UNIVERSITÉ
CÔTE D'AZUR

Enoncés - Expressions

Présentation: **Stéphane Lavirotte**
Auteurs: ... et al*

(*) Cours réalisé grâce aux documents de :
Erick Gallesio

Mail: Stephane.Lavirotte@univ-cotedazur.fr
Web: http://stephane.lavirotte.com/
Université Côte d'Azur

UNIVERSITÉ
CÔTE D'AZUR

Enoncés

UNIVERSITÉ
CÔTE D'AZUR

Enoncé vide

- ✓ C'est l'énoncé le plus simple. . .
- ✓ Juste un " ; "
- ✓ Le " ; " est un terminateur d'instruction

```
while ((c = getchar()) != ' ')
    /* ne rien faire */ ;
```

- ✓ ou encore

```
while ((c = getchar()) != ' ') {
}
```

UNIVERSITÉ CÔTE D'AZUR

Enoncé Composé

- ✓ Un énoncé composé (ou bloc) sert à
  - regrouper plusieurs énoncés

```
while (a <= b) {
    a +=1;
    b -=1;
}
```

  - restreindre la visibilité d'une variable

```
if (a > b) {
    int x = 3 * a + 2 * b;
    ....
}
```

  - dénoter le corps d'une fonction

```
int foo() {
    ...
}
```

UNIVERSITÉ CÔTE D'AZUR

Enoncé `if`

- ✓ **Syntaxe:**

```
if (<expression logique>)
   <action1>
else
   <action2>
```

  - le `else` peut être omis
  - l'*`<expression logique>`* est une expression entière
    - 0 ⇔ faux
    - autre ⇔ vrai
  - problème du *dandling else:* utiliser des blocs

```
if (x) { /* équivalent à (x != 0) */
   printf("True");
   a += b*c;
}
else printf("False");
```

UNIVERSITÉ CÔTE D'AZUR

Enoncé switch 1/2

- ✓ **Syntaxe:**

```
switch (<expression entière>) {
    case value1: <instr_list_1>
    case value2: <instr_list_2>
    ...
    case valuen: <instr_list_n>
    default : <instr_list>
}
```

  - **Quand** `<instr_list_i>` **est choisie, l'exécution continue en séquence (source de bugs chez les débutants)**
  - **le mot clé** `break` **permet d'arrêter l'exécution de la séquence (continuer après le** `switch`**)**
  - **la clause** `default` **peut être absente**

UNIVERSITÉ CÔTE D'AZUR

Enoncé switch 2/2

```
char c;
...
switch (c) { /* conv. automatique de "char" => "int" */
   case '_' : printf("For C, '_' is a ");
   case 'a' :
   case 'A' :
   ...
   case 'z' :
   case 'Z' : printf("letter\n"); break;
   case ' ' :
   case '\t': print("space\n"); break;
   default : printf("other char\n");
}
```

UNIVERSITÉ CÔTE D'AZUR

Enoncés `while` et `do`

- ✓ **Syntaxe:**

```
while (<expression logique>)
    <énoncé>;
```

```
do
    <énoncé>;
while (<expression logique>)
```

  - **un `while` peut ne pas s'exécuter**
  - **un `do` s'exécute au moins une fois**

UNIVERSITÉ CÔTE D'AZUR

Enoncé for

- ✓ **Syntaxe:**

```
for (<initialisation>; <condition>; <increment>)
    <action>
```

- ✓ **est équivalent à**

```
<initialisation>;
while (<condition>) {
    <action>;
    <increment>;
}
```

```
i = 0;
while (i < 100) {
    t[i] = 0;
    i += 1;
}
```

```
 /* peut être réécrit en */
for (i = 0; i < 100; i++) {
    t[i] = 0;
}
```

UNIVERSITÉ CÔTE D'AZUR

Enoncé `break`

- ✓ ***utilisé dans une boucle ou un*** `switch`
- ✓ ***sort de la boucle ou du*** `switch`

- ✓ ***Exemples:***

```
for (i = 0; i < MAX; i++)
   if (t[i] == item) break;
```

```
for (i = 0; i < MAX; i++)
   for (j = 0; j < MAX; j++)
      if (t[i][j] == item) break; /* Attention! */
```

UNIVERSITÉ CÔTE D'AZUR

Enoncé continue

- ✓ *utilisé dans les boucles*
- ✓ *retourne au test*

```
for (i = 0; i < MAX; i++) {
    if (t[i] < 0) continue;
    ...
}
```

```
for (i = 0; i < MAX; i++)
    switch (t[i]) {
        case 0 : ....;
        case 1 : continue; /* "passe" directement à 2 */
        ...
    }
    ...
}
```

UNIVERSITÉ CÔTE D'AZUR

Enoncé `return`

- ✓ **Sort de le fonction courante**
  - **permet de donner le résultat de la fonction**
  - **pas de valeur dans le cas d'une fonction** `void`

```
int min(int a, int b) {
   if (a < b) return a; else return b;
}
```

```
int search(int item, int t[], int t_size) {
   int i;
   for (i = 0; i < t_size; i++)
      if (t[i] == item) return i;
   return -1; /* si on est ici, item était absent
*/
}
```

UNIVERSITÉ
CÔTE D'AZUR

Expressions - Opérateurs

UNIVERSITÉ CÔTE D'AZUR

Expression vs Enoncé

- ✓ Une expression fournit une valeur
- ✓ Un énoncé change un état (effectue une action)
- ✓ En C, n'importe quelle expression peut être transformée en énoncé en lui ajoutant un " ; "

```
x = 1; 123 ; y = 2;
/* ici 123 est "oublié" */

x = getchar(); getchar(); y= getchar();
/* ici résultat du 2e getchar "oublié" */

x = getchar(); getchar; y= getchar();
/* ici getchar est "oublié" (pas d'appel) */
```

UNIVERSITÉ CÔTE D'AZUR

Affectation 1/2

- ✓ utilise le signe "="
- ✓ est à la fois un énoncé et une expression
  - a un effet de bord (valeur de l'opérande gauche est changée)
  - a un résultat (l'opérande gauche après affectation)
  - a un type (type de l'opérande gauche)
  - peut être utilisée de façon multiple

```
x = y = z = t[i] = 0
```

  - peut être composé avec un opérateur θ

```
x θ= y ⇔ x = x θ y
```

  - avec θ dans +, -, *, /, %, <<, >>, &, ^, |

```
while ((c= getchar()) != EOF) ....
t[i][j+3] += 1; /* <=> t[i][j+3] = t[i][j+3] + 1 */
x *= y + 2      /* <=> x = x * (y+2) (et pas x = x*y + 2) */
```

UNIVERSITÉ CÔTE D'AZUR

Affectation 2/2

✓ Opérateur préfixe et postfixe

```
i++ ⇔ i += 1 ⇔ i = i + 1
i-- ⇔ i -= 1 ⇔ 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++); /* ??? */
```

UNIVERSITÉ CÔTE D'AZUR

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'éléments 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 */
```

UNIVERSITÉ CÔTE D'AZUR

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` ⇒ `int`)
  - ensuite:
    - `long double`
    - `double`
    - `float`
    - `unsigned long`
    - `long`
    - `unsigned int`
    - `int`

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

UNIVERSITÉ CÔTE D'AZUR

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*

UNIVERSITÉ CÔTE D'AZUR

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" : "");
```

UNIVERSITÉ CÔTE D'AZUR

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;
}
```

UNIVERSITÉ CÔTE D'AZUR

Priorité des Opérateurs

Catégorie d'opérateurs	Opérateurs	Acco.
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
opérateur 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
opérateur d'affectation	`= += -= *= /= %= &= ^= |= <<= >>=`	D => G
opérateur virgule	`,`	G => D

UNIVERSITÉ CÔTE D'AZUR

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';
}
```

UNIVERSITÉ CÔTE D'AZUR

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';
}
```

UNIVERSITÉ CÔTE D'AZUR

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 */ ;
}
```

UNIVERSITÉ CÔTE D'AZUR

Exemple: Conversion Chaîne ⇒ entier (atoi)

```
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;
}
```

UNIVERSITÉ CÔTE D'AZUR

Résumé: Structures de Contrôle

- ✓ **Tests**
  - **Enoncé** `if`
  - **Enoncé** `switch`
  - **Expression conditionnelle** `? :`
- ✓ **Itérations**
  - **Enonce** `for`
  - **Enoncé** `while`
  - **Enoncé** `do while`
  - **Enoncé** `break`
  - **Enoncé** `continue`
  - **Enoncé** `goto`