Programmation Procédurale – Énoncés

Polytech'Nice Sophia Antipolis

Erick Gallesio

2015 – 2016

Énoncé 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()) != ' ') {
}
```

Expression *vs* énoncé

- 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) */
```

Affectation

Affectation en C:

- 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 $\theta =$ y $\Leftrightarrow$ 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) */
```

Énoncé 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() {
   ...
}
```

Énoncé if

Syntaxe:

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

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

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

Énoncé 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
- le mot clé **break** permet d'arrêter l'exécution de la séquence (continuer après le **switch**)
- la clause **default** peut être absente

Énoncé 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");
}
```

Énoncés while et do

Syntaxe:

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

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

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

Énoncés 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 ré-écrit en */
for (i = 0; i < 100; i++)
   t[i] = 0;
```

Énoncé 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! */
```

Énoncé 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++)
  swicth (t[i]) {
    case 0 : ....;
    case 1 : continue;        /* "passe" directement à 2 */
    ...
  }
  ...
}
```

Énoncé 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 */
}
```