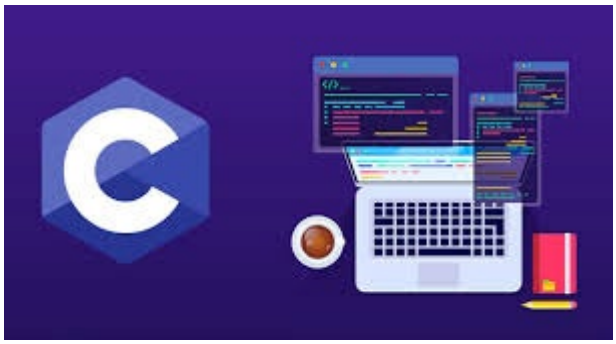
Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

HE IG VD

Département TIC

PRG2

Recueil d'exercices



René Rentsch

Table des matières

Chapitre 1 : Introduction

Exercice 1.1 Substitutions

En supposant les <?> du code ci-dessous remplacés par le code de format adéquat, indiquer ce que va afficher le programme suivant ?

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

#define A 1
#define B 2
#define AB 3
#define F(X,Y) X##Y

int main(void) {

   int n = AB;

   printf("%<?>\n", n);
   printf("%<?>\n", "AB");
   printf("%<?>\n", F(A, B));

   return EXIT_SUCCESS;
}
```

Exercice 1.2 Macros (1)

Le programme suivant contient une ou plusieurs erreurs. Corrigez-le de telle sorte qu'il affiche à l'exécution :

```
25 -1
```

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

#define A = 2;
#define B = A + 1;
#define PLUS (X,Y) X+Y;
#define MOINS(X,Y) X-Y;

int main(void) {

   printf("%d %d\n", 5*PLUS(A,B), MOINS(A+1,B+1));

   return EXIT_SUCCESS;
}
```

Exercice 1.3 Macros (2)

Soit le programme suivant (`ABS` = macro "valeur absolue") :

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

#define ABS(X) (((X) > 0) ? (X) : (-X))

int main(void) {

   int n;

   printf("1. %d\n", ABS(2));
   printf("2. %d\n", ABS(-2));
   printf("3. %d\n", ABS('A'));

   n = -2;
   printf("4. %d\n", ABS(n+1));

   n = -2;
   printf("5. %d", ABS(n++)); printf(" %d\n", n);

   n = -2;
   printf("6. %d", ABS(++n)); printf(" %d\n", n);

   n = -2;
   printf("7. %d", abs(++n)); printf(" %d\n", n);

   return EXIT_SUCCESS;
}
```

1) Que va afficher ce programme ?
2) Expliquez en quoi le programme ci-dessus est problématique.
3) Est-il possible de récrire (sans utiliser la fonction `abs` de `stdlib`) la macro `ABS` de manière à éviter le(s) problème(s) du point 2) ?

Exercice 1.4 Opérateurs de manipulation de bits (1)

Que vaut chacune des expressions suivantes ?

1) `22 | 11`
2) `0 ^ 19`
3) `2 << 3`
4) `30 & 14`
5) `8 >> 2`
6) `4 & 29`
7) `9 << 4`
8) `31 ^ 27`
9) `23 | 3`
10) `-1 >> 1`
11) `-5 >> 1`
12) `3 & ~2`
13) `6 | 5 & 4`

Exercice 1.5 Récupérer la valeur du nième bit

Sans faire usage de l'instruction *if* et en utilisant *exclusivement* des opérateurs de manipulation de bits, implémenter la fonction dont le prototype et la sémantique sont donnés ci-dessous.

```
unsigned short getBit(unsigned short pos, int n);
```

Sémantique
Retourne la valeur du bit en position *pos* dans la représentation binaire du nombre *n*
NB *pos = 0 correspond à la position du bit de poids le plus faible*

Exemples : getBit(0, 5) = 1; getBit(1, 5) = 0; getBit(2, 5) = 1

Exercice 1.6 Fixer la valeur du nième bit

Sans faire usage de l'instruction *if* et en utilisant *exclusivement* des opérateurs de manipulation de bits, implémenter la fonction dont le prototype et la sémantique sont donnés ci-dessous.

```
int* setBit(unsigned short pos, unsigned short bitValue, int* n);
```

Sémantique
Met le bit en position *pos* dans la représentation binaire du nombre *n* à la valeur *bitValue*
NB *pos = 0 correspond à la position du bit de poids le plus faible*

Exercice 1.7 Position du bit à 1 de plus faible poids

En exploitant au maximum les opérateurs de manipulation de bits, implémenter la fonction dont le prototype et la sémantique sont donnés ci-dessous.

```
short lowestOrderSetBit(int n);
```

Sémantique
Retourne la position du bit à 1 de plus faible poids. Retourne -1 si aucun bit n'est à 1.
NB *pos = 0 correspond à la position du bit de poids le plus faible*

Exercice 1.8 Représentation binaire d'un entier

En exploitant au maximum les opérateurs de manipulation de bits, implémenter la fonction dont le prototype et la sémantique sont donnés ci-dessous.

```
void decimalToBinary(int32_t n, int8_t binary[]);
```

Sémantique
Convertit en binaire le nombre entier décimal *n* et place le résultat dans le tableau *binary*.

Exercice 1.9 printf (1)

Que va afficher le programme ci-dessous ?
(Indiquez par b la présence d'un espace blanc)

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

int main(void) {

   int n = 127;
   double x = 12.3456;

   printf("R1 : |%-4d|\n", n);
   printf("R2 : |%04d|\n", n);
   printf("R3 : |%x|\n", n);
   printf("R4 : |%#o|\n", n);
   printf("R5 : |%f|\n", x);
   printf("R6 : |%5.2f|\n", x);
   printf("R7 : |%.2e|\n", x);
   printf("R8 : |%g|\n", x);

   return EXIT_SUCCESS;
}
```

Exercice 1.10 printf (2)

Soit le squelette de code suivant :

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

int main(void) {

   int n = 255;
   double x = 12.345;

   < à compléter>

   return EXIT_SUCCESS;
}
```

Compléter le code ci-dessus de sorte qu'à l'exécution il reproduise *à l'identique* l'output suivant :

```
0377
FF
+###255
1.235e+01
12.345
12.3450
```

Exercice 1.11 printf (3)

Compléter le code ci-dessous de telle sorte que celui-ci affiche à l'exécution :

```
i = 1
j = 4294967295
```

Important Le code, une fois complété, doit compiler sans warnings.

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

int main(void) {

   size_t i = 1;
   uint32_t j = UINT32_MAX;


   <à compléter>

   return EXIT_SUCCESS;
}
```

Exercice 1.12 scanf

Le code ci-dessous n'est pas correct. Le récrire complètement de sorte à corriger toutes les erreurs / maladresses commises.

```
#include <cstdlib>
#include <cstdio>
using namespace std;

int main() {

   int n, char c;

   printf("Donnez un nombre entier et un caractere : ");
   scanf("%d%c", n, c);
   printf("n = %d, c = %c\n", n, c);

   return EXIT_SUCCESS;
}
```

Exercice 1.13 Lire au maximum n caractères d'une chaîne

Soit le programme C suivant :

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

#define TAILLE_NOM 20

int main(void) {

   char nom[TAILLE_NOM + 1];

   printf("Entrez votre nom (%d caract. max) : ", TAILLE_NOM);
   scanf("%20s", nom);

   printf("Votre nom est \"%s\"\n", nom);

   system("PAUSE");
   return EXIT_SUCCESS;
}
```

1) Ce programme, pourtant très simple, comporte 3 défauts majeurs. Lesquels ?
2) Récrire le code du programme ci-dessus en corrigeant lesdits défauts.

Chapitre 2 : Pointeurs

Exercice 2.1 Les bases…

Ecrire un programme C qui :

1) Déclare une variable *n* de type *int* et l'initialise à la valeur 1
2) Déclare un pointeur *ptr* pointant sur *n*
3) Affiche à l'écran la valeur de l'objet pointé par *ptr*
4) Affiche à l'écran l'adresse contenue dans *ptr*
5) Affiche l'adresse de *ptr*

Exercice 2.2 Paramètres d'entrée-sortie

Ecrire une fonction C sans valeur de retour (*void*) qui prend comme paramètre d'entrée un nombre réel *x* (de type *double*) et qui renvoie, par paramètres d'entrée-sortie, le carré et le cube de *x*.

Exercice 2.3 Fonction avec valeur de retour de type pointeur

Que va produire à l'exécution le programme C ci-dessous ?

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

int* carres(const int tab[]);

int main(void) {
   int tab[] = {1, 2, 3};
   const size_t N = sizeof(tab) / sizeof(int);
   int* ptr = carres(tab);
   for (size_t i = 0; i < N; ++i)
      printf("%d ", ptr[i]);
   printf("\n");
   return EXIT_SUCCESS;
}

int* carres(const int tab[]) {
   const size_t N = sizeof(tab) / sizeof(int);
   int copie[N];
   for (size_t i = 0; i < N; ++i)
      copie[i] = tab[i] * tab[i];
   return copie;
}
```

Exercice 2.4 Interprétation de déclarateurs avec pointeurs

Traduire en français les déclarations C suivantes :
(Exemple : pour int ptr, il faut répondre "ptr est un pointeur sur int")*

1) `char** a[10];`
2) `double* (*b)(void);`
3) `double (*c)(double*);`
4) `int* d[10];`
5) `int (*e)[10];`
6) `int (*(*f[5])(void))[10];`
7) `double (**g)[5];`

Exercice 2.5 Ecriture de déclarateurs avec pointeurs

Ecrire les déclarations C correspondant aux énoncés suivants :

1) t est un tableau de 10 pointeurs pointant chacun sur un *int* constant
2) t est un tableau de 10 pointeurs constants pointant chacun sur un *int*
3) p est un pointeur sur une fonction prenant en paramètre un pointeur sur une fonction (prenant en paramètre un *double* et livrant un *double*) et renvoyant un pointeur sur *int*
4) p est un pointeur constant sur un tableau de 10 pointeurs sur *double*
5) t est un tableau de 10 pointeurs pointant chacun sur une fonction prenant un *double* comme paramètre et renvoyant un pointeur sur *char*
6) f est une fonction prenant en paramètre un pointeur constant sur *char* et renvoyant un pointeur constant contenant l'adresse d'un pointeur sur *char*
7) f est une fonction sans paramètre renvoyant un pointeur sur un tableau de 10 pointeurs constants sur *char*

Exercice 2.6 Affectations entre pointeurs

Soient les déclarations suivantes :

```
int i = 1, j = 2;
int* pi1 = &i;
int* pi2 = &j;
double x = 3.0;
double* pd = &x;
void* pv;
```

Quel sera dans un programme l'effet des instructions suivantes ?

1) pi1 = pi2;
2) pd = pi1;
3) pi1 = pd;
4) pv = pi1;
5) pv = &i;
6) pv = pi1;
   pi2 = pv;
7) if (pi1 = pi2)…;

Exercice 2.7 Arithmétique des pointeurs (1)

Que va afficher le programme *main* ci-dessous ?
(Conseil : Aidez-vous d'un petit dessin)

```
#include <inttypes.h>
#include <stddef.h> // pour ptrdiff_t
#include <stdlib.h>
#include <stdio.h>

int main(void) {

   int a[] = {10, 20, 30, 40, 50};
   int* p[] = {a, a+1, a+2, a+3, a+4};
   int** pp = &p[2];
   int*** ppp = &pp;

   printf("1) %d\n", *a);
   printf("2) %d\n", **p);
   printf("3) %d\n", **pp++);
   printf("4) %d\n", ***ppp);
   pp = p;
   printf("5) %d\n", **++pp);
   pp = p;
   printf("6) %d\n", ++**pp);
   pp = p;
   printf("7) %d\n", *pp[1]);
   pp = p;
   printf("8) %" PRIdPTR "\n", (ptrdiff_t) (*(p+1) - *pp));

   return EXIT_SUCCESS;
}
```

Exercice 2.8 Arithmétique des pointeurs (2)

Que va afficher le programme *main* ci-dessous ?
(Conseil : Aidez-vous d'un petit dessin)

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

int main(void) {

   int t[6], i, *p;

   for (p = &t[5]; p > &t[-1]; p--)
      *p = (int) (p - t);

   for (i = 0; i < 6; i++)
      printf("%d ", t[i]);
   printf("\n");

   for (i = 0, ++p; i < 6; i++)
      printf("%d ", *p++/2);
   printf("\n");

   return EXIT_SUCCESS;
}
```

Exercice 2.9 Arithmétique des pointeurs (3)

Soient les déclarations suivantes :

```
const char* c[] = {"elle", "mangera", "une", "petite", "tomme"};
const char** d[] = {c+1, c+2, c+3, c+4, c};
const char*** e = &d[3];
```

Quelles valeurs fournissent les expressions suivantes ?
(Conseil : Aidez-vous d'un petit dessin)

a) `c[3][0]`
b) `(**d)[5]`
c) `(**e)[*d-c]`
d) `(d[3]-3)[0][3]`
e) `**d + 5`
f) `*d[3] + 2`
g) `*(*e[-3] + 5)`
h) `**c`
i) `e[0][0][e-d]+1`
j) `0[c][0] - 'd' + 'B'`

Remarques

- Pour les chaînes de caractères (char *), donner le résultat entre guillemets. Ex : "ABC".
- Pour les caractères (char), donner la représentation littérale. Ex : ‘a’.

Exercice 2.10 Arithmétique des pointeurs (4)

Que va afficher le programme *main* ci-dessous ?
(Conseil : Aidez-vous d'un petit dessin)

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

int main(void) {

   const char* c[] = {"comprendre", "les", "pointeurs", "c\'est", "difficile"};
   const char** cp[] = {c, c + 2, c + 4, c + 1, c + 3};
   const char*** cpp = cp;
   int i;

   for (i = 0; i < 3; i++)
      printf("%c", *(**cpp + i));
   printf("%c", *(*cp[0] + 2));
   printf("%s ", *cpp[2] + 8);
   printf("%s ", *++*++cpp);
   for (i = 1; i < 4; i++)
      printf("%c", *(cpp[-1][i % 3] + 2));
   printf("%c", **--*cpp);
   printf("%s\n", **++cpp + 7);

   return EXIT_SUCCESS;
}
```

Exercice 2.11 Formalisme pointeur et tableaux 1D

Ecrire d'au moins 6 manières différentes, mais toujours en utilisant le formalisme pointeur, une fonction C *initialiser*, sans valeur de retour (*void*), permettant d'initialiser à une valeur donnée (de type *int*), tous les éléments d'un tableau à 1 dimension (1D) de taille quelconque et composé d'entiers (de type *int*).

Exercice 2.12 Inversion d'un tableau 1D (1)

Implémenter la fonction C dont le prototype et la sémantique sont donnés ci-dessous :

```
void inverser(int* debut, int* fin);
```

Sémantique
Inverse le contenu du tableau 1D défini par *début* et *fin* où *début*, resp. *fin*, désigne l'adresse du premier, resp. du dernier, élément du tableau à inverser.

Exercice 2.13 Méli-mélo

Que va afficher le programme *main* ci-dessous ?
(On suppose travailler ici avec une architecture 32 bits)

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

int main(void) {

   int i = 1, j = 2;
   int *p = &i, *q = &j;

   *p = (int) q;
   *q = (int) p;
   p = (int*) *p;
   q = (int*) *q;

   i = 3, j = 4;
   printf("*p = %d *q = %d\n", *p, *q);

   return EXIT_SUCCESS;
}
```

Exercice 2.14 Pointeurs, tableaux et fonctions

Que va afficher le programme *main* ci-dessous ?

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

void f(int* p1, int p2);

int main(void) {

   int a = 5, b = 6;
   int t[2] = {3, 4};
   int *p = NULL, *q = NULL;

   f(&a, b);
   printf("1) %d, %d\n", a, b);

   p = &a;
   q = &b;
   f(q, *p);
   printf("2) %d, %d\n", *p, *q);

   f(t + 1, *t);
   printf("3) %d, %d\n", t[0], t[1]);

   f((int*)(&p), b);
   printf("4) %d, %d\n", a, b);

   return EXIT_SUCCESS;
}

void f(int* p1, int p2) {
   *p1 = 2 * p2;
}
```

Exercice 2.15 Utilisation de calloc

Ecrire une fonction C *initialiser* permettant 1°) de créer un tableau (à 1 dimension) de taille donnée, composé d'entiers (de type *int*), 2°) d'initialiser à une valeur donnée (de type *int*) tous les éléments de ce tableau.

IMPORTANT
La fonction :

- ne doit comporter que 2 paramètres : le premier fixant la taille (le nombre d'éléments) du tableau à créer, le second fixant la valeur à donner à tous les éléments du tableau
- doit utiliser les services de la fonction `calloc`

Ecrire aussi un petit programme de test permettant de vérifier le bon fonctionnement de votre fonction.

Exercice 2.16 Inversion d'un tableau 1D (2)

Implémenter la fonction C dont le prototype et la sémantique sont donnés ci-dessous :

```
int* inverse(const int* debut, const int* fin);
```

Sémantique
Renvoie le tableau inverse du tableau 1D défini par *début* et *fin* où *début*, resp. *fin*, désigne l'adresse du premier, resp. du dernier, élément du tableau à inverser.

Exercice 2.17 Adresses du min et du max d'un tableau 1D

Ecrire une fonction C qui prend en paramètre un tableau *tab* 1D de *int* de taille quelconque et qui renvoie en valeur de retour les adresses du plus petit élément et du plus grand élément[1] de *tab* (ou *NULL* si *tab* vaut *NULL* ou si *tab* est vide).

[1] Dans le cas où il y aurait plusieurs plus petits éléments on renverra l'adresse de celui dont l'indice dans *tab* est le plus petit (idem dans le cas du plus grand élément).

Exercice 2.18 Manipulation de la mémoire (1)

Ecrire un programme C qui :

1) déclare et initialise un tableau *tab1* à une dimension de *int*
2) copie (sans utiliser de boucle) le contenu de *tab1* dans un autre tableau *tab2*
3) affiche *tab1* et *tab2*
4) se termine

NB Prévoir une fonction dédiée à l'affichage d'un tableau sous la forme [valeur1, valeur2,…]. Ecrire cette fonction en utilisant exclusivement le formalisme pointeur.

Exercice 2.19 Manipulation de la mémoire (2)

Sans utiliser de boucle, compléter la partie notée *<à compléter>* dans le programme C ci-dessous de telle sorte que celui-ci affiche à l'exécution :

```
[0, 0, 0]
[1, 1, 1]
[2, 2, 2]
```

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

#define SIZE 3

void afficher … // Le code de la fonction afficher a volontairement été omis ici

int main(void) {

   int tab[SIZE] = {0};

   afficher(tab, SIZE);

   for (size_t i = 0; i < SIZE; ++i)
      tab[i]++;

   afficher(tab, SIZE);

   <à compléter>

   for (size_t i = 0; i < SIZE; ++i)
      tab[i] += 2;

   afficher(tab, SIZE);

   return EXIT_SUCCESS;
}
```

Exercice 2.20 Manipulation de la mémoire (3)

Que va afficher à l'exécution le programme C suivant ?

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

int main(void) {
   char str[] = "memmove est tres utile......";
   memmove(str + 17, str + 12, 10);
   printf("%s\n", str);
   return EXIT_SUCCESS;
}
```

Exercice 2.21 Initialisation d'une matrice

Ecrire une fonction C sans valeur de retour (*void*) qui initialise une matrice *m x n* de *int* avec des 1 sur les 4 "bords" et des 0 partout ailleurs.

Ecrire aussi un petit programme de test permettant de vérifier le bon fonctionnement de votre fonction.

Exercice 2.22 Somme des valeurs d'une matrice

Ecrire une fonction C qui renvoie la somme des coefficients de la matrice *m x n* de *double* passée en paramètre.

Ecrire aussi un petit programme de test permettant de vérifier le bon fonctionnement de votre fonction sur la matrice 3 x 4 suivante :

$$M = \begin{pmatrix} 1 & 2.5 & 3 & 4 \\ 5 & 6 & 7.5 & 8 \\ 9.5 & 10 & 11 & 12 \end{pmatrix}$$

Exercice 2.23 Diagonale d'une matrice carrée

Ecrire une fonction C qui prend en paramètre une matrice *n x n* de *int* et qui renvoie en valeur de retour le vecteur correspondant aux éléments de la diagonale gauche de ladite matrice.

Exemple : Si matrice = $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$, la fonction doit renvoyer le vecteur $\begin{pmatrix} 1 \\ 5 \\ 9 \end{pmatrix}$

Ecrire aussi un petit programme de test permettant de vérifier le bon fonctionnement de votre fonction.

Exercice 2.24 Calcul d'une intégrale

Ecrire une fonction C permettant d'intégrer, par la méthode des trapèzes, n'importe quelle fonction f(x) (x de type *double*) entre deux bornes A et B (également de type *double*). Le nombre de pas d'intégration est aussi un paramètre de la fonction d'intégration.

Tester votre fonction d'intégration en écrivant un petit programme permettant de calculer :

$$\int_0^{\infty} e^{-x^2} dx \ \left(= \frac{\sqrt{\pi}}{2} = 0.886227 \right)$$

$$\int_0^{\frac{\pi}{2}} \sin(x) dx \ (=1)$$

Chapitre 3 : Types composés

Exercice 3.1 Correction d'erreurs

Réécrire complètement le code ci-dessous de telle sorte à corriger tous les maladresses et erreurs qu'il contient.

Une fois dûment corrigé le programme devrait afficher à l'exécution :

```
a = 1, b = "ABC"
```

NB Aucune ligne de code ne doit être ajoutée ou supprimée.

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

struct S {
   int a,
   char b[3]
}

void afficher(S* s);

int main(void) {
   S s = {a = 1; b = "ABC"};
   afficher(s);
   return EXIT_SUCCESS;
}

void afficher(S* s) {
   printf("a = %d, b = "%s"\n", s.a, s.b);
}
```

Exercice 3.2 Affichage d'une personne (1)

On suppose qu'une personne se caractérise par :

- son nom (20 caractères effectifs au maximum)
- sa taille en cm (type *uint8_t*)
- la couleur de ses yeux (type *enum*; couleurs possibles : bleu, vert, gris, marron, noir)

Ecrire un programme C qui, dans un premier temps, déclare, le plus proprement possible, tous les types nécessaires à la modélisation d'une personne (n'hésitez pas à faire usage de *typedef*) et qui, dans un second temps, déclare/définit deux fonctions (sans valeur de retour) *afficher_1* et *afficher_2*, permettant d'afficher l'ensemble des caractéristiques de la personne passée en paramètre. L'argument de la fonction *afficher_1* doit être passé par valeur, celui de *afficher_2*, par adresse.

Testez votre programme en considérant comme exemple de personne : Toto qui mesure 180 cm et qui a les yeux bleus.

Exercice 3.3 Affichage de plusieurs personnes

Essentiellement le même exercice que le précédent, mais on souhaite ici afficher non pas une personne mais plusieurs.

Plus précisément, il est demandé :

- de modéliser le concept de Personne en s'affranchissant de la contrainte des 20 caractères max pour le nom
- de déclarer un tableau contenant les 3 personnes suivantes :
  - Paul qui mesure 180 cm et qui a les yeux bleus
  - Pierre qui mesure 175 cm et qui a les yeux verts
  - Jean-Jacques qui mesure 182 cm et qui a les yeux marron
- d'écrire :
  - une fonction permettant l'affichage d'une personne (passée par adresse)
  - une fonction permettant l'affichage d'un tableau de personnes

Exemple d'exécution du programme :

```
Nom    : Paul
Taille : 180
Yeux   : bleus

Nom    : Pierre
Taille : 175
Yeux   : verts

Nom    : Jean-Jacques
Taille : 182
Yeux   : marron
```

Exercice 3.4 Lendemain d'une date

Ecrire un programme C qui détermine et affiche le lendemain d'une date donnée.

Exemple `Le lendemain du 30.04.2016 est le 01.05.2016`

Précisions

- Implémenter le concept de date sous la forme d'une structure à 3 membres : jour (de type *uint_8*), mois (de type *uint_8*) et année (de type *uint16_t*)
- Structurer le code en fonctions
- Tenir compte des années bissextiles dans le calcul du lendemain
- Afficher les dates au format jj.mm.aaaa

Exercice 3.5 Taille d'une structure

a) Soit la structure *S1* suivante :

```
struct S1 {
   int* ptr;
   int32_t a;
   int64_t b;
};
```

Quelle est la taille en bytes de *S1*

1) en architecture 32 bits ?
2) en architecture 64 bits ?

b) Soit la structure *S2* suivante :

```
struct S2 {
   int8_t a;
   int16_t b;
   char c;
   double* ptr;
};
```

Quelle est la taille en bytes de *S2*

1) en architecture 32 bits ?
2) en architecture 64 bits ?

Exercice 3.6 Affichage d'une personne (2)

On souhaite modéliser des personnes, certaines de nationalité suisse, d'autres pas.

On suppose que toute personne (qu'elle soit suisse ou non) se caractérise par un nom (20 caractères effectifs max).

Pour les personnes de nationalité suisse, on souhaite (en plus du nom) aussi enregistrer le taux d'activité exercé (entier exprimé en %).

Pour les personnes de nationalité étrangère, on souhaite (en plus du nom) aussi enregistrer le type de permis de travail dont elle dispose (permis A, B ou C; à implémenter en tant que type énuméré).

Ecrire un programme C, qui, après avoir déclaré le plus proprement possible les types nécessaires à la résolution de notre problème, déclare/initialise les 2 personnes suivantes :

- "Toto", suisse travaillant à 80%
- "Titi", étranger avec permis C

… puis affiche à l'écran l'ensemble des caractéristiques de chacune de ces 2 personnes.

Exemple d'exécution du programme :

```
Nom           : Toto
Nationalite   : Suisse
Taux activite : 80%

Nom           : Titi
Nationalite   : Etranger
Type permis   : C
```

Exercice 3.7 Bateaux

On souhaite modéliser des bateaux, plus précisément des voiliers et des bateaux à moteur.

On dispose des informations suivantes :

- Tout bateau a un nom (de longueur quelconque)
- Si le bateau est un voilier, il faut enregistrer la surface de la voilure en [m^2] (type *uint16_t*) de celui-ci
- Les bateaux à moteur se divise en 2 catégories : les bateaux de pêche et les bateaux de plaisance.
- Tout bateau à moteur se caractérise par la puissance totale de ses moteurs, exprimée en [CV] (type *uint16_t*)
- Si le bateau est un bateau de pêche, il faut enregistrer combien de tonnes de poisson celui-ci est autorisé à pêcher (de type *uint8_t*)
- Si le bateau est un bateau de plaisance, il faut enregistrer le nom de son propriétaire (de longueur quelconque)

Ecrire un programme C, qui, après avoir déclaré le plus proprement possible les types nécessaires à la résolution de notre problème, déclare/initialise les 3 bateaux suivants :

- "Alinghi", voilier ayant 300 [m2] de voilure
- "Espadon", bateau de pêche disposant de 2 moteurs de 500 [CV] chacun et pouvant pêcher 20 tonnes de poisson
- "Farniente", bateau de plaisance disposant d'un moteur de 100 [CV] et appartenant à Monsieur James Lamer

… puis affiche à l'écran l'ensemble des caractéristiques de chacun de ces 3 bateaux.

Exemple d'exécution du programme :

```
Nom          : Alinghi
Genre        : Voilier
Voilure [m2] : 300

Nom          : Espadon
Genre        : Bateau de peche
Moteurs [CV] : 1000
Peche [t]    : 20


Nom          : Farniente
Genre        : Bateau de plaisance
Moteurs [CV] : 100
Proprietaire : James Lamer
```

Exercice 3.8 Affichage des jours de la semaine (1)

En utilisant le concept de type énuméré, écrire un programme C qui :

1) demande à l'utilisateur de saisir, en toutes lettres et entièrement en minuscules, un jour de la semaine[1]
2) affiche, entièrement en minuscules et à raison d'un nom par ligne, tous les jours de la semaine, en commençant par celui saisi par l'utilisateur
3) se termine

[1] Il n'est pas demandé ici de vérifier la saisie utilisateur.

Exemple d'exécution

```
Donnez un jour de la semaine en toutes lettres et en minuscules : jeudi
jeudi
vendredi
samedi
dimanche
lundi
mardi
mercredi
```

Exercice 3.9 Affichage des jours de la semaine (2)

Même exercice que le 3.8, mais cette fois :

- l'utilisateur doit pouvoir saisir le nom d'un jour en minuscules et/ou majuscules avec d'éventuels espaces blancs devant et derrière
- le programme doit inviter l'utilisateur à refaire sa saisie si celle-ci n'est pas correcte

Exemple d'exécution

```
Donnez un jour de la semaine en toutes lettres : aaa
Saisie incorrecte. Veuillez SVP recommencer.
Donnez un jour de la semaine en toutes lettres :    venDREdi
vendredi
samedi
dimanche
lundi
mardi
mercredi
jeudi
```

Exercice 3.10 Pile dynamique

Compléter la partie notée ***<à compléter>*** du code ci-dessous de telle sorte que celui-ci affiche (si l'utilisateur entre successivement les valeurs 1, 2, 3 et 0) :

```
Donnez un entier (0 pour terminer) : 1
Donnez un entier (0 pour terminer) : 2
Donnez un entier (0 pour terminer) : 3
Donnez un entier (0 pour terminer) : 0
3
2
1
```

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

<à compléter>

int main(void) {

   Pile pile = NULL;
   Info info;

   // Saisie utilisateur
   do {
      printf("Donnez un entier (0 pour terminer) : ");
      scanf("%d", &info);
      if (info != 0)
         empiler(&pile, info);
   } while (info != 0);

   // Affichage dans l'ordre inverse des valeurs saisies par l'utilisateur
   printf("\n");
   while (!estVide(pile)) {
      desempiler(&pile, &info);
      printf("%d\n", info);
   }

   return EXIT_SUCCESS;
}
```

Chapitre 4 : Chaînes de caractères

Exercice 4.1 Implémentation de strlen

Proposez une implémentation de la fonction *strlen* dont le prototype et la sémantique sont donnés ci-dessous :

```
size_t strlen(const char* s);
```

Sémantique :
Renvoie le nombre de caractères de la chaîne *s* (le '\0' final n'est pas comptabilisé)

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 4.2 Implémentation de strcpy

Proposez une implémentation de la fonction *strcpy* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strcpy(char* to, const char* from);
```

Sémantique :
Recopie la chaîne *from* dans la chaîne *to* et retourne un pointeur sur *to*.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 4.3 Implémentation de strncpy

Proposez une implémentation de la fonction *strncpy* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strncpy(char* to, const char* from, size_t size);
```

Sémantique :
Recopie les *size* premiers caractères de la chaîne *from* dans la chaîne *to* et retourne un pointeur sur *to*. De manière plus précise :

- Si *size* <= *strlen(from)*, *size* caractères sont copiés dans *to.*
  Attention : Aucune marque de fin de chaîne ('\0') n'est copiée dans *to*
- Si *size* > *strlen(from)*, *strlen(from)* caractères ainsi que *size* - *strlen(from)* '\0' sont copiés dans *to*.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 4.4 Implémentation de strcat

Proposez une implémentation de la fonction *strcat* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strcat(char* to, const char* from);
```

Sémantique :
Concatène la chaîne *from* à la suite de la chaîne *to* et retourne un pointeur sur *to*.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 4.5 Implémentation de strncat

Proposez une implémentation de la fonction *strncat* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strncat(char* to, const char* from, size_t size);
```

Sémantique :
Concatène la chaîne *from* à la suite de la chaîne *to* et retourne un pointeur sur *to*… mais au plus *size* caractères de la chaîne *from* sont copiés. Ici, contrairement à *strncpy* (voir exercice 4.3), un caractère '\0' est toujours ajouté à la fin de la chaîne (quel que soit le critère qui arrête la concaténation).

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 4.6 Implémentation de strcmp

Proposez une implémentation de la fonction *strcmp* dont le prototype et la sémantique sont donnés ci-dessous :

```
int strcmp(const char* s, const char* t);
```

Sémantique :
Compare les deux chaîne *s* et *t* et retourne un résultat > 0 si s > t, = 0 si s = t et < 0 si s < t.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 4.7 Implémentation de strchr

Proposez une implémentation de la fonction *strchr* dont le prototype et la sémantique sont donnés ci-dessous :

```
char* strchr(const char* s, int c);
```

Sémantique :
Retourne un pointeur sur la première occurrence du "caractère" *c* dans la chaîne *s* ou *NULL* si *c* n'est pas trouvé.

Ecrire aussi un petit programme permettant de tester votre fonction.

Exercice 4.8 Prénom et nom

Ecrire une fonction C qui prend en paramètres un prénom et un nom et qui retourne une nouvelle chaîne sous la forme "prénom nom".

Exemple : "James" + "Bond" => "James Bond".

Ecrire aussi un petit programme de test qui, après avoir demandé à l'utilisateur d'entrer un prénom, puis un nom, affiche à l'écran, en utilisant les services de la fonction ci-dessus :

```
La chaine "prenom nom" comporte n caracteres.
```

Soit, dans notre exemple :

```
La chaine "James Bond" comporte 10 caracteres.
```

Exercice 4.9 Inversion d'une chaîne

Proposer une implémentation pour chacune des fonctions d'inversion d'une chaîne suivantes :

1. `void inverser_1(char* s);`
2. `char* inverser_2(const char* s);`
   NB renvoie s en cas de problème

Ecrire aussi un petit programme permettant de tester les deux fonctions.

Lister les avantages/inconvénients respectifs des 2 fonctions ci-dessus.

NB Hormis *strlen*, aucune autre fonction de *string.h* ne doit être utilisée ici.

Chapitre 5 : Fichiers

Exercice 5.1 Lecture intégrale d'un fichier texte

On suppose disposer du fichier texte suivant :

```
1
2
3
4
5
```

Ecrire le plus proprement possible un programme C qui permette d'afficher à l'écran le contenu intégral de ce fichier.

Exercice 5.2 Ecriture d'un fichier binaire

Ecrire un programme C permettant de stocker dans un fichier binaire nommé *personnes.dat* des données relatives à des personnes.

On suppose qu'une personne se caractérise par les propriétés suivantes :

- son nom (20 caractères max)
- son prénom (15 caractères max)
- son âge (de type *unsigned short*)

Les données des diverses personnes doivent être saisies par l'utilisateur. On conviendra que la saisie se termine dès lors que l'utilisateur entre un nom de personne vide.

Exercice 5.3 Lecture intégrale d'un fichier binaire

On suppose disposer du fichier binaire *personnes.dat* créé dans l'exercice précédent.

Ecrire le plus proprement possible un programme C qui recopie le contenu intégral du fichier binaire *personnes.dat* dans un fichier texte *personnes.txt*, histoire de rendre les informations contenues dans le fichier binaire lisibles par un être humain.

Faire en sorte que les données du fichier texte apparaissent alignées en colonnes, comme suit :

```
Nom                Prenom          Age
Federer            Roger            37
Nadal              Rafael           33
Djokovic           Novak            32
del Potro          Juan Martin      30
```

Exercice 5.4 Recherche séquentielle dans un fichier binaire

Ecrire un programme C permettant, à partir du fichier *personnes.dat* créé dans l'exercice 5.2, de retrouver et d'afficher à l'écran les informations relatives à une personne de *nom* donné.

Exemples d'exécution

```
Quel nom recherchez-vous ? : Wawrinka
Desole! Le nom "Wawrinka" ne figure pas dans le fichier

Quel nom recherchez-vous ? : del Potro
Juan Martin del Potro, 30 ans
```

Exercice 5.5 Recherche par accès direct dans un fichier binaire

En exploitant l'accès direct, écrire un programme C permettant, à partir du fichier *personnes.dat* créé dans l'exercice 5.2, de retrouver et d'afficher à l'écran les informations relatives à une personne de *rang*[1] donné.

[1] Convention : La première personne stockée dans le fichier a pour rang 1, la deuxième, le rang 2, etc.

Exemples d'exécution

```
Quel rang recherchez-vous ? : 0
Le fichier contient 4 enregistrement(s).
Desole! Aucune personne ayant ce rang ne figure dans le fichier.

Quel rang recherchez-vous ? : 1
Le fichier contient 4 enregistrement(s).
La personne de rang 1 est : Roger Federer, 37 ans

Quel rang recherchez-vous ? : 4
Le fichier contient 4 enregistrement(s).
La personne de rang 4 est : Juan Martin del Potro, 30 ans

Quel rang recherchez-vous ? : 5
Le fichier contient 4 enregistrement(s).
Desole! Aucune personne ayant ce rang ne figure dans le fichier.
```

Exercice 5.6 Compteurs d'octets

Ecrire un programme complet permettant d'ouvrir n'importe quel fichier (image, son, document, …) et de compter le nombre d'octets de chaque valeur (0 à 255) figurant dans ce fichier.

Le programme doit demander à l'utilisateur le nom du fichier (nom pouvant éventuellement contenir des espaces) à traiter. La saisie du nom doit être sécurisée (pas de débordement de buffer possible). Le nom du fichier (extension incluse) doit être plus petit ou égal à 30 caractères.

Avant de terminer le programme, il faut afficher le résultat à l'écran (compteurs de chaque valeur d'octet).

Exercice 5.7 Lecture d'un fichier texte sans utiliser de boucle

Ecrire un programme C permettant d'afficher à l'écran l'intégralité d'un fichier texte donné, sans jamais utiliser de boucle.

Exercice 5.8 Heures d'ensoleillement

On suppose disposer du fichier *xxx.input* suivant :
(xxx est variable; l'extension .input est fixe)

```
Heures  Lieu
--------------------
1466    Aarau
1545    Adelboden
1637    Bale
1682    Berne
1350    Engelberg
1795    Evolene
1828    Geneve
1570    Interlaken
1832    Jungfraujoch
1872    Lausanne
1424    Lucerne
2143    Montana
1641    Neuchatel
1844    Nyon
1719    Payerne
1809    Santis
1448    Schaffhouse
2094    Sion
1595    Wadenswil
1685    Zermatt
1566    Zurich
```

Ce fichier fournit le nombre d'heures annuelles d'ensoleillement de divers lieux en Suisse.

Ecrire un programme C permettant de produire, à partir du fichier *xxx.input*, le fichier *xxx.output* suivant :

```
Moyenne des heures d'ensoleillement : 1692.6

Lieux ayant plus d'heures d'ensoleillement que la moyenne :
- Evolene (1795)
- Geneve (1828)
- Jungfraujoch (1832)
- Lausanne (1872)
- Montana (2143)
- Nyon (1844)
- Payerne (1719)
- Santis (1809)
- Sion (2094)
```

Contraintes

- Le nom du fichier d'input est un paramètre de la ligne de commande
- La longueur d'une ligne du fichier d'input est de 30 caractères maximum
- Le fichier d'input contient au maximum 100 lignes de données heures – lieu
- Le nom du fichier d'output est de 50 caractères maximum (extension *.output* incluse)
- La taille maximale d'une ligne du fichier d'input ainsi que le nombre maximal de données heures – lieu sont des points garantis (peuvent être considérés comme toujours vrais)

Précision sur le comportement du programme

- Si le nom du fichier d'input n'est pas passé en paramètre de la ligne de commande,
  - le programme le signale via le message :
    `Entrez SVP le nom du fichier a lire (max. n caract.).`
  - le programme se termine
- Si le nom du fichier d'input est passé en paramètre de la ligne de commande, mais qu'il est trop long,
  - le programme le signale via le message :
    `Le nom du fichier est trop long. Il doit comporter au max. n caracteres.`
  - le programme se termine
- S'il s'avère impossible d'ouvrir le fichier d'input ou le fichier d'output,
  - le programme le signale via le message :
    `Impossible d'ouvrir le fichier "xxx.input" (ou "xxx.output").`
  - le programme se termine