Programmation : Exercices

IUT de Villetaneuse — R&T 1ère année

Laure Petrucci

6 novembre 2007

1 Premiers programmes

Exercice 1.1 : Machine à dessiner

On souhaite écrire un programme pour afficher des dessins. On va écrire plusieurs fonctions d'affichage. Les fonctions plus complexes devront utiliser les plus simples.

Question 1 : Écrire une fonction `affiche_etoile()` qui affiche une étoile `*`.

Question 2 : Écrire une fonction `affiche_espace()` qui affiche un espace.

Question 3 : Écrire une fonction `nouvelle_ligne()` qui fait passer l'affichage à la ligne suivante.

Question 4 : Écrire une fonction `affiche_un_caractere(car)` qui affiche le caractère `car`.

Question 5 : Écrire une fonction `affiche_caracteres(nb_car,car)` qui affiche `nb_car` caractères `car`.

Question 6 : Écrire une fonction `carre5()` qui affiche un carré avec 5 étoiles sur chaque coté, comme le dessin de la figure 1(a).

```
*****
*   *
*   *
*   *
*****
```
(a) Carré de coté 5

```
*
**
***
****
*****
```
(b) Triangle de base 5

```
    *
   ***
  *****
 *******
*********
```
(c) Triangle isocèle de hauteur 5

FIG. 1 – Figures à dessiner

Question 7 : Écrire la fonction `carre5` en utilisant une boucle.

Question 8 : Écrire une fonction `triangle5()` qui affiche le triangle de la figure 1(b).

Question 9 : Écrire une fonction `isocele5()` qui affiche le triangle isocèle de la figure 1(c).

Question 10 : On souhaite maintenant paramétrer ces fonctions de dessin. Écrire une fonction `carre(cote)` qui affiche un carré avec `cote` étoiles sur chaque coté.

Question 11 : Écrire une fonction `triangle(base)` qui affiche un triangle dont la base a `base` étoiles.

Question 12 : Écrire une fonction `isocele(hauteur)` affichant un triangle isocèle dont la hauteur est `hauteur` lignes.

Question 13 : Écrire un programme `C` comportant les fonctions d'affichage `affiche_un_caractere(car)`, `affiche_caracteres(nb_car, car)`, `carre(cote)`, `triangle(base)` et `isocele(hauteur)`. Le programme principal (`main()`) affichera un carré de 5 étoiles de coté, puis un triangle dont la base est de taille 6, et enfin un triangle isocèle de hauteur 7.

Exercice 1.2 : Date du lendemain

On souhaite écrire un programme calculant la date du lendemain d'un jour donné.

Question 1 : Écrire une fonction `calcul_lendemain(jour,mois,annee)` réalisant ce calcul. Elle utilisera des fonctions qui seront explicitées plus tard : `teste_dernier_jour(jour,mois,annee)` et `teste_dernier_mois(mois)`. Vous devez expliquer ce que doivent faire ces fonctions sans écrire le détail de leur algorithme, et en particulier l'utilisation des paramètres.

Question 2 : Écrire l'algorithme de la fonction `teste_dernier_mois(mois)`.

Question 3 : Écrire l'algorithme de la fonction `teste_dernier_jour(jour,mois,annee)`.

Question 4 : Écrire ces algorithmes en langage `C`. La fonction principale `main` devra calculer la date du lendemain du 31 décembre 2006, du 28 février 1900, du 28 février 2000, du 28 février 2003 et du 30 juin 2005.

Exercice 1.3 : Reproduction des lapins

Un couple de lapins a sa première portée à 2 mois, puis une portée tous les mois. Chaque portée est un couple de lapins. Tous les couples ainsi obtenus se reproduisent de la même manière.

Question 1 : On distinguera le nombre de couples de nouveaux lapins `nouveaux`, le nombre de couples de lapins ayant un mois, `un_mois`, et le nombre de couples de lapins « vieux » ayant 2 mois ou plus, `vieux`. Calculer « à la main » le nombre de couples de lapins de chaque type, ainsi que leur nombre total, pour les 10 premiers mois.

Question 2 : Écrire un algorithme `nb_lapins(nb_mois, nb_couples)` calculant le nombre de lapins obtenus au bout de `nb_mois` mois à partir de `nb_couples` couples jeunes, et renvoyant le résultat.

Question 3 : Écrire un algorithme `lapins_un_milliard` calculant au bout de combien de temps les lapins sont plus d'un milliard (on supposera qu'aucun lapin ne meurt pendant cette période), en partant d'un couple de lapins.

Question 4 : Mettre en œuvre ces algorithmes dans un programme `C`.

2 Codage binaire, opérations bit à bit

Exercice 2.1 : Codage des caractères et des entiers

Les entiers peuvent être représentés sur des nombres de bits différents :
- un *entier court* (`short`) est codé sur au moins 16 bits ;
- un `int` utilise au moins autant de bits qu'un `short` ;
- un *entier long* (`long`) est codé sur au moins 32 bits.

Question 1 : Écrire un algorithme `taille_short()` permettant, à l'aide de manipulations de bits, de déterminer le nombre de bits utilisés pour coder des entiers courts.

Question 2 : Traduire votre algorithme en programme `C`.

Question 3 : En vous inspirant de la fonction `taille_short`, écrivez en `C` une fonction générique `taille_type(num_type)`, qui calcule la taille d'un `short` si `num_type` vaut 1, d'un `int` si `num_type` vaut 2, d'un `long` si `num_type` vaut 3, d'un `char` si `num_type` vaut 4.

Exercice 2.2 : Codage VRC

Question 1 : Écrire une fonction `est_pair(nombre)` qui renvoie `vrai` si le nombre est pair, `faux` sinon.

Question 2 : Écrire une fonction `affiche_8bits(car)` qui affiche le codage binaire du caractère `car`.

Question 3 : Écrire une fonction `compte_bits(valeur, car)`, qui compte le nombre de bits valant `valeur` dans le codage binaire de `car`.

Question 4 : Écrire une fonction `force_bit1(numero, car)` qui change la valeur du bit de rang `numero` en 1 dans le codage binaire du caractère `car`.

Question 5 : Écrire une fonction `force_bit(valeur, numero, car)` qui change la valeur du bit de rang `numero` en `valeur` dans le codage binaire du caractère `car`.

Question 6 : Écrire une fonction `VRC(car,parite)` qui calcule le *bit de parité* du caractère `car` en utilisant un codage *VRC* avec la `parite` passée en paramètre (1 pour un codage pair, 0 pour un codage impair).

Question 7 : Écrire une fonction `code_VRC(car, parite)` qui code le caractère `car` avec un VRC ayant la `parite` spécifiée.

Question 8 : Mettre en œuvre ces fonctions dans un programme `C`. Écrire également une fonction `table_codage()` qui affiche la table des caractères ASCII (de numéros 32 à 126) ainsi que leur codage VRC, sous la forme suivante :

```
Décimal   Binaire   Hexadécimal   Caractère   VRC pair   VRC impair
   32     00100000       20                  10100000    00100000
   33     00100001       21          !       00100001    10100001
   ...      ...          ...         ...        ...         ...
```

Exercice 2.3 : Protection de fichiers

Question 1 : Écrire une fonction `affiche_protections(protections)` qui prend comme paramètre des protections écrites sous forme de code octal et les affiche sous forme compréhensible par un utilisateur. Par exemple, `affiche_protections(0755)` affichera `rwxr-xr-x`.

Question 2 : Programmer cette fonction en `C`. La fontion principale affichera les protections correspondant aux modes `0532`, `0750` et `0644`. En `C`, une valeur octale est indiquée en commençant son écriture par un `0`.

3 Portée des variables, paramètres de fonctions

Exercice 3.1 : Portée des variables en C

Qu'affiche le programme C suivant ? Expliquez en détail votre réponse.

```
1  #include <stdio.h>
2
3  /* schémas des fonctions */
4  void affiche1(int x);
5  int affiche2(int x);
6
7  /* déclaration des fonctions */
8
9  void affiche1(int x){
10   x = x + 1;
11   printf("affiche1 : x=%d\n",x);
12 }
13
14 int affiche2(int x){
15   x = x + 1;
16   printf("affiche2 : x=%d\n",x);
17   return(x);
18 }
19
20 int main(){
21   int x;
22
23   x = 10;
24   printf("main: x=%d\n",x);
25   affiche1(x);
26   printf("main: x=%d\n",x);
27   x = affiche2(x);
28   printf("main: x=%d\n",x);
29 }
```

Exercice 3.2 : Passage de paramètres par valeur et par adresse

Qu'affiche le programme C suivant ? Expliquez en détail votre réponse.

```
1  #include <stdio.h>
2
3  /* schémas des fonctions */
4  void affiche1(int x);
5  void affiche3(int* ptr);
6
7  /* déclaration des fonctions */
8
9  void affiche1(int x){
10   x = x + 1;
11   printf("affiche1 : x=%d\n",x);
```

```
12 }
13
14 void affiche3(int* ptr){
15   *ptr = (*ptr) + 1;
16 }
17
18 int main(){
19   int x;
20
21   x = 10;
22   printf("main: x=%d\n",x);
23   affiche1(x);
24   printf("main: x=%d\n",x);
25   affiche3(&x);
26   printf("main: x=%d\n",x);
27 }
```

4 Pointeurs

Exercice 4.1 : Le milliard de lapins revisité

On souhaite modifier la fonction `lapins_un_milliard` pour pouvoir obtenir non seulement le nombre de mois nécessaires à l'obtention d'un milliard de lapins, mais aussi le nombre exact de lapins. Pour cela, on utilise le schéma de fonction : `void lapins_un_milliard(int* nb_lapins, int* temps)`.

Question 1 : Pourquoi avec un tel schéma la fonction ne renvoie-t-elle pas de résultat ?

Question 2 : Écrire la nouvelle fonction en `C`, ainsi que la fonction principale l'appelant.

Exercice 4.2 : Échange de valeurs de variables

On souhaite écrire une fonction `échange` réalisant l'échange des valeurs de deux variables.

Question 1 : Les deux variables doivent-elles être passées en paramètre par valeur ou par adresse ? Justifiez votre réponse.

Question 2 : Écrire l'algorithme de la fonction `echange`.

Question 3 : Écrire une fonction principale qui initialise les deux variables, affiche leur valeur, appelle la fonction `echange` et enfin affiche les (nouvelles) valeurs de ces deux variables.

Question 4 : Mettre en œuvre ces fonctions dans un programme `C`.

5 Chaînes de caractères

Exercice 5.1 : Manipulation de chaînes de caractères

Question 1 : Écrire une fonction `char *append(char *ch1, char *ch2)` qui construit la concaténation des chaînes `ch1` et `ch2`. Cette fonction devra allouer dynamiquement la place nécessaire pour la chaîne résultat et renvoyer un pointeur sur cette chaîne.

Question 2 : Écrire une fonction `char *RenverseChaine(char * ch)` qui renvoie un pointeur vers une chaîne de caractères correspondant à `ch` écrit à l'envers.

Question 3 : Un *palindrome* est une chaîne de caractères qui se lit de la même façon de gauche à droite et de droite à gauche. Écrire, en utilisant `strcmp` et `RenverseChaine`, une fonction `int EstPalindrome(char *ch)`, qui revoie 1 si `ch` est un palindrome, et 0 sinon.

Question 4 : Pourquoi la solution retenue ci-dessus n'est-elle pas efficace ? Proposer une nouvelle fonction.

Question 5 : Écrire le programme principal `main` pour exécuter ces fonctions sur deux chaînes de caractères passées en paramètre.

6 Tableaux

Exercice 6.1 : Tableaux et adresses

Soit un tableau `tab` de dix éléments de type `int`.

Question 1 : Écrire un programme affichant les adresses des éléments `tab[0]` et `tab[4]`. L'affichage sera effectué de deux manières : en utilisant la notation des tableaux, et en utilisant les pointeurs.

Question 2 : Quelle est la différence entre ces deux adresses ?

Exercice 6.2 : Triangle de Pascal

Le triangle de Pascal est utilisé pour calculer les coefficients du développement de $(a+b)^n$. Il a la forme suivante :

$$\begin{array}{ccccc} 1 & & & & \\ 1 & 1 & & & \\ 1 & 2 & 1 & & \\ 1 & 3 & 3 & 1 & \\ 1 & 4 & 6 & 4 & 1 \end{array}$$

La colonne de gauche et la diagonale ne contiennent que des 1. L'élément $x_{i,j}$ à la ligne i et la colonne j vaut : $x_{i,j} = x_{i-1,j-1} + x_{i-1,j}$.

On peut, à partir de la cinquième ligne du triangle ci-dessus, déduire :

$$(a+b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$$

Écrire un programme `C` prenant comme paramètre un entier n et affichant le développement de $(a+b)^n$. On utilisera une fonction `void suivant(int *prec, int *cour)` prenant comme paramètres les deux dernières lignes calculées, effectuant le calcul d'une nouvelle ligne, et mettant à jour les paramètres pour que ce soient toujours les deux dernières ligne calculées. Cette fonction affichera également la nouvelle ligne. On commencera par réfléchir aux structures de données utilisées.

Exercice 6.3 : Tableaux et pointeurs

Qu'affiche le programme suivant ? Expliquer son fonctionnement.

```
1  #include <stdio.h>
2
3  int main(){
4    char *c[] = {"ENTER","NEW","POINT","FIRST"};
5    char **cp[] = {c+3,c+2,c+1,c};
6    char ***cpp = cp;
7
8    printf("%s",**++cpp);
9    printf("%s ",*--*++cpp+3);
10   printf("%s",*cpp[-2] +3);
11   printf("%s\n",cpp[-1][-1] +1);
12 }
```

7 Récursion

Exercice 7.1 : Factorielle

Écrire un programme calculant la factorielle d'un nombe de manière récursive.

Exercice 7.2 : Triangle de Pascal

Réécrire le programme calculant le triangle de Pascal sous forme récursive.

Exercice 7.3 : Tri fusion

On souhaite écrire un programme réalisant le tri des éléments d'un tableau par la technique de *tri-fusion*. Pour cela, un tableau à trier est divisé en deux parties la première moitié et la seconde. Chacun de ces tableaux plus petits est trié suivant le même algorithme. Ensuite, les deux tableaux triés ainsi obtenus sont fusionnés en respectant l'ordre, pour obtenir le tableau d'origine trié.

Question 1 : Appliquer (à la main) cette technique pour trier le tableau suivant :

8	12	4	5	1	2	9

Question 2 : Écrire un programme `C` implémentant l'algorithme de tri-fusion.

8 Structures, listes, entrées/sorties

Exercice 8.1 : Répertoire téléphonique

Nous souhaitons écrire un programme gérant un répertoire téléphonique. Le programme aura les fonctionnalités suivantes :

- *lecture* et *écriture* d'un répertoire téléphonique *dans un fichier* ;
- *ajout* d'une fiche dans le répertoire ;
- *affichage* de l'intégralité du répertoire ;

Pour gérer le répertoire, on utilisera les structures de données et définitions de types suivantes, stockées dans un fichier `repertoire.h` :

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

#define true 1
#define false 0

/* Définition d'une fiche du répertoire */
typedef struct{
  string nom;
  string prenom;
  string tel;
  } fiche, *ficheptr;

/* Définition d'un onglet dans le répertoire */
typedef struct ONGLET{
  ficheptr lafiche;
  struct ONGLET suivante;
  } onglet, *ongletptr;

/* Définition du répertoire */
typedef ongletptr repertoire[26];
```

Le *répertoire* téléphonique est composé de 26 onglets, un par lettre de l'alphabet. Le premier onglet, `repertoire[0]` contient donc tous les renseignements sur les personnes dont le nom commence par A. Chaque *onglet* est une liste de fiches, et une *fiche* est composée du nom, du prénom et du numéro de téléphone d'une personne.

Question 1 : Écrire les fonctions permettant l'affichage d'un répertoire. Pour cela, on effectuera un découpage en 3 fonctions :

- `void affiche_repertoire(repertoire lerepertoire)`
- `void affiche_onglet(ongletptr longlet)`
- `void affichefiche(ficheptr lafiche)`

Pour chaque fiche, on affichera sur une ligne le nom, le prénom et le numéro de téléphone. Chaque onglet aura sa lettre affichée, puis la liste de ses fiches.

Question 2 : Comment modifier ces fonctions de manière à pouvoir les utiliser pour sauvegarder le répertoire dans un fichier ? Ces fonctions seront écrites dans un fichier `affiche.c`.

Question 3 : Le fichier `ESfichiers.c` contiendra 2 fonctions gérant les entrées/sorties du programme de répertoire téléphonique dans des fichiers :

- `ficheptr lire_fiche(FILE *fichier)` permet de lie une fiche depuis un fichier et de remplir la structure de données correspondante. Cette fonction renvoie un pointeur sur la fiche nouvellement créée, ou un pointeur nul s'il n'y a pas de nouvelle fiche, c'est-à-dire en cas d'erreur ou de fin de fichier.
- `void ecrire_repertoire(FILE *fichier, repertoire lerepertoire)` sera appelée à la place de `affiche_repertoire` lorsque l'on souhaite sauvegarder le répertoire dans un fichier et non l'afficher à l'écran.

Écrire ces fonctions.

Question 4 : Écrire un fichier `gestion.c` contenant les fonctions relatives à la gestion du répertoire, c'est-à-dire l'ajout de nouvelles fiches :

- `void ajouter_fiche(repertoire lerepertoire, ficheptr lafiche)` ajoute `lafiche` au répertoire `lerepertoire`, dans l'onglet idoine. Elle aura préalablement transformé la première lettre du nom en majuscule si cela s'avère nécessaire.
- `void majuscule(ficheptr lafiche)` transforme la première lettre du nom en majuscule si c'est nécessaire.
- `void ajouter_dans_onglet(onglettr *longlet, ficheptr lafiche)` ajoute la fiche dans l'onglet. Celui-ci doit être passé par adresse, car s'il n'y a aucune fiche dans l'onglet, il sera modifié. Les fiche devront être rangées dans l'onglet selon l'ordre alphabétique sur les noms puis sur les prénoms.
- `int avant(ficheptr fiche1, ficheptr fiche2)` renvoie vrai si la `fiche1` doit être rangée avant la `fiche2`, faux sinon.

Question 5 : Écrire la partie principale du programme dans un fichier `main.c`. Il contiendra les fonctions suivantes :

- `int main()` est la fonction principale.
- `void repertoire_vide(repertoire lerepertoire)` initialise le répertoire vide.
- `void menu(repertoire lerepertoire)` propose à l'utilisateur le menu suivant, et appelle les fonctions appropriées :
  1. lecture du répertoire depuis le clavier
  2. lecture du répertoire depuis un fichier
  3. affichage du répertoire à l'écran
  4. sauvegarde du répertoire dans un fichier
  5. ajout d'une fiche
  6. quitter
- `void free_repertoire(repertoire lerepertoire)` libère toutes les structures de données allouées dynamiquement pour le répertoire. Cela permet de finir le programme proprement.

Question 6 : Écrire un fichier `makefile` pour la compilation de votre programme.