Bonnes pratiques

Informatique et Science du Numérique

Bonnes pratiques

Informatique et Science du Numérique

Codage

- Indenter le code
- Limiter la portée des variables
- Traiter les erreurs en premier
- Utiliser le switch au delà de 3 tests en cascades
- Ne jamais utiliser l'instruction continue
- Utiliser des constantes pour les nombres
- Ne pas écrire du code redondant
- Ne pas écrire des fonctions de plus de 100 lignes
- Utiliser les commentaires (pertinents !)
- Respecter les règles d'écriture

Bonnes pratiques

Informatique et Science du Numérique

Codage :: Indentation du code

```
int eau  = stock;
int soda = stock;
int choix;

do {
printf("eau : %d, soda : %d\n", eau, soda);
printf("choix boisson (1 : eau, 2 : soda) :");
scanf("%d", &choix);

if ( choix == 1 ) {
if ( eau ) {
eau--;
printf("prendre bouteille eau\n");
}
}
else
if ( choix == 2 )
if ( soda ) {
soda--;
printf("prendre bouteille soda\n");
}
} while ( eau || soda );
```

```
int eau  = stock;
int soda = stock;
int choix;

do {
   printf("eau : %d, soda : %d\n", eau, soda);
   printf("choix boisson (1 : eau, 2 : soda) :");
   scanf("%d", &choix);

   if ( choix == 1 ) {
      if ( eau ) {
         eau--;
         printf("prendre bouteille eau\n");
         }
    }
   else
      if ( choix == 2 )
         if ( soda ) {
            soda--;
            printf("prendre bouteille soda\n");
            }
} while ( eau || soda );
```

Bonnes pratiques

Informatique et Science du Numérique

Codage :: portée des variables

```
#define max_size    6000

unsigned int liste[max_size];
unsigned int i, j;
unsigned char permut;

static void bubble_sort(unsigned int liste[])
{
    j = max_size;

   do {
      permut = false;

      for (i = 0; i < j - 1; i++)
         if ( liste[i] > liste[i+1])
            permut = swap(&liste[i], &liste[i+1]);
      j--;
   } while ( j > 0 && permut );
}
```

```
#define max_size    6000

static void bubble_sort(unsigned int liste[])
{
   unsigned int i, j = max_size;
   unsigned char permut;

   do {
      permut = false;

      for (i = 0; i < j - 1; i++)
         if ( liste[i] > liste[i+1])
            permut = swap(&liste[i], &liste[i+1]);
      j--;
   } while ( j > 0 && permut );
}
```

Bonnes pratiques

Informatique et Science du Numérique

Codage :: traitement des erreurs

```
printf("Enter frame (10 bits): ");
scanf("%s" , data);

// parcours de la trame
for (i = 0 ; i < 10 ; i++) {
   // vérification des bits
   if ( data[i] != '0'&& data[i] != '1' ) {
      error = true ;

      if ( i == 0 )
         printf("start bit error\n");
      else if ( i == 9 )
         printf("stop bit error\n");
      else
         printf("frame bit error\n");
      }
   else...

   if ( error )
      return -1;
   }
```

```
printf("Enter frame (10 bits): ");
scanf("%s" , data);

// vérification bit de start
if ( data[0] != '0' ) {
   printf("start bit error\n");
   return -1;
   }

// vérification bit de stop
if ( data[9] != '1' ) {
   printf("stop bit error\n");
   return -1;
   }

// parcours de la trame
for (i = 1 ; i < 9 ; i++) {
   // vérification des bits
   if ( data[i] != '0'&& data[i] != '1' )
      //...
   else...
   }
```

Bonnes pratiques

Informatique et Science du Numérique

Codage :: if else en cascade

```
// parcours de la trame
for (i= 0 ; i < 10 ; i++) {
   // vérification des bits
   if ( data[i] != '0'&& data[i] != '1' ) {
      error = true ;

      if ( i == 0 )
         printf("start bit error\n");
      else if ( i == 9 )
         printf("stop bit error\n");
      else
         printf("frame bit error\n");
      }
   else...
   }
```

```
// parcours de la trame
for (i= 0 ; i < 10 ; i++) {
   // vérification des bits
   if ( data[i] != '0'&& data[i] != '1' ) {
      error = true ;

      switch (i) {
         case 0 :
            printf("start bit error\n");
            break;
         case 9 :
            printf("stop bit error\n");
            break;
         default :
            printf("frame bit error\n");
            break;
         }

   else...
   }
```

Bonnes pratiques

Informatique et Science du Numérique

Codage :: utilisation de constantes

```
static char *caesar(char *src, char *dst, const int size)
{
   // transformation majuscules
   majuscules(src);

   // transformation lettres -> chiffres
   int num, i = 0;
   while ( src[i] && i < size ) {
      num = src[i] - 'A';    // transformation lettres -> chiffres
      num = modulo(num + 2, 26);   // chiffrement
      dst[i++] = 'A' + num;
      }

   return dst;
}
```

```
const int mod = 26;  // nombre de lettres de l'alphabet
const int offset = 2;  // décalage

static char *caesar(char *src, char *dst, const int size)
{
   // transformation majuscules
   majuscules(src);

   // transformation lettres -> chiffres
   int num, i = 0;
   while ( src[i] && i < size ) {
      num = src[i] - 'A';    // transformation lettres -> chiffres
      num = modulo(num + offset, mod);   // chiffrement
      dst[i++] = 'A' + num;
      }

   return dst;
}
```

Bonnes pratiques

Informatique et Science du Numérique

Codage :: commentaires pertinents

```
static const unsigned int byte = 8;

static char *bitmap(char *buffer, const unsigned char c)
{
   // fonction bitmap
   unsigned int i;

   // on boucle
   for (i = 0; i < byte; i++)
      // on transforme la valeur
      buffer[i] = c & (unsigned int) pow(2, byte - i - 1) ? '1' : '0' ;
   buffer[byte] = 0;    // dernière valeur à 0

   // retourne la valeur
   return buffer;
}
```

```
static const unsigned int byte = 8;

static char *bitmap(char *buffer, const unsigned char c)
{
   // transforme un caractère ASCII en binaire
   // in : chaîne binaire vide, caractère ASCII
   // out: chaîne binaire de la conversion
   unsigned int i;

   // on boucle sur la taille d'un octet (8 bits)
   for (i = 0; i < byte; i++)
      // ET logique entre la caractère ASCII et une valeur binaire
      // valeur binaire = 2^(i-1) soit 2^[0 ; 7]
      // le masque doit retourner '1' ou '0'
      buffer[i] = c & (unsigned int) pow(2, byte - i - 1) ? '1' : '0' ;
   buffer[byte] = 0;    // une chaîne se termine par 0

   return buffer;
}
```

Bonnes pratiques

Informatique et Science du Numérique

Approche

- Lire du code
- Ré utiliser du code
- Relire son code
- Optimiser son code
- Évaluer son code

Bonnes pratiques

Informatique et Science du Numérique

Méthodes de conception

- Descendante
  Programmation procédurale
  Analyse cartésienne (fonctions)
  Langage C, Pascal, …

- Ascendante
  Programmation Orientée Objet
  Analyse systémique (relations)
  Langage C++, PHP, Python, ...

Bonnes pratiques

Informatique et Science du Numérique

Méthodes de conception :: descendante

myDataLogger

écriture myDataLogger	17/01/15	17/02/15
conception protocoles	17/01/15	17/01/15
conception GUI	18/01/15	18/01/15
onglet LED	23/01/15	24/01/15
onglet LCD	25/01/15	25/01/15
test 1	27/01/15	27/01/15
onglet courbes	30/01/15	01/02/15
test 2	03/02/15	03/02/15
lecture série filaire	06/02/15	07/02/15
enregistrement csv	08/02/15	08/02/15
test 3 : recette	10/02/15	10/02/15
mise en conformité	13/02/15	14/02/15
bilan	17/02/15	17/02/15
lecture série sans fil	17/02/15	17/02/15