Cet index a pour but de répertorier tous les exemples de sources C disponibles sur **www.developpez.com**.
Les sources proposées ici sont principalement celles des membres de Developpez.com. Si vous trouvez une erreur ou si vous avez des remarques à effectuer sur certaines de ces sources, merci de contacter l'auteur de la source s'il dispose d'un domaine sur Developpez.com ou de poster dans le **forum C**.
Si vous possédez une série de sources et souhaitez les faire apparaître dans cette liste, contactez le **responsable** après avoir pris connaissance du descriptif dans **ce thread**.
L'**équipe C** de Developpez.

Ont contribué à cette FAQ :

Anomaly (home) (Blog) - Romuald Perrot (home) - Nicolas Joseph (home) (Blog) - AjJi - lefort - Emmanuel Delahaye (home) - Trap D - beyo - Jean Christophe Beyler (home) - Franck.H (home) - Foobar1329 - D[r]eadLock - haypo (home) - Fatalis - leneuf22 - Musaran - Vincent PETIT (home) - rolkA - gl - Jean-Marc.Bourguet - souviron34 - odsen.s - Helmstetter Bernard - Melem - troumad (home) - padugas - Dark_Ebola - Skyrunner (home) - Mr_Chut - vicenzo - forthx - Bornerdogge - cledesol - diogene - mabu -

Sommaire > Algorithmes

Déterminer la parité d'un char

Auteurs : leneuf22 ,

La parité consiste à calculer le nombre de bits d'une variable (ici un *unsigned char*) égaux à 1 et si cette somme est paire, la parité de cette variable le sera aussi, sinon elle sera impaire.

```
int parite_paire (unsigned char nombre)
{
  int ret = 0;

  while (nombre)
  {
    ret ^= nombre & 1;
    nombre >>= 1;
  }
  return ret;
}
```

Cryptage selon la méthode de César

Auteurs : Nicolas Joseph ,

Le cryptage de César

```
void cesar (char *str, int decalage)
{
  if (str)
  {
    const char alphabet[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
                             'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
                             'u', 'v', 'w', 'x', 'y', 'z'};
    int maj = 0;

    while (*str)
    {
      int i;

      maj = isupper (*str);
      *str = tolower (*str);
      for (i = 0; i < 26; i++)
      {
        if (alphabet[i] == *str)
        {
          *str = alphabet[(i+decalage)%26];
          if (maj)
          {
            *str = toupper (*str);
          }
          break;
        }
      }
      str++;
    }
  }
```

```
}
```

Tri rapide d'un tableau

Auteurs : lefort ,

Cette fonction est une version simplifiée de la fonction de la bibliothèque standard qsort.

```
void echange (int *tab, int i, int j)
{
  int tampon = tab[i];

  tab[i] = tab[j];
  tab[j] = tampon;
}

void trirapide (int *tab, int G, int D)
{
  int g, d;
  int val;

  if (D <= G)
  {
    return;
  }
  val = tab[D];
  g = G;
  d = D - 1;
  do
  {
    while ((tab[g] < val))
    {
      g++;
    }
    while ((tab[d] > val) && (d>G))
    {
      d--;
    }
    if (g < d)
    {
      echange (tab, g, d);
    }
  } while (g < d);
  echange (tab, g, D);
  trirapide (tab, G, g - 1);
  trirapide (tab, g + 1, D);
}
```

Pour trier un tableau de taille N, il suffit de faire :

```
int t[N];

/*Initialisation de t*/
...

trirapide (t,0,N-1);
```

Ici le code est proposé avec un tableau d'entiers mais il est possible d'utiliser le même algorithme pour trier un tableau de type différent.

Tri par insertion

Auteurs : Romuald Perrot ,

```c
#include <stdio.h>
#include <stdlib.h>

static int swap (void *const a, void *const b, size_t size)
{
  void *const temp = malloc (size);

  if (temp == NULL)
  {
    return EXIT_FAILURE;
  }

  memcpy (temp, a, size);
  memcpy (a, b, size);
  memcpy (b, temp, size);

  free(temp);

 return EXIT_SUCCESS;
}

int sort (void * vdata, size_t nb_elt, size_t size_elt, int (*comp) (const void *, const void *))
{
  size_t i;

  /* Cast en char * pour pouvoir effectuer des calculs d'adresse */
  char * mini = NULL;
  char * data = vdata;

  for (i = 1 ; i < nb_elt ; i++)
  {
    {
      size_t j;

      /* On considere le premier element du tableau non trie comme le plus petit. */
      mini = data + i * size_elt;

      /* On recherche le plus petit element */
      for (j = i + 1 ; j < nb_elt ; j++)
      {
        if (comp ((data + j * size_elt), mini ) < 0)
        {
          mini =  data + j * size_elt;
        }
      }

      /* On echange le plus petit element avec */
      if (swap ((data + i * size_elt) , mini) == EXIT_FAILURE)
      {
        return EXIT_FAILURE;
      }
    }
  }
}

int compare (const void * a, const void * b)
{
  int ta = *(int *)a;
```

```
  int tb = *(int *)b;

  return ta - tb;
}

int main (void)
{
  int c[] = {1 , 4 , 3 , 10 , 5};

  if (sort (c, 5, sizeof (int), compare) == EXIT_FAILURE)
  {
    exit (EXIT_FAILURE);
  }

  {
    int i;

    for (i = 0; i < 5; i++)
    {
      fprintf (stderr, "%d\n", c[i]);
    }
  }

  return EXIT_SUCCESS;
}
```

Calcul des clés de Luhn

Auteurs : gl ,

Cet algorithme permet de vérifier la validité d'un numéro, tel que celui des cartes bancaires, des cartes SNCF ou encore les numéros de SIRET. Cet algorithme consiste à prendre chaque chiffre du numéro, le multiplier par deux s'il est pair et on additionne chaque chiffre ainsi obtenu. Si le résultat est un multiple de 10, le numéro est valide.

ftp://ftp-developpez.com/c/sources/c/luhn.zip

Comment calculer le nombre de chiffres d'un entier ?

Auteurs : Jean Christophe Beyler ,

Il peut être parfois utile de connaître le nombre de chiffres que contient un nombre par exemple si l'on souhaite le convertir en chaîne de caractères à l'aide de la fonction sprintf :

```
#include <math.h>
int nombre_chiffre (int i)
{
  return log10 (i) + 1;
}
```

Comment trouver les combinaisons possibles d'un tableau ?

Auteurs : Jean-Marc.Bourguet ,

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void usage (const char *name)
{
```

```c
  fprintf (stderr, "Usage : %s abcd\n", name);
}

int main (int argc, char **argv)
{
  int ret = EXIT_SUCCESS;

  if (argc == 2)
  {
    size_t size;
    char *permuted = NULL;

    size = strlen (argv[1]);
    permuted = malloc (sizeof (*permuted) * (size + 1));
    if (permuted)
    {
      int i;
      int j;
      char tmp;

      strcpy (permuted, argv[1]);
      do
      {
        printf ("%s\n", permuted);
        i = size - 1;
        while (i > 0 && permuted[i] <= permuted[i-1])
        {
          --i;
        }
        if (i > 0)
        {
          j = size - 1;
          while (permuted[j] <= permuted[i-1])
          {
            --j;
          }
          tmp = permuted[j];
          permuted[j] = permuted[i-1];
          permuted[i-1] = tmp;
        }
        j = size - 1;
        while (i < j)
        {
          tmp = permuted[j];
          permuted[j] = permuted[i];
          permuted[i] = tmp;
          ++i;
          --j;
        }
      } while (strcmp (permuted, argv[1]) != 0);
      free( permuted), permuted = NULL;
    }
    else
    {
      fprintf (stderr, "Memoire insufisante\n");
      ret = EXIT_FAILURE;
    }
  }
  else
  {
    usage (argv[0]);
    ret = EXIT_FAILURE;
  }
  return ret;
```

```
}
```

Afficher toutes les solutions au problème des N-Reines

Auteurs : Helmstetter Bernard ,

Programme qui permet de résoudre et afficher toutes les solutions au problème des N-Reines (ou N-Dames).

Le problème des N-Reines consiste à placer N reines sur un échiquier NxN sans que l'une d'elles puisse en manger une autre (avec les règles des échecs : une reine peut « manger » toute pièce située sur sa ligne, sur sa colonne ou sur l'une de ses deux diagonales).
Pour plus d'informations sur le problème des N-Reines, vous pouvez consulter cet article sur la résolution du problème des Huit Dames

ftp://ftp-developpez.com/c/sources/c/n_reines.zip

Utilisation de l'algorithme Minimax (MinMax)

Auteurs : Helmstetter Bernard ,

Programme qui joue au morpion, avec l'algorithme Minimax.

Minimax (aussi minmax) est une méthode dans la "Théorie de la décision" qui permet de minimiser un maximum de perte possible. Alternativement, il peut considérer en tant que maximum le minimum de gain (maxmin).

Pour plus d'information sur cet algorithme: Minimax

ftp://ftp-developpez.com/c/sources/c/morpion_minimax.zip

Utilisation de l'algorithme NegaMax

Auteurs : Helmstetter Bernard ,

Programme qui joue au morpion, avec l'algorithme Negamax. Cet algorithme simplifie un peu le code.

L'algorithme Negamax est une variante de Minimax qui se fonde sur la propriété de zero-sum (ou zero sommes) d'un jeu à deux joueurs.

Pour plus d'information sur cet algorithme: Negamax

ftp://ftp-developpez.com/c/sources/c/morpion_negamax.zip

Utilisation de l'algorithme d'élagage alpha-beta

Auteurs : Helmstetter Bernard ,

Programme qui joue au morpion, avec l'algorithme alpha-beta (ou aussi Elagage alpha-beta).
L'algorithme alpha-beta permet de réduire le nombre de noeuds évalués par l'algorithme Minimax (MinMax).

Pour plus d'informations sur cet algorithme: **alpha-beta**

ftp://ftp-developpez.com/c/sources/c/morpion_alphabeta.zip

Construction et résolution de labyrinthe

Auteurs : Helmstetter Bernard ,

Programme de construction et de résolution de labyrinthe. Trois méthodes de résolution sont proposées :

- **Recherche en profondeur d'abord récursive**
- **Recherche en profondeur d'abord avec une pile**
- **Recherche en largeur d'abord avec une queue**

ftp://ftp-developpez.com/c/sources/c/laby.zip

Résolution de labyrinthe avec l'algorithme A* (A Star)

Auteurs : Helmstetter Bernard ,

Méthode de résolution de labyrinthe avec l'algorithme A* implémenté à l'aide d'une queue de priorité. Ce programme montre également comment passer d'un tableau bidimensionnel à un tableau unidimensionnel pour représenter le labyrinthe.

Pour plus d'informations sur l'algorithme A*: **Recherche de chemin: A***

ftp://ftp-developpez.com/c/sources/c/laby_pqueue.zip

Résolution de Sudoku par backtracking

Auteurs : Helmstetter Bernard ,

Programme de résolution de sudoku par backtracking très basique.

ftp://ftp-developpez.com/c/sources/c/sudoku.zip

Résolution de Sudoku par backtracking avec propagation des contraintes

Auteurs : Helmstetter Bernard ,

Programme de résolution de sudoku par backtracking avec propagation des contraintes et sélection de la variable la plus contraignante.

ftp://ftp-developpez.com/c/sources/c/sudoku_mrv.zip

Programme d'Othello avec l'algorithme Negamax

Auteurs : Helmstetter Bernard ,

Programme simple d'othello, avec une recherche par Negamax et de l'approfondissement itératif

ftp://ftp-developpez.com/c/sources/c/othello.zip

Tri par minimum

Auteurs : Franck.H ,

Fonction de Tri par Minimum. Cette fonction tri dans les deux sens (croissant et décroissant).
Ici l'exemple porte sur un tableau d'entiers mais cela peut être utilisé pour tout autre type !

```c
#include <stdio.h>
#include <stdlib.h>


/*
 * order: 0 = Tri croissant,
 *        1 = Tri decroissant.
 */
void tri_minimum (int * tab, size_t size, int order)
{
   size_t i = 0;
   size_t j = 0;


   if (tab != NULL && size > 1)
   {
      /* Parcours du tableau. */
      for (i = 0; i < size; i++)
      {
         /* Recherche du minimum. */
         size_t min = i;
```

```
         for (j = i; j < size; j++)
         {
            if (order == 0)
            {
               if (tab[j] < tab[min])
               {
                  min = j;
               }
            }
            else
            {
               if (tab[j] > tab[min])
               {
                  min = j;
               }
            }
         }

         /* Echange des valeurs du tableau. */
         {
            int tmp = tab[i];
            tab[i] = tab[min];
            tab[min] = tmp;
         }
      }
   }
}


int main (void)
{
   int i = 0;
   int tab [] = { 5, 2, 9, 6, 10, 8, 3, 1, 7, 4 };


   tri_minimum (tab, 10, 0);

   for (i = 0; i < 10; i++)
   {
      printf ("%d ", tab[i]);
   }
   printf ("\n");


   return 0;
}
```

Tri par maximum

Auteurs : Franck.H ,

Fonction de Tri par Maximum. Cette fonction tri dans les deux sens (croissant et décroissant).
Ici l'exemple porte sur un tableau d'entiers mais cela peut être utilisé pour tout autre type !

```
#include <stdio.h>
#include <stdlib.h>


/*
 * order: 0 = Tri croissant,
 *        1 = Tri decroissant.
 */
void tri_maximum (int * tab, size_t size, int order)
```

```c
{
   size_t i = 0;
   size_t j = 0;


   if (tab != NULL && size > 1)
   {
      /* Parcours du tableau. */
      for (i = size - 1; i > 0; i--)
      {
         /* Recherche du maximum. */
         size_t max = i;
         for (j = 0; j <= i; j++)
         {
            if (order == 0)
            {
               if (tab[j] > tab[max])
               {
                  max = j;
               }
            }
            else
            {
               if (tab[j] < tab[max])
               {
                  max = j;
               }
            }
         }

         /* Echange des valeurs du tableau. */
         {
            int tmp = tab[i];
            tab[i] = tab[max];
            tab[max] = tmp;
         }
      }
   }
}


int main (void)
{
   int i = 0;
   int tab [] = { 5, 2, 9, 6, 10, 8, 3, 1, 7, 4 };


   tri_maximum (tab, 10, 1);

   for (i = 0; i < 10; i++)
   {
      printf ("%d ", tab[i]);
   }
   printf ("\n");


   return 0;
}
```

Tri à bulle

Auteurs : Franck.H ,

Fonction de Tri à bulle. Cette fonction tri dans les deux sens (croissant et décroissant).
Ici l'exemple porte sur un tableau d'entiers mais cela peut être utilisé pour tout autre type !

```c
#include <stdio.h>
#include <stdlib.h>


/*
 * order: 0 = Tri croissant,
 *        1 = Tri decroissant.
 */
static void echanger_valeur (int * tab, size_t a, size_t b)
{
   if (tab != NULL)
   {
      int tmp = tab[a];
      tab[a] = tab[b];
      tab[b] = tmp;
   }
}

static void tri_a_bulle (int * tab, size_t size, int order)
{
   int trie = 0;

   if (tab != NULL && size > 1)
   {
      while (! trie)
      {
         size_t i = 0;

         trie = 1;
         for (i = 0; i < size - 1; i++)
         {
            if (order == 0)
            {
               if (tab[i] > tab[i + 1])
               {
                  echanger_valeur (tab, i, i + 1);
                  trie = 0;
               }
            }
            else
            {
               if (tab[i] < tab[i + 1])
               {
                  echanger_valeur (tab, i, i + 1);
                  trie = 0;
               }
            }
         }
      }
   }
}


int main (void)
{
   int i = 0;
   int tab [] = { 5, 2, 9, 6, 10, 8, 3, 1, 7, 4 };


   tri_a_bulle (tab, 10, 0);

   for (i = 0; i < 10; i++)
   {
      printf ("%d ", tab[i]);
   }
   printf ("\n");
```

```
   return 0;
}
```

Inversion des couleurs d'un fichier BMP

Auteurs : padugas ,

Programme qui inverse les couleurs (négatif) d'un fichier BMP 24 bits. L'image doit se trouver dans le même répertoire que le programme.

ftp://ftp-developpez.com/c/sources/c/bmp-negatif.zip

Checksum 16 bits

Auteurs : Emmanuel Delahaye ,

Cette fonction calcul un checksum de 16 bits.

```
#include <stdio.h>

#define DBG 1

int cksum16 (void const *p, size_t len)
{
   /* per defaut : erreur */
   int cs = -1;

   /* parite paire ? */
   if ((len & 1) == 0)
   {
      /* acceder aux donnees en mode byte */
      unsigned char const *po = p;
      unsigned i;

      /* initialisation du CS */
      cs = 0;

      for (i = 0; i < len; i += 2)
      {
         /* convention reseau : MSB en tete */
         unsigned n = 0;
         n |= (po[i + 0] & 0xFF) << (8 * 1); /* MSB */
         n |= (po[i + 1] & 0xFF) << (8 * 0); /* LSB */

#if DBG
         printf ("po[%u+0]=%d po[%u+1]=%d\n", i, po[i + 0], i, po[i + 1]);
#endif

         cs += n;

#if DBG
         printf ("n=%04X cs=%04x\n", n, cs);
#endif
      }
      /* complement a 1 sur 16 bits */
      cs = ~cs & 0xFFFF;
   }
   return cs;
}
```

```
int main (void)
{
   unsigned char data[20] = { 1, 2, 3, 4, 5, 6, 7, 8 };

   int cs = cksum16 (data, 8);

   if (cs != -1)
   {
      printf ("cs = %04X\n", cs);
   }
   else
   {
      puts ("cs error");
   }
   return 0;
}
```

Solveur de Sudoku

Auteurs : forthx ,

Solveur de sudoku

```
#include <stdio.h>
#include <stdlib.h>

/*structure pour pouvoir retourner 2 variables*/
typedef struct _pxy{
       int x;
       int y;
       } pxy;

/* entrer ici la grille a resoudre (0 = case vide)
 (ou completer le programme avec une interface graphique :D)
ci-join un exemple de grille "dur" a resoudre par le programme
(merci a Stumpy pour la grille)

note : le fait d'utiliser 3 tables prend plus de memoire
et necessite legerement plus de temps d'ecriture, par contre
les tests de validité d'une valeur doivent etres plus simples.

*/


/* table representant les collones de la grille*/
int col[9][9]=
{
{0,0,0 ,0,0,0 ,0,0,0},
{0,0,0 ,0,0,3 ,0,8,5},
{0,0,1 ,0,2,0 ,0,0,0},

{0,0,0 ,5,0,7 ,0,0,0},
{0,0,4 ,0,0,0 ,1,0,0},
{0,9,0 ,0,0,0 ,0,0,0},

{5,0,0 ,0,0,0 ,0,7,3},
{0,0,2 ,0,1,0 ,0,0,0},
{0,0,0 ,0,4,0 ,0,0,9},
};

/*transposé de col */
int ligne[9][9];

/*valeures en coordonées de groupe*/
```

```
int part[9][9];



/*--------------------fonctions de calculs anexes-----------------*/

/*genere x et y a partire de la position pos*/
pxy genxy(int pos)
{
  pxy p;
  p.x = pos%9;
  p.y = (pos-p.x)/9;
  return p;
}

/*genere les coordonées de groupe a partir de x et y*/
pxy genpxy(int x, int y)
{
  pxy p;
  p.y = x%3+3*(y%3) ;
  p.x = (x-x%3)/3+(y-y%3);
  return p;
}

/*initialisation de col et part*/
void init(void)
{
  int x,y;
  pxy p;
  for (x=0;x<9;x++)
       for (y=0;y<9;y++){
            ligne[y][x]= col[x][y];
            p=genpxy(x,y);
            part[p.x][p.y]=col[x][y];
            }
}

/*si la position est valide en ligne, en colonne et en groupe retourne 1. sinon 0*/
int valide(int val, int x, int y, pxy p)
{
  int i;
  for (i=0;i<9;i++)
      if (ligne[y][i] == val)
         return 0;
  for (i=0;i<9;i++)
      if (col[x][i] == val)
         return 0;
  for (i=0;i<9;i++)
      if (part[p.x][i] == val) /*on reste sur le bloc et on regarde les cases*/
         return 0;
  return 1;
}

/*ajout d'un nombre validé*/
void add(int val,int x, int y,pxy p)
{
  ligne[y][x] = val;
  col[x][y] = val;
  part[p.x][p.y] = val;
}

/*retrait d'un nombre incorect (ne fait rien si la case est vide)*/
void del(int x, int y,pxy p)
{
  ligne[y][x] = 0;
  col[x][y] = 0;
  part[p.x][p.y] = 0;
```

```
}


/*--------------la fameuse fonction recurcive!----------*/

int calcul(int position)
{
  int i;
  pxy c,p;

  /*on genere les coordonnées de travail*/
  c=genxy(position);
  p=genpxy(c.x,c.y);

  /*si on sort de la grille->fini*/
  if (position == 81)
     return 1;

  /*si la case est deja pleine->suivante*/
  if (col[c.x][c.y] != 0)
     return calcul(position+1);

  /*les chiffres valides vont de 1 a 9*/
  for (i=1;i<10;i++)
      {
   /*si le chiffre rentre on le met...*/
      if (valide(i,c.x,c.y,p)==1)
         {
         add(i,c.x,c.y,p);
   /*... et si la grille s'est bien completé,
         on averti la fonction appelante que c'est ok*/
         if (calcul(position+1)==1)
            return 1;
         }
      }
  /*si aucune valeur ne convien on retire la valeur qui ne marche pas...*/
  del(c.x,c.y,p);
  /*et on signal que la grille est bloqué*/
  return 0;
}


/*--------------------------fonction d'affichage--------------------------*/

void affiche(void)
{
  int x,y;
  pxy p;
  for (x=0;x<9;x++)
      {
      if (x%3==0)printf(" --- --- ---\n");
         for (y=0;y<9;y++)
             {
             if(y%3==0)printf(" ");
             if(col[x][y] != 0)printf("%d",col[x][y]);
             else printf(".");
             }
       printf("\n");
       }
  printf("\n");
}


/*--------------------------main-------------------------*/
int main()
```

```
{
  /*on prepare toutes les grilles*/
  init();
  /*on affiche la grille initiale*/
  affiche();

  /*on lance la calcul*/
  calcul(0);
  /*on affiche la grille resolue (si la grille n'a pas de solution,
    il faudra s'armer de patience ;)*/
  affiche();


  /* cette ligne est utile si vous etes sous windows
     et que vous n'utilisez pas la console*/
  /*system("PAUSE");*/
  return 0;
}
```

Conversion binaire -> ASCII

Auteurs : Jean Christophe Beyler ,

La fonction *get_char_from_bin* permet de convertir une représentation binaire en nombre.

```
unsigned char get_char_from_bin (const char *bin)
{
  int i;
  int d;
  unsigned char res;

  res = 0;
  i = strlen(bin)-1;
  for (d = 1; (d <= 128) && (i>=0) ; d*=2, i--)
  {
    if (bin[i] == '1')
      res += d;
  }
  return (res);
}
```

Obtenir la représentation en base N d'un entier

Auteurs : Vincent PETIT ,

La fonction *itoa* retourne la représentation en base *radix* (entre 2 et 36) de l'entier *value*, sous forme d'une chaîne de caractères.

La chaîne de caractères string doit être assez grande pour contenir la représentation.

```
char *itoa (int value, char *string, int radix)
{
  char tmp[32];
  char *tp = tmp;
  int i;
  unsigned v;
  int sign;
  char *sp;

  if (radix > 36 || radix <= 1)
  {
    return 0;
  }

  sign = (radix == 10 && value < 0);

  if (sign)
  {
    v = -value;
  }
  else
  {
    v = (unsigned)value;
  }

  while (v || tp == tmp)
  {
    i = v % radix;
    v = v / radix;
```

```
    if (i < 10)
    {
      *tp++ = i+'0';
    }
    else
    {
      *tp++ = i + 'A' - 10;
    }
  }

  sp = string;

  if (sign)
  {
    *sp++ = '-';
  }

  while (tp > tmp)
  {
    *sp++ = *--tp;
  }

  *sp = 0;

  return string;
}
```

Gestion des nombres au format BCD

Auteurs : gl ,

Le format BCD est un codage hexadécimal où seul les quartets <= 9 sont utilisés. Ce qui fait qu'après \x09 ce n'est pas \x0A mais \x10.
C'est une façon de coder des chaînes de chiffres assez longues en réduisant la taille par deux par rapport à une chaîne de caractères tout en restant facilement lisible.
Les fonctions présentes dans le zip permettent une conversion de la notation ASCII vers BCD et réciproquement :

```
"1234" <==> "\x12\x34"
```

ftp://ftp-developpez.com/c/sources/c/bcd.zip

Encodage base 64

Auteurs : mabu ,

Cette fonction encode en base 64. Ce programme travaille avec stdin et stdout.

```
/**
 * @file encode64.c
 */

#include <stdio.h>
#include <string.h>

/* le type code64 contient le code base64 proprement dit.*/
typedef unsigned char code64[4];
/* le type in64 contient les octets à coder */
typedef unsigned char in64[3];
```

```c
/**
 * La fonction codeToChar converti un paquet de 6 bits en son code base 64.
 * Les deusx premiers bits de c doivent être nuls.
 * @param[in] c : paquet de 6 bits à encoder
 */
unsigned char codeToChar(const unsigned char c)
{
    if (c < 26) {
        /* le paquet de 6 bits vaut entre 0 et 25 : on le code de A à Z */
        return 'A' + c;
    } else if (c < 52) {
        /* le paquet de 6 bits vaut entre 26 et 51 : on le code de a à z */
        return 'a' + c - 26;
    } else if (c < 62) {
        /* le paquet de 6 bits vaut entre 52 et 61 : on le code de 0 à 9 */
        return '0' + c - 52;
    } else if (62 == c) {
        /* le paquet de 6 bits vaut entre 62 : on le code + */
        return '+';
    } else if (63 == c) {
        /* le paquet de 6 bits vaut entre 63 : on le code / */
        return '/';
    } else {
        /* les deux premiers bits ne sont pas nuls, ce doit être un erreur. */
        printf("\n<<%d>>\n", c);
        return '?';
    }
}

/**
 * Cette fonction encode un paquet de 3 octets en un quadruplet de caractères
 * en base 64.
 * @param[in] in : paquet d'au plus 3 octets à encoder
 * @param[out] code : encodage des octets
 * @param[in] sz : nombre d'octets à encoder (1, 2 ou 3)
 */
void encode(in64 in, code64 *code, const int sz)
{
    if (NULL != code) {
        unsigned char c;
        /* cas particulier : moins de trois octets à coder */
        switch (sz) {
        case 1:
            in[2] = 0;
        case 2:
            in[1] = 0;
        }

        /* on récupère les 6 premiers bits du premier octet */
        c = in[0] / 4;
        /* on encode le paquet de 6 bits */
        (*code)[0] = codeToChar(c);

        /* on récupère les 2 bits derniers du premier octet
           (qui n'avaient pas été codés) */
        c = (in[0] % 4) * 16;
        /* on récupère les 4 premiers bits du second octet */
        c += in[1] / 16;
        /* on encode le paquet de 6 bits */
        (*code)[1] = codeToChar(c);

        /* on récupère les 4 derniers bits du second octet */
        c = (in[1] % 16) * 4;

        /* on récupère les 2 premiers bits du troisième octet */
        c += in[2] / 64;
        /* on encode le paquet de 6 bits */
        (*code)[2] = codeToChar(c);
```

```
        /* on récupère les 6 derniers bits du troisième octet */
        c = in[2] % 64;
        /* on encode le paquet de 6 bits */
        (*code)[3] = codeToChar(c);

        /* cas particulier : moins de trois octets à coder :
            on bourre avec des = */
        switch (sz) {
        case 1:
            (*code)[2] = '=';
        case 2:
            (*code)[3] = '=';
        }
    }
}

/**
 * Point d'entrée du programme.
 * On va lire l'entrée standard par paquet de 3 octets et afficher le résultat
 * directement sur la sortie standard.
 * Si moins de trois octets sont lus on encode éventuellement ce qui reste à
 * encoder puis le programme se termine.
 */
int main(void)
{
    in64 in;
    code64 code;
    int read;

    do {
        /* lecture de trois octets depuis l'entrée standard */
        read = fread(in, 1, 3, stdin);
        if (read != 0) {
            /* des octets on été lus */
            int i;
            /* encodage des octets */
            encode(in, &code, read);
            /* affichage de ce qui a été codé */
            for(i = 0; i < 4; ++i) {
                fputc(code[i], stdout);
            }
            /* flush pour afficher sans attendre */
            fflush(stdout);
        }
    } while (read == 3);

    return 0;
}
```

Decodage base 64

Auteurs : mabu ,

Cette fonction encode en base 64. Ce programme travaille avec stdin et stdout.

```
/**
 * @file decode64.c
 */

#include <stdio.h>
#include <string.h>
```

```c
/* le type code64 contient le code base 64 à décoder*/
typedef unsigned char code64[4];
/* le type out64 contient les octets décodés */
typedef unsigned char out64[3];

/**
 * La fonction charToCode converti un caractère base 64 en paquet de 6 bits.
 */
unsigned char charToCode(const unsigned char c)
{
    if((c >= 'A') && (c <= 'Z')) {
        /* A à Z --> 0 à 25 */
        return c - 'A';
    } else if((c >= 'a') && (c <= 'z')) {
        /* a à z --> 26 à 51 */
        return c + 26 - 'a';
    } else if((c >= '0') && (c <= '9')) {
        /* 0 à 9 --> 52 à 61 */
        return c + 52 - '0';
    } else if('+' == c) {
        /* + --> 62 */
        return 62;
    } else if('/' == c) {
        /* / --> 63 */
        return 63;
    } else {
        /* caractère illegal */
        return -1;
    }
}

/**
 * Cette fonction décode un paquet de 4 caractères base 64 en 3 octets
 * @param[in] code : paquet de caractères à décoder
 * @param[out] out : octets décodé
 * @param[size] size : nombre d'octets décodés (1, 2 ou 3)
 */
void decode(const code64 code, out64 *out, int * size)
{
    if(NULL != out && NULL != size) {
        int i;
        code64 tmp;
        /* on converti les 4 caractères en 4 paquets de 6 bits */
        for(i = 0; i != 4; ++i) {
            tmp[i] = charToCode(code[i]);
        }
        /* ici, on converti les 4 paquets de 6 bits en 3 octets */
        (*out)[0] = tmp[0] * 4;
        (*out)[0] += tmp[1] / 16;
        *size = 1;
        if('=' != code[2]) {
            (*out)[1] = (tmp[1] % 16)*16;
            (*out)[1] += tmp[2] / 4;
            *size = 2;
            if('=' != code[3]) {
                (*out)[2] = (tmp[2]%4)*64;
                (*out)[2] += tmp[3];
                *size = 3;
            }
        }
    }
}

int main(void)
{
    out64 out;
    code64 code;
```

```
    int read;
    int size;

    while(1) {
        /* lecture de 4 caratères depuis l'entrée standard */
        read = fread(code, 1, 4, stdin);
        if (4 != read) {
            /* Les 4 caractères n'ont pas été lus, on sort*/
            break;
        } else {
            /* les 4 caractères ont été lus */
            int i;
            /* décodage du paquet de caractères */
            decode(code, &out, &size);
            /* affichage de ce qui a été décodé */
            for(i = 0; i < size; ++i){
                fputc(out[i], stdout);
            }
            /* flush pour afficher sans attendre */
            fflush(stdout);
        }
    }

    return 0;
}
```

Calcul de l'écart entre deux dates

Auteurs : beyo ,

A partir de deux dates la fonction *Diff* calcul le nombre de jours qui sépare ces deux dates, en tenant compte des années bissextiles.

```c
/* l'année est-elle bissextile*/
int Bissextile (int A)
{
  return A % 4 == 0 && (A % 100 != 0 || A % 400 == 0);
}

/*combien de jours se sont ecoules depuis le debut de l'annee donnee*/
int Nb_Jours (int J, int M, int A)
{
  int i, D = 0;
  const int Mois[12]= {31,28,31,30,31,30,31,31,30,31,30,31};

  if (M == 1)
  {
    D = J;
  }
  else
  {
    for (i = 0; i < (M-1); i++)
    {
      D += Mois[i];
    }
    D+=J;
  }
  if ((M > 2) && (Bissextile(A)))
  {
    D++;
  }
  return D;
}

/*la fonction diff proprement dite*/
int Diff (int j1, int m1, int a1, int j2, int m2, int a2)
{
  int NJ = 0, NJ1, NJ2, i;

  NJ1 = Nb_Jours (j1, m1, a1);
  NJ2 = Nb_Jours (j2, m2, a2);
  if (a2 == a1)
  {
    NJ = NJ2 - NJ1;
  }
  else
  {
    for (i = 0; i < (a2-a1); i++)
    {
      NJ += 364;
      if (Bissextile (a1+i))
      {
        NJ++;
      }
    }
    NJ -= NJ1;
    NJ += NJ2+1;
  }
  return NJ;
```

```
}
```

Utilisation de mktime

Auteurs : D[r]eadLock , Emmanuel Delahaye ,

```
#include <stdio.h>
#include <time.h>

int main (void)
{
   char const *jour[] =
      { "dimanche", "lundi", "mardi", "mercredi", "jeudi", "vendredi",
"samedi" };

   int y, m, d;
   /*
      afin d'eviter les ennuis, il est fortement conseille d'initialiser
      tous les champs a 0 des la creation de la structure :
    */
   struct tm t = { 0 };
   time_t tt;

   /* 17/04/2003 */
   y = 103;                     /* 2003-1900 */
   /*
      La numerotation des mois commence a 0.
    */
   m = 4 - 1;

   d = 17;
   t.tm_sec = 0;
   t.tm_min = 0;
   t.tm_mday = d;
   t.tm_mon = m;
   t.tm_year = y;
   tt = mktime (&t);
   t = *localtime (&tt);
   printf ("date : %s %.2d/%.2d/%.4d\n", jour[t.tm_wday], d, m + 1, y + 1900);
   return 0;
}
```

Savoir si une année est bissextile ou non

Auteurs : beyo ,

```
int Bissextile (int A)
{
  return A % 4 == 0 && (A % 100 != 0 || A % 400 == 0);
```

```
}
```

Gestion des dates

Auteurs : gl ,

Le fichier date.c propose la fonction *DATE_eGetTodayDate* qui permet de récupérer la date du jour selon le format désiré.

ftp://ftp-developpez.com/c/sources/c/date.zip

Comment calculer le temps d'exécution d'une fonction

Auteurs : Nicolas Joseph ,

```c
#include <stdio.h>
#include <time.h>

int main (void)
{
  clock_t start, end;

  start = clock ();
  {
    /* Portion de code a chronometrer */
  }
  end = clock ();
  printf ("Temps en secondes : %f\n", (end - start) / (double)CLOCKS_PER_SEC);
  return 0;
}
```

Si le code dure moins d'une seconde (précision de la fonction *time*), vous pouvez effectuer plusieurs appels dans une boucle for sans oublier de diviser le temps d'exécution d'autant.

Faire une pause

Auteurs : Nicolas Joseph ,

```c
#include <time.h>

void sleep (unsigned int sec)
{
#ifdef _WIN32
  Sleep (sec * 1000);
#else
#  if _POSIX_VERSION > 198808L
  sleep (sec);
#  else
  clock_t start, end;

  end = start = clock ();
  if (start != -1)
  {
    while ((end - start) / CLOCKS_PER_SEC < sec)
    {
      end = clock ();
    }
  }
#  endif /* _POSIX_VERSION */
#endif /* _WIN32 */
```

```
}
```

On privilégie les fonctions système si elles existent (sleep sous Windows et sleep sous les systèmes POSIX.1), ce qui permet de laisser le processeur libre pour les autres tâches, sinon on se contente d'une attente active.

La précision de la fonction sleep dépend de la précision de la fonction clock.

Comparateur de dates

Auteurs : souviron34 ,

Cette fonction permet de comparer deux dates sous forme de chaînes de caractères.

```
/*
 * Positif si DateToCompare > Reference.
 *
 * Precision = 16 par defaut (resultat a la milliseconde pres). Pour utiliser la
 * valeur par defaut, mettre 0 ou une valeur négative.
 */
double CompareDates (char *Reference, char *DateToCompare, int Precision)
{
   int i, Longueur, Puissance;
   double d;


   if (Reference != NULL && DateToCompare != NULL)
   {
      Longueur = strlen(DateToCompare);
      d = 0.0;


      if (Precision <= 0)
         Puissance = 16;
      else
         Puissance = Precision;


      for ( i = 0 ; i < Longueur ; i++ )
      {
         if ( (int)DateToCompare[i] != (int)Reference[i] )
         {
            d = pow(10.0,(double)(Puissance - i)) ;
            d = d * (double)((int)DateToCompare[i] - (int)Reference[i]) ;

            break ;
         }
      }
   }


   return d;
}
```

Convertir un temps GMT vers time_t

Auteurs : souviron34 ,

Cette fonction convertit une structure de temps GMT en une donnée de type time_t.

```c
time_t MktimeFromGMT (struct tm * tbrok)
{
   char     chaine_TZ[30];
   char   * TZ = NULL;
   time_t   temps = -1;


   if (tbrok != NULL)
   {
      TZ = getenv ("TZ");
      putenv ("TZ=UTC0");

      if ((temps = mktime (tbrok)) != -1)
      {
         sprintf  (& chaine_TZ[0], "TZ=%s", TZ);
         putenv   (chaine_TZ);
      }
   }


   return temps;
}
```

Récupérer le temps réel absolu

Auteurs : souviron34 ,

Récupérateur de temps (comme clock()), sauf qu'il récupère le temps réel absolu (alors que clock() récupère le temps par rapport au programme), en renvoyant une valeur réelle comportant un nombre de secondes ainsi qu'une fraction correspondant aux microsecondes.
Elle retourne -1.0 si une erreur s'est produite.

ftp://ftp-developpez.com/c/sources/c/get_clock.zip

Sommaire > GTK+

Afficher simplement une arborescence disque

Auteurs : Franck.H ,

Ce code est une "pseudo" classe qui prend en charge une très grande partie de la gestion d'un widget *gtk_tree_view* (affichage en arbre). Est gérée, la création complète du widget qui est placé lui-même dans un *gtk_scrolled_window* (barres de défilement), l'ajout/suppression d'éléments. Sont pris en compte également la gestion de certaines propriétés visuelles du widget comme l'affichage et le changement du titre de l'entête de la colonne, l'affichage de lignes colorées, etc...

ftp://ftp-developpez.com/c/sources/c/gtk_hf_Treeview.zip

Afficher un message avec GTK+

Auteurs : Franck.H ,

Ceci est une petite fonction qui permet d'afficher en un seul appel une boîte de messages en GTK.
Plus la peine de s'embêter à coder l'appel, la mise en place de la boîte de dialogue et la réception des messages etc... Il suffit d'appeler la fonction et de vérifier la valeur qu'elle retourne et la comparer à celle de la constante d'après le type de réponses que vous attendez puis c'est tout !

ftp://ftp-developpez.com/c/sources/c/gtk_dialog_message.zip

Utilisation de GtkUIManager

Auteurs : Nicolas Joseph ,

Utilisation de GtkUIManager

ftp://ftp-developpez.com/c/sources/c/gtk-uimanager.zip

Editeur de texte

Auteurs : Nicolas Joseph ,

GTK+ par l'exemple

ftp://ftp-developpez.com/c/sources/c/gtk-editeur.zip

API DOM pour la lecture de fichier XML

Auteurs : Nicolas Joseph ,

La glib propose une seule API pour lire un fichier XML : l'API SAX.
Voici une implémentation de l'API DOM basée sur GMarkup.

ftp://ftp-developpez.com/c/sources/c/gmarkup-dom.zip

Sommaire > Jeux

Trouver le bon nombre

Auteurs : AjJi , Nicolas Joseph ,

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main (void)
{
  int Ncache, Rep, i = 0;
  char s[8];

  srand (time (NULL));
  /* Nous appelons rand car les premières valeurs ne sont pas aleatoires */
  for (i = 0; i < 10; i++)
  {
    rand();
  }
  Ncache = (int)((float)rand() / RAND_MAX * (1000 - 1));
  i = 0;
  do
  {
    printf ("Entrez le nombre cache (0 < n < 1000) :\n");
    fgets (s, sizeof (s), stdin);
    Rep = (int)strtol (s, NULL, 10);
    if (Rep < Ncache)
    {
      printf ("Essayez un nombre plus grand !\n\n");
    }
    else if (Rep > Ncache)
    {
      printf("Entrez un nombre plus petit !\n\n");
    }
    i++;
  } while (Rep != Ncache);
  printf ("\nBravo vous avez trouve le bon nombre lors en %d essai%s.\n", i, (i>2)?"s":"");
  return 0;
```

```
}
```

Jeu du taquin

Auteurs : troumad ,

Jeu du taquin programmé avec la bibliothèque GTK+. Le programme permet de choisir son découpage et également de choisir l'image sur laquelle s'amuser !

ftp://ftp-developpez.com/c/sources/c/taquin.zip

Jeu du pendu

Auteurs : troumad ,

Jeu du pendu (version mathématique) programmé avec la bibliothèque GTK+. Ce programme est fait initialement pour des CM1. Chaque partie de pendu laisse une trace que l'instituteur peut parcourir grâce à un programme lecture_pendu.

ftp://ftp-developpez.com/c/sources/c/pendu.zip

Trouvez le bon mot

Auteurs : Franck.H ,

Jeu où il faut trouver le bon mot. C'est un style de pendu mais en mode console uniquement. Le programme est fournit avec une dictionnaire de 331612 mots !

ftp://ftp-developpez.com/c/sources/c/trouver-le-bon-mot.zip

Sommaire > Les structures de données

Connaître le nombre d'éléments d'un tableau

Auteurs : Nicolas Joseph ,

L'opérateur sizeof permet de connaître la taille d'un objet, pour avoir le nombre d'éléments d'un tableau, il suffit de diviser par la taille d'un élément :

```
#define NB_ELEMENTS(t) (sizeof (t) / sizeof *(t))
```

Décaler les colonnes d'un tableau

Auteurs : Musaran ,

Voici trois méthodes (de la plus simple à la plus rapide) permettant de décaler les colonnes d'un tableau vers la gauche (c'est à dire vers les indices plus faibles) en ajoutant des zéro dans la dernière colonne ainsi libérée.

```
#include <string.h> /* memmove et pas memcpy, car chevauchement. */

#define NLIG 3
#define NCOL 6

int main (void)
{
  int lig, col;
  int shiftw = 2;
  int Tab[NLIG][NCOL];

  /* decalage "a la main" */
  for (lig = 0; lig < NLIG; ++lig)
  {
    for (col = 0; col < shiftw-1; ++col)
    {
      Tab[lig][col] = Tab[lig][col+1]; /* decalage a gauche */
    }
    Tab[lig][shiftw-1] = 0 ;/* introduction de 0 à droite */
  }

  /* decalage optimise memmove */
  for (lig = 0; lig < NLIG; ++lig)
  {
    memmove (&Tab[lig][0], &Tab[lig][1], (shiftw-1)*sizeof(int)); /* decalage a gauche */
    Tab[lig][shiftw-1] = 0; /* introduction de 0 a droite */
  }

  {
    /* decalage optimise memmove+pointeurs */
    int (*plig)[NCOL];

    for (plig = &Tab[0]; plig < &Tab[NLIG]; ++plig)
    {
      memmove (&plig[0][0], &plig[0][1], (shiftw-1)*sizeof(int)); /* decalage a gauche */
      Tab[lig][shiftw-1] = 0; /* introduction de 0 a droite */
    }
  }
  return 0;
```

```
}
```

Inverser une liste chaînée

Auteurs : Jean Christophe Beyler ,

```
typedef struct liste
{
  int noeud;
  liste *next;
} liste;

liste *inverse_liste (liste *lst)
{
  liste *tmp1; /* pointeur vers le suivant dans la liste lst */
  liste *tmp2; /* pour memoriser lst de manière temporaire */

  /* initialisation */
  if (lst != NULL)
  {
    tmp1 = lst->next;
    lst->next = NULL;

    while (tmp1 != NULL)
    {
      /* on memorise l'adresse du suivant de tmp1*/
      tmp2 = tmp1->next;

      tmp1->next = lst; /* on met le next de tmp1 a jour */

      lst = tmp1;     /*On décale lst */

      tmp1 = tmp2; /* on passe au suivant de l'ancienne lst */
    }
  }
  return lst;
}
```

Fusionner le contenu de deux tableaux

Auteurs : Emmanuel Delahaye ,

```
#include<stdio.h>

#define N(a) (sizeof (a)/sizeof *(a))

static void print (char const *s, int const a[], size_t n)
{
  size_t i;

  printf ("%s: ", s);
  for (i = 0; i < n; i++)
  {
    printf ("%d ", a[i]);
  }
  printf ("\n");
}

int main(void)
{
  int a[] =
    {
```

```
      2, 5, 6, 8, 9, 11
    };
  int b[] =
    {
      3, 6, 7, 9, 12
    };
  int c[N(a) + N(b)];
  size_t ic = 0;
  {
    size_t ia = 0;
    size_t ib = 0;

    print ("a", a + ia, N(a) - ia);
    print ("b", b + ib, N(b) - ib);
    print ("c", c, ic);
    printf ("\n");

    while (ia != N(a) || ib != N(b))
    {
#if 1
      /* Elimination des doublons */
      if (a[ia] == b[ib])
      {
        if (ia < N(a))
        {
          c[ic] = a[ia];
          ia++;
        }
        if (ib < N(b))
        {
          ib++;
        }
      }
      else
#endif
      if (a[ia] < b[ib])
      {
        if (ia < N(a))
        {
          c[ic] = a[ia];
          ia++;
        }
      }
      else
      {
        if (ib < N(b))
        {
          c[ic] = b[ib];
          ib++;
        }
      }
      ic++;
      print ("a", a + ia, N(a) - ia);
      print ("b", b + ib, N(b) - ib);
      print ("c", c, ic);
      printf ("\n");
    }
  }

  print ("c", c, ic);
  return 0;
}
```

Si

```
#if 1
```

est remplacé par

```
#if 0
```

Les doublons présents seront supprimés.

Les listes simplement chaînées

Auteurs : Nicolas Joseph ,

Les listes simplement chaînées

ftp://ftp-developpez.com/c/sources/c/liste-simple.zip

Les listes doublement chaînées

Auteurs : Nicolas Joseph ,

Les listes doublement chaînées

ftp://ftp-developpez.com/c/sources/c/liste-double.zip

Les piles

Auteurs : Nicolas Joseph ,

Les piles

ftp://ftp-developpez.com/c/sources/c/pile.zip

Les files

Auteurs : Nicolas Joseph ,

Les files

ftp://ftp-developpez.com/c/sources/c/file.zip

Liste générique doublement chaînée

Auteurs : Melem ,

Liste générique doublement chaînée avec les fonctions de gestion de base.

ftp://ftp-developpez.com/c/sources/c/liste-generique-double.zip

Anneau de stockage (Buffer Circulaire)

Auteurs : diogene ,

Ce code permet de gérer un buffer circulaire.

Si vous l'utilisez avec un thread de lecture et un thread d'écriture (ce genre d'objet étant souvent utilisé ainsi), definissez CIRCBUFFER_MTHREAD (CircBuffer.h). Ce n'est pas indispensable mais améliore les performances.

Si vous l'utilisez avec plus d'un thread de lecture ou plus d'un thread d'écriture, il est impératif de definir CIRCBUFFER_MTHREAD (CircBuffer.h)

Attention, deux fonctions, CBuffLireBuffer() et CBuffEcrireBuffer(), sont bloquantes (voir commentaires dans CircBuffer.h). Si vous utilisez ces fonctions dans un cadre monothread (même thread pour la lecture et l'écriture), vous devrez vous assurer d'être dans les limites d'utilisation de ces fonctions ou sinon le programme sera gelé.

```
/* CircBuffer.h */
#ifndef CIRCBUFFER_H
#define CIRCBUFFER_H
/*-------------------------------------------------------------------*/
/* Si vous n'utilisez qu'un seul thread, commentez la ligne suivante    */
#define CIRCBUFFER_MTHREAD
/*-------------------------------------------------------------------*/
#include <stdlib.h>
/*-------------------------------------------------------------------*/
typedef struct circbuff TCircBuffer ;
/*-------------------------------------------------------------------*/
/*  Création d'un buffer circulaire acceptant 'nombreElement' objets de
    taille 'tailleElement' bytes.
    Retour :
    Adresse de la structure TCircBuffer créée ou NULL en cas d'échec */
TCircBuffer *CBuffCreer(size_t tailleElement, size_t nombreElement);

/*-------------------------------------------------------------------*/
/*  Destruction et récupération de la mémoire allouée à l'objet TCircBuffer
    d'adresse 'cbuffer' et créé par CBuffCreer */
void CBuffDetruire(TCircBuffer * cbuffer);

/*-------------------------------------------------------------------*/
/*  Vide l'objet TCircBuffer, pointé par 'cbuffer',des éléments qui y sont stockés */
void CBuffVider(TCircBuffer * cbuffer);

/*-------------------------------------------------------------------*/
/*  Renvoie le nombre d'objets actuellement stockés dans l'objet TCircBuffer
    d'adresse 'cbuffer' */
size_t CBuffElementsStockes(TCircBuffer * cbuffer);

/*-------------------------------------------------------------------*/
/*  Renvoie le nombre d'objets qu'il est encore possible de stocker dans
    l'objet TCircBuffer d'adresse 'cbuffer' */
size_t CBuffElementsLibres(TCircBuffer * cbuffer);

/*-------------------------------------------------------------------*/
/*  Ecrit dans l'objet TCircBuffer, d'adresse 'cbuffer', 'nbElement' objets
    ou jusqu'à ce que l'objet TCircBuffer soit plein. Les objets à écrire
    sont rangés dans un tableau dont l'adresse du premier élément est 'data'.
    Retour :
    Le nombre d'objets effectivement écrits. Si le buffer circulaire est plein,
    cette valeur peut être inférieure à 'nbElement'. */
size_t CBuffEcrire(TCircBuffer * cbuffer, void * data, size_t nbElement);

/*-------------------------------------------------------------------*/
/*  Extrait de l'objet TCircBuffer, d'adresse 'cbuffer', 'nbElement' objets
    ou jusqu'à ce que l'objet TCircBuffer soit vide et les copie dans le
    tableau dont l'adresse du premier élément est 'data'.
    Retour :
    Le nombre d'objets effectivement extraits et copiés. Si le buffer circulaire
    est vide, cette valeur peut être inférieure à 'nbElement'. */
size_t CBuffLire(TCircBuffer * cbuffer, void * data, size_t nbElement);
```

```
/*-------------------------------------------------------------------------*/
/*  Extrait de l'objet TCircBuffer, d'adresse 'cbuffer', 'nbElement' objets
    et les copie dans le tableau dont l'adresse du premier élément est 'data'.
    La fonction est BLOQUANTE tant que 'nbElement' objets n'ont pas été copiés. */
void CBuffLireBuffer(TCircBuffer * cbuffer, void * data, size_t nbElement);

/*-------------------------------------------------------------------------*/
/*  Ecrit dans l'objet TCircBuffer, d'adresse 'cbuffer', 'nbElement' objets.
    Les objets à écrire sont rangés dans un tableau dont l'adresse du premier
    élément est 'data'.
    La fonction est BLOQUANTE tant que 'nbElement' objets n'ont pas été écrits. */
void CBuffEcrireBuffer(TCircBuffer * cbuffer, void * data, size_t nbElement);

/*-------------------------------------------------------------------------*/
```

```
/* CircBuffer.c */
#include <string.h>
#include "CircBuffer.h"
#ifdef CIRCBUFFER_MTHREAD
#include <sched.h>
#include <pthread.h>
#endif
/*-------------------------------------------------------------------------*/
typedef enum {idRead, idWrite} TOperation;
typedef unsigned char TByte;
/*-------------------------------------------------------------------------*/
struct circbuff
{
    size_t  tailleElement;  // taille en bytes d'un élément
    size_t  nbBytes;        // nombre de bytes du buffer
    size_t  rw[2];     // position de lecture/d'écriture
#ifdef CIRCBUFFER_MTHREAD
    pthread_mutex_t mutex;  // mutex
    pthread_cond_t notfull; // condition buffer non plein
    pthread_cond_t notempty;// condition buffer non vide
#endif
    TByte   *buffer;        // les données
};
/*-------------------------------------------------------------------------*/
TCircBuffer *CBuffCreer(size_t tailleElement, size_t nombreElement)
{
    TCircBuffer * cbuffer = NULL;
    size_t dim = tailleElement * nombreElement;
    if(dim > tailleElement && dim/nombreElement == tailleElement)
 cbuffer = malloc(sizeof *cbuffer);
    if (cbuffer != NULL)
    {
        cbuffer->buffer = malloc(dim);
        if (cbuffer->buffer != NULL)
        {
            cbuffer->nbBytes = dim;
            cbuffer->tailleElement = tailleElement;
            CBuffVider(cbuffer);
#ifdef CIRCBUFFER_MTHREAD
            pthread_mutex_init(&cbuffer->mutex, NULL);
            pthread_cond_init(&cbuffer->notfull, NULL);
            pthread_cond_init(&cbuffer->notempty, NULL);
#endif
        }
        else
        {
            free(cbuffer);
            cbuffer = NULL;
        }
    }
    return cbuffer;
```

```
}
/*---------------------------------------------------------------------*/
void CBuffDetruire(TCircBuffer * cbuffer)
{
    if (cbuffer != NULL)
    {
        free(cbuffer->buffer);
#ifdef CIRCBUFFER_MTHREAD
        pthread_cond_destroy(&cbuffer->notempty);
        pthread_cond_destroy(&cbuffer->notfull);
        pthread_mutex_destroy(&cbuffer->mutex);
#endif
        free(cbuffer);
    }
}
/*---------------------------------------------------------------------*/
void CBuffVider(TCircBuffer * cbuffer)
{
  cbuffer->rw[idRead]  = 0;
  cbuffer->rw[idWrite] = cbuffer->tailleElement;
}
/*---------------------------------------------------------------------*/
static size_t getCount(TCircBuffer * cbuffer, TOperation op)
{
    size_t write = cbuffer->rw[idWrite];
    size_t read  = cbuffer->rw[idRead];
    size_t count = write - read - cbuffer->tailleElement;
    if(write <= read) count += cbuffer->nbBytes;
    if(op == idWrite) count = cbuffer->nbBytes - count - cbuffer->tailleElement;
    return count/cbuffer->tailleElement;
}
/*---------------------------------------------------------------------*/
size_t CBuffElementsStockes(TCircBuffer * cbuffer)
{
    return getCount(cbuffer, idRead);
}
/*---------------------------------------------------------------------*/
size_t CBuffElementsLibres(TCircBuffer * cbuffer)
{
    return getCount(cbuffer, idWrite);
}
/*---------------------------------------------------------------------*/
static size_t readwriteb(TCircBuffer * cbuffer, void * data, size_t nbElement, TOperation op )
{
    size_t nbTransf = 0;
#ifdef CIRCBUFFER_MTHREAD
    pthread_mutex_lock(&cbuffer->mutex);
#endif
    size_t position = cbuffer->rw[op];
    size_t nbElemDispo = getCount(cbuffer, op);
    if (nbElemDispo !=0 )
    {
        size_t byteTotal;
        size_t byteToEnd;
        size_t byteDispoToEnd = cbuffer->nbBytes - position - (op==idRead ?
 cbuffer->tailleElement : 0);
        nbTransf = nbElement > nbElemDispo ? nbElemDispo : nbElement;
        byteTotal = nbTransf*cbuffer->tailleElement;
        byteToEnd  = byteDispoToEnd > byteTotal ? byteTotal : byteDispoToEnd;
        if(op == idWrite)
        {
            if (byteDispoToEnd > 0)
  memcpy(cbuffer->buffer + position, data, byteToEnd);
            if (byteTotal > byteToEnd)
  memcpy(cbuffer->buffer, (TByte*)data + byteToEnd, byteTotal - byteToEnd);
        }
        else
        {
```

```
            if (byteDispoToEnd > 0)
  memcpy(data, cbuffer->buffer + position + cbuffer->tailleElement, byteToEnd);
            if (byteTotal > byteToEnd)
  memcpy((TByte*)data + byteToEnd, cbuffer->buffer, byteTotal - byteToEnd);
        }
        byteToEnd = cbuffer->nbBytes - position ;
        cbuffer->rw[op] = byteTotal < byteToEnd  ? position + byteTotal
                                                 : byteTotal - byteToEnd  ;
#ifdef CIRCBUFFER_MTHREAD
        pthread_cond_signal (op == idWrite ? &cbuffer->notempty : &cbuffer->notfull);
#endif
    }
#ifdef CIRCBUFFER_MTHREAD
    else  pthread_cond_wait(op ==
 idWrite ? &cbuffer->notfull : &cbuffer->notempty, &cbuffer->mutex);
    pthread_mutex_unlock(&cbuffer->mutex);
#endif
    return nbTransf;
}
/*---------------------------------------------------------------------*/
size_t CBuffEcrire(TCircBuffer * cbuffer, void * data, size_t nbElement)
{
    return readwriteb(cbuffer, data, nbElement, idWrite);
}
/*---------------------------------------------------------------------*/
size_t CBuffLire(TCircBuffer * cbuffer, void * data, size_t  nbElement)
{
   return readwriteb(cbuffer, data, nbElement, idRead);
}
/*---------------------------------------------------------------------*/
void CBuffEcrireBuffer(TCircBuffer * cbuffer, void * buffer, size_t nbElement)
{
 size_t n = 0;
 while (n < nbElement)
    n += readwriteb(cbuffer, (TByte *)buffer + n*cbuffer->tailleElement, nbElement - n, idWrite);
}
/*---------------------------------------------------------------------*/
void CBuffLireBuffer(TCircBuffer * cbuffer, void * buffer, size_t nbElement)
{
 size_t n = 0;
 while (n < nbElement)
    n += readwriteb(cbuffer, (TByte *)buffer + n*cbuffer->tailleElement, nbElement -n , idRead);
}
```

Sommaire > Manipulation de fichier

Copier un fichier

Auteurs : Nicolas Joseph ,

Copie un fichier *source* vers *dest*.

```c
#include <stdio.h>
#include <string.h>

int file_copy (const char *source, const char *dest)
{
  int ret = 0;
  FILE *src = NULL;
  FILE *dst = NULL;
  char buffer[BUFSIZ];

  src = fopen (source, "r");
  if (src)
  {
    dst = fopen (dest, "w");
    if (dst)
    {
      while (fgets (buffer, BUFSIZ, src))
      {
         fprintf (dst, "%s", buffer);
      }

      if (ferror (src))
      {
        fprintf (stderr, "Erreur lors de la lecture du fichier source : %s\n", source);
        ret = -3;
      }

      if (ferror (dst))
      {
        fprintf (stderr, "Erreur lors de l'ecriture du fichier dst : %s\n", dest);
        ret = -4;
      }

      fclose (src), src = NULL;
      fclose (dst), dst = NULL;
    }
    else
    {
      fprintf (stderr, "Impossible d'ouvrir le fichier dest : %s\n", dest);
      fclose (src);
      ret = -2;
    }
  }
  else
  {
    fprintf (stderr, "Impossible d'ouvrir le fichier source : %s\n", source);
    ret = -1;
  }
  return ret;
}
```

Compter le nombre de lignes d'un fichier

Auteurs : Emmanuel Delahaye ,

```c
#include <stdio.h>
```

```
int nb_lignes (FILE *fp)
{
  int n=0, c;

  while ((c = fgetc(fp)) != EOF)
  {
    if (c == '\n')
    {
      n++;
    }
  }
  return n;
}
```

Ce code compte le nombre de sauts de ligne contenus dans le fichier, par conséquent si la dernière ligne n'est pas complète la fonction n'en tiendra pas compte.

Gestion des fichiers clés/valeurs

Auteurs : Franck.H ,

C_IniFile est un module qui sert à créer et gérer des fichiers de configuration dans le même style que ceux de Windows. Ceci étant, ce code est indépendant du système d'exploitation, il est écrit en C Standard. Ce code permet donc de créer des fichiers de configuration, d'en ajouter des données par paire: clé/valeur et aussi de créer des groupes d'options (ou sections), de récupérer des données enregistrées et de supprimer des sections entières ou tout simplement une donnée bien précise d'après le nom de sa clé.
Cette source dans sa version 3.0, est une version très stable est testé dans un projet de petit système d'exploitation, voici sa page officielle: http://franckh.developpez.com/cinifile/

ftp://ftp-developpez.com/c/sources/c/C_IniFile.zip

Gestion des fichiers au format .ini

Auteurs : gl ,

Une seconde bibliothèque de gestion de fichiers au format ini, qui cette fois utilise les extensions Windows si elles sont disponibles. Nécessite ini.zip, str.zip et bool.zip.

ftp://ftp-developpez.com/c/sources/c/ini.zip

Lire une ligne d'un fichier

Auteurs : Nicolas Joseph ,

Cette fonction a pour but de lire la prochaine ligne du flux d'entrée *stream* et de la placer dans une zone de mémoire allouée dynamiquement dont l'adresse est stockée à l'adresse *pp_line*. En cas d'erreur ou de fin de fichier, la fonction retourne NULL.

```
char *get_next_line (FILE *stream, char **pp_line)
{
  if (stream != NULL && pp_line != NULL)
  {
    void *line = NULL;
    char tmp[BUFSIZ] = "";
    size_t size = 1;

    *pp_line = NULL;
```

```
    while (fgets (tmp, BUFSIZ, stream) != NULL)
    {
      size += BUFSIZ;
      line = realloc (*pp_line, sizeof (**pp_line) * size);
      if (line != NULL)
      {
        if (*pp_line == NULL)
        {
          ((char *)line)[0] = '\0';
        }
        *pp_line = line;
        line = NULL;
        strcat (*pp_line, tmp);
        if ((*pp_line)[strlen (*pp_line)-1] == '\n')
        {
          (*pp_line)[strlen (*pp_line)-1] = '\0';
          break;
        }
      }
      else
      {
        free (*pp_line), *pp_line = NULL;
      }
    }
  }
  return *pp_line;
}
```

Comment savoir si un fichier est vide ?

Auteurs : Foobar1329 ,

```
#include <stdio.h>

int isEmpty (char const *pFilePathStr)
{
  int rc = 0;

  if (pFilePathStr)
  {
    FILE * fp = NULL;

    fp = fopen(pFilePathStr, "r");
    if (fp)
    {
      rc = ( fgetc(fp) == EOF && !ferror(fp) );
      fclose(fp), fp = NULL;
    }
  }
  return rc;
}
```

Compter le nombre de mots dans un fichier

Auteurs : Franck.H ,

Cette fonction permet de compter le nombre de mots se trouvant dans un fichier.

```
#include <stdio.h>
#include <ctype.h>
```

```c
unsigned long file_how_much_word (const char * filename)
{
   FILE * p_file = NULL;
   long   cnt_w  = 0;


   if (filename != NULL)
   {
      p_file = fopen (filename, "r");

      if (p_file != NULL)
      {
         int c = 0;
         int mark = 0;


         while ((c = fgetc (p_file)) != EOF)
         {
            if (! isspace (c) && mark == 0)
            {
               cnt_w++;
               mark = 1;
            }
            else if (isspace (c))
            {
               mark = 0;
            }
         }

         fclose (p_file);
      }
   }


   return cnt_w;
}
```

Opérations sur les fichiers

Auteurs : Franck.H ,

Ce module permet de faire quelques opérations sur les fichiers à accès séquentiel, il propose les opérations suivantes:

- **Recherche du nombre de lignes**
- **Lecture d'une ligne par son numéro**
- **Suppression d'une ligne par son numéro**
- **Suppression d'une ligne par son contenu**
- **Recherche d'une ligne par une sous-chaîne**
- **Copie de fichiers**
- **Concaténation de fichiers**
- **Nombre d'occurrences d'une lettre**
- **Nombre d'occurrences d'un mot**
- **Test si un fichier existe**

ftp://ftp-developpez.com/c/sources/c/C_File.zip

Lister les répertoires et sous-répertoires

Auteurs : Skyrunner ,

Ce programme permet de lister les répertoires et les sous-répertoires ainsi que les fichiers qui y sont stockés.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <dirent.h>

void usage(char *name)
{
   printf("Usage : %s [directory]\n", name);
}

int lister(char *directory)
{
   DIR *dir = NULL;
   struct dirent *file = NULL;

   if((dir = opendir(directory)) == NULL)
   {
      return EXIT_FAILURE;
   }

   printf("Repertoire %s :\n", directory);

   while((file = readdir(dir)) != NULL)
   {
      if(strcmp(file->d_name, ".") && strcmp(file->d_name, ".."))
      {
         if(file->d_type == DT_DIR)
         {
            lister(strncat(directory, file->d_name, 256));
         }
         else
         {
            printf("\t%s\n", file->d_name);
         }
      }
   }

   printf("Fin repertoire %s\n", directory);

   closedir(dir);

   return EXIT_SUCCESS;
}

int main(int argc, char **argv)
{
   char directory[256];

   if(argc < 2)
   {
      usage(argv[0]);
      return EXIT_FAILURE;
   }

   strncpy(directory, argv[1], 256);

   return lister(directory);
```

```
}
```

Analyse récursive d'un répertoire

Auteurs : cledesol ,

Programme qui analyse récursivement un répertoire et fournit dans un fichier le nom des fichiers, leur taille, la date de dernier accès et la date de dernière modification.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <dirent.h>
#include <sys/stat.h>

#ifdef WIN32
#define FILE_SEPARATOR "\\"
#else
#define FILE_SEPARATOR "/"
#endif

#define MEMERROR fprintf (stderr,"%s %d : malloc() failed\n",__FILE__,__LINE__)

/*
  pour lancer le prgm :

 1e argument de l'executable : nom du repertoire a scanner
 2e argument de l'executable : fichier de sortie dans lequel l'analyse du scan sera ecrite

  Format du fichier de résultats:
  Nom_Fichier|Taille_Fichier|Date dernier accès|Date dernière modification.

  Ces champs dates sont extraits directement de la structure stat. Ce sont donc
  des time_t (délai écoulé depuis le 1er janvier 1970).
 */

static FILE *pFileOut;

static void usage(char *);
static void readRecursif(char *, char *);


int main(int argc, char **argv)
{

   if(argc < 3) {
      usage(argv[0]);
      return EXIT_FAILURE;
   }
   if ((pFileOut = fopen(argv[2], "w")) == NULL) {
       printf("\nUnable to open: %s\n", argv[2]);
       return EXIT_FAILURE;
   }
   readRecursif(argv[1], NULL);
   fclose(pFileOut); pFileOut=NULL;

   return(EXIT_SUCCESS);
}

static void usage(char *name)
{
   printf("Usage : %s directory fileOut\n", name);
```

```
}

static void readRecursif(char *parent, char *child) {
  DIR *dir = NULL;
  struct dirent *file = NULL;
  struct stat infos;
  char *path;
  char *filePath;

  if (child != NULL)
      path = malloc(strlen(parent) + strlen(child) + 2);
  else
      path = malloc(strlen(parent) + 1);

  if(path==NULL)
  {
    MEMERROR;
    exit(EXIT_FAILURE);
  }

  if (child != NULL)
    sprintf(path, "%s%s%s", parent, FILE_SEPARATOR, child);
  else
    sprintf(path, "%s", parent);

  printf("Analyzing : %s\n", path);

  if((dir = opendir(path)) != NULL)
  {
    while((file = readdir(dir)) != NULL)
    {
      if(strcmp(file->d_name, ".") && strcmp(file->d_name, ".."))
      {
        filePath = malloc(strlen(path) + strlen(file->d_name) + 2);
 if(filePath==NULL)
        {
          MEMERROR;
          exit(EXIT_FAILURE);
        }
        sprintf(filePath, "%s%s%s", path, FILE_SEPARATOR, file->d_name);
        stat(filePath, &infos);

        if (S_ISDIR(infos.st_mode))
          readRecursif(path, file->d_name);
 else
          fprintf(pFileOut, "%s|%lu|%lu|%lu\n", filePath, infos.st_size, infos.st_atime,
 infos.st_mtime);

        free(filePath); filePath=NULL;
      }
    }
  }

  closedir(dir); dir=NULL;
  free(path); path=NULL;

}
```

Copier une chaîne de caractères

Auteurs : Nicolas Joseph ,

```
#include <stdlib.h>
char *str_dup (const char *src)
{
  char *dest = NULL;

  if (src != NULL)
  {
    dest = malloc ((strlen (src) + 1) * sizeof (*dest));
    if (dest != NULL)
    {
      strcpy (dest, src);
    }
  }
  return dest;
}
```

Créer une chaîne de caractères formatée

Auteurs : Nicolas Joseph ,

Cette fonction permet de créer une nouvelle chaîne de caractères en utilisant les spécificateurs de format à la *printf*.

```
#include <stdio.h>

char *str_dup_printf (const char *format, ...)
{
  char *s = NULL;

  if (format)
  {
    char t[1];
    va_list pa;
    size_t size = 0;

    va_start (pa, format);
    size = vsnprintf (t, 1, format, pa);
    size++;
    s = malloc (sizeof (*s) * size);
    if (s)
    {
      vsnprintf (s, size, format, pa);
    }
  }
  return s;
}
```

Extraire une sous-chaîne d'une chaîne de caractères

Auteurs : D[r]eadLock ,

***subString* retourne la sous-chaîne de *chaine* comprise entre les indices *debut* et *fin*.**

```
#include <string.h>

int subString (const char *chaine, int debut, int fin, char *result)
```

```
{
  result[fin+1-debut] = '\0';
  memcpy (result, (char *)chaine+debut, fin+1-debut);
  return (fin+1-debut);
}
```

Gestion de chaînes de caractères

Auteurs : Franck.H ,

C_Str est un objet String qui connaît lui même la taille de la chaîne de caractères qu'il contient.
Outre la faculté de connaître automatiquement la taille de sa chaîne, il dispose de plus de 30 fonctions de traitement sur les chaînes de caractères. Ces fonctions sont des idées reprises de différents langages (Java, Python) : concaténation entre l'objet string et de multiples chaînes de caractères, concaténation entre plusieurs objets string, différentes fonctions de changement de la casse de la chaîne, inversion de la chaîne, clonage de l'objet string, etc...
L'objet string est dynamique, ce qui veut dire que si vous changez la chaîne de caractères au sein même de l'objet avec les fonctions fournies, sa taille est mise à jour. La chaîne contenue dans un objet string est terminée avec un zéro de fin de chaîne.
Page officielle de cette source: http://franckh.developpez.com/cstr/

ftp://ftp-developpez.com/c/sources/c/C_Str.zip

Remplacer toutes les occurrences d'une sous-chaîne

Auteurs : haypo ,

Remplace toutes les occurences de *Avant* par *Apres* dans la chaine *txt*, puis renvoie un pointeur sur la nouvelle chaîne créée. Renvoie NULL si *txt* ne contient aucune occurence de *Avant*.

```
#include <string.h>
#include <stdlib.h>

char *str_replace (const char *txt, const char *Avant, const char *Apres)
{
  const char *pos;      /* Position d'une occurance de Avant dans txt */
  char *TxtRetour;      /* La chaine retournée */
  size_t PosTxtRetour;  /* Position du prochain caractère à écrire */
                        /* dans TxtRetour */
  size_t Long;          /* Long d'une chaine à écrire dans TxtRetour */
  size_t TailleAllouee; /* Taille allouée à TxtRetour */

  /* Cherche la première occurence */
  pos = strstr (txt, Avant);

  /* Aucune occurrences : renvoie simplement une copie de la chaine */
  if (pos == NULL)
  {
    return NULL;
  }

  /* Alloue une nouvelle chaine */
  Long = (size_t)pos - (size_t)txt;
  TailleAllouee = Long + strlen (Apres) + 1;
  TxtRetour = malloc (TailleAllouee);
  PosTxtRetour = 0;

  /* Copie la première partie de la chaîne sans occurrence */
  strncpy (TxtRetour + PosTxtRetour, txt, Long);
  PosTxtRetour += Long;
  txt = pos + strlen (Avant);
```

```
  /* Ajoute la chaîne de remplacement Apres */
  Long = strlen (Apres);
  strncpy (TxtRetour + PosTxtRetour, Apres, Long);
  PosTxtRetour += Long;

  /* Cherche la prochaine occurrence */
  pos = strstr (txt, Avant);
  while (pos != NULL)
  {
    /* Agrandit la chaine */
    Long = (size_t)pos - (size_t)txt;
    TailleAllouee += Long + strlen (Apres);
    TxtRetour = (char *)realloc (TxtRetour, TailleAllouee);

    /* Copie ce qu'il y a entre la dernier occurrence et la nouvelle */
    strncpy (TxtRetour + PosTxtRetour, txt, Long);
    PosTxtRetour += Long;

    /* Passe l'occurrence */
    txt = pos + strlen (Avant);

    /* Ajoute la chaîne de remplacement */
    Long = strlen (Apres);
    strncpy (TxtRetour + PosTxtRetour, Apres, Long);
    PosTxtRetour += Long;

    /* Cherche la prochaine occurrence */
    pos = strstr (txt, Avant);
  }

  /* Ajoute le reste de la chaîne (il reste au moins '\0') */
  Long = strlen (txt) + 1;
  TailleAllouee += Long;
  TxtRetour = realloc (TxtRetour, TailleAllouee);
  strncpy (TxtRetour + PosTxtRetour, txt, Long);
  return TxtRetour;
}
```

Remplacer une partie d'une chaîne

Auteurs : rolkA ,

Ce code permet de rechercher une sous-chaîne dans une chaîne de caractères et, si elle est trouvée, de la remplacer par une autre.

```
#include <string.h>

char *RemplacerFragment (char *source, const char *vieux, const char *nouveau)
{
  char *original = source;
  char temp[256];
  int ancienne_long = strlen (vieux);
  int i, j, k, place = -1;

  for (i = 0; source[i] && (place == -1); ++i)
  {
    for (j = i, k = 0; source[j] == vieux[k]; j++, k++)
    {
      if (!vieux[k+1])
      {
        place = i;
      }
    }
```

```
    }
    if (place != -1)
    {
      for (j=0; j<place; j++)
      {
        temp[j] = source[j];
      }
      for (i=0; nouveau[i]; i++, j++)
      {
        temp[j] = nouveau[i];
      }
      for (k = place + ancienne_long; source[k]; k++, j++)
      {
        temp[j] = source[k];
      }
      temp[j] = 0;
      for (i=0; source[i] = temp[i]; i++)
      {
      }
    }
    return original;
  }
```

Gestion des chaînes de caractères

Auteurs : gl ,

Le fichier str.c propose des fonctions pour supprimer les espaces en début et fin de chaîne, pour mettre une chaîne de caractères en minuscule ou en majuscule.

ftp://ftp-developpez.com/c/sources/c/str.zip

Modifier la casse d'une chaîne de caractère

Auteurs : Nicolas Joseph ,

Le fichier d'entête ctype.h propose les fonctions tolower et toupper pour mettre un caractère respectivement en minuscule et en majuscule, il est intéressant de proposer la même chose mais pour une chaîne de caractères :

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

char *str_tolower (const char *ct)
{
  char *s = NULL;

  if (ct)
  {
    int i;

    s = malloc (sizeof (*s) * (strlen (ct) + 1));
    if (s)
    {
      for (i = 0; ct[i]; i++)
      {
        s[i] = tolower (ct[i]);
      }
      s[i] = '\0';
    }
  }
  return s;
```

```
}

char *str_toupper (const char *ct)
{
  char *s = NULL;

  if (ct)
  {
    int i;

    s = malloc (sizeof (*s) * (strlen (ct) + 1));
    if (s)
    {
      for (i = 0; ct[i]; i++)
      {
        s[i] = toupper (ct[i]);
      }
      s[i] = '\0';
    }
  }
  return s;
}
```

Connaître l'indice d'une sous-chaîne

Auteurs : Nicolas Joseph ,

Il existe la fonction strstr qui permet de trouver l'adresse d'une sous-chaîne mais je trouve plus intéressant de connaître l'indice de celle-ci dans le tableau :

```
int str_istr (const char *cs, const char *ct)
{
  int index = -1;

  if (cs && ct)
  {
    char *ptr_pos = NULL;

    ptr_pos = strstr (cs, ct);
    if (ptr_pos)
    {
      index = ptr_pos - cs;
    }
  }
  return index;
}
```

Il faut vérifier le retour de la fonction car si la sous-chaîne n'est pas trouvée, l'indice vaut -1 ce qui provoquera un comportement indéfini en cas d'utilisation de l'indice dans un tableau.

Découper une chaîne

Auteurs : Nicolas Joseph ,

Cette fonction permet de découper une chaîne de caractère suivant un délimiteur et de placer chaque sous-chaîne dans un tableau terminé par NULL.

```
char **str_split (const char *cs, const char *delim, size_t *p_size)
{
  size_t size = 0;
```

```
  char **ret = NULL;

  if (cs != NULL && delim != NULL)
  {
    int start = 0;
    int end = 0;
    size_t length = 0;
    const char *t = NULL;
    void *tmp = NULL;

    while ((t = strstr (&cs[start], delim)))
    {
      end = t - cs;
      length = end - start;
      if (length > 0)
      {
        size++;
        tmp = realloc (ret, sizeof (*ret) * size);
        if (tmp != NULL)
        {
          ret = tmp;
          ret[size-1] = malloc (sizeof (**ret) * (length + 1));
          if (ret[size-1])
          {
            strncpy (ret[size-1], &cs[start], length);
            ret[size-1][length] = '\0';
          }
          else
          {
            fprintf (stderr, "Memoire insufisante\n");
            exit (EXIT_FAILURE);
          }
        }
        else
        {
          fprintf (stderr, "Memoire insufisante\n");
          exit (EXIT_FAILURE);
        }
      }
      start = end + 1;
    }
    size += 2;
    tmp = realloc (ret, sizeof (*ret) * size);
    if (tmp != NULL)
    {
      ret = tmp;
      ret[size-1] = NULL;
      length = strlen (cs) - start;
      if (length > 0)
      {
        ret[size-2] = malloc (sizeof (**ret) * (length + 1));
        if (ret[size-2])
        {
          strcpy (ret[size-2], &cs[start]);
        }
      }
      else
      {
        ret[size-2] = NULL;
      }
    }
    else
    {
      fprintf (stderr, "Memoire insufisante\n");
      exit (EXIT_FAILURE);
    }
  }
  if (p_size)
```

```
    {
      *p_size = size-1;
    }
    return ret;
  }
```

⚠ *La variable s passée en paramètre est modifiée par la fonction strtok. De plus le tableau de pointeurs renvoyé par notre fonction fait référence à la chaîne passée en paramètre, par conséquent, elle ne doit pas être modifiée ni détruite si vous utilisez le tableau de sous-chaînes.*

Fusionner plusieurs chaînes de caractères

Auteurs : Nicolas Joseph ,

Cette fonction réunit des chaînes de caractères grâce à un séparateur.

```
char *str_join (char **field, const char *delim)
{
  char *ct = NULL;

  if (field != NULL && delim != NULL)
  {
    int i;
    size_t size = 0;
    size_t size_delim = 0;

    size_delim = strlen (delim);
    for (i = 0; field[i] != NULL; i++)
    {
      void *tmp = NULL;

      size += strlen (field[i]) + size_delim + 1;
      tmp = realloc (ct, sizeof (*ct) * size);
      if (tmp != NULL)
      {
        if (ct == NULL)
        {
          ct = tmp;
          strcpy (ct, field[i]);
        }
        else
        {
          ct = tmp;
          strcat (ct, delim);
          strcat (ct, field[i]);
        }
      }
      else
      {
        free (ct), ct = NULL;
        break;
      }
    }
  }
  return ct;
}
```

Pour pouvoir gérer un nombre variable d'arguments, la liste doit être terminée par la valeur NULL.

Eliminer les espaces superflus

Auteurs : Jean Christophe Beyler ,

Cette fonction supprime les espaces superflus dans une chaîne de caractères.

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

char* supprime (const char *src)
{
  char *dest = NULL, *res;
  int n = strlen (src);
  int i, dansunmot, j = 0;

  dest = malloc (n+1);
  if(dest == NULL)
  {
    return NULL;
  }

  /* Cherche le premier non espace */
  i = 0;
  while ((i < n) && (src[i] == ' '))
  {
    i++;
  }

  /* On est dans un mot */
  dansunmot = 1;

  /* Jusqu'a la fin de la chaine */
  while (i < n)
  {
    /* Si c'est un espace, on fait remarquer qu'on n'est plus dans un mot */
    if (src[i] == ' ')
    {
      dansunmot = 0;
    }
    else
    {
      /* Si on n'est pas dans un mot, on met un espace */
      if (dansunmot == 0)
      {
        dest[j]=' ';
        j++;
      }
      /* On est dans un mot */
      dansunmot = 1;
      /* On copie la lettre */
      dest[j] = src[i];
      /* On incremente l'indice de destination */
      j++;
    }
    /* On incremente l'indice de la source */
    i++;
  }

  /* Mettre le '\0' a la fin */
  dest[j] = '\0';
```

```
    j++;

    /* On realloue pour avoir la bonne taille */
    res = realloc (dest, j);
    if (res)
    {
      return res;
    }
    else
    {
      return dest;
    }
  }
```

Enlever le début d'une chaîne

Auteurs : Franck.H ,

Cette fonction permet de supprimer le début de la chaîne passée en paramètre par un simple procédé de décalage des caractères et complétion avec des zéros de fin de chaîne. La fonction permet de faire ce remplacement à partir d'un caractère délimiteur qui est alors passé en tant que second argument. La chaîne passée en argument est modifiée !

```
void remove_begin (char * str, const char delim)
{
   if (str != NULL)
   {
      char * p1 = str;
      char * p2 = str;


      /* Recherche du caractre contenu dans delim */
      while (*p2++ != delim);


      /* Deplacement des caractères de la chaine. */
      do
      {
         *p1++ = *p2++;
      }
      while (*p2);


      /* On rempli le reste avec des zeros de fin. */
      do
      {
         *p1++ = 0;
      }
      while (p1 != p2);
   }
}
```

Concaténer des chaînes

Auteurs : Mr_Chut , Emmanuel Delahaye ,

Cette fonction permet de concaténer un nombre indéfini de chaînes de caractère comme le fait la fonction standard strcat.

```
static char *str_concat (char *cs,...)
```

```
{
   char *s = NULL;
   size_t size = 1;
   {
      const char *ct;
      va_list va;
      va_start (va, cs);
      while ((ct = va_arg (va, char *)) != NULL)
      {
         size += strlen (ct);
      }
      va_end (va);
   }

   s = malloc (size);

   if (s == NULL)
   {
      printf ("in str_concat : malloc() failed.\n");
   }
   else
   {
      const char *ct;
      va_list va;
      va_start (va, cs);
      strcpy (s, cs);

      while ((ct = va_arg (va, char *)) != NULL)
      {
         strcat (s, ct);
      }
      va_end (va);
   }
   return s;
}
```

Comment faire une rotation des bits vers la droite ?

Auteurs : Jean-Marc.Bourguet ,

```
unsigned rotate_right (unsigned x, int count)
{
   /* assume unsigned has no padding bits */
   int const unsigned_bit_count = CHAR_BIT * sizeof unsigned;
   count %= unsigned_bit_count;

   if (count < 0)
      count += unsigned_bit_count;

   /* but don't assume anything about << unsigned_bit_count */
   if (count == 0)
      return x;
   else
      return (x >> count) | (x << unsigned_bit_count-count);
}
```

Comment récupérer le motif binaire d'une variable ?

Auteurs : Franck.H ,

La fonction *get_binary* permet de récupérer le motif binaire d'une variable de type unsigned int *(les types signés ne sont pas recommandés pour ce genre de manipulations)*. Elle remplit donc un *tableau de char* dynamique *(ce qui implique une libération avec free)*.
Le motif binaire est inséré dans le tableau dans le sens de lecture normal d'un être humain soit de gauche à droite *(le motif binaire est récupéré de droite à gauche)*. Le second paramètre de la fonction permet d'accéder à l'adresse d'une variable pour pouvoir également récupérer la taille du tableau pour pouvoir faire éventuellement un futur parcours de celui-ci !

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>


char * get_binary (unsigned int val, size_t * tab_size)
{
   size_t size = sizeof (unsigned int) * CHAR_BIT;    /* Taille du type.      */
   size_t i = 0;
   char * tab = NULL;


   tab = malloc (size);

   if (tab != NULL)
   {
      for (i = 0; i < size; i++)
      {
         /*
          * On recupere les bits de la variable un par un en commencant
          * par les bits de poids faible (bits de droite) mais on les insere
          * dans le sens de lecture normal d'un etre humain soit de gauche
          * a droite.
          */
         tab[size - i - 1] = (val << (size - i - 1)) >> (size - 1);
      }
```

```
      if (tab_size != NULL)
      {
         *tab_size = size;
      }
   }


   return tab;
}


int main (void)
{
   char * tab = NULL;
   size_t size = 0;
   size_t i = 0;


   tab = get_binary (29, & size);

   if (tab != NULL)
   {
      for (i = 0; i < size; i++)
      {
         printf ("%d", tab[i]);
      }
      printf ("\n");
   }

   free (tab);


   return EXIT_SUCCESS;
}
```

Recherche dichotomique pour la puissance de 2 supérieure

Auteurs : Jean Christophe Beyler ,

La fonction pow2sup fait une recherche dichotomique pour la puissance de 2 supérieure à celle passée en argument. En utilisant des décalages de bits, elle calcule rapidement le bon résultat. Enfin, le programme exécute la fonction 30 fois pour montrer son fonctionnement.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>


unsigned int pow2sup (unsigned int nbr)
{
    int bs,bi;
    int min = 0;
    int max = 31;

    int mid = (min + max)/2;

    /* Borne inferieure */
    if(nbr <= 1) {
        return 1;
    }

    /* Borne superieure */
    if( nbr > (1<<30)){
```

```
        return (1<<31);
    }

    do
    {
        bs = (1<<mid);
        bi = (1<<(mid-1));

        if(nbr < bi) {
            max = mid;
            mid = (min+max)/2;
        }
        else {
            if(nbr > bs) {
                min = mid;
                mid = (min+max)/2;
            }
            else {
                if(bi == nbr) {
                    return bi;
                }
                return bs;
            }
        }
    }
    while (min != max);

    return 0;
}

int main(void)
{
    int i;
    unsigned int nbr,sol;

    srand (time(NULL));

    for(i=0;i<30;i++) {
        nbr = rand();
        sol = pow2sup(nbr);

        if ((sol >= nbr) && (sol/2 < nbr)) {
            printf("%u -> %u \n",nbr,sol);
        }
        else {
            printf("Erreur avec %u -> %u\n",nbr, sol);
        }
    }

    return EXIT_SUCCESS;
}
```

Calcul de la racine carrée d'un nombre

Auteurs : Anomaly ,

Ce code permet de calculer la racine carrée entière d'un nombre entier (>= 1) sans utiliser les flottants, en utilisant la méthode des approximations successives. Cette fonction est plus efficace que sqrt() avec gcc sans optimisations. Par contre, avec les optimisations, sqrt() écrase cette fonction. L'avantage principal est ici d'éviter d'utiliser les flottants et la bibliothèque mathématique et d'assurer une bonne vitesse de calcul sur les machines avec des co-processeurs faibles ou sans co-processeur.

```
int racine(int nombre)
{
  int r1, r2;

  r1 = (nombre + 1) / 2;

  for (;;)
  {
    r2 = (r1 + nombre / r1) / 2;

    if (r2 >= r1)
      return r1;

    r1 = r2;
  }
}
```

Résoudre une équation du second degré

Auteurs : Emmanuel Delahaye ,

```
#include <stdio.h>
#include <math.h>               /* pour la racine ==>  sqrt(); */
#include <string.h>
#include <errno.h>
#include <stdlib.h>

/* saisie d'une ligne (chaine de caracteres) */
int get_s (char *const s, size_t const size)
{
  /* lecture d'une ligne */
  int err = fgets (s, size, stdin) == NULL;

  if (!err)
  {
    /* s'il n'y a pas eu d'erreur de lecture
     * recherche du '\n'' final
     */
    char *p = strchr (s, '\n');

    if (p != NULL)
    {
      /* Si on l'a trouve, destruction */
       *p = 0;
    }
    else
    {
      /* sinon, c'est que la chaine de destination est trop courte.
       * lecture de tous les caracteres non lus (nettoyage de stdin)
       */
      int c;
```

```
      while ((c = getchar ()) != '\n' && c != EOF)
      {
      }
      err = 1;              /* ligne incomplete */
    }
  }
  return err;
}

/* saisie d'un flottant */
int get_d (double *const p)
{
  int err = 1;

  if (p != NULL)
  {
    /* Si on a bien passe un pointeur non NULL */
    char s[64];

    /* lecture d'une ligne */
    err = get_s (s, sizeof s);

/*
 * L'etat de la globale 'C' errno est incertain.
 * On le force a 0 (= pas d'erreur).
 * En effet au demarrage du programme, errno est initialisee a 0,
 * mais aucune fonction standard ne positionne errno a 0, alors
 * que beaucoup d'entre elles, (y compris des fonctions
 * non-standard ou applicatives) peuvent le modifier a une autre valeur.
 */
    errno = 0;

    if (*s != 0)
    {
      /* Si la chaine saisie n'est pas vide ("") */
      /* ce pointeur sera mis a jour par strtod().
       * Il indiquera la position du premier caractere non converti
       */
      char *p_end;

      /* Conversion de la chaine en double */
      double x = strtod (s, &p_end);

      /* On verifie que tout s'est bien passe :
       * - p_end pointe bien sur 0, c'est a dire la fin de la chaine
       * - errno n'indique pas d'erreur
       */
      if (*p_end == 0 && errno == 0)
      {
        /* on retourne la valeur a l'appelant */
        *p = x;
      }
      else
      {
        err = 1;            /* erreur de conversion */
      }
    }
    else
    {
      err = 1;              /* chaine vide */
    }
  }
  return err;
}

int saisie (double *const pa, double *const pb, double *const pc)
{
  int err;
```

```c
  printf ("ENTRER A:");
  err = get_d (pa);

  if (!err)
  {
    printf ("ENTRER B:");
    err = get_d (pb);

    if (!err)
    {
      printf ("ENTRER C:");

      if (!err)
      {
        err = get_d (pc);
      }
    }
  }
  return err;
}

void calcul (double const a, double const b, double const c)
{
  double delta = ((b * b) - (4 * a * c));

  printf ("delta=%f\n", delta);

  if (delta < 0)
  {
    printf ("il n'y a pas de solution Reelle\n");
  }
  else if (delta == 0)
  {
    printf ("X=%f\n", (-b) / (2 * a));
  }
  else
  {
    printf ("X1=%f\n"
            "X2=%f\n"
            ,((-b) - (sqrt (delta))) / (2 * a)
            ,((-b) + (sqrt (delta))) / (2 * a)
       );
  }
}

int main (void)
{
  double a, b, c;

  int err = saisie (&a, &b, &c);
  if (!err)
  {
    calcul (a, b, c);
  }
  return 0;
}
```

Comment savoir si un nombre est premier ?

Auteurs : Jean Christophe Beyler ,

Ressemblant à un hybride entre le crible d'Eratosthène et la méthode classique, cette solution utilise les nombres premiers en dessous de 100 pour savoir si un nombre est premier ou non. Si jamais on a fini le parcours, on utilise la méthode classique...

On suppose que nbr >= 1

```c
#include <stdio.h>
#include <math.h>

int estPremier (int nbr)
{
  /*Les nombres premiers < 100*/
  static int prems[] = {2,3,5,7,11,13,17,19,23,29,31,37,41,
                        43,47,53,59,61,67,71,73,79,83,89,97};
  int i, n;
  double d;


  /* On suppose que 1 est premier */
  if (nbr == 1)
  {
    return 1;
  }

  n = sizeof (prems) / sizeof (*prems);

  /* D'abord on regarde si n est divisible par les nombres premiers dans le tableau */
  for (i = 0; i < n; i++)
  {
    if (nbr == prems[i])
    {
      return 1;
    }
    if (nbr % prems[i] == 0)
    {
      return 0;
    }
  }

  /* Ensuite, on doit regarder a partir du dernier element du tableau+2 jusqu'a sqrt(nbr)... */
  d = sqrt (nbr) + 0.5; /* Le 0.5 permet de tester si c'est un carre parfait... */
  i = prems[i-1] + 2;

  while (i < d)
  {
    if (nbr % i == 0)
    {
      return 0;
    }
    i += 2;
  }
  return 1;
}

int main(void)
{
  int i;

  for (i = 101; i < 500; i++)
  {
    if (estPremier (i))
    {
      printf ("%d\n",i);
    }
  }
  return 0;
```

```
}
```

Calcul du plus grand diviseur commun de deux entiers relatifs

Auteurs : odsen.s ,

Cette fonction renvoie le PGCD (plus grand diviseur commun) de deux entiers relatifs, selon l'algorithme d'Euclide.
La division euclidienne s'écrit comme suit :
dividende = diviseur * quotient + reste
avec
0 <= reste < diviseur
Exemple :
Division euclidienne de 20 par 3
20 = 3 * 6 + 2
avec
0 <= 2< 3
En utilisant des divisions euclidiennes successives, on peut trouver le PGCD de deux entiers relatifs.

```
unsigned long pgcd(long a, long b, int voirAlgorithme)
{
    long dividende = labs(a); /* le dividende contient la valeur absolue de a */
    long diviseur = labs(b);  /* le diviseur contient la valeur absolue de b  */
    long quotient;
    long reste;

    int fin = 0;


    /*
     * on ne calcule le pgcd de deux nombres que s'ils sont différents de zéro
     */
    if(a != 0 && b != 0)
    {
       while(!fin)
       {
           /* On applique la division euclidienne */
           reste = dividende % diviseur;
           quotient = (dividende - reste) / diviseur;


           /*
            * Si l'affichage de l'algorithme est souhaité,
            * on affiche chaque ligne
            */
           if (voirAlgorithme)
           {
               printf (
                  "%ld = %ld * %ld + %ld\n",
                  dividende, diviseur, quotient, reste
               );
           }

           /* Si le reste est différent de 0, on continue l'algorithme */
           if(reste !=0)
           {
               dividende = diviseur;
               diviseur = reste;
           }
           else
           {
               fin = 1;
```

```
            }
        }
    }
    else
    {
        /* Erreur ... */
        diviseur = 0;
    }


    return diviseur;
}
```

Interpreteur d'expression mathématique

Auteurs : Melem ,

Evaluateur d'expression mathématique passée sous forme de chaîne de caractères

ftp://ftp-developpez.com/c/sources/c/interpreteur.zip

[ncurses] Comment ne pas afficher à l'écran ce que l'on entre au clavier ?

Auteurs : Musaran ,

Ce code permet de ne pas afficher à l'écran les caractères saisis (ils sont remplacés par le caractère *mask*).
Nécessite *getch*, disponible dans la bibliothèque curses.

```c
#include <stdlib.h>

/* Saisit une ligne terminée par [Entrée] comme chaîne C, en masquant l'entrée.
   Renvoie la longueur de l'entrée. */
size_t getpassword (char *pbuff, size_t buffsize, char mask)
{
  char *pcur= pbuff;

  for (; buffsize>1; --buffsize)
  {
    *pcur = getch ();
    if (*pcur == '\r')
    {
      break;
    }
    putch (mask);
    pcur++;
  }
  *pcur = '\0';
  return pcur-pbuff;
}
```

Sommaire > Gestion dynamique de la mémoire

Gestion des allocations dynamiques de mémoire.

Auteurs : Franck.H ,

Gère les allocations dynamiques de mémoire faites par le biais des wrappers des fonctions standards.
Ce module garde une trace dans une liste chaînée, de toutes les adresses allouées dynamiquement ce qui lui permet lors de sa fermeture de libérer les éventuelles adresses qui aurait été oubliées d'être libérées par l'utilisateur.
D'autres possibilités sont offertes comme la création d'un log des allocations/réallocations/libérations.

ftp://ftp-developpez.com/c/sources/c/CMM.zip

Allouer dynamiquement un tableau à 2 dimensions

Auteurs : Franck.H ,

Cette fonction alloue un tableau 2D dynamique de chaînes de caractères.
Chaque chaîne est ici pourvue du même nombre de caractères. Si la fonction échoue, elle renvoie NULL.
Ce principe est applicable à tout type de données.

```
char ** char_tab_alloc (size_t n, size_t m)
{
   char     err = 0;
   char **  tab = malloc (n * sizeof (* tab));

   if (tab != NULL)
   {
      size_t i = 0;

      for (i = 0; i < n; i++)
      {
         tab[i] = malloc (m * sizeof (** tab));

         if (tab[i] == NULL)
         {
            err = 1;
            break;
         }
      }


      /*
       * Si une erreur d'allocation a eu lieu on supprime tout le
       * tableau et la fonction renvoie alors la valeur NULL.
       */
      if (err)
      {
         size_t i = 0;

         for (i = 0; i < n; i++)
         {
            if (tab[i] != NULL)
               free (tab[i]);
         }
         free (tab);
         tab = NULL;
      }
   }

   return tab;
```

```
}
```

Exemple d'utilisation des Threads

Auteurs : Jean Christophe Beyler ,

Programme montrant le fonctionnement et l'utilisation des Threads POSIX.1

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *thread_aff(void *arg)
{
    int *pi = arg;


    /* Si l'argument est NULL */
    if(pi == NULL) {
        return NULL;
    }

    /* Affichage de l'argument */
    printf("Nouveau thread avec un argument valant %d\n",*pi);

    /* Liberation de la memoire */
    free(pi), pi=NULL;


    return NULL;
}

void *thread_sommeEntiersVersDouble(void *arg)
{
    double *res = NULL;


    /* Recuperation de la valeur de l'argument */
    int *pi = arg;

    /* Si l'argument est NULL */
    if(pi == NULL) {
        return NULL;
    }

    /* Creation d'une section memoire pour le double */
    res = malloc(sizeof(*res));

    /* Gestion de l'erreur */
    if(res == NULL) {
        return res;
    }

    *res = *pi + *(pi+1);

    /* Liberation de la memoire */
    free(pi), pi=NULL;


    return res;
}


int main(void)
{
   pthread_t t1, t2;
```

```
    int *pi_t1 = NULL, *pi_t2 = NULL;
    int res;
    void *res_thread = NULL;
    double *d = NULL;


    /*
     * Pour passer un argument a un thread, il vaut mieux lui allouer
     * la memoire et passer le pointeur.
     */
    pi_t1 = malloc(sizeof(*pi_t1));

    if(pi_t1 == NULL) {
        printf("Erreur dans l'allocation de pi_t1\n");
        return EXIT_FAILURE;
    }
    *pi_t1 = 123;

    res = pthread_create(&t1, NULL, thread_aff, pi_t1);

    if (res != 0) {
        printf("Erreur dans la creation du thread 1 : %d\n",res);
    }

    /*
     * Pour passer un argument a un thread, il vaut mieux lui allouer
     * la memoire et passer le pointeur.
     */
    pi_t2 = malloc(2*sizeof(*pi_t2));

    if(pi_t2 == NULL) {
        printf("Erreur dans l'allocation de pi_t2\n");
        return EXIT_FAILURE;
    }
    pi_t2[0] = 35;
    pi_t2[1] = 14;

    res = pthread_create(&t2, NULL, thread_sommeEntiersVersDouble, pi_t2);

    if (res != 0) {
        printf("Erreur dans la creation du thread 1 : %d\n",res);
    }


    /*
     * On attend la fin du premier thread, le resultat du
     * thread ne nous interesse pas.
     */
    res = pthread_join(t1, NULL);

    if(res!=0) {
        printf("Erreur dans l'attente pour le premier thread\n");
    }


    /*
     * On attend la fin du deuxieme thread,
     * le resultat du thread nous interesse.
     */
    res = pthread_join(t2, &res_thread);

    if(res!=0) {
        printf("Erreur dans l'attente pour le deuxieme thread\n");
    }


    /* Afficher le resultat */
    d = res_thread;
```

```
    if(d != NULL) {
        printf("Resultat : %f\n",*d);
    }


    /*
     * Liberation de la memoire :
     *  - On ne libere pas pi_t1 ou pi_t2, c'est le travail des deux
     *    autres threads de le faire.
     */
    free(d);
    d = NULL;


    return EXIT_SUCCESS;
}
```

Exemple d'utilisation des fork

Auteurs : Jean Christophe Beyler ,

Programme montrant le fonctionnement et l'utilisation de fork.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>


void fct_aff(int arg)
{
   /* Affichage de l'argument */
   printf("Nouveau thfr avec un argument valant %d\n",arg);
}


double thfr_sommeEntiersVersDouble(int a, int b)
{
   return a + b;
}


void sortiePere(char *s, FILE *fr, FILE *fw)
{
   printf("%s",s);

   /* Fermeture des tubes */
   fclose(fr);
   fclose(fw);

   /* Attente des deux fils */
   wait(NULL);
   wait(NULL);
}

int main(void)
{
   pid_t p, p2;
   char buf[128];
   char *endptr = NULL;

   int pf[2], fp[2];
```

```
FILE *fr = NULL, *fw = NULL;


/* Un premier fork */
p = fork();

if (p) {
   /*
    * C'est le pere... Preparation du pipe pere -> fils
    */
   if(pipe(pf) == -1) {
      perror("Erreur avec la creation du tube pere -> fils : ");

      /* Attente du premier fils */
      wait(NULL);
      return EXIT_FAILURE;
   }

   /* Preparation du pipe fils -> pere */
   if(pipe(fp) == -1) {
      perror("Erreur avec la creation du tube fils -> pere : ");

      /* On ferme le premier pipe */
      close(pf[0]);
      close(pf[1]);

      /* Attente du premier fils */
      wait(NULL);
      return EXIT_FAILURE;
   }


   /* Deuxieme fork */
   p2 = fork();

   if(p2) {
      /*
       * C'est encore le pere...
       */
      int res;


      /* On recupere les FILE et on ferme ce dont on n'utilise plus */
      close(pf[0]);
      close(fp[1]);
      fr = fdopen(fp[0],"r");
      fw = fdopen(pf[1], "w");

      if( (fr == NULL) || (fw == NULL) ) {
         sortiePere("Erreur dans l'ouverture de fr et fw\n",fr,fw);
         return EXIT_FAILURE;
      }

      /* Ecriture des deux operandes */
      fprintf(fw, "%d %d\n", 12, 34);
      fflush(NULL);

      /* Attente de la reponse */
      if(fgets(buf, sizeof(buf), fr) == NULL) {
         sortiePere("Erreur dans la lecture du pipe\n",fr,fw);
         return EXIT_FAILURE;
      }

      res = strtol(buf, &endptr, 0);

      if( (buf[0] != '\0') && (*endptr != '\n') ) {
         sortiePere("Erreur dans la lecture de l'entier\n", fr, fw);
         return EXIT_FAILURE;
```

```
            }


            printf("Resultat %d\n",res);
            sortiePere("Pere se termine bien\n", fr, fw);
        }
        else {
            /*
             * C'est le deuxieme fils...
             */
            int op1, op2;


            /* On recupere les FILE et on ferme ce dont on n'utilise plus */
            close(pf[1]);
            close(fp[0]);
            fr = fdopen(pf[0],"r");
            fw = fdopen(fp[1], "w");


            /* On recupere deux entiers du pere */
            /* Attente de la reponse */
            if(fgets(buf, sizeof(buf), fr) == NULL) {
                printf("Probleme du fils a lire les operandes\n");

                /* On sort */
                fclose(fr);
                fclose(fw);

                return EXIT_FAILURE;
            }

            op1 = strtol(buf, &endptr, 0);

            if( (buf[0] != '\0') && (*endptr != ' ') ) {
                printf("Erreur du fils dans la lecture de l'entier\n");

                /* On sort */
                fclose(fr);
                fclose(fw);

                return EXIT_FAILURE;
            }

            op2 = strtol(endptr, &endptr, 0);

            if(*endptr != '\n') {
                printf("Erreur du fils dans la lecture du 2eme entier\n");
                /* On sort */
                fclose(fr);
                fclose(fw);

                return EXIT_FAILURE;
            }

            /* Ecriture du resultat */
            fprintf(fw,"%d\n",op1+op2);
            fflush(NULL);

            /* On sort */
            fclose(fr);
            fclose(fw);
        }
    }
    else {
        /* Sinon c'est le fils */
        printf("Bonjour, je suis le premier fils\n");
    }
```

```
    return EXIT_SUCCESS;
}
```

Utilisation de la mémoire partagée

Auteurs : Nicolas Joseph ,

Ce programme montre comment utiliser la mémoire partagée.

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/wait.h>


typedef struct
{
  int nb;
  const char *msg;
}
shared_s;


int main (int argc, char **argv)
{
  int id;
  int pid;
  shared_s *s = NULL;
  key_t cle;


  errno = 0;
  cle = ftok (argv[0], 'a');

  if (cle != -1)
  {
    errno = 0;
    id = shmget (cle, sizeof (shared_s), IPC_CREAT | 0666);

    if (id != -1)
    {
      errno = 0;
      s = shmat (id, NULL, 0);

      if (s != (void *)-1)
      {
        s->nb = 2;
        s->msg = "Shared hello world";

        errno = 0;
        pid = fork ();

        if (pid > 0)
        {
          /* Processus pere */
          int i;

          for (i = 0; i < s->nb; i++)
          {
            printf ("%s\n", s->msg);
```

```
          }

          waitpid (pid, NULL, 0);

          for (i = 0; i < s->nb; i++)
          {
            printf ("%s\n", s->msg);
          }
        }
        else if (pid == 0)
        {
          /* Processus fils */
          shared_s *ss = NULL;

          /*
           * On laisse le temps au processus pere d'afficher le message
           * d'origine.
           */
          sleep (1);
          ss = shmat (id, NULL, 0);

          if (ss != (void *)-1)
          {
            ss->nb = 5;
            ss->msg = "Msg modifier !";
          }
          else
          {
            perror ("shmat");
            return EXIT_FAILURE;
          }
        }
        else
        {
          perror ("fork");
          return EXIT_FAILURE;
        }
      }
      else
      {
        perror ("shmat");
        return EXIT_FAILURE;
      }
    }
    else
    {
      perror ("shmget");
      return EXIT_FAILURE;
    }
  }
  else
  {
    perror ("ftok");
    return EXIT_FAILURE;
  }


  (void)argc;
  return EXIT_SUCCESS;
```

```
}
```

Gestion de signal basique avec sigaction

Auteurs : Jean Christophe Beyler ,

Ce programme s'endort pendant un certain nombre de secondes. Ce nombre est augmenté à chaque tour de boucle. Nous redirigeons le signal SIGINT pour permettre au programme de terminer son cycle avant de sortir.

```
#include <signal.h>
#include <unistd.h>

#include <string.h>
#include <stdlib.h>
#include <stdio.h>

/*
 * Volatile permet de dire au compilateur de ne pas optimiser le code
 * lorsque la variable continuer est en jeu.
 * Cela force la verification de la valeur de cette variable dans la boucle
 * de la fonction main
 */
volatile int continuer = 1;

int fct(int n)
{
    int restant = n;

    /*
     * On veut forcer l'attente du nombre de secondes voulu par l'utilisateur,
     * on met donc une boucle en attendant le retour 0 de la fonction sleep
     *
     * Par contre, si on s'amuse a envoyer un signal trop rapidement, le decompte ne
     * se fera jamais assez rapidement avec sleep.
     *
     * On peut eviter cela en decrementant restant
     */
    while(restant > 0) {
        restant = sleep(restant);
        restant --;
    }

    printf("Fct endormi pour %d secondes\n",n);
    return n;
}

void gestion_signal(int signum)
{
    /* Pas besoin de regarder la valeur de signum, nous avons un seul signal */
    continuer = 0;

    /* On n'utilise pas signum mais pour eviter un warning */
    (void) signum;
}

int main(void)
{
    int n, sum=0;
    struct sigaction act;

    /* Prepare la structure sigaction */
    memset(&act,0,sizeof(act));
    act.sa_handler = gestion_signal;
```

```
    if(sigaction(SIGINT,&act,NULL)==-1) {
        fprintf(stderr,"Erreur avec sigaction\n");
        exit(EXIT_FAILURE);
    }

    /* La boucle generale */
    n = 1;
    while(continuer) {
        n *= 2;

        sum += fct(n);
    }

    /* Un affichage de fin de programme */
    printf("En total : %d\n", sum);
    return EXIT_SUCCESS;
}
```

Comment exécuter une tâche avec un signal ?

Auteurs : Jean Christophe Beyler ,

Ce programme consiste à faire faire une tâche à un programme lors de la réception d'un signal. Nous redirigeons SIGINT de la même façon qu'avant pour terminer correctement le programme, permettant de faire du nettoyage (s'il le faut) et ensuite le signal SIGUSR1 pour afficher un message lors de sa réception.
Enfin, les deux signaux sont redirigés vers la même fonction, et nous utilisons le premier paramètre pour distinguer les deux signaux.

```
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

/*
 * Volatile permet de dire au compilateur de ne pas optimiser le code
 * lorsque la variable continuer est en jeu.
 * Cela force la verification de la valeur de cette variable dans la boucle
 * de la fonction main
 */
volatile int continuer = 1;

void gestion_signal(int signum)
{
    /* Si c'est SIGINT, on met continuer a 0 */
    if(signum == SIGINT) {
        continuer = 0;
    }
    else {
        /* Sinon si c'est SIGUSR1, on affiche bonjour */
        if(signum == SIGUSR1) {
            printf("Hello World !!!\n");
        }
    }
}

int main(void)
{
    struct sigaction act;

    /* Affichage du pid */
```

```
    printf("Le programme commence : %d\n", getpid());

    /* Prepare la structure sigaction */
    memset(&act,0,sizeof(act));
    act.sa_handler = gestion_signal;

    if(sigaction(SIGINT,&act,NULL)==-1) {
        fprintf(stderr,"Erreur avec sigaction\n");
        exit(EXIT_FAILURE);
    }

    if(sigaction(SIGUSR1,&act,NULL)==-1) {
        fprintf(stderr,"Erreur avec sigaction\n");
        exit(EXIT_FAILURE);
    }

    /* La boucle generale */
    while(continuer) {
        /* Dormir pour une journee en attente d'un signal */
        sleep(86400);
    }

    /* Un affichage de fin de programme */
    printf("Le programme se termine\n");
    return EXIT_SUCCESS;
}
```

Sommaire > Divers

Analyser les options passées en ligne de commande

Auteurs : Musaran ,

Cet exemple montre comment récupérer les options passées en ligne de commande sous la forme *-a option_a -b option_b*.

```
int main (int argc, char *argv[])
{
  int i;

  /* parcourir les arguments */
  for (i = 1; i != argc; ++i)
  {
    /* si le premier caractère de l'argument est '-'... */
    if (argv[i][0] == '-')
    {
      /* ...c'est une option. */
      char* popt;

      /* parcourir les caractères de l'option */
      for (popt = &argv[i][1]; *popt != '\0'; ++popt)
      {
        switch (*popt)
        {
          case 'a':
            /*option a*/
          break ;
          case 'b':
            /*option b*/
          break ;
          default :
            /*option inconnue*/
          break;
        }
      }
    }
    else
    {
      /* ...c'est un argument */
      argv[i]; /* traiter cet argument */
    }
  }
  return 0;
}
```

Echanger la valeur de deux variables

Auteurs : Nicolas Joseph ,

Cette macro permet d'échanger le contenu de deux variables quelque soit leur type.

```
#include <stdio.h>

#define SWAP(type, var1, var2) \
do{ \
   type tmp = (var1); \
   (var1) = (var2);   \
   (var2) = tmp;      \
}while(0)

int main (void)
{
```

```
  int a = 0, b = 1;

  printf ("a=%d b=%d\n", a, b);
  SWAP (int, a, b);
  printf ("a=%d b=%d\n", a, b);
  return 0;
}
```

[VT100] Positionnement curseur et code ANSI

Auteurs :

```
/* VT100 : Actions sur l'écran d'un terminal */
#define CLEARSCR        "\x1B[2J\x1B[;H"        // Clear SCReen
#define CLEAREOL        "\x1B[K"                // Clear End Of Line
#define CLEAREOS        "\x1B[J"                // Clear End Of Screen
#define CLEARLCR        "\x1B[0K"               // Clear Line Cursor Right
#define CLEARLCL        "\x1B[1K"               // Clear Line Cursor Left
#define CLEARELN        "\x1B[2K"               // Clear Entire LiNe
#define CLEARCDW        "\x1B[0J"               // Clear Curseur DoWn
#define CLEARCUP        "\x1B[1J"               // Clear Curseur UP
#define GOTOYX          "\x1B[%.2d;%.2dH"       // Goto at (y,x)

#define INSERTMOD       "\x1B[4h"               // Mode insertion
#define OVERWRITEMOD    "\x1B[4l"               // Mode de non insertion
#define DELAFCURSOR     "\x1B[K"
#define CRLF            "\r\n"                  // Retour à la ligne

/* VT100 : Actions sur le curseur */
#define CURSON          "\x1B[?25h"             // Curseur visible
#define CURSOFF         "\x1B[?25l"             // Curseur invisible

/* VT100 : Actions sur les caractères affichables */
#define NORMAL          "\x1B[0m"               // Normal
#define BOLD            "\x1B[1m"               // Gras
#define UNDERLINE       "\x1B[4m"               // Souligné
#define BLINKING        "\x1B[5m"               // Clignotant
#define INVVIDEO        "\x1B[7m"               // Inverse vidéo

/* VT100 : Couleurs */
#define CL_BLACK "\033[22;30m"   // Noir
#define CL_RED "\033[22;31m"   // Rouge
#define CL_GREEN "\033[22;32m"   // Vert
#define CL_BROWN "\033[22;33m"  // Brun
#define CL_BLUE "\033[22;34m"   // Bleu
#define CL_MAGENTA "\033[22;35m"  // Magenta
#define CL_CYAN "\033[22;36m"   // Cyan
#define CL_GRAY "\033[22;37m"   // Gris
#define CL_DARKGRAY "\033[01;30m"  // Gris foncé
#define CL_LIGHTRED "\033[01;31m"  // Rouge clair
#define CL_LIGHTGREEN "\033[01;32m" // Vert clair
#define CL_YELLOW "\033[01;33m"  // Jaune
#define CL_LIGHTBLUE "\033[01;34m"  // Bleu clair
#define CL_LIGHTMAGENTA "\033[01;35m" // Magenta clair
#define CL_LIGHTCYAN "\033[01;36m"  // Cyan clair
```

```
#define CL_WHITE "\033[01;37m"   // Blanc
```

Gestion du type boolean

Auteurs : gl ,

Un simple fichier d'en-tête qui permet d'avoir un type boolean (disponible uniquement à partir du C99).

ftp://ftp-developpez.com/c/sources/c/bool.zip

Module de trace

Auteurs : gl ,

La fonction *TRACE_vReportTrace* permet d'enregistrer une trace dans un fichier de log avec diverses informations telles que la date et l'id du processus.

ftp://ftp-developpez.com/c/sources/c/traces.zip

Gestion des options de la ligne de commande

Auteurs : gl ,

La fonction *OPT_iGetOpt* permet, à chaque appel, de récupérer les options de la ligne de commande. Nécessite bool.zip.

ftp://ftp-developpez.com/c/sources/c/opt.zip

Déterminer le nombre d'arguments d'une fonction variadic

Auteurs : Nicolas Joseph ,

Pour déterminer le nombre d'arguments d'un fonction variadic, on utilise généralement une sentinelle (par exemple un pointeur NULL) ou l'on précise en premier argument le nombre d'arguments passés.
Voici une macro, postée par Laurent Deniau sur comp.std.c, qui retourne le nombre d'arguments passés en paramètre :

```
#include <stdio.h>
#include <stdarg.h>

/* The PP_NARG macro returns the number of arguments that have been
  * passed to it.
  */
#define PP_NARG(...) \
         PP_NARG_(__VA_ARGS__,PP_RSEQ_N())
#define PP_NARG_(...) \
         PP_ARG_N(__VA_ARGS__)
#define PP_ARG_N( \
          _1, _2, _3, _4, _5, _6, _7, _8, _9,_10, \
         _11,_12,_13,_14,_15,_16,_17,_18,_19,_20, \
         _21,_22,_23,_24,_25,_26,_27,_28,_29,_30, \
         _31,_32,_33,_34,_35,_36,_37,_38,_39,_40, \
         _41,_42,_43,_44,_45,_46,_47,_48,_49,_50, \
         _51,_52,_53,_54,_55,_56,_57,_58,_59,_60, \
         _61,_62,_63,N,...) N
#define PP_RSEQ_N() \
         63,62,61,60,                   \
         59,58,57,56,55,54,53,52,51,50, \
         49,48,47,46,45,44,43,42,41,40, \
         39,38,37,36,35,34,33,32,31,30, \
```

```
          29,28,27,26,25,24,23,22,21,20, \
          19,18,17,16,15,14,13,12,11,10, \
          9,8,7,6,5,4,3,2,1,0
/* Note: using PP_NARG() without arguments would violate 6.10.3p4 of ISO C99. */

#define FOO(...) foo (PP_NARG(__VA_ARGS__), __VA_ARGS__)

static void foo (int nb, ...)
{
  va_list ap;

  printf ("%d : ", nb);
  va_start (ap, nb);
  for (int i = 0; i < nb; i++)
  {
    char *x = va_arg (ap, char *);

    printf ("%s, ", x);
  }
  va_end (ap);
  printf ("\n");
}

int main (void)
{
  FOO ("test", "toto", "titi");
  FOO ("test", "toto", "titi", "tata");
  return 0;
}
```

Utilisation de RSA avec la libgcrypt

Auteurs : Dark_Ebola ,

Programme et fonctions permettant d'utiliser plus facilement la bibliothèque GNU de chiffrage des données. Ce code contient de quoi:

- **générer une paire de clef et les sauvegarder dans des fichiers**
- **chiffrer une chaîne**
- **déchiffrer une chaîne**

Vous devez obligatoirement avoir installé la bitliothèque *gcrypt* pour pouvoir compiler correctement ce programme !

ftp://ftp-developpez.com/c/sources/c/utilisation-gcrypt.zip

Librairie d'encapsulation de Services Windows

Auteurs : vicenzo ,

Cette librairie est une encapsulation complète de l'API de Services Microsoft Windows NT (NT4, 2k, XP and Vista). Pour implémenter un service, il suffit juste de fournir une seule fonction de traitement !
Exemple :

```
#include <NTService.h>

void WINAPI MyServiceMain()
{
    while (!ServiceIsTerminated())
    {
```

```
        if (ServiceGetCurrentStatus() == SERVICE_RUNNING)
        {
            /* code application ... */
        }
        ServiceSleep(1000);
    }
}

int main(int argc, char** argv)
{
    return ServiceRun("MyServiceName", MyServiceMain, 1000, SERVICE_ACCEPT_STOP);
}
```

Pour plus de contrôle et/ou information sur le service et ses évènements, il est possible de fournir une fonction callback.
Exemple :

```
DWORD WINAPI MyServiceCallback(DWORD dwEvent, DWORD dwState)
{
    switch (dwEvent)
    {
        case SE_INSTALL:
            if (dwState == SS_SUCCESS)
                ServiceLog(EVENTLOG_INFORMATION_TYPE, _T("Installation ok"));
            break;
        case SE_DELETE:
            if (dwState == SS_SUCCESS)
                ServiceLog(EVENTLOG_INFORMATION_TYPE, _T("Uninstallation ok"));
            break;
        case SE_START:
            /*...*/
            break;
        case SE_STOP:
            /*...*/
            break;
        case SE_PAUSE:
            /*...*/
            break;
        case SE_CONTINUE:
            /*...*/
            break;
        case SE_CUSTOM:
            return ServiceLog(EVENTLOG_INFORMATION_TYPE,  _T("Event %s - Status %s"), dwEvent,
 dwState);
    }

    return SS_SUCCESS;
}
```

Pour utiliser le logging par défaut dans le journal système de windows, il faut inclure à votre projet les fichiers NTEventLog.xxx fournis.
Sinon pour une gestion de logging personnalisée ou déja existante via des ressources stockées dans une DLL, il faut fournir sa propre dll et fournir les fichier de messages (*.mc, *.rc, *.h, ...). Cela est intéréssant uniquement quand il faut pouvoir gérer des messages en plusieurs langues.
La librarie fournie toute les fonctions nécessaires pour contrôler des services en local ou distants.
Exemple pour lister les services en local :

```
LPSERVICE_ITEM list = LocalServiceGetList(SERVICE_WIN32, SERVICE_STATE_ALL, &count);

for (i = 0; i < count; i++)
{
    printf("%s\n", list[i].lpDisplayName);
    printf("%s\n", list[i].lpServiceName);
```

```
    LPSERVICE_INFO info = LocalServiceGetInfo(list[i].lpServiceName);

    if (info)
    {
        printf("%s\n", info->lpBinaryPathName);
        printf("%s\n", info->lpDescription);
        printf("%s\n", info->lpServiceStartName);

        ServiceFreePointer(info);
    }
}
```

La page officielle du projet: http://sourceforge.net/projects/ntwinserv

ftp://ftp-developpez.com/c/sources/c/WinServices-1.6.0.zip

Chiffres en lettres

Auteurs : Melem ,

Ce petit programme vous demande un nombre et va afficher deux mille cinq cent soixante et onze si vous entrez 2571 et un si vous entrez 1

ftp://ftp-developpez.com/c/sources/c/chiffres-lettres.zip

SuperPrime - un petit benchmark

Auteurs : Bornerdogge ,

Un petit programmer permettant de calculer tous les nombres premiers jusqu'à 500 milliards (ça peut prend un peu de temps).
Principe d'utilisation:
Au départ du programme, on vous demande d'entrer un chiffre. C'est le nombre de chiffres qui doivent être testés, premiers ou pas (exemple: tapez "100" suivi de la touche "enter" pour calculer tous les nombres premiers jusqu'à 100).
Le programme détecte lui-même le nombre de coeurs du sytème.
Une fois le calcul terminé, le programme indique le temps nécessaire au calcul et inscrit tous les nombres premiers trouvés dans le fichier "PrimeNumbers.txt". Attention, en fonction du nombre de nombres permiers trouvés, le fichier peut devenir assez volumineux...
Si vous désirez quitter le programme pendant son exécution, tapez simplement "Ctrl"+"C".
En principe, les machines Intel devraient être beaucoup plus puissante que les AMD la-dessus (-> en entiers plutot qu'en virgule flottante...).
Pour effectuer une comparaison de performances, entrez 500000 et retenez le Checksum.
Le programme existe pour Windows et Linux, voici les liens:

- **ftp://ftp-developpez.com/c/sources/c/SuperPrimeWindows_v2.2b.rar**
- **ftp://ftp-developpez.com/c/sources/c/SuperPrimeLinux_v2.2.rar**