

Writing the Program

Pertemuan 10

Writing good programs

› Why?

- If the writing style is bad:
 - It is easier to make a logical error
 - It is more difficult to find errors
 - It is more difficult to understand the code
 - It is more difficult to add new features
 - It is more difficult to maintain the program

What does it mean to write a good program?

› A code must be

- Well documented
- Readable
- General
- Efficient
- Simple
- Easy to update with new features
- It must handle all correct and incorrect input
- *Think: you should write your program as you had got it for maintenance from someone else*

Well documented code – module/file information

› Identify the code (your module)

- File Name, Author, Date, Summary, History

```

//-----
// <FILENAME> - <short description>
//
// Author, date, version
//
// ABSTRACT
// <description>
//
// HISTORY
// date
// change
//.....
//
//-----

```

Well documented code - comments

› In-line comments (full line comments)

› Before

- Classes
- Functions
- Sections within functions
- Complex data structures

```

//-----
// openSession
// Open a new work session. Read a tree structure for the specified session
// The default values from the recent sessions are taken
// The current directory can be overwritten by reqCurrentDir if different from ""
// Parameters:
//   Session: A top directory of the session
//   reqCurrentDir: the current directory of the session
// Return: Pointer to the session tree structure
//-----
treeStructure *openSession(String Session, String reqCurrentDir)

```

Comments

› End-of-line comments follow instructions

› Explain the code, data in the line

› Do not explain the trivial things

```

MyPocket.money = MyBank.Account; // get my money from the bank
Request_t request; // type of request
int returnCode; // return code from server
unsigned char flag; // general purpose flag
unsigned char accessMode; // used for open w,r, a
dirEntry fileNode; // file descriptor from dir
char data[BLOCK_SIZE]; // data to transfer

```

Comments

- › Use tools for automatic document generation from comments
- › Important for API definitions
- › Example: Javadoc

```
//
// TheClassName
/** A summary sequence about the class goes here
<p>
* More comments about the class...
* Use <em>Emphasize</em> or other HTML tags
*/
```

Comments - conclusion

- › When reading comments placed on the file header a reader must have a clear idea what is the module about
- › When reading comments of classes, functions, blocks, the reader must have a clue what is the purpose of the class/function without reading the code
- › The comments should be additional information to the reader
- › The comments should help reader to faster understand the code
- › Keep the comments and the code synchronized

The code must be readable

- › Use consistent naming convention

- Classes public class Car
- Functions public void eatBananas() {}
- Variables private Color myColor;

- › Begin with capitals for class names and class variables (static) and all caps for constants
- › Begin everything other than class names and variables with a lower case letter.
- › Combine several words with starting capital letter
- › Use names that are understandable (isOpen instead of opn)
- › Use short names for temporary variables (for (int i=1; i<noDevices; i++)

Naming convention - example

```
class Circle
{
enum Colors {Red, Blue, Green};
. . .
};
struct Point {int x, int y};
typedef long TimeUnit;
const double PI = 3.1415926535897932384626;
const int result = DoCalculation();
int noOfLines;
Symbol FindSymbol(String name);
```

The code must be readable

- › **Important**
 - Not too much information at the same time
 - › Logical blocks (function, {...}) should not be larger than 1 page
 - Inputs and outputs to blocks should be visible and not ambiguous (avoid return in if blocks)
 - Use empty lines to distinguish logical parts (similar to paragraphs in documents)
 - Use consistent style with indentation
- ```
If (isPrinted(mypaper))
{
 sendMessage (paperOK);
}
```

## Consistency

- › Use the same principles through the program
  - Naming convention
  - Passing parameters, return values
  - Processing non-valid inputs
  - Error management in the same way
  - Use the same styles in loops,
  - Etc.
  - NEVER EVER USE THE SAME VARIABLE OR STRUCTURE ELEMENT FOR DIFFERENT PURPOSES

## The code must be general enough

- › **Think the most appropriate usage of the code**
  - (not what is easiest to write at the moment)
- › **The code must work for different environment requirements**
  - Example: It should work in any directory, not just in your home Directory
  - NO HARCODED NAMES and NUMBERS
  - Use predefined constants
- › **The program should work well for any amount of data**
  - Be careful with predefine arrays, buffer sizes
  - Input values, predefined names, Filenames, etc.
- › **NO DUPLICATION OF ALLMOST THE SAME CODE**

## Simplicity

- › **If you:**
  - Have difficulties to explain what the code actually does
  - Must always think to remember what is the code about
  - You cannot see directly the inputs and outputs
  - You have many temporary variables
- › **your code is too complicated**
- › **How to make code simpler**
  - consider the design
  - Analyse input/output
  - Divide functions in several sub-functions

## Processing Input/outputs

- › **Task;**
- › **Write a function that counts the number of equal sides of a Triangle**
  - Input: the three sides of a triangle
  - Output: return the number of sides of equal sizes
  - If all three sides are different return 1

## Input variables handling

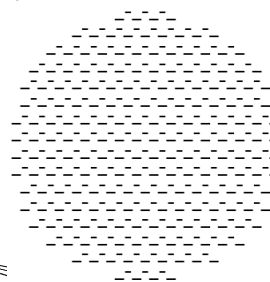
- › **Check the correctness of input variables**
  - Especially when a user enters values
  - What happens when data is not correct?
- › **Error handling**
  - Use the same way of handling errors
  - Exception handling
- › **Internal functions return error code, they do not display the error message**

## Good/bad code is direct visible

- › **A short look at a program code can give you a**
  - Good feeling
  - Bad feeling
- › **A good, nice program is like a nice story, or a nice picture**
- › **Good programmers are proud of their programs**
- › **Read your code!**
- › **Give your code to a friend to read it!**

## A nice example of a code

```
#define _ F-->00 || F-00--;
long F=00,00=00;
main(){F_00();printf("%i.3f\n", 4.*-F/00/00);}F_00()
{
```



What is the program doing?