Introduction au Langage C

Vincent Vidal

Maître de Conférences
Enseignements : IUT Lyon 1 - pôle AP - Licence ESSIR - bureau 2ème étage
Recherche : Laboratoire LIRIS - bât. Nautibus
E-mail : vincent.vidal@univ-lyon1.fr

Supports de cours et TPs : `http://claroline connect.univ-lyon1.fr` espace d'activités "M1102 M1103 C - Introduction au langage C"

46H prévues $\approx$ 42H de cours+TPs, 2H - interros, et 2H - examen final

Évaluation : Contrôle continu + examen final + Bonus/Malus TP

Plan

Plan

Les 3 types d'instruction en C

- Les *instructions simples* qui sont obligatoirement terminées par un point virgule " **;**".
- Les *blocs* { }.
- Les *instructions de structuration/de contrôle* permettant de réaliser des choix (e.g. **if**, **switch**) et des boucles/itérations (e.g. **while**, **for**).

Les *blocs* et les *instructions de structuration* sont **récursifs**, car ils peuvent à leur tour, contenir une ou plusieurs de ces 3 formes.

Les 3 types d'instruction en C

- Les *instructions simples* qui sont obligatoirement terminées par un point virgule " **;**".
- Les *blocs* **{ }**.
- Les *instructions de structuration/de contrôle* permettant de réaliser des choix (e.g. **if**, **switch**) et des boucles/itérations (e.g. **while**, **for**).

Les *blocs* et les *instructions de structuration* sont **récursifs**, car ils peuvent à leur tour, contenir une ou plusieurs de ces 3 formes.

Les 3 types d'instruction en C

- Les *instructions simples* qui sont obligatoirement terminées par un point virgule " **;**".
- Les *blocs* **{ }**.
- Les *instructions de structuration/de contrôle* permettant de réaliser des choix (e.g. **if**, **switch**) et des boucles/itérations (e.g. **while**, **for**).

Les *blocs* et les *instructions de structuration* sont **récursifs**, car ils peuvent à leur tour, contenir une ou plusieurs de ces 3 formes.

Les 3 types d'instruction en C

- Les *instructions simples* qui sont obligatoirement terminées par un point virgule " **;**".
- Les *blocs* **{ }**.
- Les *instructions de structuration/de contrôle* permettant de réaliser des choix (e.g. **if**, **switch**) et des boucles/itérations (e.g. **while**, **for**).

Les *blocs* et les *instructions de structuration* sont **récursifs**, car ils peuvent à leur tour, contenir une ou plusieurs de ces 3 formes.

- *Instruction vide* : **syntaxe** : **;**
- *Instruction expression* : **syntaxe** : <*expression*> **;**

expression inclut les expressions et les instructions-expressions (basées sur des opérateurs et qui ont une valeur).

Un bloc est une suite d'instructions placées entre **{** et **}**. Un bloc peut se ramener à une seule instruction, voire être vide. **Évitez d'ajouter des ' ;' intempestifs à la suite d'un bloc**.

```
{
   [ suite_de_déclarations ]
   [ suite_d_instructions ]
}
```

Les crochets **[ ]** signifient optionnel.

L'instruction conditionnelle **if** (1/4)

```
if (expression)
   instruction_1
```

instruction_1 est exécutée seulement si *expression* est vraie (non nulle). *expression* est une expression quelconque. *instruction_1* est une instruction quelconque.
Exemples d'expression : 5 est vraie, $6 == 6$ est vraie, $5 < 4$ est fausse, $2 - 2$ est fausse...

L'instruction conditionnelle **if** (2/4)

```
if (expression)
   instruction_1
else
   instruction_2
```

instruction_1 est exécutée si *expression* est vraie (non nulle), sinon *instruction_2* après le else est exécutée. *instruction_1* et *instruction_2* sont des instructions quelconques.

L'instruction conditionnelle **if** (3/4)

```
if (expression_1)
if (expression_2)
   instruction_1
else
   instruction_2
```

Avec quel if le else est-il associé ?

L'instruction conditionnelle **if** (3/4)

```
if (expression_1)
if (expression_2)
   instruction_1
else
   instruction_2
```

Avec quel if le else est-il associé ?
Le second, la règle est : **Un else se rapporte toujours au dernier if rencontré auquel un else n'a pas encore été attribué.**

L'instruction conditionnelle **if** (4/4)

```c
#include <stdio.h>  /* pour printf et scanf */
#include <math.h>   /* pour sqrt */

int main(void) {
   double x;
   printf("Donner un nombre :");
   scanf("%lf", &x);
   if (x > 0.0)
      printf("La racine carrée de %f est %f\n", x, sqrt(x));
   else
      if (x < 0.0)
         printf("%f n'a pas de racine carrée\n", x);
      else
         printf("La racine carrée de %f est 0\n", x);

   return 0;
}
```

L'instruction de branchement conditionnel **switch** (1/3)

```
switch(expression_entière)
{
   case constante_1 : [ suite_d_instructions_1 ]
   case constante_2 : [ suite_d_instructions_2 ]
   ........
   case constante_n : [ suite_d_instructions_n ]

   [ default : suite_d_instructions ]
}
```

expression_entière est une **expression entière** quelconque. Les étiquettes constantes *constante_i* sont entières.

L'instruction de branchement conditionnel **switch** (2/3)

Si *expression_entière* est évaluée à la valeur d'une étiquette présente, il y a branchement vers cette étiquette.
Sinon, il y a branchement vers l'étiquette **default** si elle est présente, et sinon on ne rentre pas dans le bloc du switch.

suite_d_instructions_i est une séquence d'instructions arbitraires, éventuellement terminée par l'instruction **break** *pour sortir du bloc* **switch** et ne pas exécuter les instructions des étiquettes suivantes.

L'instruction de branchement conditionnel **switch** (2/3)

Si *expression_entière* est évaluée à la valeur d'une étiquette présente, il y a branchement vers cette étiquette.
Sinon, il y a branchement vers l'étiquette **default** si elle est présente, et sinon on ne rentre pas dans le bloc du switch.

suite_d_instructions_i est une séquence d'instructions arbitraires, éventuellement terminée par l'instruction **break** *pour sortir du bloc* **switch** et ne pas exécuter les instructions des étiquettes suivantes.

L'instruction de branchement conditionnel **switch** (3/3)

```c
#include <stdio.h>  /* pour printf et scanf */

int main(void) {
   int choix;
   printf("Donner un entier entre 0 et 3 :");
   scanf("%d", &choix);
   switch(choix)
   {
      case 0 :
      case 2 : printf("Un chiffre pair.\n");
               break;
      case 1 :
      case 3 : printf("Un chiffre impair.\n");
               break;
      default : printf("Réponse incorrecte.\n");
   }

   return 0;
}
```

Donner les exécutions de ce programme dans les cas suivants : l'utilisateur entre 0 ; 1 ; 2 ; 3 ; 4.

Les 3 instructions de boucle

- **do ... while**
- **while**
- **for**

L'instruction de répétition **do ... while** (1/2)

```
do
   instruction
while (expression) ;
```

instruction est exécutée au moins une fois et elle est répétée tant que *expression* est vraie (non nulle). **Cette condition de poursuite est évaluée après l'instruction**. *instruction* est une instruction quelconque. *expression* est une expression quelconque. Si *expression* est toujours vraie, alors il y a une boucle infinie.

L'instruction de répétition **do ... while** (2/2)

```c
#include <stdio.h>  /* pour printf et scanf */

int main(void) {
   double somme=0.0, note;
   int cpt=0;
   do
   {
      printf("Donner une note (nombre négatif pour terminer) :");
      scanf("%lf", &note);
      somme += note;
      cpt++;
   }while(note >= 0.0) ;
   somme -= note; if(cpt > 1) cpt--;
   printf("La moyenne des notes est %f\n", somme/cpt);

   return 0;
}
```

Modifier le programme précédent pour qu'il donne en plus de la moyenne, la note min et la note max.

L'instruction de répétition **do ... while** (2/2)

```c
#include <stdio.h>  /* pour printf et scanf */

int main(void) {
   double somme=0.0, note;
   int cpt=0;
   do
   {
      printf("Donner une note (nombre négatif pour terminer) :");
      scanf("%lf", &note);
      somme += note;
      cpt++;
   }while(note >= 0.0) ;
   somme -= note; if(cpt > 1) cpt--;
   printf("La moyenne des notes est %f\n", somme/cpt);

   return 0;
}
```

Modifier le programme précédent pour qu'il donne en plus de la moyenne, la note min et la note max.

L'instruction de répétition **while** (1/2)

```
while (expression)
    instruction
```

instruction est exécutée seulement si *expression* est vraie, et elle est répétée tant que *expression* est vraie. **Cette condition de poursuite est évaluée avant l'instruction**. *instruction* est une instruction quelconque. *expression* est une expression quelconque. Si *expression* est toujours vraie, alors il y a une boucle infinie.

L'instruction de répétition **while** (2/2)

```c
#include <stdio.h>  /* pour printf et scanf */

int main(void) {
   double somme=0.0, note;
   int cpt=0;
   printf("Donner une note (nombre négatif pour terminer) :");
   scanf("%lf", &note);
   while(note >= 0.0)
   {
      somme += note;
      cpt++;
      printf("Donner une note (nombre négatif pour terminer) :");
      scanf("%lf", &note);
   }
   if(cpt == 0) cpt=1;
   printf("La moyenne des notes est %f\n", somme/cpt);

   return 0;
}
```

L'instruction de répétition **for** (1/3)

```
for ( [ expression_1 ] ; [ expression_2 ] ; [
    expression_3 ])
   instruction
```

Les trois expressions du **for** sont facultatives. Si *expression_2* est absente, elle est considérée comme vraie.

- *expression_1* joue souvent le rôle d'initialisation des variables de contrôle à partir desquelles la condition de poursuite est construite.
- *expression_2* est la condition de poursuite.
- *expression_3* joue souvent le rôle de mise à jour des variables de contrôle.

L'instruction de répétition **for** (2/3)

for (expression_1 ; expression_2 ; expression_3) instruction est équivalent à :

```
expression_1 ;
while (expression_2)
{
   instruction
   expression_3 ;
}
```

L'instruction de répétition **for** (3/3)

```c
#include <stdio.h>  /* pour printf et scanf */

int main(void) {
   int table, i;
   printf("Quelle table de multiplication afficher?");
   scanf("%d", &table);
   for( i=0 ; i <= 10 ; i++ )
      printf("%d x %d = %d ; ", table, i, table*i);

   return 0;
}
```

Les instructions **break**, **continue** et **goto**

break et **continue** s'emploient principalement au sein des boucles tandis que **goto** est d'un usage libre mais peu répandu.
Au sein d'une boucle :

- **break** permet d'interrompre le déroulement de la boucle englobante. Si un break apparaît dans un switch imbriqué dans une boucle, il ne fait sortir que du switch.
- **continue** permet de passer prématurément au tour de boucle suivant.
- **break** et **continue** ne se rapportent qu'à la boucle dans laquelle ils se trouvent (e.g. la plus interne parmi $n \geq 2$ boucles imbriquées).

Les instructions **break**, **continue** et **goto**

break et **continue** s'emploient principalement au sein des boucles tandis que **goto** est d'un usage libre mais peu répandu.
Au sein d'une boucle :

- **break** permet d'interrompre le déroulement de la boucle englobante. Si un break apparaît dans un switch imbriqué dans une boucle, il ne fait sortir que du switch.
- **continue** permet de passer prématurément au tour de boucle suivant.
- **break** et **continue** ne se rapportent qu'à la boucle dans laquelle ils se trouvent (e.g. la plus interne parmi $n \geq 2$ boucles imbriquées).

Les instructions **break**, **continue** et **goto**

break et **continue** s'emploient principalement au sein des boucles tandis que **goto** est d'un usage libre mais peu répandu.
Au sein d'une boucle :

- **break** permet d'interrompre le déroulement de la boucle englobante. Si un break apparaît dans un switch imbriqué dans une boucle, il ne fait sortir que du switch.
- **continue** permet de passer prématurément au tour de boucle suivant.
- **break** et **continue** ne se rapportent qu'à la boucle dans laquelle ils se trouvent (e.g. la plus interne parmi $n \geq 2$ boucles imbriquées).

L'instruction **break**

```c
#include <stdio.h>  /* pour printf et scanf */

int main(void) {
   double somme=0.0, note;
   int cpt=0;
   do
   {
      printf("Donner une note (nombre négatif pour terminer) :");
      scanf("%lf", &note);
      if(note < 0.0) break;
      somme += note;
      cpt++;
   }while( 1 ) ;
   if(cpt == 0) cpt=1;
   printf("La moyenne des notes est %f\n", somme/cpt);

   return 0;
}
```

L'instruction **continue**

```c
#include <stdio.h>  /* pour printf et scanf */

int main(void) {
   int somme=0, n, i;
   printf("Donner une valeur de n : ");
   scanf("%d", &n);
   for(i=1 ; i <= n ; i++)
   {
      if( i%2==1 ) continue;
      somme += i;
   }
   printf("La somme des entiers naturels pairs <= %d est %d\n", n, somme);

   return 0;
}
```

L'instruction **goto** (1/3)

L'instruction **goto** permet un branchement vers un emplacement quelconque du programme, qui est localisé à l'aide d'une étiquette.

```
...
goto etiquette;
...
etiquette :
...
```

L'instruction **goto** (2/3)

```c
#include <stdio.h>  /* pour printf et scanf */

int main(void) {
   int choix;

   debut :

   printf("Bienvenue!\n");

   /* Ici on peut avoir d'autres instructions */
   printf("Taper 1 pour aller au debut du programme et tout autre valeur pour aller a la fin\n");
   scanf("%d", &choix);
   if(choix==1)
      goto debut;
    else
      goto fin;
   /* Ici on peut avoir d'autres instructions */

   fin :

   printf("Au revoir et a bientot!\n");

   return 0;
}
```

L'instruction **goto** (3/3)

```c
#include <stdio.h>  /* pour printf et scanf */

int main(void) {

   /* simuler une boucle infinie et la condition pour en sortir */
   debut_boucle :

   [ suite_d_instructions ]
   if(expression) goto fin;

   goto debut_boucle;

   fin :

   return 0;
}
```