Coding_Standard_u4

Naming

Variables

1. Variable names start with a **non-capital** letter.
2. The variable names should convey meaning and be as short as possible (not more than 2 words).
3. If the variable name is made up of two words, the second word should start with a capital letter
4. No underscores should be used in the middle of the variable name
5. No numbers should be used.
6. Single letter variables are allowed if used only in cycles as counters.
7. Multiple variables of the same type can be declared (but not initialized) in the same line

```
int counter = 0;

char flagCounted = 0;

int i;

for(i = 0; i < something; ++i)
    ;
```

Functions

1. Function names should start with a **non-capital** letter
2. Function names have to convey meaning. There is no limit to how long the name should be
3. Every word starting from the second should start with a capital letter
4. No underscores should be used
5. No numbers should be used
6. A boolean function should start with a prefix 'is'
7. A function name has to be as short as possible without loosing meaning
8. Ideally a function name is a *verb*

```
int count(int);

char isFlagOn(void);

int countPolynomialOrder(struct Poly);
```

Structures

1. Structure names should start **with a capital letter**
2. All structures should be type-defined[1] with the same name
3. A structure name should be no longer than two words
4. No underscores should be used
5. No numbers should be used
6. If present, the second word should start with a capital letter

```
typedef struct {
    int something;
    char flag;
} List;
```

Pointers

1. The star should be attached to the front of the variable's name
2. If there is no variable name, the star should be space separated from the type
3. All other variable conventions apply

```
int *input, **output;

void a(int *);
```

Declaration, Definition & Initialization

Variables

1. Variables should be initialized during declaration **when possible**
2. First rule does not apply to variables whose position of declaration is constricted by the c89 standard
3. A separate block should be devoted for user input which does not require initialization

```
/* User input */
int age, level;
/* END of User input */
```

Functions

1. All function declarations have to reside in a separate header file
2. If `main` function is defined in a file, it should be the first function in that file
3. All *testable* function definitions except for `main` have to reside in a separate library[2]
4. Functions, whose return type is `void` should always have a `return` sentence at the end

Pointers

1. Pointers should **always be initialized**. Default is the `NULL` value
2. All other variable conventions apply

```
/* User input */
int age, level;
char *name = NULL;
char *surname = NULL;
/* END of User input */
```

Structure

1. All structures should be defined in a header file

Global variables

1. Global variables should not be used

Formatting

Control structures

1. Any control structure must take up at least 2 lines
2. Left indentation of ~3 whitespace characters should be used
3. Opening brace should be inline with the definition of the control structure
4. Closing brace should be aligned with the start of the definition of the control structure
5. A control structure has to be braced[3] if the body of it is more than 1 sentence
6. A nested control structure requires the parent structure to be braced

```
if(condition) {
    ;
    ;
}

if(contidion)
    ;

if(condition) {
    while(!condition) {
        for(; contidion; )
            ;
    }
}
```

Semicolon & Comma Spacing

1. There is always a space after a semicolon, unless the semicolon is the last character in the line
2. The first rule applies to the comma operation as well

```
int array[] = {1, 2, 3, };

for(; ; )
    ;
```

Operator Formatting

1. Unary operators should not be spaced
2. Binary operators should be single-spaced from both sides
3. If more than one binary operation is being used in a sentence, the priorities should be made obvious using parentheses.

4. Objects connected with structure operators "." and/or "->" should be considered as one object. The "." and "->" operators themselves should not be spaced from either side, nor should they be in parentheses.

Testing

Unit Tests

1. Every function defined in the header file must have at least 2 key tests in the same test suite
2. Testing should be done using the Google Test framework
3. CMake should be used for building
4. A separate Makefile can reside in the `src` directory if needed

Miscellaneous

Labels & GOTO

1. The use of labels and goto statements is only allowed when validating user input

Includes & Macros

1. All includes and macros should be defined at the top of the file

---

1. a `typedef` sentence should be used ↩
2. This is to alleviate the unit testing process↩
3. A braced control structure has it's body wrapped in `{}` (curly braces)↩