Instructions conditionnelles et boucles en C [1]

Salem BENFERHAT

Centre de Recherche en Informatique de Lens (CRIL-CNRS)
email : benferhat@cril.fr

[1] Version préliminaire du cours. Tout retour sur la forme comme sur le fond est le bienvenu.

Instructions conditionnelles en Langage C

La notion de bloc

- Un bloc est une suite d'instructions délimitées par les accolades : { et }
  - Rappels :
    - Les tabulations/indentations et les espaces n'ont pas d'importance au niveau de la compilation (contrairement à Python).
    - Ils restent importants pour la lisibilité des programmes

La notion de bloc

- Un bloc ne se termine pas un point virgule ";"
- Remarque :
  - Mettre un point virgule ";" revient à exécuter, juste après le bloc, une instruction vide :-)
- Un bloc joue un rôle important dans les définitions des corps des boucles et des conditionnelles.

Instructions conditionnelles

Instructions conditionnelles

La syntaxe générale est :

```
if (condition)
 {
  Bloc d'instructions
 }
```

Instructions conditionnelles

La syntaxe générale est :

```
if (condition)
 {
  Bloc d'instructions
 }
```

- Condition est une expression booléenne (0 pour faux et le reste pour vrai).

Instructions conditionnelles

La syntaxe générale est :

```
if (condition)
 {
  Bloc d'instructions
 }
```

- Condition est une expression booléenne (0 pour faux et le reste pour vrai).
- Si l'évaluation de la condition donne une valeur différente de 0, alors le bloc d'instructions est exécuté.

Instructions conditionnelles

La syntaxe générale est :

```
if (condition)
 {
  Bloc d'instructions
 }
```

- Condition est une expression booléenne (0 pour faux et le reste pour vrai).
- Si l'évaluation de la condition donne une valeur différente de 0, alors le bloc d'instructions est exécuté.
- Sinon le bloc d'instructions est ignoré.

Instructions conditionnelles

La syntaxe générale est :

```
if (condition)
 {
  Bloc d'instructions
 }
```

- Condition est une expression booléenne (0 pour faux et le reste pour vrai).
- Si l'évaluation de la condition donne une valeur différente de 0, alors le bloc d'instructions est exécuté.
- Sinon le bloc d'instructions est ignoré.
- Les accolades ne sont pas nécessaires si le bloc d'instructions contient une seule instruction.

Exercice

La séquence suivante :

```
if (a>0):
 {
  i+=1;
 }
```

contient une erreur de syntaxe. Pouvez-vous la corriger ?

Instructions conditionnelles

Remarque : Contrairement à Python, il n'y a pas de ":" après (condition).

Instructions conditionnelles

Remarque : Contrairement à Python, il n'y a pas de ":" après (condition).

De ce fait, l'écriture suivante :

```
if (a>0):
 {
  i+=1;
 }
```

Instructions conditionnelles

Remarque : Contrairement à Python, il n'y a pas de ":" après (condition).

De ce fait, l'écriture suivante :

```
if (a>0):
 {
  i+=1;
 }
```

générera une erreur de syntaxe!

Un autre exercice facile

Qu'affiche la séquence suivante d'instructions (la variable a est déclarée comme int) ?

```
a=-1;
if (a>0);
 {
  printf ("%d \n", a);
 }
```

Un autre exercice facile

La séquence suivante d'instructions (la variable a est déclarée comme int) :

```
a=-1;
if (a>0);
 {
  printf ("%d \n", a);
 }
```

affichera :

- -1
- A la compilation un "warning" est édité : if statement has empty body [-Wempty-body]

Encore un exercice facile

Un autre exercice facile

Qu'affiche la séquence suivante (la variable a est déclarée comme int) ?

```
a=-1;
if (a)
 {
  printf ("Impression \n");
 }
```

Un autre exercice facile

La séquence suivante (la variable a est déclarée comme int) :

```
a=-1;
if (a)
 {
  printf ("Impression \n");;
 }
```

affichera :

- Impression
- En effet, la condition (a) est considérée comme vraie.

Instructions conditionnelles avec alternatives

La syntaxe générale est :

```
if (condition)
 {
  Bloc 1 d'instructions
 }
else
 {
  Bloc 2 d'instructions
 }
```

Instructions conditionnelles avec alternatives

La syntaxe générale est :

```
if (condition)
 {
  Bloc 1 d'instructions
 }
else
 {
  Bloc 2 d'instructions
 }
```

- Si l'évaluation de la condition donne une valeur différente de 0, alors le bloc 1 d'instructions est exécuté.

Instructions conditionnelles avec alternatives

La syntaxe générale est :

```
if (condition)
 {
  Bloc 1 d'instructions
 }
else
 {
  Bloc 2 d'instructions
 }
```

- Si l'évaluation de la condition donne une valeur différente de 0, alors le bloc 1 d'instructions est exécuté.
- Sinon (l'évaluation de la condition donne 0) le bloc 2 d'instructions est exécuté.

Instructions conditionnelles imbriquées (dangling else)

Soit la séquence suivante :

```
int a, b;
scanf("%d %d", &a, &b);
if (a>0)
 if (b>0) printf ("%d \n", 3*a);
 else printf ("%d \n", 4*a);
```

Instructions conditionnelles imbriquées (dangling else)

Soit la séquence suivante :

```
int a, b;
scanf("%d %d", &a, &b);
if (a>0)
 if (b>0) printf ("%d \n", 3*a);
 else printf ("%d \n", 4*a);
```

- Qu'afficherait la suite d'instructions ci-dessus avec :
  - a=12, b=-2
  - a=12, b=2
  - a=-12, b=2
- A quel "if" (ou "condition") le "else" est rattaché?

Instructions conditionnelles imbriquées (dangling else)

```
int a, b;
scanf("%d %d", &a, &b);
if (a>0)
 if (b>0) printf ("%d \n", 3*a);
 else printf ("%d \n", 4*a);
```

Solutions

a	b	affichage
12	-2	48
12	2	36
-12	2	–

- Le ”else” est associé au ”if” le plus proche.
- Utiliser les blocs pour lever les ambiguïtés.

Instructions conditionnelles imbriquées (dangling else)

- Utiliser les blocs pour lever les ambiguïtés.

```
int a, b;
scanf("%d %d", &a, &b);
if (a>0)
{
 if (b>0) printf ("%d \n", 3*a);
}
else printf ("%d \n", 4*a);
```

L'instruction Switch

Exercice

Ecrire un programme C, qui :

- lit un entier positif, qui représente le code d'un département français, et
- affiche le nom du département associé.

L'instruction Switch

Exercice

Ecrire un programme C, qui :

- lit un entier positif, qui représente le code d'un département français, et
- affiche le nom du département associé.

Il est facile de l'écrire avec des "if-else" imbriqués.
Nous verrons que dans ce type d'exemple, il est plus agréable de le faire en utilisant l'instruction "switch".

L'instruction Switch

- Le switch est une instruction de sélection.
- Ecriture compacte des "if else" en cascade.
- Très utilisé lors des choix multiples.
  - Echelle à valeurs discrètes, typiquement des entiers et des caractères (qui sont en réalité codés comme des petits entiers).
  - Pour chaque valeur de cette échelle, une suite d'instructions est exécutée.

L'instruction Switch

L'instruction `switch` permet d'exécuter différents blocs de code selon la valeur d'une expression.

Syntaxe générale du switch

```
switch (expression)
{
```

Ici, `expression` est une variable ou une valeur qui sera comparée aux différentes constantes.

Différents cas possibles

```
switch (expression)
{
  case constante1:
        suite d'instructions 1;
        [break;]

    case constante2:
        suite d'instructions 2;
        [break;]
```

Chaque `case` définit un bloc d'instructions exécuté si `expression` correspond à la constante.

Ajouter d'autres cas

```
switch (expression)
{
  case constante1:
        suite d'instructions 1;
        [break;]

    case constante2:
        suite d'instructions 2;
        [break;]

   case constante3:
        suite d'instructions 3;
        [break;]

    ...
    case constanten:
        suite d'instructions n;
        [break;]
```

On peut ajouter autant de cas que nécessaire pour couvrir toutes les valeurs possibles.

Rajouter le cas par défaut

```
switch (expression)
{
  case constante1:
        suite d'instructions 1;
        [break;]
    case constante2:
        suite d'instructions 2;
        [break;]
   case constante3:
        suite d'instructions 3;
        [break;]
    ...
    case constanten:
        suite d'instructions n;
        [break;]
   default:
        suite d'instructions à exécuter par défaut;
}
```

Le bloc `default` est exécuté si aucune des constantes ne correspond à `expression`.

Le rôle du break

- Le mot-clé `break` permet de sortir du `switch` après un cas.
- Sans `break`, le programme continue d'exécuter les cas suivants (phénomène appelé "fall-through").

switch (en résumé)

- L'instruction switch évalue d'abord l'expression donnée entre parenthèse du switch.
- Ensuite il la compare, dans l'ordre, aux constantes données juste après le mot clef "case"
- Si une constante est égale à l'expression alors la suite d'instructions, associée à la constante appariée, est exécutée.
- Si aucune constante n'est égale à l'expression alors la suite d'instructions donnée dans le cas "default" est exécutée.
- L'instruction "break", quand elle est présente, permet de quitter le bloc de switch.

Switch : exemples des départements

```
#include <stdio.h>
int main (void)
{
    unsigned short departement;
    printf("Merci d'introduire le code département : ");
    scanf("%hu", &departement);
    switch (departement) {
        case 1:
            printf("Le nom du département est : Ain. \n");
            break;
        case 2:
            printf("Le nom du département est : Aisne. \n");
            break;
            .
            .
        case 95:
            printf("Le nom du département est : Val-d'Oise. \n");
            break;
        case 971:
            printf("Le nom du département est : Guadeloupe \n");
            break;
            .
            .
        case 974:
            printf("Le nom du département est : La Réunion \n");
            break;
        default:
            printf("Le numéro du département est inéxistant. \n");
            break;
}
return(0);
}
```

switch : remarques

- Lorsque l'instruction "break" n'est pas présente, le programme continue l'exécution des instructions des "cases" suivantes.
- La valeur donnée dans les "cases" doit-être une constante (typiquement de type int ou char).
- Les constantes de type float (et les chaînes de caractères) ne sont pas acceptées.

Opérateur du conditionnement ternaire

Autre forme d'instructions conditionnelles

On parle aussi d'opérateur ternaire. Sa syntaxe est :

condition ? expression 1 : expression 2;

- Condition est un expression booléenne. Si elle est vraie, l'expression 1 est exécutée sinon l'expression 2 est exécutée.
- Très pratique pour avoir des affectations conditionnelles:

  variable = condition ? expression 1 : expression 2;
- La variable aura la valeur de expression 1 si la condition est vraie, sinon elle aura la valeur de expression 2.

Autre forme d'instructions conditionnelles

Exercice

Ecrire un programme C, qui :

- lit un entier positif, qui représente une année civile, et
- affiche le nombre de jours associé au mois de février.

Le programme doit utiliser le conditionnement basé sur l'opérateur ternaire "?".

Remarque : Une année est dite bissextile si i) elle est divisible par 4 mais pas divisible par 100 ou ii) elle est divisible par 400.

Autre forme d'instructions conditionnelles

```
#include <stdio.h>
int main (void)
{
 unsigned short annee, fevrier;
 printf("Merci d'introduire une année civile ? ");
 scanf("%hu", &annee);
 fevrier = (((annee%4==0)&&(annee%100!=0))||(annee%400==0)) ? 29 : 28;
 printf ("Pour l'année %d, le mois de février a %d jours.\n",
         annee, fevrier);
 return(0);
}
```

Exercice: autre forme d'instructions conditionnelles

Compléter le programme suivant (en remplissant les .....) qui convertit une lettre majuscule en une lettre minuscule (et inversement).

```
#include <stdio.h>
int main (void)
{
    char car;
    printf ("\nMerci d'introduire une lettre majuscule ou minuscule: ");
    scanf ("%c", &car);
    printf ("\nLa conversion (majuscule vs minuscule) de %c donne %c.\n ",
     ......, .... ? ....... : ......);
}
```

Solution :autre forme d'instructions conditionnelles

```
#include <stdio.h>
int main (void)
{
    char car;
    printf ("\nMerci d'introduire une lettre majuscule ou minuscule: ");
    scanf ("%c", &car);
    printf ("\nLa conversion (majuscule vs minuscule) de %c donne %c.\n ",
     car, ((car>='A') && (car<='Z')) ? car -'A' + 'a' : car - 'a'+'A');
}
```

Les instructions d'itération

Les instructions d'itération

- La boucle
- La boucle do ... while
- La boucle for

La boucle while

La syntaxe est :

```
while (condition){
 Suite d'instructions du while
}
```

- Condition est une expression booléenne.

La boucle while

La syntaxe est :

```
while (condition){
 Suite d'instructions du while
}
```

- Condition est une expression booléenne.
- Tant que la condition est vraie (valeur de l'évaluation de la condition est différente de 0), la suite d'instructions du "while" est exécutée.

La boucle while

La syntaxe est :

```
while (condition){
 Suite d'instructions du while
}
```

- Condition est une expression booléenne.
- Tant que la condition est vraie (valeur de l'évaluation de la condition est différente de 0), la suite d'instructions du "while" est exécutée.
- Cette condition est ré-évaluée à la fin de chaque itération.

La boucle while

La syntaxe est :

```
while (condition){
 Suite d'instructions du while
}
```

- Condition est une expression booléenne.
- Tant que la condition est vraie (valeur de l'évaluation de la condition est différente de 0), la suite d'instructions du "while" est exécutée.
- Cette condition est ré-évaluée à la fin de chaque itération.
- On quitte la boucle dès que la condition est fausse (valeur égale à 0).

La boucle while

Exercice

Qu'affiche le programme C suivant :

```
#include <stdio.h>
int main (void)
{
  int i;
  i=-3;
  while (i){
    printf("La valeur de i est %d. \n", i);
    i++;
  }
return(0);
}
```

La boucle while

Exercice

Qu'affiche le programme C suivant :

```
#include <stdio.h>
int main (void)
{
  int i;
  i=-3;
  while (i){
    printf("La valeur de i est %d. \n", i);
    i++;
  }
return(0);
}
```

Solution

- Il affiche les trois lignes suivantes :
  - La valeur de i est -3.
  - La valeur de i est -2.
  - La valeur de i est -1.

La boucle do ... while

La syntaxe est :

```
do {
 Suite d'instructions du "do ... while"
} while (condition);
```

La boucle do ... while

La syntaxe est :

```
do {
 Suite d'instructions du "do ... while"
} while (condition);
```

- La différence entre une boucle ”while” et une boucle ”do ... while” est que :
  - le corps de la boucle ”do ... while” est exécuté au moins une fois.

La boucle do ... while

La syntaxe est :

```
do {
 Suite d'instructions du "do ... while"
} while (condition);
```

- A chaque exécution du corps de la boucle "do ... while", la condition est ré-évaluée.

La boucle do ... while

La syntaxe est :

```
do {
 Suite d'instructions du "do ... while"
} while (condition);
```

- A chaque exécution du corps de la boucle "do ... while", la condition est ré-évaluée.
- Si elle est vraie (valeur différente de 0), la suite d'instructions du "do ... while" est de nouveau exécutée.

La boucle do ... while

La syntaxe est :

```
do {
 Suite d'instructions du "do ... while"
} while (condition);
```

- A chaque exécution du corps de la boucle "do ... while", la condition est ré-évaluée.
- Si elle est vraie (valeur différente de 0), la suite d'instructions du "do ... while" est de nouveau exécutée.
- L'exécution de la boucle se termine dès que la condition est fausse.

La boucle do ... while

La syntaxe est :

```
do {
 Suite d'instructions du "do ... while"
} while (condition);
```

- A chaque exécution du corps de la boucle ”do ... while”, la condition est ré-évaluée.
- Si elle est vraie (valeur différente de 0), la suite d'instructions du ”do ... while” est de nouveau exécutée.
- L'exécution de la boucle se termine dès que la condition est fausse.
- N'oubliez pas le ”;” après le while.

La boucle for

La syntaxe est :

```
for ([initialisation]; [condition]; [expression]) {
 suite d'instructions du for.
}
```

- D'abord la séquence d'initialisation (qui est une expression) est exécutée (mais une et une seule fois).

La boucle for

```
for ([initialisation]; [condition]; [expression]) {
 suite d'instructions du for.
}
```

Ensuite, de manière répétitive, les étapes suivantes sont exécutées dans l'ordre :

- La condition (qui est une expression) est évaluée.

La boucle for

```
for ([initialisation]; [condition]; [expression]) {
 suite d'instructions du for.
}
```

Ensuite, de manière répétitive, les étapes suivantes sont exécutées dans l'ordre :

- La condition (qui est une expression) est évaluée.
- Si cette condition est fausse (exemple elle est égale à 0), la boucle for n'est pas exécutée.

La boucle for

```
for ([initialisation]; [condition]; [expression]) {
 suite d'instructions du for.
}
```

Ensuite, de manière répétitive, les étapes suivantes sont exécutées dans l'ordre :

- La condition (qui est une expression) est évaluée.
- Si cette condition est fausse (exemple égale à 0), la boucle for n'est pas exécutée.
- Si cette condition est vraie (valeur différente de 0),
  - la suite d'instructions du "for" est de nouveau exécutée, puis
  - le troisième champs de la boucle (une expression dite d'incrémentation de la boucle for) est exécutée (évaluée).

La boucle for

Typiquement, dans la syntaxe de la boucle for :

```
for ([initialisation]; [condition]; [expression]) {
 suite d'instructions du for.
}
```

- Le premier champs contient une initialisation d'une variable de parcours (soyons créatifs et appelons cette variable *i*).

La boucle for

Typiquement, dans la syntaxe de la boucle for :

```
for ([initialisation]; [condition]; [expression]) {
 suite d'instructions du for.
}
```

- Le premier champs contient une initialisation d'une variable de parcours (soyons créatifs et appelons cette variable *i*).
- Le deuxième champs contient la condition d'arrêt de la boucle.

La boucle for

Typiquement, dans la syntaxe de la boucle for :

```
for ([initialisation]; [condition]; [expression]) {
 suite d'instructions du for.
}
```

- Le premier champs contient une initialisation d'une variable de parcours (soyons créatifs et appelons cette variable *i*).
- Le deuxième champs contient la condition d'arrêt de la boucle.
- Le troisième champs contient une opération d'incrémentation, par exemple $i++$.

La boucle for

Exercice

Ecrire un programme C qui :

- lit un nombre strictement positif n et
- calcule la somme de tous les nombres pairs inférieurs à n.

La boucle for

```
#include <stdio.h>
int main(void){
    int n, i, somme=0;
    scanf("%d", &n);
    for (i=2; i<=n; i=i+2) {
        somme+=i;
    }
    printf ("\n La somme des nombres pairs inférieurs à %d est : %d. \n",
        n, somme);
    return(0);
}
```

La boucle for et ”do ... while”

Exercice

Ecrire un programme C qui :

- lit un nombre strictement positif n
  - Il est demandé d'utiliser la boucle
    do ... while
    jusqu'à ce que l'utilisateur rentre un nombre n positif.
- et calcule la somme de tous les nombres pairs inférieurs à n.

La boucle for

```
#include <stdio.h>
int main(void){
    int n, i, somme=0;
    do {
        printf ("\n Merci d'introduire un nombre strictement positif :");
        scanf("%d", &n);
    } while (n<=0);
}
```

- Cette boucle est exécutée au moins une fois et est répétée jusque' qu'un nombre strictement positif est introduit.
- Remarque : On fait l'hypothèse que l'utilisateur rentre des entiers.

La boucle for

```
#include <stdio.h>
int main(void){
    int n, i, somme=0;
    do {
        printf ("\n Merci d'introduire un nombre strictement positif :");
        scanf("%d", &n);
    } while (n<=0);
    for (i=2; i<=n; i=i+2) {
        somme+=i;
    }
    printf ("\n La somme des nombres pairs inférieurs à %d est : %d. \n",
        n, somme);
    return(0);
}
```

Question

Que se passe-t-il si par erreur un caractère est saisi au lieu d'un nombre ?

La boucle for : exemple

```
#include <stdio.h>
int main(void){
    int n, i, somme=0;
    do {
        printf ("\n Merci d'introduire un nombre strictement positif :");
        scanf("%d", &n);
    } while (n<=0);
    for (i=2; i<=n; i=i+2) {
        somme+=i;
    }
    printf ("\n La somme des nombres pairs inférieurs à %d est : %d. \n",
        n, somme);
    return(0);
}
```

Question

- Que se passe-t-il si par erreur un caractère est saisi au lieu d'un nombre ?
- Cela dépendra de la valeur par défaut de *n*.
  - Si elle est positive, c'est cette valeur qui sera utilisée dans la boucle.
  - Si elle est négative, la première boucle sera exécutée à l'infini.

La boucle for : exemple avec erreur de saisie

```
#include <stdio.h>
int main(void){
    int n=-1, i, somme=0;
    do {
        printf ("\n Merci d'introduire un nombre strictement positif :");
        scanf("%d", &n);
    } while (n<=0);
    for (i=2; i<=n; i=i+2) {
        somme+=i;
    }
    printf ("\n La somme des nombres pairs inférieurs à %d est : %d. \n",
        n, somme);
    return(0);
}
```

Remarque

- Avec ce programme, si par erreur un caractère est saisi au lieu d'un nombre alors la première boucle sera exécutée à l'infini.
- Plus précisément, l'affichage suivant :
  Merci d'introduire un nombre strictement positif :
  sera répété à l'infini.

La boucle for

Dans la boucle for :

```
for ([initialisation]; [condition]; [expression]) {
 suite d'instructions du for.
}
```

- Aucun des champs n'est obligatoire.
- L'absence du champs condition signifie que la condition est tout le temps vraie.

La boucle for

- En particulier, la boucle for suivante :

```
for (; ; ) {
 suite d'instructions du for.
}
```

  - représente une boucle infinie.
  - On sort de cette boucle grâce à l'instruction break (ou goto à la Fortran).
- L'instruction "continue" permet d'ignorer juste l'itération en cours.

Exercices

La somme des chiffres d'un nombre

- Ecrire un programme C :
  - qui lit un entier positif n et
  - qui affiche la somme des chiffres qui le composent.

  Par exemple, si $n = 2095$ la fonction retournera le nombre 16.

Rappels

- $a\%b$: donne le reste de la division de a par b.
- a / b : donne le résultat de la division entière de a par b lorsque a et b sont des variables de type entier.

La somme des chiffres d'un nombre

```c
#include <stdio.h>
int main(void)
 {
  unsigned int n, Res=0;
  scanf("%u", &n);
  while  (n!=0)
  {
  Res = Res + (n % 10);
  n = n / 10;
  }
  printf("%u", Res);
  return Res;
 }
```

Exercice

- Ecrire un programme C qui compte le nombre de caractères dans une seule ligne.

Exercice

- Reprendre l'exercice précédant en ne déclarant qu'une seule variable (de type unsigned char; un "petit" entier positif qui code le nombre de caractères lus).