USTL

Licence d'informatique
Module de C/Unix

Premiers TD de programmation en C

Nouvelle révision, décembre 2001

1 Premiers programmes en C

Ces premiers exercices ont pour but de vous familiariser avec la syntaxe du langage C :

- les types de base : `char`, `int`, `unsigned`...
- les structures de contrôle : `if`, `while`, `for`...
- quelques fonctions d'entrée/sortie : `printf()`, `scanf()`, `getchar()`...
- le codage des entiers positifs en binaire.

Exercice 1 (Factorielle)
Question 1.1 Donner le code d'une fonction

```
unsigned int factorielle (unsigned int n) ;
```

qui calcule $n!$. On développera une version itérative et une version récursive de la fonction.

Question 1.2 Donner le code d'un programme qui lit un entier et affiche sa factorielle.

Exercice 2 (Parenthésage d'une expression)
On lit sur l'entrée standard (`stdin`) un texte, terminé par `EOF`, dont on veut vérifier que le parenthésage est correct. On sortira sur la sortie standard (`stdout`) un message pour le résultat.

La fonction `getchar()` renvoie le code ASCII du caractère lu sur `stdin`, la valeur `EOF` en fin de fichier ou ou cas d'erreur.

Exercice 3 (Exemples de macros)
Définir à l'aide de macros `cpp` les « constantes » VRAI et FAUX.

Écrire ensuite, le plus simplement possible, une macro `cpp` qui teste si un caractère `c` passé en paramètre est un chiffre ou une lettre majuscule : `CHIFROUMAJ(c)`.

Exercice 4 (Conversion caractères/entier)
On calcule la valeur d'un entier entré sur `stdin` sous forme d'une suite de caractères (terminée par un caractère autre qu'un chiffre).

L'arithmétique « classique » peut s'appliquer sur le type `char`. La valeur numérique d'un caractère est le code ASCII de ce caractère (65 pour `'A'`, 66 pour `'B'`...).

Exercice 5 (Format des entiers positifs)
On veut calculer le nombre de bits sur lequel sont codés les `unsigned int`. Le format des données n'est pas précisé dans les spécifications du langage C, il peut dépendre de la machine ou du compilateur.

L'opérateur arithmétique `x << lg` calcule un décalage sur `x` d'une longueur `lg` bits vers les bits de poids fort. Les `lg` bits de poids forts sont perdus, ceux de poids faible sont mis à 0.

On remarquera qu'on peut utiliser cet opérateur pour une multiplication (ou une division avec >>) par 2, 4..., à condition de faire attention aux pertes éventuelles d'informations.

Exercice 6 (Conversion décimal/binaire)
On entre sur `stdin` une valeur `unsigned int`. On sort sur `stdout` sa traduction en binaire.

Id : tdc1.tex,v 1.11 2001/12/20 07 :42 :43 marquet Exp

2 Manipulation de tableaux

Les exercices proposés illustrent l'utilisation simple de tableaux. En particulier, les chaînes de caractères peuvent être vues comme des tableaux.

Exercice 7 (Compteur de chiffres et de blancs)
On veut compter les occurrences de chaque chiffre (`'0'` à `'9'`) et des blancs (`' '` ou `\t`) d'un texte lu sur `stdin`. Le résultat sera stocké dans un tableau de 10 entiers pour les chiffres et dans une variable entière pour les blancs.

Exercice 8 (Recherche par dichotomie)
On écrit une fonction qui recherche l'indice d'un nombre `x` dans un tableau `v` de `n` entiers triés par ordre croissant :

```
int search (int x, int v[], int n) ;
```

Exercice 9 (Inversion d'une chaîne de caractères)
En langage C, les chaînes de caractères sont rangées sous forme de tableaux de caractères, terminés par le caractère `'\0'`. On veut inverser une chaîne de caractères, sans déclarer un second tableau. Pour ce faire, on pourra avoir besoin d'une fonction `strlen(const char s[])` qui retourne la longueur d'une chaîne (non compris le caractère `'\0'`).

```
void reverse(char s[]) ;
```

Exercice 10 (Conversion décimal/caractères)
La fonction `itoa(int n, char s[])` retourne dans `s` la chaîne de caractères représentant l'entier signé passé en paramètre. Pour l'écrire, on se méfiera du signe, et du fait que l'on ne veut pas de zéros inutiles. Pour cela, on pourra réutiliser la fonction `reverse()`.

Exercice 11 (Chaînes de caractères)
Réécrire des fonctions de la bibliothèque de manipulation des chaînes de caractères. Ces fonctions ne gèrent pas l'allocation mémoire, le programmeur doit s'assurer de la validité avant l'appel à ces fonctions.

```
/* recopie le contenu de src dans dest */
char * strcpy (char *dest, char *src) ;

 /* recopie src a la fin de dest (concat) */
char * strcat (char *dest, char *src) ;

/* compare les contenus de s2 et s1 ;
   renvoie la difference entre les deux premiers caracteres differents
*/
int strcmp (char *s1, char *s2) ;
```

3 Pointeurs et allocation dynamique

Les exercices proposés ici introduisent la manipulation de pointeurs. La structure des tableaux à plusieurs dimensions est détaillée. On introduit l'allocation dynamique de mémoire et la manipulation de structures chaînées.

Exercice 12 (Passage de paramètres)
Écrire une fonction qui échange les valeurs de deux variables entières.

Exercice 13 (Tableaux à plusieurs dimensions)
Soit la déclaration d'un tableau

```
int b[3][5] ;
```

En considérant que l'allocation du tableau se fait linéairement en mémoire (les 3 « tranches » de `b` sont allouées à des adresses contiguës), donner l'état du tableau `b` après l'exécution de :

```
int b[3][5] ;
int *a = *b , c ;
for ( c=0 ; c<15 ; *a++ = c++ ) ;
**b = 15 ;         **(b+1) = 16 ;     *(b[0]+1) = 17 ;   *(*b+8) = 18 ;
*(b[1]+2) = 19 ;   *(*(b+1)+5) = 20 ; *(b[2]+3) = 21 ;   *(*(b+2)+2) = 22 ;
```

Exercice 14 (Copie de tableau/chaîne de caractères, allocation dynamique de mémoire)
Donnez le code C de la fonction de la bibliothèque `string`

```
char *strdup(const char *s);
```

qui retourne une copie fraîchement allouée de la chaîne de caractère `s`.

On fera appel à la fonction de bibliothèque

```
#include <stdlib.h>
void *malloc(size_t size);
```

qui renvoie un pointeur de type `void *` sur une zone mémoire de longueur de `size` octets. On pourra aussi utiliser les autres fonctions de la bibliothèque `string`.

Que retourne cette fonction `strdup()` en cas de problème ?

Écrire une fonction qui renvoie une copie d'un tableau de `n` entiers passé en paramètre.

Exercice 15 (Liste triée)
On manipule une liste d'entiers triés dans l'ordre croissant. Une liste est représentée par une structure chaînée. Écrire le type `sorted_list_t`. Quelle est la représenatuon de la liste vide ? Écrire chacune des fonctions suivantes (justifier les prototypes proposés, on parle d'une implantation fonctionnelle) :

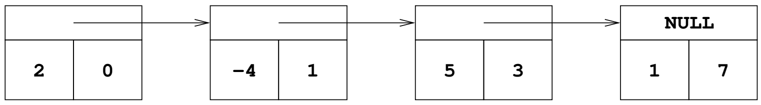
```
int member (const int n, const sorted_list_t l);
/* renvoie vrai si n est present dans l  */

sorted_list_t add_iterative (const int n, const sorted_list_t l);
/* insertion itérative de n dans l       */

sorted_list_t add_recursive (const int n, const sorted_list_t l);
/* idem. version recursive               */

sorted_list_t del (const int n, const sorted_list_t l) ;
/* si n present alors supprime n de l    */

void freelist (sorted_list_t l) ;
/* detruit la liste et libere la memoire */
```

Réécrire la fonction d'insertion d'un élément en modifiant le prototype :

```
void add (int n, sorted_list_t *l);
```

Exercice 16 (Manipulation de polynômes)
Un polynôme est une liste de monômes. Un monôme est caractérisé par un coefficient (considéré entier) et un degré. On ne rangera en mémoire que les monômes de coefficient différent de zéro.

Par exemple le polynôme $x^7 + 5x^3 - 4x + 2$ constitué de 4 monômes sera représenté par :



Question 16.1 Écrire les types de données `monome_t` et `polynome_t`.

Question 16.2 Écrire une fonction qui évalue un polynôme pour une valeur passée en paramètre.

Question 16.3 Écrire une fonction qui incrémente le degré de tous les monômes d'un polynôme.

Question 16.4 Écrire sur la sortie standard la représentation d'un polynôme.

Question 16.5 Écrire un fonction qui lit en entrée standard un polynôme entré sous forme de suite de coefficients des monômes de degrés décroissants. Le premier entier entré représente le coefficient du monôme de degré le plus élevé ; le nombre de coefficients entrés est égal au degré du polynôme plus 1. Par exemple, le polynôme précédent est entré par la suite :

```
1 0 0 0 5 0 -4 2
```

Question 16.6 Écrire une fonction qui libère la mémoire occupée par un polynôme.

Question 16.7 Écrire la fonction qui additionne un monôme à un polynôme. Attention, on ne stocke jamais les monômes dont le coefficient est nul.

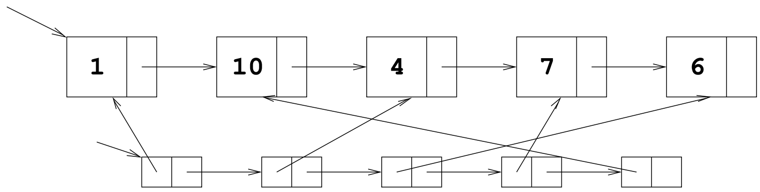
Question 16.8 Décrire l'entête d'une fonction qui additionne (ou multiplie) deux polynômes.

Exercice 17 (Tri d'une liste)
On a une liste d'objets de type `obj_t`. On dispose d'une fonction qui calcule le poids d'un objet de type `obj_t` :

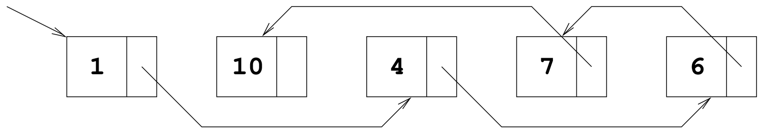
```
typedef ... obj_t ;
unsigned int poids (obj_t *o) ;

typedef struct mobj_s mobj_t, *listobj_t ;
struct mobj_s {
  obj_t     o;
  listobj_t next;
};
```

On veut trier cette liste par ordre croissant de poids, mais le type des objets `obj_t` est volumineux (il occupe beaucoup de place mémoire). On ne fait donc aucune copie de ces objets.

Pour cela, on contruit une liste dont les éléments pointent sur les objets de type `mobj_t`, dans l'ordre croissant des poids.



Puis, on « reconstruit » la liste dans le bon ordre :



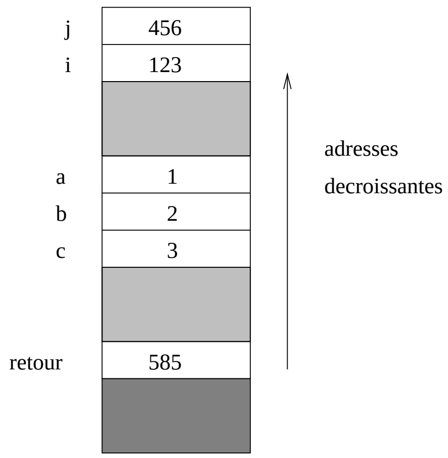
Pour développer cet algorithme, on devra donc utiliser deux listes de types différents. On pourra réfléchir à trier directement une liste sans l'utilisation d'une autre liste (tout en respectant la contrainte de ne pas copier les objets ontenus dans la liste).

4 Implantation des appels de fonction ; la bibliothèque `stdarg`

Exercice 18 (Pile d'appels de fonctions)
Un appel de fonction a pour effet de modifier le contexte courant. La pile d'exécution stocke les valeurs à sauvegarder avant un appel de fonction :

– la valeur qui sera retournée par la fonction est allouée sur la pile ;
– des informations liées à l'exécutif (*run-time*) sont sauvegardées ; (lesquelles ?)
– les valeurs des paramètres effectifs sont empilées ;
– les variables locales sont allouées sur le sommet de pile.

```
int
f (int a,
   int b,
   int c)
{
  int i=123, j=456 ;
  return a+b+c+i+j ;
}

void main () {
  f (1,2,3) ;
}
```



La figure ci-contre représente la pile suite à l'appel de la fonction `f()`.

Soient les fonctions suivantes :

```
int
facrec (unsigned int n)
{
  if (n>1)
    return n * fact (n-1) ;
  return 1 ;
}
```

```
int
faciter (unsigned int n)
{
  int res = 1 ;
  while (n>1)
    res *= n-- ;
  return res;
}
```

Quels sont les états successifs de la pile au cours de l'évaluation de `facrec(4)` et de `faciter(4)` ? Conclusion ?

Exercice 19 (Nombre de paramètres inconnu)
Certaines fonctions acceptent un nombre de paramètres qui peut varier d'un appel à l'autre :

```
int
masomme (int nbpara, ...)
{
   int somme ;
   /* ... */
   return somme ;
}

int
main()
{
   int i, j, k, s ;
   /* ... */
   s = masomme (3,i,j,k) ;
   s = masomme (2,i,s) ;
}
```

Dessinez l'état de la pile à l'entrée dans la fonction `masomme()`. Complétez la fonction `masomme()`.

Exercice 20 (La bibliothèque `stdarg.h`)
Habituellement, on utilise les macros de `<stdarg.h>` pour gérer ces appels :

```
int
masomme(int nbpara,...)
{
   va_list pvar;
   int i,somme=0;

   va_start(pvar,nbpara);
   for (i=0;i<nbpara;i++)
      somme += va_arg(pvar,int);
   return somme;
}
```

Écrivez les macros `va_list`, `va_start`, et `va_arg`. Citez des fonctions qui utilisent ce mécanisme.