLength( );
    int getWidth( );
    int getHeight( );

    void setLength( );
    void setWidth( );
    void setHeight( );
  private:
    int length, width, height;
};

class LabeledBox : public Box {
  public:
    LabeledBox( int l, int w, int h, char lab );
  private:
    char label;
};
```

```
class Box {
  public:
    Box( int l, int w, int h );
    int getVolume( );

    int getLength( );
    int getWidth( );
    int getHeight( );

    void setLength( );
    void setWidth( );
    void setHeight( );
  protected:
    int length, width, height;
};

class LabeledBox : public Box {
  public:
    LabeledBox( int l, int w, int h, char lab );
  private:
    char label;
};
```

Overerving

`virtual`

- Staat als keyword voor de method van de parent class
- Maakt dat bij aanroep van een functie dynamisch wordt bepaald van welke class deze wordt aangeroepen
- Virtual constructor wordt niet gebruikt
- Virtual destructor is essentieel voor het opruimen van members van de child class! Definieer daarom altijd een virtual destructor

Overerving

Een Abstract Base Class (ABC) is een klasse waarin functies niet gemplementeerd zijn

- Kan dus niet geïnstantieerd worden
- Meeste functies zijn `virtual`
- `virtual int doSometing() = 0;`
- Child classes implementeren alle functies
- Goede start van een klasse
- Geschikt voor implementeren van datastructuur wanneer meerdere implementaties vereist zijn

Const correctness

- Correctheid bij het werken met constanten heeft voordelen
  - Werkt als een stukje commentaar
  - Voorkomt misbruik via references en pointers
- Werk vanaf het begin const correct!
- Later terugwerken kost veel moeite

Const correctness

- Het verschil tussen:
  - `int const* p; // p is pointer to const int`
  - `int* const p; // p is const pointer to int`
  - `int const* const p;//p is const pointer to const int`

Const correctness

- Het verschil tussen:
  - `int const* p; // p is pointer to const int`
  - `int* const p; // p is const pointer to int`
  - `int const* const p;//p is const pointer to const int`
- Lees van rechts naar links
- In veel standaarden wordt de naam van een var die const is in hoofdletters geschreven. Niet verplicht, wel raadzaam

Const correctness

- Methods uit classes kunnen ook constant zijn! Een const method moet de class onveranderd laten.
- Een correcte getter functie:

```
MyObj const & MyClass::getMyObj() const {
    return myobj;
}
```

- En de setter variant:

```
void MyClass::setMyObj(MyObj const & myobj) {
    this->myobj = myobj;
}
```

Standard C++ library

Bestaand uit:

- C libraries: cmath, cstdlib, ctime, ...
  - cmath: M_PI, sin()
  - cstdlib: (s)rand()
  - ctime: time()
- STL libraries
  - vector, list, stack, map, deque, ...
  - Algorithm
- iostream, fstream, stringstream, ...
- Other libraries: string

Standard C++ library

- In cstdlib geeft `rand()` een integer terug in het bereik `[0, RAND_MAX]`
- Elk random getal is pseudorandom
- Reëel random getal in `[min, max)`?
  - `(double)rand() / ((double)RAND_MAX + 1.0)` geeft een getal `r` in `[0.0, 1.0)`
  - `min + r * (max - min)` is redelijk
  - deling van `RAND_MAX` door `r` is beter
- `rand()` is voldoende voor Datastructuren

Standard C++ library

```
void testStack() {
  stack<string> allwords;
  allwords.push( voorbeeld );
  cout << "Number of words = " << allwords.size() << endl;

  while (!allwords.empty()) {
    cout << allwords.top() << endl;
    allwords.pop();
  }
}
```

Templates

```
int addInteger( int const x, int const y ) {
  return x + y;
}

double addDouble( double const x, double const y ) {
  return x + y;
}
```

Templates

Templates

- Bestaan voor functies en voor classes
- Geven een of meer opties voor datatypes
  - bv `vector<int>` of `vector<float>`
- Creëren een nieuwe instantie van de class of functie tijdens het compileren, wanneer deze ergens gebruikt wordt
- Standard C++ library maakt intensief gebruik van templates

Templates

```
template<typename T>
T const add( T const &t1, T const &t2 ) {
  return t1 + t2;
}
```

- Vereiste: de gebruikte operators moeten gedefinieerd zijn voor het betreffende datatype
- Kun je ook zelf definieëren.

Samenvattend

Vereisten voor de programmeeropdrachten:

- Teams van twee personen
- Code in `C++`, moet compileren op de LIACS Linux omgeving
- Maak indien nodig gebruik van een Makefile
- Opdracht compileerd zonder errors / warnings (test altijd met `-Wall` en `-Wextra` flags aan)
- Zorg dat je functies niet te lang zijn, deel ze zo nodig op
- Goede scheiding tussen `private`/`protected`/`public` members en variabelen.
- Gebruik abstracte classes voor ADT's waar nuttig
- (Soms) optioneel: gebruik van templates

Samenvattend

Waar je verder op beoordeeld wordt:

- Datastructuur
  - Werk volgens de verstrekte ADT
- Algoritmes
- Modulariteit
- Abstractie
- Const correctness
- Commentaar
- Meegeleverde bestanden, netheid van code

Samenvattend

Bij vragen:

- `http://www.parashift.com/c++-faq/`
- Tijdens het werkcollege
- Per email naar `j.n.van.rijn@liacs.leidenuniv.nl`