Licence informatique et vidéoludisme

Programmation avancée
de C à C++

Revekka Kyriakoglou

Cours et exercices

21 septembre 2022

Plan

Présentation générale

(les points cités ci-après ne sont pas nécessairement abordés dans l'ordre)

- Rapides révisions de notions abordées en *Programmation Impérative*.
- Maîtriser les concepts avancés de la programmation C : renforcement pointeurs, allocation mémoire et organisation multi-fichiers ; structures de données efficaces ; mesure de performances.
- Apprendre à utiliser une sélection d'outils : `make`, utilisation/création de bibliothèques, `gnuplot`, `GraphViz`, `gprof`, `valgrind`, ...
- La généricité en C (usage des `void *`).
- Utilisation poussée des macros (*C preprocessor*).
- Introduction au C++ sans paradigme objet.
- Utilisation de la STL (C++).

Compilation

Compilation et génération de l'exécutable :

1. **`gcc -Wall -Wextra -Wfatal-errors exo1.c -o exo1`**
   - `Wall` : "tous" les warnings
   - `Wextra` : parce que "-Wall" ne contiens pas tout ! (voir man gcc)
   - `-Wfatal-errors` : arrêt de la compilation dès la 1ère erreur
2. Exécution : `./exo1`

Compilation

Compilation et génération de l'exécutable :

1. **`gcc -Wall -Wextra -Wfatal-errors exo1.c -o exo1`**
   - `Wall` : "tous" les warnings
   - `Wextra` : parce que "-Wall" ne contiens pas tout ! (voir man gcc)
   - `-Wfatal-errors` : arrêt de la compilation dès la 1ère erreur
2. Exécution : `./exo1`

? Avez-vous créé un répertoire pour ce cours et cette seance ?

Compilation

Compilation et génération de l'exécutable :

1. **`gcc -Wall -Wextra -Wfatal-errors exo1.c -o exo1`**
   - `Wall` : "tous" les warnings
   - `Wextra` : parce que "-Wall" ne contiens pas tout ! (voir man gcc)
   - `-Wfatal-errors` : arrêt de la compilation dès la 1ère erreur
2. Exécution : `./exo1`

? Avez-vous créé un répertoire pour ce cours et cette seance ?
`~/work/cours/ProgAvancee/Seance01/print_argv/`


Au fait, le `~/` est la référence vers votre *home directory* (racine de votre compte utilisateur

stdio.h

? Quelle bibliothèque faut-il appeler pour utiliser `printf` ?

stdio.h

? Quelle bibliothèque faut-il appeler pour utiliser `printf` ?

Il fait taper **`#include <stdio.h>`** en haut de votre code.

stdio.h

? Quelle bibliothèque faut-il appeler pour utiliser `printf` ?

Il fait taper **`#include <stdio.h>`** en haut de votre code.

? Ecrire un programme qui imprime votre nom.

Variables

Une variable est le nom donné à une zone de stockage que nos programmes peuvent manipuler.

Variables

Une variable est le nom donné à une zone de stockage que nos programmes peuvent manipuler.

Une variable est une entité "constituée" des cinq éléments :

Variables

Une variable est le nom donné à une zone de stockage que nos programmes peuvent manipuler.

Une variable est une entité "constituée" des cinq éléments :

- un identificateur (nom),
- type (intervalle de valeurs possibles),
- une valeur (à un moment donné),
- une adresse (emplacement mémoire),
- scope (sa durée de vie dans le programme).

Variables

Une variable est le nom donné à une zone de stockage que nos programmes peuvent manipuler.

Une variable est une entité "constituée" des cinq éléments :

- un identificateur (nom),
- type (intervalle de valeurs possibles),
- une valeur (à un moment donné),
- une adresse (emplacement mémoire),
- scope (sa durée de vie dans le programme).

Il faut déclarer toutes les variables avant de les utiliser.

Type	Size (bytes)	octect	bits	Format Specifier
char	1	1	8	%c
float	4	4	32	%f
double	8	8	64	%lf
int	4	4	32	%d, %i

- Le mot **bit (b)** vient de l'anglais « binary digit ». Un bit est un nombre binaire qui peut prendre les valeurs **0** ou **1**.
- Un **byte (B)** est composé de **8 bits**.
  Un bit peut prendre 2 valeurs (0 ou 1), donc un byte peut prendre $2^8 = 256$ différents valeurs.
- Un **octet (o)** est 1 byte qui est 8 bits.

void signifie "rien" ou "aucun type". Vous pouvez considérer void comme une absence.
Si une fonction ne renvoie rien, son type de retour doit être void.

l-value et **r-value**

Comment affecter une **valeur** à une **variable** ?

- À l'aide de l'opérateur affectation « = », lors de la déclaration de la variable ou plus tard, **ou bien**, par passage de paramètres (*i.e.* valable pour les arguments d'une fonction). Il y a d'autres moyens tels que `memset/memcpy` ou autres manipulations passant par des pointeurs : à voir plus tard.
- Dans tous les cas il faut bien distinguer ce qu'est une **l-value** et une **r-value** ; **l** et **r** faisant respectivement référence à *left* et *right*, et ceci est lié à l'emplacement du symbole par rapport à l'opérateur d'affectation « = ». Ainsi, une **l-value** est un contenant, comme par exemple une boîte, et une **r-value** fait référence à une donnée, ou une information, qui correspond *in fine* à une valeur.

l-value et **r-value**

```c
#include <stdio.h>

int main(void){
    // declare 'a', 'b' an object of type 'int'
    int a = 1, b;

    // declare pointer variable 'p', and 'q'
    int *p, *q; // *p, *q are lvalue
    *p = 1; // valid l-value assignment

    // Invalid since "p + 2" is not an l-value
    // p + 2 = 18;

    q = p + 5; // valid - "p + 5" is an r-value

    // Below is valid - dereferencing pointer
    // expression gives an l-value
    *(p + 2) = 18;

    p = &b;

    int arr[20]; // arr[12] is an lvalue; equivalent to *(arr+12)

    // Note: arr itself is also an lvalue
    return 0;
}
```

Code source 1.1 – Exemples **r-value** et **l-value** `r-value_l-value.c`

Operateurs

Opérateur

Les opérateurs sont des symboles qui permettent de manipuler des variables.
Il y a plusieurs types d'opérateurs :

- Opérateurs de calcul.
- Opérateurs d'assignation.
- Opérateurs d'incrémentation.
- Opérateurs de comparaison.
- Opérateurs logiques.
- Opérateurs bit-à-bit.
- Opérateurs de décalage de bit.

Opérateurs de calcul

Opérateur	Dénomination	Effet
+	Addition	Ajoute deux valeurs
-	Soustraction	Soustrait deux valeurs
*	Multiplication	Multiplie deux valeurs
/	Division	Divise deux valeurs
=	Affectation	Affecte une valeur à une variable

Opérateurs d'assignation et opérateurs d'incrémentation

Opérateur	Effet
+=	Additionne deux valeurs et stocke le résultat dans la variable (à gauche)
-=	Soustrait deux valeurs et stocke le résultat
*=	Multiplie deux valeurs et stocke le résultat
/=	Divise deux valeurs et stocke le résultat

Opérateurs d'incrémentation

Opérateur	Effet
++	Augmente d'une unité la variable
–	Diminue d'une unité la variable

x++ permet de remplacer x=x+1 avec x+=1.

Opérateurs de comparaison

Opérateur	Dénomination	Effet
==	Egalité	Vérifie l'égalité entre deux valeurs
<	Infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur
<=	Infériorité	vérifie qu'une variable est Inférieure ou égale à une valeur
>	Supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur
>=	Supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur
!=	Différence	Vérifie qu'une variable est différente d'une valeur

La valeur 1 correspond à **Vrai** et la valeur 0 à **Faux**.

Opérateurs logiques

Opérateur	Dénomination	Effet
||	OU	Vérifie qu'une des conditions est réalisée
&&	ET	Vérifie que toutes les conditions sont réalisées
!	NON	Inverse l'état d'une variable booléenne

Opérateurs bit

Opérateur	Dénomination
&	AND
|	OR
∧	XOR
< <	left shift
> >	right shift
~	Inverse

- **& :** Prend deux nbr et effectue le AND sur chaque bit des deux nbr.
- **| :** Prend deux nbr et effectue un OR sur chaque bit de deux nbr.
- **∧ :** (XOR au sens des bits) prend deux nbr et effectue un XOR sur chaque bit de deux nbr.
  Le résultat de XOR est 1 si les deux bits sont différents.
- **< < :** Prend deux nbr, décale à gauche les bits du premier, le second décide du nbr de places à décaler.
- **> > :** Prend deux nbr, décale à droite les bits du premier, le second décide du nmbr de places à décaler.
- **~ :** Prend un nbr et inverse tous les bits de celui-ci.

Operateur `sizeof`

`sizeof`

Opérateur qui donne la **quantité de stockage**, en octets, obligatoire pour stocker un objet du type de l'opérande. Cet opérateur vous permet d'éviter de spécifier les tailles de données dépendantes de l'ordinateur dans vos programmes. Le résultat de `sizeof` est de type intégral non signé, généralement désigné par `size_t`.

Syntaxe :

```
sizeof( type )
sizeof expression
```

Opérateurs


Saisir ce code, sans faire un copier/coller, dans un éditeur convenable ayant au minimum une coloration syntaxique et une capacité à l'indenter correctement!!!

```c
// C Program to demonstrate use of bitwise operators
#include <stdio.h>
int main()
{
    unsigned char a = 5, b = 9;
    // a = 5(00000101), b = 9(00001001)
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a & b);
    printf("a|b = %d\n", a | b);
    printf("a^b = %d\n", a ^ b);
    printf("~a = %d\n", a = ~a);
    printf("b<<1 = %d\n", b << 1);
    printf("b>>1 = %d\n", b >> 1);

    return 0;
}
```

Code source 1.2 – Exemples Operateurs `bitwise_operator_ex1.c`


Expliquer les opérations de ce programme.

Exercices

Exercise

Essayez d'expliquer ce qui se passe avec le code suivant.

```c
#include <stdio.h>

int main(void){
    int x = 5;
    (x & 1) ? printf("Odd\n") : printf("Even\n");
    return 0;
}
```

Code source 1.3 – Pair ou impair `bitwise_operator_ex2.c`

Licence **I**nformatique et **V**idéoludisme
Université Paris 8