Programming Style Guide

This guide is a compilation of helpful recommendations to increase the quality of your code. These recommendations are related to the structure, format and naming conventions that are widely used in C programs. While most of them do not result in wrong programs, they will be taken into account in the course evaluation.

General recommendations

Indentation

Use **ALWAYS** code indentation. If your IDE does not support automatic indentation, do manual indentation. Use a tab size between 3 and 5 spaces.

`main` function

The `main` function must have `int` returning type. Do not forget to include a `return` at the end. If the `main` function does not use arguments, it must explicitly declare them as `void`.

```
int main(void) {
   // …
   return 0;
}
```

Comments

Include comments in your programs when the code is not self-explicative. If possible, do not comment single lines but complete sections of the program.

Brackets

Use TBS variant of K&R indent style. E.g.:

```
int main(void) {
   int a = 5;
   if(a > 10) {
      printf("Larger");
   } else {
```

```
        printf("Smaller");
    }
    return 0;
}
```

Variables and constants

Labels and identifiers

Identifiers must be consistent to the use of variables and constants. However, do not use very long names.

C is a case-sensitive language, which means that uppercase and lowercase letters are different. For variables, use CamelCase notation with first lowercase letter. For constants, use uppercase letters and underscores. E.g.:

```
float studentGrade;
#define MAX_SIZE 100
```

Scope

Variables must be declared at the beginning of their scope; i.e., at the beginning of the block in which they are used. In short functions, you must declare all the variables at the beginning of the function, even though it includes nested blocks. This is particularly convenient in the `main` function when it has less than 80-100 lines.

Global variables are completely forbidden.

Expressions

Use parentheses when the evaluation of the expression is not clear from the precedence rules of the language.

Loops

break, continue, goto

Do not use `break` inside loop instructions (`break` is only allowed inside a `switch`). Use `continue` only in those cases that it is absolutely the best choice.

***goto* is completely forbidden.**

indexes and counter variables

Do not end a loop --and in particular, a `for` loop-- by modifying the index/counter variable when other condition holds. Use *flag* variables instead.

For example:

```
/* Read until 0 is introduced */
// BAD
int c, i = 0;
while(i < 10) {
    scanf("%i", &c);
    if(c==0) {
        i=10;
    }
}
// OK
int c, i = 0, end = 0;
while(i < 10 && !end) {
    scanf("%i", &c);
    if(c==0) {
        end = 1;
    }
}
```

Functions

Arrays as function parameters

Functions with one-dimension arrays as parameters must use the array size as a parameter of the function. Inside the function, this parameter will be used to process all the elements of the array.

In the declaration of the formal parameters, the size of the array will not be specified along with the array name and type.

```
// OK
float sum(float A[], int n);
...
float sum(float A[], int n) {
   int i;
   for(i=0; i<n; i++)
   ...
}
```

```
// BAD
float sum(float A[MAX_ELEMENTS]);
...
float sum(float A[MAX_ELEMENTS]) {
   int i;
   for(i=0; i<MAX_ELEMENTS; i++)
   ...
}
```

Functions with two- (or more) dimension arrays as parameters must include the size of all dimensions in the declaration of the formal parameters.

Structs as function parameters

When passing structs to a function by reference, the arrow notation (`->`) is preferred over the indirection operator (`*`).