Révisions en Programmation Impérative

1 Tableaux et calculs

a. Écrire la fonction `int ivmax(int *t, int n)` qui retourne **l'indice de la plus grande valeur** contenue dans le tableau passé en argument.

b. Écrire la fonction `float somme(float * t, int n)` qui retourne **la somme** des `n` éléments du tableau passé en argument.

c. Écrire la fonction `float pp(float * t, int n)` qui retourne **le plus petit** des `n` éléments du tableau passé en argument.

d. Écrire la fonction récursive `long long puissance_r(int x, unsigned int n)` qui retourne x^n.

e. Écrire la fonction itérative `long long puissance_i(int x, unsigned int n)` qui retourne x^n.

2 Chaînes de caractères

a. Écrire la fonction `int cpt_par(char * s)` qui retourne "le nombre de parenthèses ouvrantes moins le nombre de parenthèses fermantes" de la chaîne de caractères pointée par `s`.

b. Écrire la fonction `int verif_par(char * s)` qui retourne 1 si le parenthésage de la chaîne de caractères pointée par `s` est correcte et 0 sinon. Vous pouvez utilisez les indications suivantes :

```
* Initialiser une variable P à 0 ;
* En parcourant la chaîne de caractères :
Si parenthèse ouvrante, incrémenter P d'une unité ;
Si parenthèse fermante, decrémenter P d'une unité ;
(Remarque : Pendant le parcours, si P est négatif, retourner 0)
* En fin de parcours :
si P est strictement positif retourner 0 ;
sinon retourner 1 ;
```

3 Compréhension de programme (faire tourner à la main)

Que donne l'exécution réussie du programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void foo(char * str1, char * str2) {
  char * d = str1;
  int i = 0, l = strlen(str2);
  while(*str1) {
    *(str1++) += str2[(i++)%l] - '0';
    printf("%s\n", d);
  }
}
int main(int argc, char ** argv) {
  char s[] = "AAAAAAAAAA";
  if(argc != 2 || strlen(argv[1]) != 6) {
    fprintf(stderr, "usage : %s <votre numero d'etudiant>\n", argv[0]);
    exit(1);
  }
  foo(s, argv[1]);
  return 0;
}
```

4 Binaire

Écrire un programme récupérant son premier et unique argument (utiliser `argc` et `argv`), teste qu'il est bien entier (utiliser `atoi` ou `strtol`) et l'affiche en binaire (utiliser l'opérateur de décalage `>>` et l'opérateur binaire `&`).

5 Un peu plus loin

a. Écrire la fonction `inv_i(int t[], int n)` qui inverse l'ordre des éléments dans `t` (`n` étant le nombre d'éléments de `t`) ;

b. Écrire la fonction `inv_s(char * t)` qui inverse l'ordre des éléments dans `t` ;

c. Réécrire la fonction `char * strcpy(char * dest, const char * src);` qui copie la chaîne pointée par `src` dans la zone mémoire pointée par `dest` (voir la *man page* plus loin) ;

d. Réécrire la fonction `char * strdup(const char * s);` qui renvoie un pointeur sur une nouvelle chaîne de caractères qui est dupliquée depuis `s` (voir la *man page* plus loin) ;

e. Écrire en `C` et en n'utilisant que les types standards, un algorithme capable de calculer sans débordement mémoire la plus grande série possible de combinaisons C_n^p. Les valeurs de la série doivent être stockées dans des `unsigned long long`.

A préparer (finir) pour la semaine 2 – TP :

— Écrire un programme (main + fonction(s)) pour le calcul rapide de la puissance : x^n avec x et n entiers.

— Écrire en `C` une implémentation du Crible d'Ératosthène. Ce programme prendra en entrée N (en argument de l'exécutable), la borne supérieure des nombres premiers à trouver ; et il imprimera (printf) tous ces nombres. Merci de vous inspirer de l'algorithme décrit sur la page correspondante sur Wikipédia :

L'algorithme procède par élimination : il s'agit de supprimer d'une table des entiers de 2 à N tous les multiples d'un entier. En supprimant tous les multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier, et qui sont donc les nombres premiers.

On commence par rayer les multiples de 2, puis à chaque fois on raye les multiples du plus petit entier restant.

On peut s'arrêter lorsque le carré du plus petit entier restant est supérieur au plus grand entier restant, car dans ce cas, tous les non-premiers ont déjà été rayés précédemment.

À la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à N.

L'ensemble doit être prêt sous la forme d'une archive `tgz` (ou tar.gz et NON zip, rar ou autres) contenant le dossier regroupant les deux programmes plus un `Makefile` – aucun binaire n'est admis dans l'archive.

```
STRCPY(3)       Linux Programmer's Manual       STRCPY(3)

NAME
       strcpy, strncpy - copy a string
SYNOPSIS
       #include <string.h>

       char *strcpy(char *dest, const char *src);

       char *strncpy(char *dest, const char *src, size_t n);
DESCRIPTION
       The strcpy() function copies the string pointed to by src,
       including the terminating null byte ('\0'), to the buffer
       pointed to by dest.  The strings may not overlap, and the
       destination string dest must be large enough to receive the
       copy.

       The strncpy() function is similar, except that at most n bytes
       of src are copied.  Warning: If there is no null byte among the
       first n bytes of src, the string placed in dest will not be
       null-terminated.

       If the length of src is less than n, strncpy() pads the
       remainder of dest with null bytes.

       A simple implementation of strncpy() might be:
           char *
           strncpy(char *dest, const char *src, size_t n) {
               size_t i;

               for (i = 0; i < n && src[i] != '\0'; i++)
                   dest[i] = src[i];
               for ( ; i < n; i++)
                   dest[i] = '\0';

               return dest;
           }
RETURN VALUE
       The strcpy() and strncpy() functions return a pointer to the
       destination string dest.
CONFORMING TO
       SVr4, 4.3BSD, C89, C99.
NOTES
       Some programmers consider strncpy() to be inefficient and error
       prone.  If the programmer knows (i.e., includes code to test!)
       that the size of dest is greater than the length of src, then
       strcpy() can be used.

       If there is no terminating null byte in the first n characters
       of src, strncpy() produces an unterminated string in dest.
       Programmers often prevent this mistake by forcing termination
       as follows:
           strncpy(buf, str, n);
           if (n > 0)
               buf[n - 1]= '\0';
BUGS
     If the destination string of a strcpy() is not large enough, then
     anything might happen.  Overflowing fixed-length string buffers
     is a favorite cracker technique for taking complete control of
     the machine.  Any time a program reads or copies data into a
     buffer, the program first needs to check that there's enough
     space.  This may be unnecessary if you can show that overflow is
     impossible, but be careful: programs can get changed over time,
     in ways that may make the impossible possible.
SEE ALSO
     bcopy(3), memccpy(3), memcpy(3), memmove(3), stpcpy(3),
     stpncpy(3), strdup(3), string(3), wcscpy(3), wcsncpy(3)
```

FIGURE 1 – Man-page de la fonction strcpy.

```
STRDUP(3)          Linux Programmer's Manual                  STRDUP(3)
NAME
       strdup, strndup, strdupa, strndupa - duplicate a string
SYNOPSIS
       #include <string.h>
       char *strdup(const char *s);
       char *strndup(const char *s, size_t n);
       char *strdupa(const char *s);
       char *strndupa(const char *s, size_t n);
   Feature Test Macro Requirements for glibc (see
   feature_test_macros(7)):
       strdup():
           _SVID_SOURCE || _BSD_SOURCE || _XOPEN_SOURCE >= 500
           || _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
           || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
       strndup():
           Since glibc 2.10:
               POSIX_C_SOURCE >= 200809L || _XOPEN_SOURCE >= 700
           Before glibc 2.10:
               _GNU_SOURCE
       strdupa(), strndupa(): _GNU_SOURCE
DESCRIPTION
       The strdup() function returns a pointer to a new string which
       is a duplicate of the string s.  Memory for the new string is
       obtained with malloc(3), and can be freed with free(3).

       The strndup() function is similar, but only copies at most n
       characters.  If s is longer than n, only n characters are
       copied, and a terminating null byte ('\0') is added.

       strdupa() and strndupa() are similar, but use alloca(3) to
       allocate the buffer. They are only available when using the GNU
       GCC suite, and suffer from the same limitations described in
       alloca(3).
RETURN VALUE
       The strdup() function returns a pointer to the duplicated
       string, or NULL if insufficient memory was available.
ERRORS
       ENOMEM Insufficient memory available to allocate duplicate
       string.
CONFORMING TO
       strdup() conforms to SVr4, 4.3BSD, POSIX.1-2001.  strndup()
       conforms to POSIX.1-2008.  strdupa() and strndupa() are GNU
       extensions.
SEE ALSO
       alloca(3), calloc(3), free(3), malloc(3), realloc(3),
       string(3), wcsdup(3)
```

FIGURE 2 – Man-page de la fonction strdup.