POLYTECH NICE-SOPHIA

SI 3 - 2017-2018

Programmation Procédurale

Feuille 3

Chaînes de caractères, Manipulations de bits, Fonctions à arité variable

Exercice 1: Palindrome

Écrire la fonction `int palindrome(const char str[])` qui renvoie 1 si la chaîne `str` est un palindrome et 0 sinon.

```
palindrome("ressasser") ⟶ 1
palindrome("kayak")     ⟶ 1
palindrome("X")         ⟶ 1
palindrome("test")      ⟶ 0
```

Exercice 2: Suppression dans une chaîne

Écrire la fonction `void suppression(char str[], const char suppr[])` qui permet de supprimer la chaîne `suppr` dans la chaîne `str`:

```
Suppression de 'on' dans 'Bonjour' => 'Bjour'
Suppression de 'B' dans 'Bonjour' => 'onjour'
Suppression de 'Bonjour' dans 'Bonjour' => ''
Suppression de 'jour' dans 'Bonjour' => 'Bon'
Suppression de 'abc' dans 'Bonjour' => 'Bonjour'
```

Pour écrire cette fonction, vous essaierez de ne pas utiliser de chaîne temporaire (la suppression se fait directement dans la chaîne `str`). Pour cela, vous pouvez vous définir une fonction qui décale une chaîne de caractère vers la gauche, à partir d'une position donnée, de n caractères.

Exercice 3: Impression de nombres entiers

1. Écrire une fonction permettant d'imprimer un nombre entier, en utilisant seulement la fonction `putchar`.
2. Généraliser votre fonction pour qu'elle puisse imprimer un nombre dans une base **b** quelconque ($2 <= b <= 36$)

Exercice 4: décomposition binaire

Écrire la fonction `en_binaire` qui affiche le nombre `n` qui lui est passé en paramètre.

1. écrire une première version récursive travaillant par divisions successives (cette version affichera `n` avec le nombre minimal de bits nécessaires pour représenter `n`).

2. écrire une version utilisant les opérateurs sur les bits de C (cette fonction affichera `n` sur le nombre de bits nécessaire pour représenter un `int` sur votre machine.

Exercice 5: grands ensembles

On décide de définir un type abstrait de données permettant de représenter de grands ensembles d'entiers (compris entre 0 et 999). Pour l'implémentation de ce type, on décide de représenter un grand ensemble par un tableau de bits (si un nombre est présent dans l'ensemble, le bit correspondant à ce nombre sera mis à 1, sinon le bit sera à 0). Pour cela, nous avons les définitions suivantes:

```
#define CHAR_SIZE  8                        /* nombre de bits dans un char */
#define MAX_BIGSET 125                      /* nombre de cellules dans un ensemble */
#define MAX_VAL    (CHAR_SIZE * MAX_BIGSET)

typedef unsigned char BIGSET[MAX_BIGSET]; /* un ensemble dans [0 .. MAX_VAL[  */
```

Ecrire les fonctions suivantes.

1. `void BIGSET_init(BIGSET s) /* créer l'ensemble vide */`
2. `void BIGSET_add(BIGSET s, int i) /* ajouter i dans s */`
3. `int BIGSET_is_in(BIGSET s, int i) /* 1 si i dans s et 0 sinon */`
4. `void BIGSET_print(BIGSET s) /* afficher les éléments de s */`
5. `void BIGSET_inter(BIGSET s1, BIGSET s2, BIGSET res)`
   `/* range dans res le résultat de l'intersection des ensembles s1 et s2 */`

Exemple de code utilisant les fonctions sur les ensembles

```
{
  BIGSET e1, e2, e3;
  int i;

  BIGSET_init(e1); BIGSET_init(e2);
  for (i = 0; i < 40; i += 5) BIGSET_add(e2, i);
  for (i = 0; i < 40; i += 3) BIGSET_add(e1, i);
  BIGSET_inter(e1, e2, e3);

  printf("e1 = "); BIGSET_print(e1);   /* => e1 = {0 3 6 9 12 15 18 21 24 27 30 33 36 39} */
  printf("e2 = "); BIGSET_print(e2);   /* => e2 = {0 5 10 15 20 25 30 35} */
  printf("e3 = "); BIGSET_print(e3);   /* => e3 = {0 15 30} */
}
```

Exercice 6: calculatrice

On désire réaliser une petite calculatrice en C. Pour cela, on a besoin de la fonction à nombre variable de paramètres `evaluer` dont le prototype est

```
int evaluer(char operateur, int operande, ...);
```

Cette fonction permet d'appliquer `operateur` à sa liste d'opérandes. On supposera ici que cette fonction ne travaille que sur des nombres positifs et que la fin de sa liste d'opérandes sera dénotée par un nombre négatif. D'autre part, cette fonction n'implémente que les quatre opérations suivantes: '+', '-', '*' et '/'. Ainsi,

```
evaluer('+', 1, 2, 3, -1))                          → 6
evaluer('-', 10, evaluer('*', 2, 2, 2, -1), 2, -1))  → 0
```

Exercice 7: une version simplifiée de printf

Il s'agit ici de coder des fonctions à nombre variable d'arguments, à l'aide des macros du fichier standard `<stdarg.h>`.

- coder la fonction `void CatStrings(char *str1, ...)` qui affiche à la suite tous ses paramètres jusqu'à trouver le pointeur nul. Par exemple pour afficher "`essai`" on veut pouvoir coder:
  `CatStrings("es", "sai", NULL)`.
- coder la fonction `void Printf(char *format, ...)` qui se comporte comme printf et reconnait dans son format les séquences "`%%`", "`%c`", "`%s`", "`%x`" et "`%d`".

Exercice 8 (facultatif): manipulation des tabulations

Les terminaux Unix possèdent des taquets de tabulation, placés tous les 8 caractères à partir du début de la ligne. A l'affichage, une tabulation (caractère `'\t'`) positionne le curseur au prochain taquet.

- Ecrire le filtre `detab` qui remplace chaque tabulation par des espaces, de telle sorte que l'entrée et la sortie aient la même apparence à l'affichage.
- Ecrire le programme `entab` qui remplace des séquences d'espaces par des tabulations. Par souci de simplification on ne remplacera que la plus longue séquence d'espaces commençant au début d'une ligne.

On peut généraliser ce mécanisme à des taquets de tabulation placés tous les N caractères. Intégrer cette généralisation dans `entab` et `detab`: par exemple on tape "`detab -3 < foo > bar`" pour supprimer les tabulations en considerant que $N = 3$.

Exercice 9 (facultatif): suppression des commentaires C

Ecrire le filtre `uncomment` qui remplace chaque commentaire C "`/*...*/`" par un espace. On ne tiendra pas compte des chaînes de caractères C. De plus, les commentaires ne peuvent pas être imbriqués. Voici un exemple de transformation:

```
/* foo.c */
#if 0
int ident/* comm */ificateur;
char *s = "/* chaine */";
1/*/2 3/*/4
/*1 comm /*2 mentaire 3*/ 4*/
#endif
```

$\Rightarrow$

```
#if 0
int ident ificateur;
char *s = " ";
1 4
  4*/
#endif
```

Exercice 10 (facultatif): calcul de x^n

La méthode de calcul $x^n = \underbrace{x \times x \times \ldots \times x}_{n}$ est peu efficace car elle requiert n multiplications.

Une méthode rapide se base sur le fait qu'un entier n est codé en binaire dans la machine:

$$n \equiv b_k b_{k-1} \ldots b_1 b_0 \text{ où } b_i = 0 \text{ ou } 1 \text{ et } n = \Sigma_i 2^i b_i = b_0 + 2b_1 + 4b_2 + \ldots + 2^k b_k$$

On exprime donc x^n par:

$$x^n = x^{\left\{\begin{array}{l}1 \text{ si } b_0 = 1\\ 0 \text{ si } b_0 = 0\end{array}\right\} + \left\{\begin{array}{l}2 \text{ si } b_1 = 1\\ 0 \text{ si } b_1 = 0\end{array}\right\} + \left\{\begin{array}{l}4 \text{ si } b_2 = 1\\ 0 \text{ si } b_2 = 0\end{array}\right\} + \left\{\begin{array}{l}8 \text{ si } b_3 = 1\\ 0 \text{ si } b_3 = 0\end{array}\right\} + \ldots}$$

soit:

$$x^n = \left\{ \begin{array}{l} x \text{ si } b_0 = 1 \\ 1 \text{ si } b_0 = 0 \end{array} \right\} \times \left\{ \begin{array}{l} x^2 \text{ si } b_1 = 1 \\ 1 \text{ si } b_1 = 0 \end{array} \right\} \times \left\{ \begin{array}{l} x^4 \text{ si } b_2 = 1 \\ 1 \text{ si } b_2 = 0 \end{array} \right\} \times \dots$$

Par exemple, $13 = 8 + 4 + 1 = 1_3 1_2 0_1 1_0$ en binaire, donc $x^{13} = x^8 \times x^4 \times 1 \times x$.
Introduisant la variable z valant successivement x, x^2, x^4, x^8, ... et y le produit des "bons" z, on obtient la méthode suivante:

$$\left\{ \begin{array}{l} n \text{ se décompose en } n \equiv b_k b_{k-1} \dots b_o \\ y \leftarrow 1 \\ z \leftarrow x \\ \text{pour } i \leftarrow 0 \text{ à } k\text{: si } b_i = 1 \text{ alors } y \leftarrow yz;\ z \leftarrow z^2 \\ y \text{ vaut alors } x^n \end{array} \right.$$

Coder la fonction `double Puissance(double x, unsigned int n)` qui implémente cette méthode. On pourra utiliser les opérateurs C de manipulation de bits "`<<`", "`>>`", "`&`", "`|`", "`~`".