Universiteit Antwerpen

Inleiding C++

Coding Conventions

Opleiding Bachelor of Science in Informatica, van de Faculteit Wetenschappen, Universiteit Antwerpen.
Nota's bij de cursus voor academiejaar 2012 - 2013.

Ruben Van den Bossche, Sam Verboven

Onderzoeksgroep Computationeel Modelleren en Programmeren

Inhoudsopgave

HOOFDSTUK 1

Codeer conventies

1.1 Makefiles

In elk project komt minstens 1 makefile voor die minimaal volgende rules bevat:

- rule om het project te compileren, te linken en de doxygen documentatie te genereren (via `make` of `make all`)
- rule om de objectfiles te verwijderen (via `make clean`)

1.2 Organisatie van bronbestanden

Elk C++ bronbestand bestaat uit de volgende onderdelen:

1. voor een headerfile, de *multiple definition guard* die meervoudige definitie door het meermaals includeren van het file verhindert
2. een *algemeen commentaarblok* in javadoc stijl dat volgende elementen bevat
   - korte beschrijving
   - @author
   - @date
   - @version
3. een *include* blok
4. de globale *using* statements

5. het *body*-gedeelte

Plaats een witregel tussen elk van deze onderdelen.

1.3 Algemeen

Gebruik altijd slechts één statement per regel! Dus niet:

```
a++;b++;
```

maar wel

```
a++;
b++;
```

Let er ook op dat je bij het aanroepen van procedures en methoden steeds de ronde haakjes plaatst, ook al geef je geen parameters mee!

1.4 Indentatie

Alle onderdelen van sectie 1.2 worden tegen de kantlijn geplaatst.

1.4.1 Accolades

Hoewel er nog meerdere stijlen bestaan voor wat betreft het plaatsen van accolades, zullen wij één van onderstaande conventies volgen.

```
1 int somefunction (int someargument) {
2   ...
3   for (i = 0; i < 100; i++) {
4     ...
5   }
6   ...
7 }
```

De openingsaccolade staat op dezelfde lijn als zijn voorganger en de sluitingsaccolade op een nieuwe regel. Een andere stijl gebruikt die conventie voor blokken in een functie- of klassedefinitie, maar zet de openingsaccolade voor functie-body's en klassedefinities op een nieuwe regel:

```
1 int somefunction (int someargument)
2 {
3   ...
4   for (i = 0; i < 100; i++) {
5     ...
6   }
7   ...
8 }
```

1.4.2 Regels afbreken

Zorg ervoor dat de regels in uw code niet te lang worden (zodanig dat je niet moet gaan *scrollen* om een regel te lezen). Dikwijls is dit echter onvermijdelijk. Breek in die situaties de regel af en indenteer hem een tabpositie meer dan de eerste regel. Neem bij het afbreken volgende regels in acht:

- breek af *na* een komma
- breek af *voor* een operator
- verkies *higher-level*-afbreken boven *lower-level* (zoals bijvoorbeeld in lange expressies

1.5 Commentaar

Op sommige plaatsen (zoals bij lange, ingewikkelde expressies) kan het nodig zijn een woordje commentaar toe te voegen. Aarzel niet dit dan ook te doen. Functies, klassen en methods worden ook voorzien van een commentaarblok in javadoc stijl met volgende gegevens:

- korte beschrijving
- @param voor elke gebruikte parameter
- @return
- @exception indien er exceptions gethrowd worden

Een voorbeeld:

```
1  /*
2   * This function adds two integers.
3   * @param a The first term of the sum
4   * @param b The second term of the sum
5   * @return The sum of a and b
6   */
7  int sum (int a, int b)
8  {
9    return a + b;
10 }
```

1.6 Naamgeving

Goede naamgeving is zeer belangrijk en dus is het aangewezen zo weinig mogelijk afkortingen te gebruiken en namen zo logisch mogelijk te kiezen (dus zorg ervoor dat ze overeenstemmen met de realiteit!).

1.6.1 Namespaces

Namespaces beginnen steeds met een hoofdletter. In geval van meerdere woorden worden deze onderscheiden door een hoofdletter.

Bijvoorbeeld:

```
namespace MyNameSpace {
  ...
}
```

1.6.2 Types, Acroniemen

Begin de naam van een een type en een acroniem steeds met een hoofdletter. In geval van meerdere woorden worden deze onderscheiden door een hoofdletter.

Bijvoorbeeld:

```
typedef int MyTypeDef                          // Type
SWLString myString ("Secret World Live!")      // Acroniem
```

1.6.3 Templateparameters

Templateparameters moeten steeds beginnen met de letter `A`. In geval van meerdere woorden worden deze onderscheiden door een hoofdletter.

Bijvoorbeeld:

```
template <typename AType>  // AType is het template argument
class FunctorTrait {
  ...
};
```

1.6.4 Enumeraties

Enumeraties beginnen steeds met de letter `E` en in geval van meerdere woorden worden deze onderscheiden door een hoofdletter.

Bijvoorbeeld:

```
enum EFreezeLevels {kFrozen=1, kThawn, kNone}
```

1.6.5 Klassen

- Een basisklasse representeert fundamentele functionele objecten en begint steeds met een hoofdletter.
- Een virtuele basisklasse begint steeds met een `V`.
- Een pure virtuele klasse of interface begint steeds met de letter `I`.
- Een klasse die sommige, maar niet alle, functionaliteit van een virtuele basisklasse implementeerd, wordt een *mixin class* genoemd. Deze klassen beginnen met de letter `M`.
- Exceptionklassen beginnen steeds met een hoofdletter en eindigen op het woord `Exception`.

- Klassen die design patterns implementeren beginnen steeds met een hoofdletter en eindigen met de naam van het design pattern.

Bijvoorbeeld

```
class BaseClass {              // Basisklasse
  ...
};

class VBaseClass {             // Virtuele basisklasse
  ...
};

class IUserInterface {         // Interface
  ...
};

class MPrintable {             // Mixin class
  ...
};

class StandardException {      // Exceptionklasse
  ...
};

class IconFactory {            // Factory pattern
  ...
};
```

1.6.6 Procedures, methodes

Procedures en methodes beginnen steeds met een kleine letter. In geval van meerdere woorden worden deze onderscheiden door een hoofdletter. Getters en setters beginnen steeds met *set...*, *get...* of *is...* (bij Booleans).

Bijvoorbeeld:

```
int entier (double x) {                  // functie
  return ((int) x);
}

class Full {
  bool stop (long int size);                                  // methode
  void setMessage (std::string& msg);                         // setter
  std::string& msg getMessage ();                             // getter
  bool isMessageSet ();                                       // boolean
};
```

1.6.7 Variabelen

Variabelen en parameters beginnen steeds met een kleine letter. Bovendien geldt:

- globale en static variabelen beginnen met de letter `g`.
- constanten beginnen met de letter `k`.
- data members beginnen met de letter `f`.
- static data members beginnen met de letters `fg`.

Bijvoorbeeld:

```
1   int gCount;                          // globale variabele
2
3   void foo () {
4     int counter;                       // lokale variabele
5     static int gCounter;               // static variabele
6     const int kMaxCount(100);          // constante variabele
7     ...
8   }
9
10  class MyClass {
11    private:
12      std::string fName;               // data member
13      static Date fgDefaultDate;       // static data member
14    ...
15  };
```

1.7 Preprocessing

1.7.1 Multiple Inclusion Guard

De multiple inclusion guard zorgt ervoor dat er geen meervoudige definities kunnen voorkomen door het meermaals includeren van eenzelfde headerfile. Gebruik steeds `INC_BESTANDSNAAMINHOOFDLETTERS_EXTENSIE`

Bijvoorbeeld:

```
1   # ifndef INC_VIDEO_H
2   # define INC_VIDEO_H
3   ...
4   # endif // INC_VIDEO_H
```

1.7.2 Conditionele compilatie

Makro's laten toe om te switchen tussen broncode varianten. Gebruik ze enkel wanneer ze echt nodig zijn. Gebruik steeds het prefix `DEF`.

Bijvoorbeeld:

```
1    # define DEF_ARCH X86
2    ...
3
4    # ifdef DEF_ARCH == 86
5    ...
6    else
7    ...
8    # endif // DEF_ARCH
```