IN200 – Fondements de l'Informatique II

Sandrine Vial
sandrine.vial@uvsq.fr
Bureau 306 D (3ème étage) Bât. Descartes.

Janvier 2016

Emploi du temps

- **Cours** : Mardi (BI, DLBI, MPCI1) ou Mercredi (MIASHS, DLMP, MPCI2) de 15h15 à 16h45 en Amphi D.
- **TD** : (à partir du 1er février).
  - cf. emploi du temps sur votre ENT.

Toutes les informations sur e-campus.

Absences à signaler à la scolarité auprès de **Stéphanie Quesnot** et/ou **Charlotte Le Maire** (Scolarité, Bât. Fermat).

Organisation

- Pour les TD : utilisation du cartable numérique ou de votre ordinateur portable.
- 3 **contrôles** sur table dans le semestre durant le créneau de cours :
  - Mardi 16 Février (BI, DLBI, MPCI1) ou Mercredi 17 Février (MIASHS, DLMP, MPCI2)
  - Mardi 22 Mars (BI, DLBI, MPCI1) ou Mercredi 23 Mars (MIASHS, DLMP, MPCI2)
  - Mardi 19 Avril (BI, DLBI, MPCI1) ou Mercredi 20 Avril (MIASHS, DLMP, MPCI2)
- 1 **khôlle** sur machine durant le créneau de TD :
  - Semaine du 9 au 13 Mai 2016
- Des évaluations lors des TDs : vous ne serez pas forcément avertis !

Semaine du 29 février : semaine de vacances (ni TD, ni Cours).
Semaine du 25 avril : semaine de vacances (ni TD, ni Cours).

But de l'UE

- Approfondir la programmation en langage C.
  - Utilisation d'une bibliothèque graphique
- Faire ses premiers pas en algorithmique.

Variables de type `int`

- Permet la représentation des entiers signés en machine.
- Codage des entiers de -2^{31} à $2^{31} - 1$ sur 32 bits.
- Représentation des entiers en base *décimale*, *hexadécimale* et *octale*.

Exemples

```
int j;
int p;

j = 2;
p = j;
p = 3;
j = 0x7F;
p = j;
j = 06;
```

Variables réelles

Les nombres réels sont représentés en machine (virgule flottante) :

- en trois parties : un signe (s), une mantisse (m) et un exposant (e).

$$reel = s \times m \times b^e$$

où b est la base de la représentation (généralement 2).

Avantages et Inconvénients

- Représentation de très petits et de très grands nombres
- La précision est finie => Tous les nombres ne sont pas représentables
- Problèmes d'arrondi dans les calculs

$$(2^{60} + 1) - 2^{60} \neq 1$$

Les priorités

Du plus prioritaire au moins prioritaire

opérateur	Symbole	Arité	Associativité
appel de fonction	()		
signes	+ -	1	$D \Rightarrow G$
multiplication, division, modulo	* / %	2	$G \Rightarrow D$
addition, soustraction	+ -	2	$G \Rightarrow D$
opérateurs relationnels	< <= > >=	2	$G \Rightarrow D$
opérateurs comparaison	== !=	2	$G \Rightarrow D$
affectation	=	2	$D \Rightarrow G$

Les instructions

- L'ordre d'exécution des instructions est séquentiel.

```
int a, b;

a = 10;
b = a * 2;
```

- L'instruction vide existe

```
int a;
a = a * 2;;;
```

- Un bloc d'instructions { ... }

```
int a, b;

a = 2;
{
  b = a * 2;
  a = 12;
}
```

`if ... else`

- Si une condition est remplie exécuter des instructions sinon exécuter d'autres instructions
- `if (condition) { un bloc d'instructions 1 } else { un bloc d'instructions 2 }`
- `condition` doit être vraie pour que le bloc d'instructions 1 soit exécuté et fausse pour que le bloc d'instructions 2 soit exécuté.

Exemple 1

```
#include "graphics.h"
int main()
{
     POINT p;

     init_graphics(900,600);

     p = wait_clic();
     if (p.x < 450)
     {
          draw_fill_circle(p,50,rouge);
     }

     wait_escape();
     exit(1);
}
```

Exemple 2

```
#include "graphics.h"
int main()
{
     POINT p;

     init_graphics(900,600);

     p = wait_clic();
     if (p.x < 450)
     {
          draw_fill_circle(p,50,rouge);
     }
     else
     {
          draw_fill_circle(p,50,vert);
     }
     wait_escape();
     exit(1);
}
```

Tests Simples

- Tous les opérateurs de comparaisons sont utilisables dans une condition
- `if (a < 10) ...`
- `if (b >= 12) ...`
- `if (i == j) ...`

Attention ! Piège !

- L'expression `if (a = valeur_non_nulle) ...` est une expression toujours **vraie**

```
POINT p;

p.x = 100;
p.y = 200;

if (p.x = 200)
  {
    draw_fill_circle(p,50,vert);
  }
```

Attention ! Piège !

- L'expression `if (a = valeur_non_nulle) ...` est une expression toujours **vraie**
- L'expression `if (a = 0) ...` est une expression toujours **fausse**

```
POINT p;

p.x = 0;
p.y = 200;

if (p.x = 0)
  {
    draw_fill_circle(p,50,vert);
  }
```

Conditions plus complexes

- On peut combiner les conditions
  - Les deux conditions sont vraies en même temps : `ET` noté `&&`
  - Une des deux conditions soit vraie : `OU` noté `||`

Conditions plus complexes

- On peut combiner les conditions
  - Les deux conditions sont vraies en même temps : `ET` noté `&&`
  - Une des deux conditions soit vraie : `OU` noté `||`

A && *B*	TRUE	FALSE
TRUE	**TRUE**	*FALSE*
FALSE	*FALSE*	*FALSE*

Conditions plus complexes

- On peut combiner les conditions
  - Les deux conditions sont vraies en même temps : `ET` noté `&&`
  - Une des deux conditions soit vraie : `OU` noté `||`

A && *B*	TRUE	FALSE
TRUE	**TRUE**	*FALSE*
FALSE	*FALSE*	*FALSE*

A || *B*	TRUE	FALSE
TRUE	**TRUE**	**TRUE**
FALSE	**TRUE**	*FALSE*

Exemple : trouver b tel que $a \leq b \leq c$

```
int x,y,z;
int milieu;

init_graphics(400,200);
x = lire_entier_clavier();
y = lire_entier_clavier();
z = lire_entier_clavier();

if (x <= y)
  {
    if (y <= z) milieu = y;
    else if (x >= z) milieu = x;
    else milieu = z;
  }
else
  {
    if (x <= z) milieu = x;
    else if (y >= z) milieu = y;
    else milieu = z;
  }
write_int(milieu);
wait_escape();
```

Un programme

- Toutes les lignes de code sont exécutées séquentiellement.
- **Embranchement** `if (...) ... else ...` : Permet de choisir les lignes de codes à éxécuter en fonction d'une condition.
- **Boucles** : Permet de **répéter** un *bloc d'instructions* en fonction d'*une condition*.

L'instruction `while`

Tant qu'une *condition* est remplie, exécuter un *bloc d'instructions*.

```
while (condition)
{
   bloc d'instructions
}
```

Exemple

```
POINT p1,p2;

    init_graphics(600,300);

    p1.x = 10;
    p2.x = p1.x;
    p1.y = 10;
    p2.y = 290;
    while (p1.x < 590)
    {
         draw_line(p1,p2,bleu);
         p1.x = p1.x + 40;
         p2.x = p1.x;
    }
    wait_escape();
    exit(0);
```

L'Instruction `for`

Deux définitions

1. Après *initialisation* et tant qu'une *condition* est remplie, exécuter un *bloc d'instructions*.
2. Pour une valeur comprise entre une *valeur minimale* et une *valeur maximale* faire *bloc d'instructions*.

```
for (initialisation; condition; poursuite)
{
   bloc d'instructions
}
```

Exemple

```
POINT p1,p2;

init_graphics(600,300);
p1.y = 10;
p2.y = 290;
for (p1.x = 10; p1.x < 590 ; p1.x = p1.x + 40)
{
    p2.x = p1.x;
    draw_line(p1,p2,bleu);
}
wait_escape();
exit(0);
```

Equivalence entre le `while` et le `for`

```
for(initialisation;condition;poursuite)
{
   bloc d'instructions
}
```

```
initialisation;
while(condition)
{
   bloc d'instructions
   poursuite;
}
```

do .... while

- Faire un *bloc d'instructions* et ensuite le répéter tant qu'une *condition* est vraie.

```
do
{
   bloc d'instructions
} while(condition);
```

Exemple

```
POINT p1,p2;

init_graphics(600,300);
p1.y = 10;
p2.y = 290;
p1.x = 10;
do
{
    p2.x = p1.x;
    draw_line(p1,p2,bleu);
    p1.x = p1.x + 40;
} while (p1.x < 590);
wait_escape();
exit(0);
```

Exemple

```
POINT p1,p2;
int i;
init_graphics(600,300);
p1.x = 10;
p2.x = 590;
p1.y = 10;
for(i = 0; i < 10; i = i+1)
{
   p2.y = p1.y;
   if (i%2 == 0)
   {
      draw_line(p1,p2,vert);
   }
   else
   {
      draw_line(p1,p2,bleu);
   }
   p1.y = p1.y + 30;
}
```

Exemple

```
POINT p1;
init_graphics(600,300);

for(p1.y = 10; p1.y < 290; p1.y = p1.y + 30)
{
   for(p1.x = 10; p1.x < 290; p1.x = p1.x + 30)
   {
      if (p1.x == p1.y)
      {
         draw_fill_circle(p1,10,rouge);
       }
   }
}
```