Norme C

Version du 19 septembre 2003

Assistants C/Unix

Cette norme a pour but d'uniformiser l'écriture des programmes au sein d'EPITA.

Table des matières

1 Nomination des objets

- Les noms de variables, fonctions, macros, types, fichiers ou répertoires doivent être explicites et en anglais.
- Les abréviations sont *tolérées* dans la mesure où elles permettent de réduire significativement la taille du nom sans en perdre le sens. Les parties des noms composites seront séparées par '_'.

```
#define MAX_LINE_SIZE           256
#define COMMENT_CHAR            '#'
#define MAX_ANIM_NAME_LENGTH    64
#define MAX_FILE_NAME_LENGTH    1024
```

- Les noms de variables, de fonctions, fichiers et de répertoires doivent être décrits par l'expression régulière suivante:

```
[a-z][a-z.0-9_]*
```

- Les noms des variables de macros doivent être écrits en majuscules. Leurs arguments ont des noms comme 'Ceci', pas comme 'CELA', et pas plus 'ainsi'.

```
#define XFREE(Var)      \
  do                    \
    {                   \
      if (Var)          \
        free(Var);      \
    }                   \
while (0)
```

  avec les '\' bien alignés par des tabulations (avec M-i dans Emacs).
- Les macros ne doivent pas servir à contourner la norme.
- Dans le cas de la déclaration d'une structure le nom commencera par 's_', par 't_' pour un alias de type ('typedef'), par 'u_' pour une union et 'e_' pour une énumération.

```
typedef unsigned char   t_cmap[CMAP_NBR_COL * CMAP_SIZE];
typedef unsigned char   t_pixel;

struct          s_picture
{
  int           width;
  int           height;
  t_cmap        cmap;
  t_pixel       *picture;
};
```

- Les noms de globales (si elles sont autorisées) commenceront par 'gl_'.

2 Présentation globale

- L'indentation sera celle obtenue avec Emacs (avec la configuration dans '`/u/a1/.env/.Emacs`')

```
int     example_align(char *str)
{
  int   i;

  for (i = 0; i < 4; ++i)
    if (i > 2)
      return 1;
  return 0;
}
```

- On passe toujours à la ligne après '`{`', '`}`' ou une structure de contrôle. On indente une première fois pour les accolades, puis une seconde pour leur contenu.

```
if (cp) return cp;            => Incorrect

if (cp) {return cp;}          => Incorrect

if (cp)                         => Correct
{
  return cp;
}

if (cp)                         => Correct
  {
    return cp;
  }
```

- On sautera une ligne entre la déclaration de variable et les instructions.

```
int     example_foo(char c)
{
  int   i;

  for (i = 0; i < 42; ++i)
    ;

  return i;
}
```

- Une ligne ne doit pas excéder 79 caractères.
- On fera attention à avoir des fonctions courtes et claires, de 25 lignes maximum (entre les 2 accolades de la fonction, les lignes vides comptent).
- Il vous est interdit d'avoir plus de 10 fonctions par fichier dont 5 maximum peuvent être exportées (non déclarées *static*).
- Il ne doit pas y avoir de commentaires dans les fonctions. S'il y a besoin de commentaires ils doivent se trouver avant la fonction et être rédigés en anglais.

```
/*
** this function is useless
*/
void    example_foo(void)
{
  return;
}
```

- Les commentaires sont commencés et terminés par une ligne seule. Toutes les lignes intermédiaires s'alignent sur elle, et commencent par '**'.

```
/*
** Comment      => Correct
*/

/*
 * Comment      => Incorrect
 */
```

- La déclaration d'arguments se fera dans le style ISO/ANSI C.

```
void    example_func(struct s_list *p1, struct s_tree *p2)
{
}
```

  et si cela dépasse les 79 colonnes:

```
void    example_func(struct s_list             *p1,
                     struct s_tree             *p2,
                     struct s_double_list      *p3)
{
}
```

  en respectant l'alignement par tabulations (avec M-i dans Emacs).

- Cas des fonctions ne prenant pas d'argument:

```
void            example_foo()           => Incorrect
{
}

void            example_foo(void)       => Correct
{
}
```

3 Présentation locale

- Un espace derrière la ‘,’ et derrière le ‘;’, sauf en fin de ligne:

```
void            example_foo(struct s_acu *acu)
{
  int           i;

  for (i = 0; i < 255; ++i)
    {
      int       j;

      j = i + 2;
      example_sum(acu, j);
    }
}
```

- Il n’y a pas d’espace entre le nom d’une fonction et la parenthèse. Cependant, il faut toujours un espace entre un mot clé C (avec argument) et la parenthèse.

```
str = xmalloc(sizeof (char) * 256);
exit(0);
```

  Ceci concerne ‘`for`’ *etc.* mais aussi ‘`sizeof`’.

- Le mot-clé ‘`return`’ ne sera pas suivi de parenthèses.

```
return 1;
return str;
return;
```

  Pour les autres mots-clefs sans argument, il ne faut pas d’espace avant le point-virgule:

```
break;
continue;
...
```

- Une fonction doit avoir au plus 4 arguments.
- Le symbole de pointeur ‘`*`’ porte toujours sur la variable (ou fonction), et jamais sur le type:

```
char    *cp;                    => Correct
char*   cp;                     => Incorrect
```

- On respectera un alignement par fonction des déclarations avec le nom de la fonction, obtenu avec des tabulations (M-i avec Emacs).

```
char c; struct s_toto *example;
```

- Lors de la déclaration les variables peuvent être initialisées ; à cet effet les appels de fonctions et les expressions composées sont interdites. En outre, une seule déclaration par ligne est autorisée.

```
int             titi = strlen("toto");  => Incorrect
char            c = (str++, *str);      => Incorrect
unsigned int    i, j;                   => Incorrect
unsigned int    *toto = &titi;          => Correct
unsigned int    foo = 1;                => Correct
static int      bar = -1;               => Correct
unsigned int    baz;                    => Correct
```

- Affectations : il doit y avoir un espace entre la variable gauche et le signe d’affectation, de même pour la variable à droite.

```
var1 = var2
var1 += var2
var1 *= var2
```

- Opérateurs unaires. Pas d'espace.

```
*cp
&cp
-n
```

- Opérateurs bi/ternaires. Tous les opérateurs binaires et ternaires sont séparés des arguments par un espace de part et d'autre.

```
if (a % 10)
  return a;
return str ? str : DEFAULT_STRING;
```

- Les '`#if`' et '`#ifdef`' indentent d'un caractère les directives cpp qui suivent. Les '`#else`' et '`#endif`' marquent les conditions dont ils sont issus.

```
#ifndef DEV_BSIZE
# ifdef BSIZE
#  define DEV_BSIZE BSIZE
# else /* !BSIZE */
#  define DEV_BSIZE 4096
# endif /* !BSIZE */
#endif /* !DEV_BSIZE */
```

4 Les headers

- La norme dans son intégralité est valide pour les headers.
- On les protégera d'une double inclusion. Si le fichier est '`foo.h`', la macro témoin est '`FOO_H_`'.

```
#ifndef FOO_H_
# define FOO_H_
/*
** Content of foo.h
*/
#endif /* !FOO_H_ */
```

- Les fonctions et variables non exportées sont dites déclarées *static*.
- Toutes les fonctions exportées par '`foo.c`' doivent être correctement prototypées dans un entête inclus par '`foo.c`'. De même, les variables globales exportées par '`foo.c`' doivent être déclarées '`extern`' dans un entête inclus par '`foo.c`'.
- Les headers doivent inclure les autres headers dont ils sont directement dépendants, et aucun autre.

5 Les fichiers

- Les fichiers doivent avoir l'entête ci-dessous (obtenu à Epita avec C-c C-h sous Emacs).

```
/*
** <filename> for <project> in <location>
**
** Made by <author>
** Login   <login name@site>
**
** Started on  <date> <author>
```

- Les fichiers ne doivent pas contenir de caractère CR (ASCII 13) en fin de ligne.

6 Interdictions

- switch (sauf les switches sur enum).
- goto
- le cast
- les `typedef` sur `struct` et `union`.

7 Recommandations

- Il est fortement recommandé de préfixer les noms des fonctions par le nom du fichier dans lequel elles sont définies.
- Certaines fonctions de la bibliothèque C peuvent retourner une valeur particulière lorsqu'elles échouent. Le code principal ne doit pas être pollué par le traitement de ces valeurs exceptionnelles, aussi faut-il introduire des versions de ces fonctions qui traitent les erreurs elles-mêmes. Ces fonctions doivent porter le nom de la fonction de la lib C standard, préfixé par "x".

```
void    *xmalloc(size_t n)
{
  void  *p;

  p = malloc(n);
  if (NULL == p)
    exit(1);

  return p;
}
```