Code Readability

OSSS - Day 2

Teodora Băluță

teobaluta@rosedu.org *teobaluta@gmail.com*

June 19, 2013

Overview

Code Readability

A set of rules and best practices

- naming

Code Readability

A set of rules and best practices

- naming
- general aspect = aesthetic

Code Readability

A set of rules and best practices

- naming
- general aspect = aesthetic
- simplify code & modularity

Code Readability

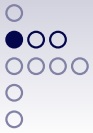
A set of rules and best practices

- naming
- general aspect = aesthetic
- simplify code & modularity
- commenting & documentation

Code Readability

A set of rules and best practices

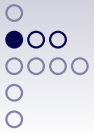
- naming
- general aspect = aesthetic
- simplify code & modularity
- commenting & documentation
- refactoring



What's in a name?

```
class BinaryTree {
  int Size();
};
```

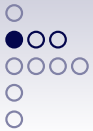
- Size() ??



What's in a name?

```
class BinaryTree {
  int Size();
};
```

- Size() ??
- could refer to the tree's height, number of nodes or the memory footprint on the disk



What's in a name?

```
class BinaryTree {
  int Size();
};
```

- Size() ??
- could refer to the tree's height, number of nodes or the memory footprint on the disk
- use specific, descriptive words

Naming - cont

Word	**Alternatives**
send	deliver, dispatch, announce, distribute, route
find	search, extract, locate, recover
start	launch, create, begin, open
make	creat, set up, build, generate, compose, add, new

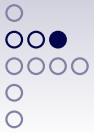
Table : Useful alternatives

Naming - cont

- avoid generic names

Naming - cont

- avoid generic names
  - for example, **tmp** și **retval**



Naming - cont

- avoid generic names
  - for example, **tmp** și **retval**
  - do we never use them?

Naming - cont

- avoid generic names
  - for example, **tmp** și **retval**
  - do we never use them?
- use a naming convention

Naming - cont

- avoid generic names
  - for example, **tmp** și **retval**
  - do we never use them?
- use a naming convention
  - lowercase + underscore: run_simple_cmd

Naming - cont

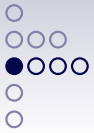
- avoid generic names
  - for example, **tmp** și **retval**
  - do we never use them?
- use a naming convention
  - lowercase + underscore: run_simple_cmd
  - CamelCase: runSimpleCmd, RunSimpleCmd()

Naming - cont

- avoid generic names
  - for example, **tmp** și **retval**
  - do we never use them?
- use a naming convention
  - lowercase + underscore: run_simple_cmd
  - CamelCase: runSimpleCmd, RunSimpleCmd()
  - Hungarian notation: arru8NumberList (not recommended now)

Naming - cont

- avoid generic names
  - for example, **tmp** și **retval**
  - do we never use them?
- use a naming convention
  - lowercase + underscore: run_simple_cmd
  - CamelCase: runSimpleCmd, RunSimpleCmd()
  - Hungarian notation: arru8NumberList (not recommended now)
- ... be consistent!



Aspect

```
class StatsKeeper {
public:
  void Add(double d);
private:  int count;
/* how many so far
*/ public:
  double Average();
private:  double minimum;
list<double>
  past_items
    ;double maximum;
};
```

```
class StatsKeeper {
  public:
    void Add(double d);
    double Average();
  private:
    list<double> past_items;
    // how many so far
    int count;

    double minimum;
    double maximum;
};
```

Aspect - cont

Indentation

- outline the flow of the code

Aspect - cont

Indentation

- outline the flow of the code
- make code easier to read

Aspect - cont

Indentation

- outline the flow of the code
- make code easier to read
- could be a part of the language (Python, Haskell)

Aspect - cont

Indentation

- outline the flow of the code
- make code easier to read
- could be a part of the language (Python, Haskell)

Aspect - cont

Indentation

- outline the flow of the code
- make code easier to read
- could be a part of the language (Python, Haskell)

Indent Styles

- K&R (see *The C Programming Language* de Kernighan & Ritchie)

Aspect - cont

Indentation

- outline the flow of the code
- make code easier to read
- could be a part of the language (Python, Haskell)

Indent Styles

- K&R (see *The C Programming Language* de Kernighan & Ritchie)
- Allman - brackets under each instruction

Aspect - cont

Indentation

- outline the flow of the code
- make code easier to read
- could be a part of the language (Python, Haskell)

Indent Styles

- K&R (see *The C Programming Language* de Kernighan & Ritchie)
- Allman - brackets under each instruction
- BSD KNF (Unix-like)

Aspect - cont

Indentation

- outline the flow of the code
- make code easier to read
- could be a part of the language (Python, Haskell)

Indent Styles

- K&R (see *The C Programming Language* de Kernighan & Ritchie)
- Allman - brackets under each instruction
- BSD KNF (Unix-like)
- ... and combinations of these!

Aspect - cont

Spacing

- readable code

Aspect - cont

Spacing

- readable code
- declaring variabiles

```
a = 3, not a=3
```

Aspect - cont

Spacing

- readable code
- declaring variabiles

  `a = 3, not a=3`
- declaring pointers

  `int* a vs int *a`

Aspect - cont

Spacing

- readable code
- declaring variabiles

  `a = 3, not a=3`
- declaring pointers

  `int* a vs int *a`

Aspect - cont

Spacing

- readable code
- declaring variabiles

  `a = 3, not a=3`

- declaring pointers

  `int* a vs int *a`

Spaces vs tabs

- tabs depend on editor, spaces same everywhere

Trailing whitespaces

Aspect - cont

Spacing

- readable code
- declaring variabiles

  `a = 3, not a=3`

- declaring pointers

  `int* a vs int *a`

Spaces vs tabs

- tabs depend on editor, spaces same everywhere

Trailing whitespaces

- commit noise

Aspect - cont

Spacing

- readable code
- declaring variabiles

  `a = 3, not a=3`

- declaring pointers

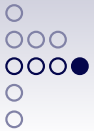
  `int* a vs int *a`

Spaces vs tabs

- tabs depend on editor, spaces same everywhere

Trailing whitespaces

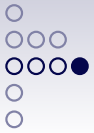
- commit noise
- positioning the cursor at the end of the line



Aspect - cont

Lines $<$ 80 characteres

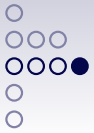
- historical reasons



Aspect - cont

Lines $<$ 80 characteres

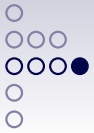
- historical reasons
- easier to read (short attention span?)



Aspect - cont

Lines $<$ 80 characteres

- historical reasons
- easier to read (short attention span?)



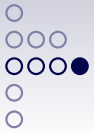
Aspect - cont

Lines $<$ 80 characteres

- historical reasons
- easier to read (short attention span?)

Column/Vertical Alignment

```
$search      = array('a',   'b',   'c',   'd',   'e');
$replacement = array('foo', 'bar', 'baz', 'quux');
```



Aspect - cont

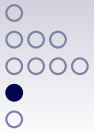
Lines $<$ 80 characteres

- historical reasons
- easier to read (short attention span?)

Column/Vertical Alignment

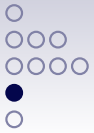
```
$search      = array('a',   'b',   'c',   'd',   'e');
$replacement = array('foo', 'bar', 'baz', 'quux');
```

Key Word: Consistency!



Functions, simplify your code

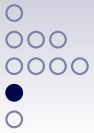
Naming functions: concrete and descriptive!



Functions, simplify your code

Naming functions: concrete and descriptive!
Short, specific, easy to follow

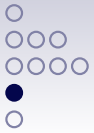
- one functionality per function



Functions, simplify your code

Naming functions: concrete and descriptive!
Short, specific, easy to follow

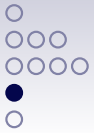
- one functionality per function
- short (aprox $<$ 40 lines)



Functions, simplify your code

Naming functions: concrete and descriptive!
Short, specific, easy to follow

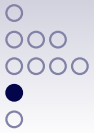
- one functionality per function
- short (aprox $<$ 40 lines)
- should be generic/reusable



Functions, simplify your code

Naming functions: concrete and descriptive!
Short, specific, easy to follow

- one functionality per function
- short (aprox $<$ 40 lines)
- should be generic/reusable

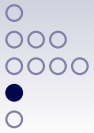


Functions, simplify your code

Naming functions: concrete and descriptive!
Short, specific, easy to follow

- one functionality per function
- short (aprox $<$ 40 lines)
- should be generic/reusable

Modularity



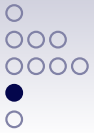
Functions, simplify your code

Naming functions: concrete and descriptive!
Short, specific, easy to follow

- one functionality per function
- short (aprox $<$ 40 lines)
- should be generic/reusable

Modularity
Simplify the code's logic

- ternary operator: time_str += (hour >= 12) ? "pm" : "am";



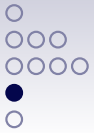
Functions, simplify your code

Naming functions: concrete and descriptive!
Short, specific, easy to follow

- one functionality per function
- short (aprox $<$ 40 lines)
- should be generic/reusable

Modularity
Simplify the code's logic

- ternary operator: time_str += (hour >= 12) ? "pm" : "am";
- while (cond) ... ; vs do ... while (cond);



Functions, simplify your code

Naming functions: concrete and descriptive!
Short, specific, easy to follow

- one functionality per function
- short (aprox $<$ 40 lines)
- should be generic/reusable

Modularity
Simplify the code's logic

- ternary operator: time_str += (hour >= 12) ? "pm" : "am";
- while (cond) ... ; vs do ... while (cond);
  - Bjarne Stroustrup: *In my experience, the do-statement is a source of errors and confusion. ... I prefer the condition "up front where I can see it." Consequently, I tend to avoid do-statements.*

Commenting & Documenting

- One language only

Commenting & Documenting

- One language only
- Describe what's not obvious

Commenting & Documenting

- One language only
- Describe what's not obvious
    - algorithms, tricks

Commenting & Documenting

- One language only
- Describe what's not obvious
  - algorithms, tricks
  - hacks

Commenting & Documenting

- One language only
- Describe what's not obvious
  - algorithms, tricks
  - hacks
  - everything you think you won't understand in 3 months time

Commenting & Documenting

- One language only
- Describe what's not obvious
  - algorithms, tricks
  - hacks
  - everything you think you won't understand in 3 months time
- Clear and corect

Commenting & Documenting

- One language only
- Describe what's not obvious
  - algorithms, tricks
  - hacks
  - everything you think you won't understand in 3 months time
- Clear and corect
- Don't exaggerate!

```
// add the two numbers
suma = a + b;
```

Why?

- We read more code than we write

Why?

- We read more code than we write
- We're usually working in a team

Why?

- We read more code than we write
- We're usually working in a team
- We want others to understand our code

Why?

- We read more code than we write
- We're usually working in a team
- We want others to understand our code
- We want to understand the other's code

Be better!

- Efficient code (but not optimized)

Be better!

- Efficient code (but not optimized)
- Macros

Be better!

- Efficient code (but not optimized)
- Macros
  - instead of inline functions

Be better!

- Efficient code (but not optimized)
- Macros
  - instead of inline functions
  - make the code easier to understand

Be better!

- Efficient code (but not optimized)
- Macros
  - instead of inline functions
  - make the code easier to understand
  - may insert bugs (they are not type safe)

Be better!

- Efficient code (but not optimized)
- Macros
  - instead of inline functions
  - make the code easier to understand
  - may insert bugs (they are not type safe)
  - inside do ... while(0)

Be better!

- Efficient code (but not optimized)
- Macros
  - instead of inline functions
  - make the code easier to understand
  - may insert bugs (they are not type safe)
  - inside do ... while(0)
    - use ";" after macro

Be better!

- Efficient code (but not optimized)
- Macros
  - instead of inline functions
  - make the code easier to understand
  - may insert bugs (they are not type safe)
  - inside do ... while(0)
    - use ";" after macro
- Include guards

Be better!

- Efficient code (but not optimized)
- Macros
  - instead of inline functions
  - make the code easier to understand
  - may insert bugs (they are not type safe)
  - inside do ... while(0)
    - use ";" after macro
- Include guards
- No typecast malloc in C

```
C:    char *str = malloc(256 * sizeof(char));
C++:  char *str = (char*) malloc(256 * sizeof(char)
```

Be better!

- Initialize variabiles cu 0, 0.0 sau NULL

Be better!

- Initialize variabiles cu 0, 0.0 sau NULL
- Test possible return values

Be better!

- Initialize variabiles cu 0, 0.0 sau NULL
- Test possible return values
  - smaller chances to have bugs

Be better!

- Initialize variabiles cu 0, 0.0 sau NULL
- Test possible return values
  - smaller chances to have bugs
- Get reviews!

Tools

Vim

- Auto-indent:

  ```
  :set cindent
  ```

- Trailing whitespaces:

  ```
  :set list
  ```

Gprof

- profiliing = analyze execution time per blocks
- -pg flags at compilation:

  ```
  gcc example1.c -pg -o example1 -O2 -lc
  ```

- C, C++, Pascal

Lint (splint)

- spot some bugs!

Links

- Google C++ Style Guides
- Linux Kernel Coding Style
- How To Write Unmaintainable Code
- *The Art of Readable Code* by Dustin Boswell, Trevor Foucher, O'Reily Media, 2011

?