2- LANGAGE C

STRUCTURE DE CONTRÔLE

Loïc Cuvillon
l.cuvillon@unistra.fr

Structures de contrôle du flot d'exécution

- **Structure de test** : `if...else`, `else if`, et `switch`
- Structure de boucle : `while`, `do..while` ; et `for`
- Rupture du flot : `continue`, `break`

Les conditions

- En C, pas de type booléen.
- La condition est une valeur numérique entière,

  généralement le résultat d'une expression:

0	⟷ FAUX
toute valeur différente de 0	⟷ VRAI

- Exemples de condition :
  - `n < 3`
  - `x < y || z < y`
  - `x`
  - `!x` (négation logique de x)
  - `c == 'a'`
  - `0` (toujours faux)
  - `function1()`, son résultat

Opérateurs relationnels

- Comparaison de 2 opérandes pour produire un résultat « booléen »

operateur	signification	exemples	
>	supérieur à	3 > 2;	/* évalué 1 */
		2.99 > 3;	/* évalué 0 */
>=	supérieur ou égal à	3 >= 3;	/ * évalué 1 */
		2.99 >= 3;	/* évalué 0 */
<	inférieur à	3 < 3;	/* évalué 0 */
		'A' < 'B';	/* évalué 1 */
<=	inférieur ou égal à	3 <= 3;	/* évalué 1 */
		3.99 < 3;	/* évalué 0 */

operateur	signification	exemples	
==	égal à	3 == 3	/* évalué 1 */
		'A' == 'a'	/* évalué 0 */
!=	différent de	3 != 3	/* évalué 0 */
		2.99 != 3	/* évalué 1 */

Opérateurs logiques

operateur	signification	exemples
&&	ET	((9/3) == 3) && (2*3 == 6) /* évalué 1 */
		('A' == 'a') && (3 == 3) ; /* évalué 0 */
||	OU	2 == 3 || 'A' == 'A' ; /* évalué 1 */
		2.99 > 3 || 0 /* évalué 0 */
!	NOT	!(3==3) /* évalué 0 */
		!(2.99 >= 3) /* évalué 1*/

- ET est vrai si les deux opérandes/expressions sont vraies
- OU est vrai si l'une des opérandes est vraie

if

- Le test `if` : exécution conditionnelle

```
if ( /*condition */)
{
   /* instruction(s) a executer si condition vraie*/
}
```

if

```
if (age > 18)
{
   printf(" vous êtes majeur !\n");
   auth=1;
}
```

- Evaluer le condition `age > 18`
- Si vrai, exécuter les instructions
  ```
  printf(" ...");
  auth=1;
  ```
- Sinon, rien

if…else

```
if ( /*condition */)
{
     /*instructions a executer si condition vraie*/
}
else
{
     /*instructions a executer si fausse*/
}
```

if…else

```
if (age > 18)
{
      printf(" Vous etes majeur !\n");
}
else
{
      printf(" Vous etes mineur…");
}
```

else if

```
if (age > 18)
{
      printf(" vous etes majeur !");
}
else if (age > 4)
{
      printf("Bon, t'es pas trop jeune...");
}
else
{
      printf("Aga aga gaa");
      /* langage nourrisson */
}
```

Note: l'ordre des conditions est important dans cette exemple

- conditions exclusives avec `else if` :

Si plusieurs conditions vraies, seules les instructions de -la première condition vraie rencontrée- sont exécutées

- `else if` est équivalant à 2 `if…else` imbriqués :

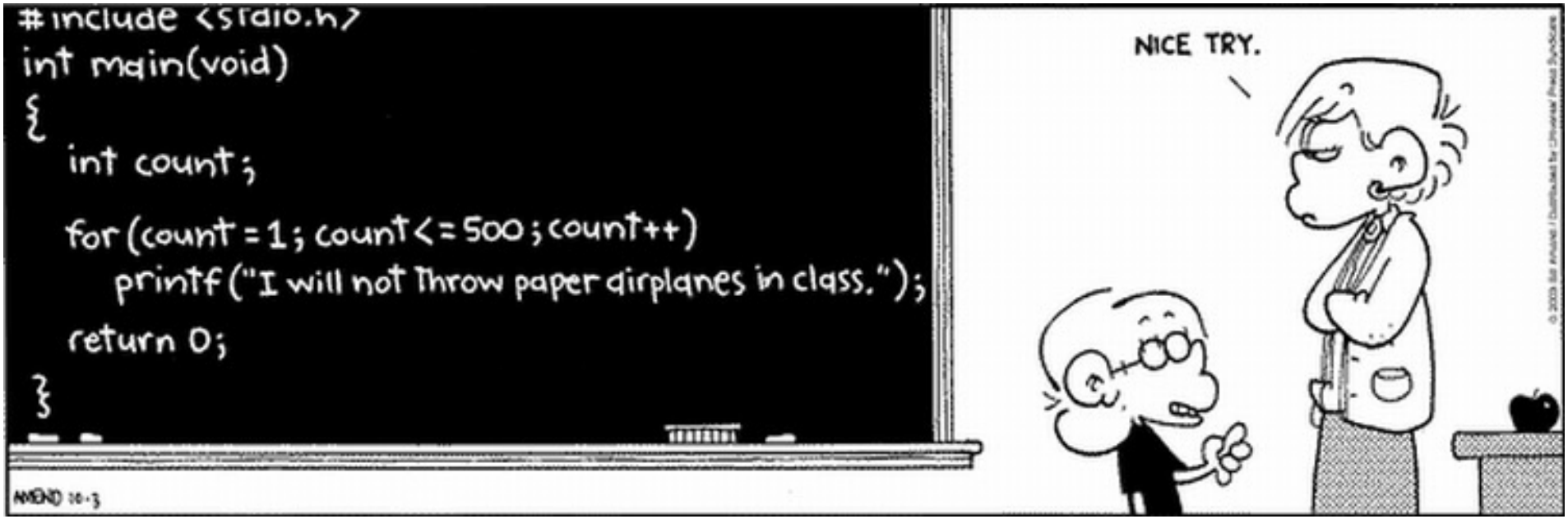
```
if
else { if … else  }
```

```
if (age > 18)
{
      printf(" vous etes majeur !");
}
else
{       if (age > 4)
        { printf("Bon, t'es pas trop jeune...");}
        else
        { printf("Aga aga gaa"); }
}
```

Structures de contrôle du flot d'exécution

- Structure de test : `if...else`, `else if`, et `switch`
- **Structure de boucle : `while`, `do..while` ; et `for`**
- Rupture du flot : `continue`, `break`

Structures de contrôle du flot d'exécution



La boucle `while`

```
while ( /*condition */)
{
        /* corps de la boucle :
                instruction(s) à répéter */
}
```

- Exécuter **tant que** la condition est vraie.
- Condition évaluée en premier, donc le corps peut ne jamais être exécuté

- Attention à la boucle infinie : `while (1)`

(ctrl+c, la combinaison de touche pour tuer le programme)

La boucle `while`

```
int i=0;

while ( i < 4 )
{
        printf(”Valeur de i : %d\n”, i);
        i++;
                /*incrémentation*/
}
```

- A l’exécution :

```
Valeur de i : 0
Valeur de i : 1
Valeur de i : 2
Valeur de i : 3
```

(ensuite `i` vaut 4 et le test est faux)

do…while ;

```
/*CALCUL SOMME DES ENTIERS SAISIS TANT QUE DIFFERENT DE 0*/
int number=0, sum=0;  /*initialisation importante de sum*/
do
{
      /* corps de la boucle*/
      printf("Enter a number: ");
      scanf("%d", &number);
      sum += number;

} while(number != 0) ;
```

Attention au point-virgule final du `do...while`

- Différence par rapport à `while` : condition testée à la fin.
- Le corps de la boucle est **exécuté au moins une fois**

De la boucle `while` vers `for`

```
int compte=0;    /*initialisation*/

while (compte < 4)
{
      printf(”compte value : %d\n”, compte);
      compte++;
}
```

équivaut à la boucle `for` suivante :

```
int compte;
for (compte=0; compte<4; compte++)
{
            printf(”compte value : %d\n”,compte);
}
```

La boucle `for`

```
for ( initialisation ; condition ; transition)
{
      /*Instruction(s) de la boucle*/
}
```

- *initialisation* :

  expression exécutée une unique fois au départ
- *condition* :

  test d'arrêt de la boucle (évalué au début de chaque itération)
- *transition* :

  expression exécutée à la fin de chaque itération

Note: chacune de ces 3 expressions est facultative. (On peut laisser vide.)

La boucle `for` : exemple

- Calcul itératif de la factorielle de `n`

```
int n=6;
int i, factoriel=1;

for (i=1 ; i <= n ; i++)
{
      factoriel = factoriel * i;
}

printf ("%d! vaut %d \n", n, factoriel);
```

Erreurs classiques

- Erreur 1 : confondre == et =

```
if ( age  =  18)     /*affectation : age vaut 18      */
{ }                  /* puis age est testé            */
                     /* 18 -> condition toujours VRAI*/
```

- Erreur 2 : le point virgule de trop

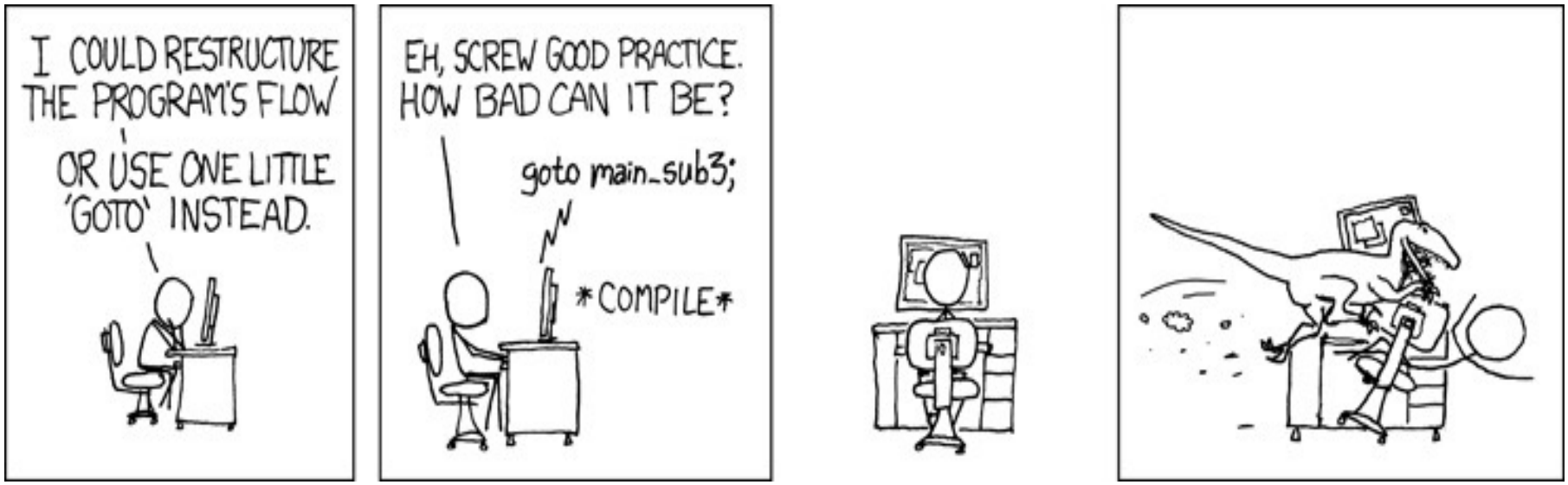
```
if ( age == 18);     /*test sans instructions associé*/
{
   printf(" Vous etes majeur");   /*toujours executé*/
}


for (i=0; i<5; i++);  /*boucle sans instructions*/
{
   printf("%i",i);    /* bloc exécuté une seule fois */
}
```

Structures de contrôle du flot d'exécution

Les ruptures du flot d'exécution

- `break`
- `continue`
- `goto` (à ne pas utiliser)



xkcd.com

L'instruction `break`

- Pour terminer prématurément une boucle
- **`break`** fait sortir immédiatement de la boucle la plus locale ou un `switch`

```
/* CALCUL SOMME de 5 NOMBRES */
double number, sum = 0.0;
for(i=1; i <= 5; ++i)
{
    printf("Enter number no%d: ",i);
    scanf("%lf",&number);

    /* If user enters a negative value, the loop ends */
    if(number < 0)
    { break; }
    sum += number;          /* sum = sum + number; */
}
```

L'instruction `break`

- Sortie du programme précédent :

```
Enter number no1: 3.4
Enter number no2: 1.6
Enter number no3: -1.6
Sum: 5.000
```