Programmation Procédurale – Expressions – Opérateurs

Polytech'Nice Sophia Antipolis

Erick Gallesio

2015 – 2016

Affectation (suite)

Opérateur préfixe et postfixe

- `i++` $\Leftrightarrow$ `i += 1` $\Leftrightarrow$ `i = i + 1`
- `i--` $\Leftrightarrow$ `i -= 1` $\Leftrightarrow$ `i = i - 1`

Note: L'évaluation de i n'est faite qu'une seule fois

Exemple:

```
i = 3; j = 3;
printf("i = %d j = %d", i++, ++j);   /* 3 and 4 */
printf("i = %d j = %d", i, j);       /* 4 and 4 */
```

Attention: ordre d'évaluation inconnu

```
t[i++] = v[i++];                     /* ??? */
printf("%d %d", i++, i++);           /* ??? */
```

Opérateurs sur les types

Taille d'un type: sizeof

```
sizeof(type)
sizeof(variable)
```

Exemples:

```
sizeof(x)
sizeof(struct personne)
sizeof(a) /sizeof(a[0])     /* nbre d'élements dans a */
```

Conversion explicite: cast

```
(type) expr          type = type dans lequel expr est convertie
```

Exemples:

```
(int) 2.0          /* force l'expression à être int */
3 / (float) 4      /* = 0.75 alors que  3/4 = 0 */
```

Conversions de type implicites (1 / 2)

- Les conversions de type ont lieu quand les opérandes sont de types différents
- Les règles sont assez complexes
- *En gros*, convertir vers le type le plus grand
  - promotion entière (sous-types de int $\Rightarrow$ int)
  - ensuite:
    - long double
    - double
    - float
    - unsigned long
    - long
    - unsigned int
    - int

Note: On omet les *long long* et les complexes ici

Conversions de type implicites (2 / 2)

- Dans une affectation
  - Les bits supplémentaires sont perdus quand une expression mélange *char*, *short*, *int* et *long*
  - conversion de la partie droite de l'affectation dans le type de la partie gauche

```
the_char = 0xabcdef        /* the_char == 0xef */

the_int = 2.3;             /* the_int = 2 */
the_float = 2;             /* the_float = 2.0 */
```

- Passage de paramètre:
  - règles identiques à celles de l'affectation
- Les conversions 'value preserving' sont toujours légales (mais la précision peut ne pas être préservée)
- Les conversions non 'value preserving' provoquent un *warning*

Opérateur de condition

- C'est un opérateur ternaire
- Comme un *if-then-else* mais qui a une valeur

Syntaxe:

```
condition? expr1: expr2
```

Exemples:

```
int min (int a, int b)
{
  return (a<b) ? a : b;
}

abs_of_x = (x < 0) ? -x : x;

printf("%d error%s\n", n, (n>1) ? "s" : "");
```

Opérateur virgule

Syntaxe:

```
expr1, expr2
```

- le résultat de l'évaluation de *expr2*
- *expr1* est évaluée mais son résultat est perdu
- utile pour mettre deux expressions là où la syntaxe n'en permet qu'une

Exemples:

```
x  =  4, y = 3;                    /* pas très utile ! */

for (i=0,j=MAX; i<j; i++,j--) { /* retourne tableau t */
  int tmp = t[i];

  t[i] = t[j];
  t[j] = tmp;
}
```

Priorité des opérateurs

Catégorie d'opérateurs	Opérateurs	Assoc.
postfixe	() [] . ->	G=>D
opérateurs unaires	++ – ! ~ * - & sizeof (cast)	D=>G
division, multiplication modulo	/ * %	G=>D
addition, soustraction	+ -	G=>D
opérateurs binaires de décalage	<< >>	G=>D
opérateurs relationnels	< <= > >=	G=>D
opérateurs de comparaison	== !=	G=>D
et binaire	&	G=>D
ou exclusif binaire	^	G=>D
ou binaire	|	G=>D
et logique	&&	G=>D
ou logique	||	G=>D
opérateur conditionnel	? :	D=>G
opérateurs d'affectation	= += -= *= /= %= &= ^= |= <<= >>=	D=>G
opérateur virgule	,	G=>D

15 niveaux de priorités!!

Exemple: strcat (1 / 3)

Vieille version

```
void strcat(char s1[], char s2[]) {
  int i=0, j=0;

  while (s1[i] != '\0') i += 1;
  while (s2[j] != '\0') {
    s1[i] = s2[j];
    i += 1; j += 1;
  }
  /* Don't forget to set the final null char */
  s1[i] = '\0';
}
```

Exemple: strcat (2 / 3)

Vieille version

```
void strcat(char s1[], char s2[]) {
  int i=0, j=0;

  while (s1[i] != '\0') i += 1;
  while (s2[j] != '\0') {
    s1[i] = s2[j];
    i += 1; j += 1;
  }
  /* Don't forget to set the final null char */
  s1[i] = '\0';
}
```

Nouvelle version

```
void strcat(char s1[], char s2[]) {
  int i=0, j=0;

  while (s1[i]) i += 1;
  while (s2[j]) s1[i++] = s2[j++];
  /* Don't forget to set the final null char */
  s1[i] = '\0';
}
```

Exemple: strcat (3 / 3)

Nouvelle version

```
void strcat(char s1[], char s2[]) {
  int i=0, j=0;

  while (s1[i]) i += 1;
  while (s2[j]) s1[i++] = s2[j++];
  /* Don't forget to set the final null char */
  s1[i] = '\0';
}
```

Version améliorée

```
void strcat(char s1[], char s2[]) {
  int i=0, j=0;

  while (s1[i]) i += 1;
  while (s1[i++] = s2[j++]) /* Nothing */;
}
```

Exemple: conversion chaîne ⇒ entier (atoi)

```c
int atoi(char s[])
{
  int i, n, sign=1;

  for (i=0; s[i]=='\t' || s[i]=='\n' || s[i]==' '; i++)
    /* Do nothing */;

  if (s[i] == '+' || s[i] == '-')
    sign = (s[i++] == '+') ? 1 : -1;

  for (n = 0; s[i]>='0' && s[i] <= '9'; i++)
    n = 10*n + (s[i]-'0');

  return sign * n;
}
```