Techniques de programmation
CM n° 2
Bases du langage C

Samson Pierre

<samson.pierre@univ-pau.fr>

09/09/2018

L2 informatique

Université de Pau et des Pays de l'Adour

Précédents responsables et auteurs de ce cours

- A. Aoun
- A. Benzekri
- J.-M. Bruel
- N. Belloir
- M. Mrissa

Références bibliographiques

- Brian W. KERNIGHAN et Dennis M. RITCHIE. *The C Programming Language*. 2e éd. Prentice Hall, 1988. ISBN : 9780131103627
- Herbert SCHILDT. *C: The Complete Reference*. 4e éd. McGraw-Hill Education, 2000. ISBN : 9780072121247

Normalisation

- Organismes de normalisation
  - American National Standards Institute (ANSI)
  - International Organization for Standardization (ISO)
- Normes
  - 1989 : C89 (ou ANSI C) par l'ANSI
  - 1990 : C90 par l'ISO
  - 1995 : C95 (ou AMD1 ou encore C94) par l'ISO
  - 1999 : C99 par l'ISO
  - 2011 : C11 par l'ISO
  - 2018 : C18 par l'ISO
- Certains compilateurs offrent des extensions à ces normes
- Meilleure portabilité : anciennes normes et sans extension
- Ce cours traite de la norme C89 sans extension

Interprétation et compilation

- Nécessité de traduire le code source vers le langage machine
- Langages interprétés
  - Programme traduit à chaque exécution
  - Traduction instruction par instruction « à la volée »
  - Bash, Javascript, PHP, Python, Scheme, ...
- Langages compilés
  - Programme traduit une fois pour générer un binaire exécutable
  - Traduction du programme dans son ensemble
  - C, C++, Pascal, ...
- Langages semi-compilés
  - Compilation puis interprétation du résultat de la compilation
  - Java, ...

Interprétation et compilation

- Interprétation
  - Avantages
    - Programme indépendant de la plateforme
    - Typage dynamique
  - Inconvénients
    - Perte de performances (programme plus lent)
    - Installation d'un interpréteur requis
    - Code source visible

Interprétation et compilation

- Compilation
  - Précompilation : transformations textuelles par le préprocesseur
  - Compilation : traduit le code source en code assembleur
  - Assemblage : traduit le code assembleur en binaire (fichier objet)
  - Édition des liens : lie les différents objets

Interprétation et compilation

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      printf("Hello world\n");
5      return 0;
6  }
```

- Dans ce code
  - Fichier d'en-tête `stdio.h` inclus (ligne 1)
  - Commentaire expliquant la raison de cette inclusion (ligne 1)
  - Fonction `main` définie (lignes 2 à 6)
  - Début d'un bloc d'instructions (ligne 3)
  - Fonction `printf` appelée avec un paramètre (ligne 4)
  - Valeur `0` retournée par la fonction `main` (ligne 5)
  - Fin d'un bloc d'instructions (ligne 6)

Interprétation et compilation

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o hello-
    world.out hello-world.c
$
```

- Description des options
  - `-std=c89` : applique la norme C89
  - `-pedantic` : génère les alertes demandées par la norme
  - `-Wall` : génère toutes les alertes
  - `-Werror` : transforme les alertes en erreurs
  - `-g` : produit les informations de débogage
  - `-o hello-world.out` : génère le fichier de sortie `hello-world.out`

Interprétation et compilation

```
$ ./hello-world.out
Hello world
$
```

Identificateurs et mots clés

- Identificateurs
  - Suite de caractères permettant de reconnaître une entité dans un programme (variable, fonction, ...)
  - Composés d'un ou plusieurs caractères
    - Le premier parmi `[A-Z]`, `[a-z]` et `[_]`
    - Les suivants parmi `[A-Z]`, `[a-z]`, `[_]` et `[0-9]`
  - Exemples : `max_value`, `MAX_VALUE_2`, `maxValue3`, ...
  - Ne peuvent pas être un mot clé du langage C

Identificateurs et mots clés

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

- Mots clés réservés pour
  - Types
  - Types complexes
  - Instructions
  - Opérateurs

Identificateurs et mots clés

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

- Mots clés réservés pour
  - Types
  - Types complexes
  - Instructions
  - Opérateurs

Identificateurs et mots clés

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

- Mots clés réservés pour
  - Types
  - Types complexes
  - Instructions
  - Opérateurs

Identificateurs et mots clés

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

- Mots clés réservés pour
  - Types
  - Types complexes
  - Instructions
  - Opérateurs

Identificateurs et mots clés

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

- Mots clés réservés pour
  - Types
  - Types complexes
  - Instructions
  - Opérateurs

Commentaires

- Tout caractère entre $/*$ et $*/$ est ignoré par le compilateur
- Utilisés pour expliquer ou documenter le code

Variables, types, déclarations et portée

- Variables
  - Identificateurs associés à une valeur en mémoire
  - Possèdent un type décrit par
    - Spécificateurs de type
    - Qualificateurs de type
    - Classes de stockage
  - Doivent être déclarées pour être utilisées

Variables, types, déclarations et portée

- Spécificateurs de types
  - `char` : caractère
  - `int` : entier
  - `float` : réel simple précision
  - `double` : réel double précision
  - `void` : vide
  - `short` : court
  - `long` : long
  - `signed` : signé
  - `unsigned` : non signé
- Remarques
  - Tailles en octets différentes selon le compilateur
  - Entiers signés par défaut
  - Caractères signés ou non par défaut selon le compilateur
  - Type `int` implicite (exemple : `short` = `short int`)
  - Certaines combinaisons de spécificateurs de types interdites

Variables, types, déclarations et portée

Type	Signification
`char`	caractère
`signed char`	caractère signé
`unsigned char`	caractère non signé
`short`	entier court
`unsigned short`	entier court non signé
`int`	entier
`unsigned int`	entier non signé
`long`	entier long
`unsigned long`	entier long non signé
`float`	réel simple précision
`double`	réel double précision
`long double`	réel précision étendue

Variables, types, déclarations et portée

Type	Obtention de la taille (octets)
`char`	`sizeof(char)`
`signed char`	`sizeof(signed char)`
`unsigned char`	`sizeof(unsigned char)`
`short`	`sizeof(short)`
`unsigned short`	`sizeof(unsigned short)`
`int`	`sizeof(int)`
`unsigned int`	`sizeof(unsigned int)`
`long`	`sizeof(long)`
`unsigned long`	`sizeof(unsigned long)`
`float`	`sizeof(float)`
`double`	`sizeof(double)`
`long double`	`sizeof(long double)`

Variables, types, déclarations et portée

Type	Taille habituelle (octets)
`char`	1
`signed char`	1
`unsigned char`	1
`short`	2
`unsigned short`	2
`int`	4
`unsigned int`	4
`long`	8
`unsigned long`	8
`float`	4
`double`	8
`long double`	16

Variables, types, déclarations et portée

Type	Obtention de la plage de valeurs
`char`	`CHAR_MIN` à `CHAR_MAX`
`signed char`	`SCHAR_MIN` à `SCHAR_MAX`
`unsigned char`	0 à `UCHAR_MAX`
`short`	`SHRT_MIN` à `SHRT_MAX`
`unsigned short`	0 à `USHRT_MAX`
`int`	`INT_MIN` à `INT_MAX`
`unsigned int`	0 à `UINT_MAX`
`long`	`LONG_MIN` à `LONG_MAX`
`unsigned long`	0 à `ULONG_MAX`
`float`	`FLT_MIN` à `FLT_MAX`
`double`	`DBL_MIN` à `DBL_MAX`
`long double`	`LDBL_MIN` à `LDBL_MAX`

Variables, types, déclarations et portée

Type	Plage de valeurs minimale
`char`	celle de `signed char` ou `unsigned char`
`signed char`	-127 à 127
`unsigned char`	0 à 255
`short`	-32767 à 32767
`unsigned short`	0 à 65535
`int`	-32767 à 32767
`unsigned int`	0 à 65535
`long`	-2147483647 à 2147483647
`unsigned long`	0 à 4294967295
`float`	1×10^{-37} à 1×10^{37}
`double`	1×10^{-37} à 1×10^{37}
`long double`	1×10^{-37} à 1×10^{37}

Variables, types, déclarations et portée

Type	Plage de valeurs habituelles
`char`	-128 à 127
`signed char`	-128 à 127
`unsigned char`	0 à 255
`short`	-32768 à 32767
`unsigned short`	0 à 65535
`int`	-2147483648 à 2147483647
`unsigned int`	0 à 4294967295
`long`	9×10^{-18} à 9×10^{18}
`unsigned long`	0 à 2×10^{19}
`float`	1×10^{-38} à 3×10^{38}
`double`	2×10^{-308} à 2×10^{308}
`long double`	3×10^{-4932} à 1×10^{4932}

Variables, types, déclarations et portée

- Fichiers d'en-tête pour les macros des plages de valeurs
  - `limits.h`
  - `float.h`

Variables, types, déclarations et portée

- Déclaration d'une variable
  - Type suivi du nom de la variable
  - Doit être placée avant toute instruction

```
char my_char;
signed char my_schar;
unsigned char my_uchar;
short my_short;
unsigned short my_ushort;
int my_int;
unsigned int my_uint;
long my_long;
unsigned long my_ulong;
float my_float;
double my_double;
long double my_ldouble;
```

Variables, types, déclarations et portée

- Déclaration d'une variable avec initialisation

```
char my_char = 'a';
signed char my_schar = 'a';
unsigned char my_uchar = 'a';
short my_short = 32767;
unsigned short my_ushort = 65535;
int my_int = 32767;
unsigned int my_uint = 65535;
long my_long = 2147483647;
unsigned long my_ulong = 4294967295;
float my_float = 0;
double my_double = 0;
long double my_ldouble = 0;
```

Variables, types, déclarations et portée

- Déclaration multiple de variables

```
char char1, char2, char3;
signed char schar1, schar2, schar3;
unsigned char uchar1, uchar2, uchar3;
short short1, short2, short3;
unsigned short ushort1, ushort2, ushort3;
int int1, int2, int3;
unsigned int uint1, uint2, uint3;
long long1, long2, long3;
unsigned long ulong1, ulong2, ulong3;
float float1, float2, float3;
double double1, double2, double3;
long double ldouble1, ldouble2, ldouble3;
```

Variables, types, déclarations et portée

- Portée d'une variable
  - Partie du code où la variable est visible et peut être utilisée
  - Déterminée par la position de la déclaration
  - Variable globale (déclarée en dehors de tout bloc)
    - Portée : le fichier dans lequel elle est déclarée
  - Variable locale (déclarée dans un bloc)
    - Portée : le bloc dans lequel elle est déclarée

Variables, types, déclarations et portée

```c
#include <stdio.h> /* for printf */
int varg = 10; /* global */
int main()
{
    int varl = 10; /* local */
    printf("varg = %d\n", varg);
    printf("varl = %d\n", varl);
    return 0;
}
```

Variables, types, déclarations et portée

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o scope.
    out scope.c
$ ./scope.out
varg = 10
varl = 10
$
```

Variables, types, déclarations et portée

- Qualificateurs de types
  - `const` : interdiction de changer la valeur de la variable
  - `volatile` : interdiction d'optimiser le code pour la variable

Variables, types, déclarations et portée

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      const int varc = 10;
5      volatile int varv = 10;
6      printf("varc = %d\n", varc);
7      printf("varv = %d\n", varv);
8      return 0;
9  }
```

Variables, types, déclarations et portée

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o type-
    qualifiers.out type-qualifiers.c
$ ./type-qualifiers.out
varc = 10
varv = 10
$
```

Variables, types, déclarations et portée

- Classes de stockage
  - `auto`
    - La variable est créée au début du bloc dans lequel elle est déclarée
    - La variable est détruite à la fin du bloc dans lequel elle est déclarée
    - Les variables locales le sont par défaut
  - `extern`
    - La variable référence une variable déclarée ailleurs
  - `static`
    - La variable est créée au début de l'exécution du programme
    - La variable est détruite à la fin de l'exécution du programme
  - `register`
    - La variable est stockée dans un registre du processeur
    - Pas en mémoire
    - Améliore la rapidité d'accès à la variable
    - Impossible d'accéder à l'adresse de la variable

Variables, types, déclarations et portée

```
1  #include <stdio.h> /* for printf */
2  int vare = 10;
3  int main()
4  {
5      auto int vara = 10;
6      extern int vare;
7      static int vars = 10;
8      register int varr = 10;
9      printf("vara = %d\n", vara);
10     printf("vare = %d\n", vare);
11     printf("vars = %d\n", vars);
12     printf("varr = %d\n", varr);
13     return 0;
14 }
```

Variables, types, déclarations et portée

```
gcc -std=c89 -pedantic -Wall -Werror -g -o storage-
    classes.out storage-classes.c
$ ./storage-classes.out
vara = 10
vare = 10
vars = 10
varr = 10
$
```

Instructions, expressions, opérateurs et constantes

- Instructions
  - Syntaxe

```
statement:
    labeled-statement
    expression-statement
    compound-statement
    jump-statement
    selection-statement
    iteration-statement
```

  - Spécifient une action à réaliser
  - Exécutées en séquence (sauf cas particuliers)

Instructions, expressions, opérateurs et constantes

- Instructions étiquetées
  - Syntaxe

```
labeled-statement:
    identifier : statement
    case constant-expression : statement
    default : statement
```

  - Permettent d'étiqueter pour
    - Instruction de saut `goto`
    - Instruction de sélection `switch`
  - Contiennent le caractère « : »

Instructions, expressions, opérateurs et constantes

- Instructions d'expression
  - Syntaxe

    `expression-statement :`
    `expression`$_{opt}$ `;`

  - Terminent par le caractère « ; »
  - Contiennent 0 ou 1 expression
  - Si 0 alors instruction nulle

Instructions, expressions, opérateurs et constantes

- Instructions composées
  - Syntaxe

```
compound-statement:
    { declaration-list_opt statement-list_opt }
declaration-list:
    declaration
    declaration-list declaration
statement-list:
    statement
    statement-list statement
```

  - Permettent de regrouper un ensemble d'instructions
  - Aussi appelées « blocs »
  - Commencent par le caractère « { »
  - Terminent par le caractère « } »
  - Contiennent 0, 1 ou plusieurs déclarations
  - Contiennent 0, 1 ou plusieurs instructions

Instructions, expressions, opérateurs et constantes

- Instructions de saut
  - Syntaxe

```
jump-statement:
    goto identifier;
    break;
    continue;
    return expressionopt;
```

  - Permettent de sauter sans condition
  - Terminent par le caractère « ; »
  - Instructions de saut `break`
    - Dans un `switch` ou une boucle
    - Permet de sortir du `switch` ou de la boucle
  - Instructions de saut `continue`
    - Dans une boucle
    - Permet de réitérer dans la boucle

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      goto engine_ready_actions;
5      printf("Engine not ready\n");
6      return 1;
7      engine_ready_actions:
8          printf("Engine ready\n");
9      return 0;
10 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o jump-
    statements.out jump-statements.c
$ ./jump-statements.out
Engine ready
$
```

Instructions, expressions, opérateurs et constantes

- Instructions de sélection
  - Syntaxe

```
selection-statement:
    if ( expression ) statement
    if ( expression ) statement else statement
    switch ( expression ) statement
```

  - Permettent de sélectionner des instructions parmi un ensemble d'instructions
  - Contiennent les mots clés `if`, `else` ou `switch`
  - Condition du `if` est fausse si elle vaut 0, sinon vraie

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int engine_ready = 1;
5      if(engine_ready == 1)
6          printf("Engine ready\n");
7      else
8      {
9          printf("Engine not ready\n");
10         return 1;
11     }
12     return 0;
13 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    selection-statements-if.out selection-
    statements-if.c
$ ./selection-statements-if.out
Engine ready
$
```

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int engine_ready = 1;
5      switch(engine_ready)
6      {
7          case 1:
8              printf("Engine ready\n");
9              break;
10         default:
11             printf("Engine not ready\n");
12             return 1;
13     }
14     return 0;
15 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    selection-statements-switch.out selection-
    statements-switch.c
$ ./selection-statements-switch.out
Engine ready
$
```

Instructions, expressions, opérateurs et constantes

- Instructions d'itération
  - Syntaxe

```
iteration-statement:
    while (expression) statement
    do statement while (expression);
    for (expression_opt; expression_opt; expression_opt)
        statement
```

  - Permettent de répéter un ensemble d'instructions sous condition
  - Aussi appelées « boucles »
  - Ont leur condition évaluée
    - Avant le corps de la boucle pour le `while` et le `for`
    - Après le corps de la boucle pour le `do while`

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int speed = -3;
5      while(speed < 5)
6      {
7          speed++;
8          if(speed < 0)
9              continue;
10         printf("speed = %d km/h\n", speed);
11     }
12     return 0;
13 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    iterative-statements-while.out iterative-
    statements-while.c
$ ./iterative-statements-while.out
speed = 0 km/h
speed = 1 km/h
speed = 2 km/h
speed = 3 km/h
speed = 4 km/h
speed = 5 km/h
$
```

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int speed = -3;
5      do
6      {
7          speed++;
8          if(speed < 0)
9              continue;
10         printf("speed = %d km/h\n", speed);
11     } while(speed < 5);
12     return 0;
13 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    iterative-statements-do-while.out iterative-
    statements-do-while.c
$ ./iterative-statements-do-while.out
speed = 0 km/h
speed = 1 km/h
speed = 2 km/h
speed = 3 km/h
speed = 4 km/h
speed = 5 km/h
$
```

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int speed;
5      for(speed = -3; speed < 5; speed++)
6      {
7          if(speed < 0)
8              continue;
9          printf("speed = %d km/h\n", speed);
10     }
11     return 0;
12 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    iterative-statements-for.out iterative-
    statements-for.c
$ ./iterative-statements-for.out
speed = 0 km/h
speed = 1 km/h
speed = 2 km/h
speed = 3 km/h
speed = 4 km/h
$
```

Instructions, expressions, opérateurs et constantes

- Expressions
  - Combinaisons d'opérateurs et d'opérandes

Instructions, expressions, opérateurs et constantes

- Opérateurs
  - D'affectation
  - Arithmétiques
  - D'incrémentation et de décrémentation
  - Relationnels
  - Logiques
  - De traitements binaires
  - Autres

Instructions, expressions, opérateurs et constantes

- Opérateurs d'affectation

Opérateur	Signification
=	affecter
+=	calculer la somme et affecter
-=	calculer la différence et affecter
*=	calculer le produit et affecter
/=	calculer le quotient et affecter
%=	calculer le reste et affecter
<<=	décaler à gauche et affecter
>>=	décaler à droite et affecter
&=	appliquer un ET binaire et affecter
|=	appliquer un OU binaire inclusif et affecter
^=	appliquer un OU binaire exclusif et affecter

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int var;
5      var = 32;
6      var += 10; /* var = var + 10 = 42 */
7      var -= 10; /* var = var - 10 = 32 */
8      var *= 10; /* var = var * 10 = 320 */
9      var /= 10; /* var = var / 10 = 32 */
10     printf("var %% 10 = %d %% 10 = %d\n",
11            var,
12            var % 10);
13     return 0;
14 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    assignment-operators.out assignment-operators.c
$ ./assignment-operators.out
var % 10 = 32 % 10 = 2
$
```

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int var;
5      var = 32; /* 00000000 00100000 = 32 */
6      var <<= 3; /* 00000001 00000000 = 256 */
7      var >>= 3; /* 00000000 00100000 = 32 */
8      var |= 256; /* 00000001 00100000 = 288 */
9      var ^= 256; /* 00000000 00100000 = 32 */
10     var &= 256; /* 00000000 00000000 = 0 */
11     printf("var = %d\n", var);
12     return 0;
13 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    assignment-operators-bitwise.out assignment-
    operators-bitwise.c
$ ./assignment-operators-bitwise.out
var = 0
$
```

Instructions, expressions, opérateurs et constantes

- `Lvalue` (vient de `left value`)
  - Désigne une zone mémoire
  - Peut être placée à gauche d'un opérateur d'affectation
  - Un identificateur de variable est une `lvalue`
  - L'opérateur de déréférencement `*` retourne une `lvalue`
- `Rvalue` (vient de `right value`)
  - Désigne la valeur d'une expression
  - Placée à droite d'un opérateur d'affectation

Instructions, expressions, opérateurs et constantes

- Opérateurs arithmétiques

Opérateur	Signification
$+$	calculer la somme
$-$	calculer la différence
$\star$	calculer le produit
$/$	calculer le quotient
$\%$	calculer le reste

Instructions, expressions, opérateurs et constantes

```c
#include <stdio.h> /* for printf */
int main()
{
    int var;
    var = 32;
    printf("%d + 10 = %d\n", var, var + 10);
    printf("%d - 10 = %d\n", var, var - 10);
    printf("%d * 10 = %d\n", var, var * 10);
    printf("%d / 10 = %d\n", var, var / 10);
    printf("%d %% 10 = %d\n", var, var % 10);
    return 0;
}
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    arithmetic-operators.out arithmetic-operators.c
$ ./arithmetic-operators.out
32 + 10 = 42
32 - 10 = 22
32 * 10 = 320
32 / 10 = 3
32 % 10 = 2
$
```

Instructions, expressions, opérateurs et constantes

- Opérateurs d'incrémentation et de décrémentation

Opérateur	Signification
$++$	incrémenter
$--$	décrémenter

Instructions, expressions, opérateurs et constantes

- Opérateurs relationnels

Opérateur	Signification
==	est égal ?
!=	est différent ?
<	est inférieur ?
<=	est inférieur ou égal ?
>	est supérieur ?
>=	est supérieur ou égal ?

Instructions, expressions, opérateurs et constantes

- Opérateurs logiques

Opérateur	Signification
& &	appliquer un ET logique
| |	appliquer un OU logique
!	appliquer un NON logique

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int pu = 1; /* plutonium */
5      int mph = 88; /* speed in miles per hour */
6      float gw = 1.21; /* electricity in gigawatts */
7      if((pu || (!pu && gw >= 1.21)) && mph >= 88)
8          printf("You traveled through time\n");
9      else
10         printf("You did not travel through time\n");
11     return 0;
12 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o logic-
    operators.out logic-operators.c
$ ./logic-operators.out
You traveled through time
$
```

Instructions, expressions, opérateurs et constantes

- Opérateurs de traitements binaires

Opérateur	Signification
&	appliquer un ET binaire
|	appliquer un OU binaire inclusif
^	appliquer un OU binaire exclusif
~	appliquer un NON binaire
<<	décaler à gauche
>>	décaler à droite

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int var;
5      var = 32; /* 00000000 00100000 = 32 */
6      printf("32 << 3 = %d\n", var << 3); /* 256 */
7      printf("32 >> 3 = %d\n", var >> 3); /* 4 */
8      printf("32 | 256 = %d\n", var | 256); /* 288 */
9      printf("32 ^ 256 = %d\n", var ^ 256); /* 288 */
10     printf("32 & 256 = %d\n", var & 256); /* 0 */
11     printf("~32 = %d\n", ~var); /* -33 */
12     return 0;
13 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
   bitwise-operators.out bitwise-operators.c
$ ./bitwise-operators.out
32 << 3 = 256
32 >> 3 = 4
32 | 256 = 288
32 ^ 256 = 288
32 & 256 = 0
~32 = -33
$
```

Instructions, expressions, opérateurs et constantes

- Autres opérateurs

Opérateur	Signification
`&`	obtenir l'adresse mémoire
`*`	déréférencer (« indirection »)
`sizeof`	obtenir la taille en octets
`()`	appeler une fonction
`[]`	accéder à un élément d'un tableau
`.`	accéder à un membre d'une structure
`->`	déréférencer et accéder à un membre d'une structure
`(type)`	convertir le type (« coercition » ou « cast »)
`,`	évaluer plusieurs expressions
`?:`	opérateur ternaire (si, alors, sinon)

Instructions, expressions, opérateurs et constantes

```
1  #include <stdio.h> /* for printf */
2  int main()
3  {
4      int var = 32;
5      int *ptr = &var;
6      printf("var = %d\n", var);
7      printf("*ptr = %d\n", *ptr);
8      printf("&var = %p\n", (void*) &var);
9      printf("ptr = %p\n", (void*) ptr);
10     printf("(var = 1, var + 2) = %d\n",
11            (var = 1, var + 2));
12     var == 32 ? var-- : var++;
13     printf("var = %d\n", var);
14     return 0;
15 }
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o other-
    operators.out other-operators.c
$ ./other-operators.out
var = 32
*ptr = 32
&var = 0x7ffee836da34
ptr = 0x7ffee836da34
(var = 1, var + 2) = 3
var = 2
$
```

Instructions, expressions, opérateurs et constantes

- Priorité des opérateurs

Opérateurs	Priorité
`(), [], ->, .`	1 (la plus forte)
`!, ~, ++, --, +, -, (type), *, &, sizeof`	2
`*, /, %`	3
`+, -`	4
`<<, >>`	5
`<, <=, >, >=`	6
`==, !=`	7
`&`	8

Instructions, expressions, opérateurs et constantes

- Priorité des opérateurs

Opérateurs	Priorité
^	9
|	10
&&	11
||	12
?:	13
=, +=, -=, *=, /=, %= &=, ^=, |=, <<=, >>=	14
,	15 (la plus faible)

Instructions, expressions, opérateurs et constantes

```c
#include <stdio.h> /* for printf */
int main()
{
    int var = 32;
    printf("1 + var * 2 - 3 = %d\n",
           1 + var * 2 - 3); /* 1 + 64 - 3 = 62 */
    printf("var * 2 + 1 - 3 = %d\n",
           var * 2 + 1 - 3); /* 64 + 1 - 3 = 62 */
    printf("1 - 3 + var * 2 = %d\n",
           1 - 3 + var * 2); /* 1 - 3 + 64 = 62 */
    return 0;
}
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    operator-precedence.out operator-precedence.c
$ ./operator-precedence.out
62
62
62
$
```

Instructions, expressions, opérateurs et constantes

```c
#include <stdio.h> /* for printf */
int main()
{
    printf("3 / 2 * 5 = %d\n",
           3 / 2 * 5); /* 1 * 5 = 5 */
    printf("3 * 2 / 5 = %d\n",
           3 * 2 / 5); /* 6 / 5 = 1 */
    return 0;
}
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
   operator-precedence-same-consecutive.out
   operator-precedence-same-consecutive.c
$ ./operator-precedence-same-consecutive.out
3 / 2 * 5 = 5
3 * 2 / 5 = 1
$
```

Instructions, expressions, opérateurs et constantes

- Constantes
  - Valeurs fixes que le programme ne peut pas modifier

Type de constante	Exemple
Décimal	`10`
Octal	`012`
Hexadécimal	`0x10`
Entier	`1000`
Entier long	`1000l`
Réel simple précision	`1000.0f` ou `1e3f`
Réel double précision	`1000.0` ou `1e3`
Réel précision étendue	`1000.0l` ou `1e3l`
Caractère	`'a'`
Chaîne de caractères	`"abc"`

  - Nombres insensibles à la casse
  - Aussi appelées « valeurs littérales »

Instructions, expressions, opérateurs et constantes

```c
#include <stdio.h> /* for printf */
int main()
{
    printf("10 = %d\n", 10); /* 10 */
    printf("012 = %d\n", 012); /* 10 */
    printf("0x10 = %d\n", 0x10); /* 16 */
    printf("1000l = %ld\n", 1000l); /* 1000 */
    printf("1000.0f = %f\n", 1000.0f); /* 1000 */
    printf("1e3 = %f\n", 1e3); /* 1000 */
    printf("1000.0l = %Lf\n", 1000.0l); /* 1000 */
    printf("'a' = %c\n", 'a');
    printf("\"abc\" = %s\n", "abc");
    return 0;
}
```

Instructions, expressions, opérateurs et constantes

```
$ gcc -std=c89 -pedantic -Wall -Werror -g -o
    constants.out constants.c
$ ./constants.out
10 = 10
012 = 10
0x10 = 16
1000l = 1000
1000.0f = 1000.000000
1e3 = 1000.000000
'a' = a
"abc" = abc
$
```

Instructions, expressions, opérateurs et constantes

Séquence d'échappement	Signification
`\a`	caractère d'appel (bip sonore)
`\b`	retour arrière
`\f`	nouvelle page
`\n`	nouvelle ligne
`\r`	retour chariot
`\t`	tabulation horizontale
`\v`	tabulation verticale
`\'`	apostrophe
`\"`	guillemet
`\?`	point d'interrogation
`\\`	antislash
`\NNN`	valeur octale `NNN`
`\xNN`	valeur hexadécimale `NN`