Langage C

Dr El Hadji Bassirou TOURE
Ecole Supérieure Polytechnique (DGI-ESP-UCAD)
2020 - 2021

Blocs d'instructions

- Une instruction se termine par un point-virgule
  - z = foo(x+y);
- Bloc d'instructions : {…}
  - Composé de plusieurs instructions (ou d'une simple instruction)
  - Compilé comme une unite
  - Variables peuvent être déclarées dans un bloc

```
{
    int temp = x+y ;
    z = foo (temp );
}
```

  - Peut être vide {}
  - Pas de point-virgule à la fin

Blocs d’instructions imbriqués

- Un bloc peut comporter d’autres blocs et ainsi de suite

```
{
    int temp = x+y ;
    z = foo (temp );
    {
        float temp2 = x*y ;
        z += bar (temp2 );
    }
}
```

Conditions

- Nativement pas de type booléen
  - Depuis C99, le type bool utilisable dans stdbool.h
  - Une condition est une (série d') expression(s)
  - Exemple: n < 3 ou bien x < y || z < y
  - Une expression différente de 0 ➔ condition est vraie
  - L'expression doit être numérique (ou un pointeur)

```
const char str [ ] = "du texte";
if ( str ) / * str n'est pas null * /
    return 0;
```

Instructions Conditionelles

- Instructions if
- Instruction switch

Instructions if

```
if (x % 2)
        y += x /2;
```

- Evaluation de la condition
  - if (x % 2 != 0)
  - Si la condition est vraie, alors exécuter l'instruction interne
    - y += x/2
  - Sinon
    - ne fait rien

… else

```
if (x % 2)
        y += x /2;
else
        y+=(x+1)/2;
```

- else n'est pas obligatoire dans une instruction if
  - if (x % 2 != 0)
  - Si la condition est fausse, alors exécuter l'instruction interne
    - y += x/2
  - L'instruction interne peut éventuellement être un bloc

… else if

```
if (x % 2 == 0)
        y += x/2;
else if (x % 4 == 1)
        y += 2* ((x +3)/4);
else
        y += (x+1)/2;
```

- Offre plus d'alternatives
- Conditions évaluées dans l'ordre jusqu'à ce que l'une soit vraie
  - Son instruction interne exécutée
- Si plusieurs conditions vraies , seule la première est exécutée
- Equivalent à des instructions if imbriquées

… Instructions if imbriquées

```
if (x % 4 == 0)
        if (x % 2 == 0)
                y = 2;
else
        y = 1;
```

- A quel **if** (interne ou externe), le **else** se rapporte ?
- Pour l'associer au if externe alors utiliser { }

```
if (x % 4 == 0) {
        if (x % 2 == 0)
                y = 2;
} else
        y = 1;
```

Instructions switch

- Alternative à l'instruction conditionnelle
- Prend en entrée une variable de type entier (ou caractère)
- Considère des cas selon les valeurs de cette variable

```
switch (ch) {
    case 'Y' : / * ch == 'Y ' * /
    / * do something * /
    break ;
    case 'N' : / * ch == 'N' * /
    / * do something else * /
    break ;
    default : / * otherwise * /
    / * do a third thing * /
    break ;
}
```

Cas multiples

- Comparaison pour chaque cas (effectuée dans l'ordre)
  - Si correspondance, exécution du code interne jusqu'au prochain break;
  - Si pas de break, l'exécution "glisse" sur les cas suivants

```
switch (ch) {
        case 'y':
        case 'Y' :
        / * do something
              if ch == 'y' ou ch == 'Y'
        * /
        break ;
}
```

```
switch (ch) {
        case 'Y' :
        / * do something if ch == 'Y' * /
        case 'N' :
        / * do something
        if ch == 'Y' ou ch == 'N'
        * /
        break ;
}
```

Instructions Répétitives (boucles)

- Boucle for
- Boucle while
- Boucle do ... while
- continue et break

Boucle while

```
while ( / * condition * / )
    /* instructions */
```

- Evalue **instructions** tant que **condition** est vraie
- La condition est évaluée en premier
  - Donc **instructions** peut ne jamais être évaluée
  - **instructions** peut être un bloc.

Boucle for

```
int i , j = 1;
for ( i = 1; i <= n; i ++)
        j *= i ;
```

- i est appelé le compteur de la boucle
- Entre les parenthèses, trois expressions séparées un point-virgule:
  - Initialisation: i = 1
  - Condition: i <= n
  - Incrémentation: i++
- Une expression non renseignée, est considérée comme vraie

Boucle for

- Equivalent à une boucle while

```
int j = 1;
int i = 1; / * initialisation * /
while ( i <= n / * condition * / ) {
      j *= i ;
      i ++; / * incrémentation * /
}
```

- Expressions composées séparées par des virgules (opérateur)
  - Cet opérateur a une faible priorité, a une associativité de gauche à droite
  - L'instruction ; (skip) indique aucune action

```
int i , j ;
for ( i = 1, j = 1; i <= n; j *= i , i ++) ;
```

Boucle do… while

```
char c ;
do {
    / * corps de la boucle * /
    puts ( "Keep going? (y/n) " );
    c = getchar ( ) ;
    / * autres traitements * /
} while (c == 'y'  && /* autres conditions */) ;
```

- Différence avec la boucle while : la condition est évaluée après chaque itération
- L'instruction interne est exécutée au moins une fois
- Noter le point-virgule à la fin

break :

- Pour sortir d'une boucle plus tôt que prévu
- break sort d'une boucle ou d'une instruction switch
- Ce programme est-il équivalent au précédent ?

```
char c ;
do {
    / * corps de la boucle * /
    puts ( "Keep going? (y/n) " );
    c = getchar ( ) ;
    if (c != 'y' )
        break ;
    / * autres traitements * /
} while ( / * autres conditions * / );
```

continue :

- Utilisé pour passer(ignorer) une itération
  - Ignore le reste du corps d'une boucle, retournant vers la condition de la boucle
  - Exemple:

```
int gcd ( int a, int b) {
    int i , ret = 1, minval = a > b ? b : a;
    for ( i = 2; i <= minval ; i ++) {
        if (a % i ) / * i n'est pas diviseur de a * /
            continue ;
        if (b % i == 0) / * i est, à la fois, diviseur  de a et b* /
            ret = i ;
    }
    return ret ;
}
```