**Licence Informatique** — **Initiation à la Programmation**
**Université Bordeaux I** — **Session de Janvier 2002**

*L'épreuve dure 3 heures et aucun document n'est autorisé.*
*Les programmes sont exclusivement à rédiger en langage C - ansi. On respectera le mode d'indentation utilisé en cours. On utilisera systématiquement la macro-fonction* `assert()` *pour signaler toute anomalie d'utilisation. La notation tiendra compte de la correction des programmes (en particulier de la bonne utilisation de la mémoire dynamique) mais surtout de la qualité du codage : lisibilité, maintenabilité, extensibilité et concision.*
*Barème indicatif : 3 - 4 - 4 - 9*

**Exercice 1.** Écrire le texte que produirait l'exécution du programme suivant :

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

int
f(int a, int *b)
{
  static int c = 0;
  c = c + a + 1;
  a = *b + c;
  *b = c;
  return a;
}

int
main(void)
{
  int a[2], *c = a + 1;
  for (a[0] = a[1] = 1; a[1] < 7; a[1]++ )
    {
      printf("%d␣%d␣", a[0], a[1]);
      printf("%d␣", f(a[0], a));
      printf("%d\n", f(c[0], c));
    }
  return EXIT_SUCCESS;
}
```

**Exercice 2.** Écrire le programme correspondant à la commande `str2ascii` qui affiche la valeur ascii des chaînes passées en argument. Cet affichage peut se faire sous trois formats différents : octal (base 8, le format à utiliser pour la fonction printf est `"\\%03o"`), décimal (base 10, format `"\\%03d"`) et hexadécimal (base 16, format `"\\%02X"`). Pour exemples :

```
> ./str2ascii -o Des peaux rouges
\104\145\163 \160\145\141\165\170 \162\157\165\147\145\163
>  ./str2ascii -d Des peaux rouges
\068\101\115 \112\101\097\117\120 \114\111\117\103\101\115
> ./str2ascii -h Des peaux rouges
\44\65\73 \70\65\61\75\78 \72\6F\75\67\65\73
> ./str2ascii Porteur de blés flamands
usage : ./str2ascii directive liste_de_chaines
-o pour écrire les chaînes en octal
-d pour écrire les chaînes en décimal
-h pour écrire les chaînes en hexadécimal
>
```

**Exercice 3.** Écrire le code des deux fonctions suivantes (soyez particulièrement attentif à la gestion de la mémoire) :

1. `char *concatener(const char *s1, const char *s2)`
   cette fonction retourne une nouvelle chaîne de caractères contenant la juxtaposition des chaînes `s1` et `s2` ; par exemple la concaténation de `"philo"` et `"sophie"` vaut `"philosophie"`.
2. `char *projeter(const char *s, const char *alphabet)`
   retourne une nouvelle chaîne de caractères contenant la suite de caractères formée à partir de `s` en effaçant les lettres n'apparaissant pas dans la chaîne `alphabet` ; par exemple `"pot"` est la projection de `"projeter"` sur `"opt"`.

**Problème** Il s'agit de présenter un module générique traitant des arbres binaires de recherche (abr). On rappelle que pour chaque sommet d'un abr les valeurs des sommets du sous-arbre gauche sont inférieures à la valeur du sommet considéré qui est elle-même inférieure à toutes les valeurs des sommets du sous-arbre droit.

Pour ce problème, on pourra s'inspirer librement du module abr traité en cours. On rappelle que dans ce module, l'aspect générique avait été traité en utilisant des pointeurs de fonctions, définis par les types suivants :

```
typedef void *objet;
typedef int (*comparer)(objet o1, objet o2);
typedef void (*liberer)(objet element_a_liberer, objet parametre);
typedef void (*appliquer)(objet element, objet parametre);
```

Les fonctionnalités du module abr sont : la création et la destruction d'un abr ; l'ajout, la suppression et la recherche d'un élément dans un abr ; l'application d'une fonction paramétrée à tous les éléments d'un abr et ce en suivant l'ordre infixe (appliquer la fonction à l'élément du noeud considéré après la visite de son sous-arbre gauche mais avant la visite de son sous-arbre droit).

1. En appliquant le principe du masquage de l'implémentation définir l'interface (le fichier `abr.h`) du module abr.
2. En supposant que le programme `la_famille.c` fasse appel au module abr écrire son fichier makefile.
3. Écrire les déclarations de type du fichier source du module abr.
4. Écrire le code des fonctions de création d'un abr et d'insertion d'un élément dans un abr.
5. Écrire la fonction de destruction d'un abr qui "libère" tous les éléments conservés.
6. Illustrer par un programme l'utilisation de toutes les fonctions du module en utilisant le type suivant :
```
typedef struct individu *individu;
struct individu
{
   char *nom;
   char *prenom;
   int date_naissance[3]; // jour mois annee
};
```

## Annexes

```
void assert (int expression);                           <assert.h>
int strcmp(const char *s1, const char *s2);             <string.h>
size_t strlen (const char *s);                          <string.h>
void *malloc(size_t size);                              <stdlib.h>
void free(void *ptr);                                   <stdlib.h>
void *realloc(void *ptr, size_t size);                  <stdlib.h>
```

— Fin —

# Correction

**Exercice 1.**

Remarques :
- 2 lignes suffisent,
- le correcteur n'a lu aucun commentaire.

```
1 1 3 5
2 5 9 18
```

**Exercice 2.**

Remarques :
- on préférera usage() à assert() pour signaler les erreurs dues à l'utilisateur ;
- on évitera la redondance des tests des options en utilisant un tableau (transformer la chaîne en numero d'option, puis tester ce numéro pour traiter la directive est redondant) ;
- la fonction d'usage termine le programme à l'aide de la fonction exit() ... elle ne doit pas retourner.

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

struct option {
  char *nom;
  char *format;
  char *commentaires;
};

struct option les_options[] =
  {
    {"-o","\\%03o","octal"},
    {"-d","\\%03d","édcimal"},
    {"-h","\\%02X","éhexadcimal"}
  };

const size_t NB_OPTIONS = sizeof(les_options) / sizeof(struct option);

char *argv0;

void
usage()
{
  size_t i;
  fprintf(stderr, "usage␣:␣%s␣option␣chaine1␣chaine2␣...␣\n",argv0);
  for(i = 0 ; i < NB_OPTIONS; i++)
    fprintf(stderr, "%s␣pouré␣crire␣les␣îchanes␣en␣%s\n",
              les_options[i].nom, les_options[i].commentaires);
  exit(EXIT_FAILURE);
}

size_t
numero_option(char *opt_arg)
{
  size_t i;
  for(i = 0; i < NB_OPTIONS; i++)
    if(!strcmp(opt_arg,les_options[i].nom))
      return i;
  usage();
}
```

```
int
main(int argc, char *argv[])
{
  size_t opt,i,j;
  argv0 = argv[0];
  if (argc < 3)
    usage();
  opt = numero_option(argv[1]);
  for(i=2; i < argc; i++)
    {
      for(j=0; argv[i][j] != '\0'; j++)
        printf(les_options[opt].format, (int) argv[i][j]);
      printf("␣");
    }
  printf("\n");
  return EXIT_SUCCESS;
}
```

**Exercice 3.**

Remarques :
- la fonction strlen retourne la longueur de la chaîne caractère, nul non compris (de toute façon l'allocation `malloc(strlen(s1) + strlen(s2))` ne pas être optimale pour la concaténation) ;
- on minimisera l'appel à la fonction strlen() qui est en $O(n)$ :
  utiliser `for(lg = strlen(s), i = 0 ; i < lg; i++)` plutôt que `for(i = 0 ; i < strlen(s); i++)` ;
- on favorisera la lisibilité : `*(tab + i + 1)` est moins lisible que verb'tab[i+1]'.

```
char *
chaine_concatener (const char *str1, const char *str2)
{
  size_t lg1 = strlen (str1);
  char *tampon = malloc (lg1 + strlen (str2) + 1);
  assert (tampon != NULL);
  strncpy (tampon, str1, lg1);
  strcpy (tampon + lg1, str2);
  return tampon;
}

int
apparait (char c, const char *s)
{
  while (*s != '\0' && *s != c)
    s++;
  return *s == c;
}

char *
projeter (const char *s, const char *alphabet)
{
  char *tampon = malloc (strlen (s) + 1);
  size_t i, j;
  for (i = j = 0; s[i] != '\0'; i++)
    if (apparait (s[i], alphabet))
      tampon[j++] = s[i];
  tampon[j] = '\0';
  return realloc (tampon, j + 1);
}
```

**problème** On trouvera sur le site www le module abr vu en cours.