TD/TP 8 : allocation dynamique de mémoire

Gilles Peskine

29 novembre 2002

1 Expressions

Dans chacun des fragments de programmes indépendants suivants, indiquer quelle est la valeur affichée, ou expliquer pourquoi le programme est incorrect.

```
int i1 = 3;
printf("r1 = %d\n", i1++);

int i2 = 3;
printf("r2 = %d\n", ++i2);

int i3 = 3;
printf("r3 = %d\n", i3++ + i3++);

int i4 = 3;
printf("r4 = %d\n", i4++ + i4);

int i5 = 3;
printf("r5 = %d\n", i5 ^ i5);

int i6;
i6 = 4 | 2;
printf("r6 = %d\n", i6);

int i7;
i7 = 4 || 2;
printf("r7 = %d\n", i7);

int i8;
i8 = (4 || 2);
printf("r8 = %d\n", i8);

int i9 = 3, j9 = 3, k9 = 3;
printf("r9 = %d\n", i9 == j9 == k9);

int i10 = 3;
printf("r10 = %d\n", i10 == 4 | 2);
```

2 Notation binaire

Écrire une fonction de conversion d'un nombre entier en écriture binaire. Écrire une fonction qui fait la conversion inverse.

```
char *ecriture_binaire(unsigned x);
unsigned entier_de_binaire(char *s);
```

3 Opérations binaires

(a) Écrire une fonction qui compte le nombre de bits 1 dans un entier. Par exemple :
`nombre_de_bits(entier_de_binaire("001100101"))` $\Longrightarrow$ 4

(b) Écrire une fonction qui prend un argument entier $n > 0$ et qui calcule l'entier k tel que $2^k \leq n < 2^{k+1}$. k est la position du plus grand bit de n qui est à 1. Pour l'argument 0, renvoyer -1.

```
int logarithme_entier(unsigned n)
```

4 Découpage d'une chaîne en mots

(a) Écrire une fonction qui découpe une chaîne de caractères en mots (séparés par un ou plusieurs espaces, sauts de lignes, etc). Cette fonction devra renvoyer un pointeur vers un tableau de pointeurs vers des chaînes de caractères (les mots en question). Utiliser un pointeur nul pour marquer la fin du tableau.

```
char **decoupe(char *s);
```

C'est (en simplifié) le travail qu'effectue le shell pour passer d'une ligne de commande à `argv`.

(b) Écrire une fonction qui libère la mémoire allouée par `decoupe`.

(c) Modifier la fonction `decoupe` pour pouvoir placer des espaces dans un « mot ». Utiliser une ou plusieurs méthodes d'échappement du shell : caractère \, guillemets doubles, guillemets simples.

5 UTF-8 et UCS

UCS est une représentation des caractères par des nombres sur 31 bits, ce qui permet de coder les caractères de toutes les écritures existantes. Comme beaucoup de programmes existants manipulent des caractères sur 8 bits, il existe un codage des caractères UCS en caractères sur 8 bits. Ce codage est nommé UTF-8.

Chaque caractère UCS (sur 31 bits, donc de 0 à 0x7FFFFFFF) est représenté par un à six caractères sur 8 bits en UTF-8 :

```
0x00000000-0x0000007F : 0bbbbbbb
0x00000080-0x000007FF : 110bbbbb 10bbbbbb
0x00000800-0x0000FFFF : 1110bbbb 10bbbbbb 10bbbbbb
0x00001000-0x001FFFFF : 11110bbb 10bbbbbb 10bbbbbb 10bbbbbb
0x00200000-0x03FFFFFF : 111110bb 10bbbbbb 10bbbbbb 10bbbbbb 10bbbbbb
0x40000000-0x7FFFFFFF : 1111110b 10bbbbbb 10bbbbbb 10bbbbbb 10bbbbbb 10bbbbbb
```

(On remplit les *b* par les bits utiles de la représentation UTF8 du caractère.)

Par exemple, le caractère UCS numéro 0x000007d2 est codé en UTF-8 par les caractères 11011111 10010010 (en binaire, soit 0xdf 0x92).

Écrire une fonction qui fait la traduction d'un caractère UCS en une chaîne UTF-8. Écrire la traduction inverse.

```
char *ucs_vers_utf8(long c)
long utf8_vers_ucs(char *s)
```