IRAM-NNNNN

Revision: 0
2005-05-29

Contact Author

Institut de RadioAstronomie Millimétrique

C Coding Standards at IRAM

Owner Alain Perrigouard (perrigou@iram.fr)

Keywords: C coding standards

Approved by:	Date:	Signature:
A.Perrigouard	May 2004	

Change Record

REVISION	DATE	AUTHOR	SECTION/PAGE AFFECTED	REMARKS

Content

1 Preamble

This document is based on several documents which can be found on the web:
Alma-C-Coding-Standards.pdf
available at http://www.alma.nrao.edu/development/computing/docs/joint/0010/2001-08010.pdf
and in /computer/doc/coding
gnu-coding-standard.pdf
linux-kernel-coding-style.txt

The first document available in pdf, written by Alan Bridger, Mick Brooks and Jim Pisano, will be used as the reference document. In the following sections only the standards and guidelines of the reference document will be either commented or emphasized.
As mentioned in the reference document introduction it is clear also that our recommended standards must be followed and one expects that everyone in the computer group will adhere to our guidelines.

2 Naming conventions

One follows the conventions of the reference document. One emphasizes on the following extensions;

2.1 Function names

Guidelines
The names should be a concatenation of words with in principle one word to indicate the action (verb) plus others for the objects (names). Each word is capitalized (all cases lower except the first one which should be uppercase) but the 1st word is all lowercase.
Ex : setNextSubscanSlewAzimuth

2.2 Variable names

Guidelines
For the local variables, the convention follows the function name convention.
For the global variables, the only difference is each word (including the first one) should be capitalized.
The pointers are identified with the extension _p.
Ex : struct observation_s * observation_p ;

2.3 Other names

Guidelines
Some declared type should be clearly identified:
A structure tag will be identified with the extension _s.
Ex: `struct observation_s {int mode; double timeSubscanStart;} ;`
However it is suggested to use only typedef struct.

A new type introduced with typedef has an extension _t and a capital letter for the first character of its name in a way similar to a new class declaration in C++.
Ex: `typedef struct {double MJD; long actualAz;} AntennaTraceFast_t;`
Ex: `typedef char CANByte_t;`

Constant names should be in all uppercase. Undercore "_" will be used to separate the words composing the name.

Ex: `#define PI 3.14`

Ex: `enum traceFlagEnum {TF_ON, TF_REF};`

Again here for enum it is suggested to use typedef enum and to use this new type for the variables which are defined only for the enum named constants:

Ex: `typedef enum {TF_ON, TF_REF} TraceFlag_t;`

3 Functions

See reference document.
ANSI C convention should be used.
Ex:

```
int driveSync(int axes)
   {
   ..........
   return i ;
   }
```

A function should have always a type which can be `void` in the case no return value is expected.

4 Statements and Control Flow

In a switch statement it is suggested to break each case.
However in the case it is desirable to fall through the next case it is mandatory to replace the line " `break;` " with the comment " `/* fall-through */` ".

5 Comments

See reference document.

6 Readability and indentation

Some rules already in the reference document are emphasized here.

Use blanks around all binary operators "+", "-", "*"…. except "." and "->".
Use blank between if, switch, do, while or for and the following expression inside parentheses.
Do not use blanks between function name and arguments inside parentheses.
Do not use blanks between "[" or "(" and an identifier or between an identifier and "]" or ")".
Ex:

```
function(arg1, arg2);
aVar = bVar * (sqrt(cVar) + sin(d[i]);
printf("%d\n", (a + b);
if (I == j)
```

No line should exceed 80 characters. Longer line should continue on next line with proper alignment.
Examples from the reference document:

```
a = (b + c) *
    (c + d);
if ((a == b) &&
    (c == d))
```

indent should be 4 characters.
See examples in the reference document (if, else if , switch, while, for and do while).

```
if (.......)
    {
    ......;
    ......;
    }

switch (......)
    {
    case ......:
        ......;
        ......;
        break;

    .
    .

    default ......:
        ......;
        ......;
        break;
```

Exception for typedef struct:

```
typedef struct
{
    ...... ......;
    ...... ......;
} ......_t;
```