Université de Nice-Sophia Antipolis
Polytech/Xidian

Mardi 27 octobre 2015
Durée: 1h30

Fonction d'entrée/sortie scanf, printf
Travaux Dirigés – Séance n. 2

---

1 Fonction d'entrée/sortie scanf, printf

En C, vous avez vu lors du TD précédent que l'écriture à l'écran, c'est-à-dire sur la sortie standard, pouvait être effectuée à l'aide de la fonction `printf`. Une documentation complète de cette fonction est disponible avec la commande `man 3 printf`. Dans cette commande le 3 signifie de consulter la section 3 du manuel, qui contient les fonctions C.

De la même manière qu'il est possible d'écrire à l'écran, il est aussi possible de lire une ou plusieurs données au clavier, c'est-à-dire sur la l'entrée standard, à l'aide de la fonction `scanf`. Le premier paramètre de cette fonction est une chaîne de caractères qui représente ce qui doit être lu depuis le clavier. Les spécifications de conversion de la fonction `scanf` sont semblables à celles de `printf` :

— %d qui sera remplacé par un entier,
— %o qui sera remplacé par un entier notation octale,
— %x qui sera remplacé par un entier notation hexadécimale,
— %f qui sera remplacé par un réel (float),
— %lf qui sera remplacé par un réel (double),
— %c qui sera remplacé par un caractère.

Les variables qui désigneront les valeurs lues devront être précédées par le symbole `&`. La raison en sera expliquée ultérieurement. Lisez l'exemple suivant :

```
int main(void) {
    int i;
    printf("Entrez␣un␣chiffre␣:␣");
    scanf("%d", &i);
    printf( "Le␣chiffre␣lu␣est␣%d.\n", i);
    return EXIT_SUCCESS;
}
```

affichera :

```
Entrez un chiffre : 3
Le chiffre lu est : 3.
```

exercice 1) Écrivez un programme qui lit sur l'entrée standard (le clavier) 3 entiers, respectivement, sous forme décimale, octale et hexadécimale ; puis qui les affiche sous forme décimale.

exercice 2) Écrivez un programme qui lit sur l'entrée standard 1 réel double, et qui affiche sous forme décimale sa partie entière. Faites `man floor`.

exercice 3) Écrivez un programme qui lit 3 entiers au clavier séparés par le signe `%`, et qui

écrit le second sur la sortie standard (l'écran).

2 Types énumérés

Les types énumérés permettent de construire un type par énumération de ses valeurs. En C, les valeurs sont des identificateurs de constante. Par exemple, on peut définir le type `couleur` suivant :

```
enum couleur { ROUGE, VERT, BLEU, JAUNE };
```

En réalité, les valeurs des types **enum** sont représentées par des entiers. Par défaut, les valeurs `ROUGE`, `VERT`, `BLEU`, `JAUNE` sont représentées par des entiers de 0 à 3.

exercice 4) Testez les programmes suivants :

```
int main(void)
{
  enum booleen {FAUX, VRAI};
  enum booleen a;
  enum booleen b;
  a = VRAI;
  b = FAUX;
  printf("a␣=␣%d\n",a);
  printf("b␣=␣%d\n",b);
  return EXIT_SUCCESS;
}
```

Ce qui est strictement équivalent au programme suivant :

```
int main(void)
{
  enum booleen {FAUX, VRAI} a,b;
  a = VRAI;
  b = FAUX;
  printf("a␣=␣%d\n",a);
  printf("b␣=␣%d\n",b);
  return EXIT_SUCCESS;
}
```

Cependant, il est possible de modifier le codage par défaut des valeurs des identificateurs de constante lors de la déclaration du type énuméré. Ces valeurs doivent obligatoirement être des entiers naturels dans un ordre croissant :

```
enum booleen {FAUX = 12, VRAI = 23};
```

Dans l'exemple suivant, un type énuméré est utilisé pour représenter les jours de la semaine. Par défaut, `samedi` reçoit la valeur 0. `dimanche` est explicitement fixé à 0. Les autres jours reçoivent une valeur par défaut entre 1 et 5.

```
enum jour
{
    SAMEDI,
    DIMANCHE = 0,
```

```
    LUNDI,
    MARDI,
    MERCREDI,
    JEUDI,
    VENDREDI
};
```

exercice 5) Avec la déclaration du type `booleen`, affichez les valeurs `FAUX` et `VRAI`.

exercice 6) Affichez la valeur `FAUX+1`.

exercice 7) Modifiez la déclaration du type `couleur` pour que les identificateurs possèdent respectivement les valeur 10, 20, 30 et 40.

exercice 8) Ajoutez au type `couleur` la couleur `ORANGE` juste après la `VERT`. Affichez sa valeur.

exercice 9) Déclarez le type énuméré `car_spéciaux` pour définir l'ensemble des caractères non-imprimables du langage C.

```
'\a' (audible bell)
'\b' (backspace)        '\f' (form feed)
'\n' (newline)          '\r' (carriage return)
'\t' (horizontal tab)   '\v' (vertical tab)
```

exercice 10) Déclarez une variable du type précédent, affectez-lui une valeur, et écrivez le contenu de cette variable à l'écran.

3 Conversion de types

Les objets, mais pas tous, peuvent être convertis d'un type vers un autre. Généralement, on distingue deux types de conversion. Les conversions *implicites* pour lesquelles l'opérateur décide de la conversion à faire en fonction de la nature des opérandes ; les conversions *explicites*, pour lesquelles le programmeur est responsable de la mise en œuvre de la conversion à l'aide d'une notation adéquate.

3.1 Conversion implicite

La conversion implicite ne requiert aucun opérateur particulier. Elle est automatiquement appliquée lorsqu'une valeur est affectée à un type compatible. Étudions les exemples suivants :

```
int a = 5.6; float b = 7;
```

Dans le premier exemple, une expression de type **float** est convertie automatiquement en **int**. Dans le second, l'expression de type **int** est convertie en **float**.

Il y a deux types de conversions automatiques : la promotion et la démotion.

La promotion correspond à la conversion d'une expression d'un type plus petit vers un type plus grand :

```
float a = 4;     /* 4 est un entier converti en float */
long b = 7;      /* 7 est un entier converti en long */
double c = a;    /* a est un float converti en double */
```


Il n'y a en général pas de problème de conversion avec la promotion.
La démotion correspond à la conversion d'une expression d'un type plus grand vers un type plus petit :

```
int a  = 7.5;  /* float converti en int */
int b  = 7.0;  /* float converti en int */
char c = b;    /* int converti en char */
```

La démotion peut conduire à une perte d'information. En effet, dans le premier exemple ci-dessus, `a` recevra la valeur 7 dans la mesure où une expression de type **int** ne peut recevoir de valeur non entière. Les compilateurs récents produisent un message d'alerte dans un tel cas.

3.2 Conversion explicite

Le langage C est un langage fortement typé. Beaucoup de conversions, spécialement celles qui impliquent différentes interprétations de la valeur, requièrent une conversion explicite.

```
int a;
int b;
double c;
a = 2;
b = 3;
c = (double) a/b;
```

Dans cet exemple, sans la conversion explicite, `c` aurait reçu comme valeur 0 au lieu de 0.6666...

exercice 11) Écrivez un programme qui lit sur l'entrée standard une valeur entière en degré Celsius et qui affiche sa conversion en degré Fahrenheit. La relation qui lie ces deux unités est :

$$F = \frac{9 \times C}{5} + 32$$

Les valeurs converties seront toujours arrondies à des valeurs entières. Vous utiliserez pour cela :

- soit la fonction `floor` (compilation avec option `-lm`)
- soit un cast

exercice 12) Écrivez un programme qui lit sur l'entrée standard une valeur réelle représentant des secondes et des centièmes de seconde (*e.g.* `1004.54` égal 1004 secondes et 54 centièmes de secondes) et qui affiche sur la sortie standard une durée dans un format : heures, minutes, secondes et centièmes.