

C Coding Guidelines

Variables

Local: `{name_of_var}`
Global: `g_{detailed_name_of_var}`

```
int i;  
extern int g_alien_cnt;
```

Functions

Names

`{libLabel}_{object}[_]{action_or_verb}` or `{libLabel}_{action_or_verb}_{object}`

```
sfSyl_welcome_txt_print ()  
sfSyl_print_welcome_txt ()
```

Definitions

```
[attributes ]{type}  
{function_name}( {args} )  
{  
    /.../  
}
```

```
noreturn void  
usage( int status )  
{  
    /*...*/  
    exit( status );  
}
```

- Rationale: easier to `grep ("^func_name") ##` Pointers `{type} *{var}`

```
int *var1, *var2;    /* 2 pointers */  
int *var1, var2;     /* 1 pointer, 1 int ! */
```

Typedefs

```
[libLabel_] {name}_t  
  
superint_t i;  
sfs_superint_t j;
```

Structures

```
{name_of_struct}_s, {name_of_struct_var}  
datstruct_s this_is_a_struct;
```

Enums

```
{name_of_enum}_e, {ENUM_CONST}, {name_of_enum_var}  
enum mood_e { TAKE_IT, GIVE_IT, KEEP_IT } my_mood;
```

Gotos

```
[GT_]{ThisPart} or [GT_]{this_part}  
EmergencyClosure:  
GT_EmergencyClosure:  
GT_emergency_closure:
```

Define

```
[TYPE_]{NAME_OF_DEF}  
#define ALIENS_ON_PLANET_CNT    1234
```

Macro (DEFINES)

```
[M_]{OBJECT}_{VERB} or [M_]{VERB}_{OBJECT}  
#define ALIENS_ON_PLANET_LOCATE ()  
#define M_ALIENS_ON_PLANET_LOCATE ()  
#define LOCATE_ALIENS_ON_PLANET ()
```

Macro (header guards)

```
{NAME_OF_HEADER}_H  
MY_COOL_LIB_H
```

- Rationale: _ and __-starting header guards are used by standard library headers ## Parenthesis / braces {func}({args});

```
printf( "spaces btwn args and parenthesis : %d", true_dat );
```

```

{statement} ({condition}) {
/.../
}

if (true_dat == 1) {
    /*...*/
} else {
    /*...*/
}

```

Code example

```

#ifndef THAT_GUARD_THOUGH_H
#define THAT_GUARD_THOUGH_H

#include "myheader.h"

#include <header1.h>
#include <header2.h>

#define STR_SIZE_OF_PLANET  "BIG"

noreturn void
f_datFunc( void )
{
    unsigned int aliens_cnt = 100;
    int happn = 0;

    printf( "This planet is %s.\n", STR_SIZE_OF_PLANET );
    if (aliens_cnt > 50) {
        puts( "it's happening" );
        happn = 1;
    } else if (aliens_cnt > 0) {
        puts( "we still have time" );
        happn = 0;
    } else { puts( "ERROR" ); goto GT_Habbening; }

    switch (happn) {
    case 0:
        return( EXIT_SUCCESS );
    default:
    GT_Habbening:
        return( EXIT_FAILURE );
    }
}

```

```
#endif      /* undef _THAT_GUARD_THOUGH_ */
```

General advices

- *snake_case*: easier to type, harder to read
 - Though: some of the best ever written softwares were made in *snake_case*
- *camelCase*: harder to type, easier to read
 - Microsoft uses it...
- Dividing the code in functions increase its comprehension and readability.
- Code must not be generic, but very specific to what exactly you're doing.
- Code for debug purpose must be removed from the final form of the code.
- Always use header guards in header files.
- Put braces even on one-line statements.

Linux kernel coding style

GNU `indent` now has an option to format your code according to the Linux kernel coding style: `indent -linux [file]`. Doc: <https://www.kernel.org/doc/html/v4.10/process/coding-style.html>

References/resources

- *Notes on Programming in C*, Rob Pike
- *C Header File Guidelines*, David Kieras, University of Michigan