Plan

Intérêt des structures de contrôle

Contrôler l'exécution des instructions du programme :
↪ notion d'algorithmes

- exécution conditionnelle (si alors sinon)
- exécuter plusieurs fois les mêmes instructions (boucle pour)
- exécuter des instructions tant que (boucle tant que)

Plan

if...else

Definition

```
if (exp)
    instr1
else
    instr2
```

- *exp* est une expression booléenne ou implicit. convertie en `_Bool` !
- $instr_1$ et $instr_2$ sont :
  - une instruction
  - un bloc d'instructions

Exemple: if ... else

Exemple

Déterminer si un entier est pair ou impair

```c
#include <stdio.h>

int main() {
   int a;
   a = 17;
   if (a%2 == 0)
      printf("%d est pair\n", a);
   else
      printf("%d est impair\n", a);
   return 0;
}
```

`if...else` (forme générale)

Definition

```
if (exp1)
   instr1
else if (exp2)
   instr2
else if (exp3)
   instr3
...
else
   instrn
```

Remarque : Le dernier `else` est facultatif, la forme la plus simple est donc :

```
if (exp)
   instr
```

Attention : L'indentation n'est pas interprétée, faire des blocs si besoin :

identique à

```
if (exp1)
   if (exp2)
      instr2
   else
      instr3
```

différent de

```
if (exp1)
   if (exp2)
      instr2
else
   instr3
```

```
if (exp1) {
   if (exp2)
      instr2
   else
      instr3
}
```

Exemple: if...else

Exemple

Déterminer le maximum de 3 entiers

```c
#include <stdio.h>
int main() {
   int a, b, c, max;
   scanf("%d%d%d",&a,&b,&c);
   if (a > b) {
      if(a > c)
          max = a;
      else
          max = c;
   }
   else if (c > b)
      max = c;
   else
      max = b;
   printf("le max est %d\n", max);
   return 0;
}
```

Conditionnelle à choix multiple : `switch`

Lorsqu'on a besoin de faire un choix parmi plus de 2 possibilités, on peut

- utiliser la conditionnelle `if .. else`

```
1  unsigned int a;
2  scanf("%d", &a);
3  if (a==1)
4      printf("a=1\n");
5  else if (a==2)
6      printf("a=2\n");
7  else if (a==3)
8      printf("a=3\n");
9  else
10     printf("a>3\n");
```

- faire une conditionnelle à choix multiple

Conditionnelle à choix multiple : `switch`

Choix multiple `switch`

```
switch (exp) {
  case cst1:
       instr1; break;
  ...
  case cstn:
       instrn; break;
  default:
       instrdefault;
}
```

- **exp** est une expression à valeur entière
- **cst1, ..., cstn** sont des constantes entières
- **instr1, ..., instrn, instrdefault** sont des séquences d'instructions terminées par un `break;`

Les instructions sont exécutées en fonction de la valeur **exp**

Si la valeur de **exp** est une des constantes **cst1,...,cstn** on exécute la suite d'intructions correspondante sinon on exécute **instrdefault**.

Exemple : switch

```
1  unsigned int a;
2  scanf("%d",&a);
3  switch (a){
4    case 1:
5        printf("a=1\n"); break;
6    case 2:
7        printf("a=2\n"); break;
8    case 3:
9        printf("a=3\n"); break;
10   default:
11       printf("a>3\n"); break;
12 }
```

L'affichage sera

- a=1 si la valeur de a est 1
- a=2 si la valeur de a est 2
- a=3 si la valeur de a est 3
- a>3 si la valeur de a est différent de 1, 2 ou 3

Exemple : `switch`

On peut grouper les cas, ne pas mettre le `break`, ni le `default`.

```c
#include <stdio.h>
int main() {
  unsigned char n;
  printf("Saisir un nombre de 1 a 10\n");
  scanf("%d",&n);
  switch (n) {
    case 2 :
    case 4 :
    case 8 :
      printf("c'est une puissance de 2\n");
    case 6 :
    case 10 :
      printf("il est pair\n");
      break :
    case 1 : case 3 : case 5 : case 7 : case 9 :
      printf("il est impair\n");
  }
  return 0;
}
```

Plan

Répétition d'instructions : la boucle pour

Intérêt : répéter un nombre de fois donné une même suite d'instructions.

Exemple

calculer la somme des entiers entre 1 et 10

```
s := 0 ;
pour i de 1 à 10 (par pas de 1) faire
    s := s+i ;
fin pour ;
```

En C, les boucles **pour** sont effectuées avec l'instruction `for`

L'instruction `for`

Definition

for (`exp1 ; exp2 ; exp3`)
`instr`

- `exp1` est une expression quelconque évaluée une seule fois au début de la boucle. Elle permet d'initialiser l'indice de boucle.
- `exp2` est une expression booléenne évaluée au début de chaque tour de boucle. Elle définit la poursuite/l'arrêt de la boucle.
- `exp3` est une expression quelconque évaluée à la fin de chaque tour de boucle. Elle permet d'incrémenter l'indice de boucle.

- `instr` est une instruction ou un bloc d'instructions

L'instruction `for`

Definition

for (`exp1 ; exp2 ; exp3`)
`instr`

- `exp1` est une expression quelconque évaluée une seule fois au début de la boucle (souvent une affectation)

On se sert de exp1 pour initialiser la variable de boucle.

Exemple

exp1 est remplacée par `int i=1` (ou `i=1` si `i` est déjà déclaré).

L'instruction `for`

Definition

for (`exp1 ; exp2 ; exp3`)
`instr`

- `exp2` est une expression booléenne

Si `exp2` est :

- vrai : la boucle `for` continue et on exécute `instr`
- faux : on sort de la boucle `for` sans exécuter `instr`

Exemple

exp2 est remplacée par `i<11` ou `i<=10`

L'instruction `for`

Definition

for (`exp1 ; exp2 ; exp3`)
`instr`

- `exp3` est une expression quelconque évaluée à chaque tour de boucle (après exécution de `instr` mais avant réévaluation de `exp2`) ;
- on passe à la valeur suivante de l'indice de boucle.

Exemple

exp3 est remplacée par `i=i+1` ou `i++`

L'instruction `for` : schéma d'exécution

Definition

for (`exp1 ; exp2 ; exp3`)
`instr`

1. évaluation de `exp1`
2. évaluation de `exp2` :
   - si `exp2` est faux `sortie de boucle`
   - si `exp2` est vrai :
     3. évaluation de `instr`
     4. évaluation de `exp3`

   on recommence en 2

L'instruction `for` : exemple 1

Exemple

calculer la somme des entiers entre 1 et 10

```
1  #include <stdio.h>
2
3  int main() {
4     int i,s;
5     s = 0;
6     for (i=1 ; i<11 ; i++) // i++ est equivalent a i=i+1
7        s = s+i;
8     printf("La somme vaut %d\n",s);
9     return 0;
10 }
```

ce programme affiche : La somme vaut 55

Du pour d'algorithmique au for

pour haut

pour i de *a* à *b* par pas de *k* faire
    instr
fin pour ;

```
1 for(int i=a ; i<=b ; i=i+k) {
2     instr
3 }
```

pour bas

pour i de *a* bas *a* par pas de *k* faire
    instr
fin pour ;

```
1 for(int i=b ; i>=a ; i=i-k) {
2     instr
3 }
```

L'instruction `for` : exemple de pour descendant

Exemple

afficher les entiers entre 10 et 1 qui sont multiples de 2 ou de 3

```c
#include <stdio.h>

int main(){
   for (int i=10;i>0;i--) {
      if (i%2==0 || i%3==0)
         printf("%d ",i);
   }
   printf("\n");
   return 0;
}
```

Ce programme affiche : 10 9 8 6 4 3 2

Remarque : on peut déclarer la variable de boucle dans `exp1` ; dans ce cas, on ne peut plus y accéder après la boucle.

L'instruction `for` : ce qu'il ne faut pas faire

Attention aux boucles qui ne se terminent jamais ! ! !
les boucles infinies ...

```
1  int i,s;
2  s=0;
3  for (i=1 ; i<11 ; s=s+i)
4     ...
```

→ la variable de boucle n'est pas incrémentée

```
1  int i;
2  for (i=1 ; i!=10 ; i+=2)
3     ...
```

→ la condition d'arrêt de la boucle n'est jamais atteinte

Plan

Répétition d'instructions : la boucle tant que

Intérêt : répéter une instruction tant qu'une condition est vérifiée

Exemple

calculer la somme des entiers entre 1 et 10

```
s := 0 ;
i := 1 ;
tant que i<11 faire
    s := s+i ;
    i := i+1 ;
fin tant que ;
```

En C, les boucles **tant que** se font avec l'instruction `while`

L'instruction `while`

Definition

while (`exp`)
  `instr`

- `exp` est une expression booléenne contrôlant la poursuite de la boucle
- `instr` est une instruction ou un bloc d'instructions qui doit agir sur la valeur de `exp` pour qu'une sortie de boucle soit possible

L'instruction `while` : schéma d'exécution

Definition

while (`exp`)
  `instr`

1. évaluation de `exp` :
   - si `exp` est faux `sortie de boucle`
   - si `exp` est vrai :
     2. évaluation de `instr`

   on recommence en 1

L'instruction `while` : exemple 1 - simulation d'un pour

Exemple

calculer la somme des entiers entre 1 et 10

```
1  #include <stdio.h>
2
3  int main(){
4     int i,s;
5     s = 0;
6     i = 1;
7     while (i<11) {
8        s = s+i;
9        i = i+1;
10    }
11    printf("la somme est %d\n",s);
12    return 0;
13 }
```

ce programme affiche : la somme est 55

L'instruction `while` : ex. avec nb. d'itérations non connu

Exemple

calcul de la plus petite puissance de 2 supérieure à un entier

```
1  #include <stdio.h>
2
3  int main(){
4     int a, p = 1;
5     printf("Tapez un entier");
6     scanf("%d",&a);
7     while (p<a)
8        p = 2*p;
9     printf("%d est la plus petite puiss. de 2 sup. a %d",p,a);
10    return 0;
11 }
```

Pour 27, il affiche : 32 est la plus petite puiss. de 2 sup. a 27.

Mais il peut ne pas s'arrêter si la plus petite puissance de 2 supérieure à l'entier saisi est supérieure au plus grand entier codable !

Plan

L'instruction `do while`

Parfois, il est souhaitable d'exécuter le corps de boucle avant la condition de boucle (`instr` avant `exp`).

Dans ce cas, on peut utiliser l'instruction `do ... while`

Definition

```
do {
    instr
} while (exp);
```

- `instr` et `exp` sont identiques à ceux utilisés dans la boucle `while` classique

L'instruction `do{} while` : schéma d'exécution

Definition

```
do {
   instr
} while (exp);
```

1. évaluation de `instr`
2. évaluation de `exp` :
   - si `exp` est faux `sortie de boucle`
   - si `exp` est vrai on recommence en 1

L'instruction do{} while : exemple

Exemple

Le jeu du trouve mon code secret !

```
1  #include <stdio.h>
2
3  int main() {
4      int secret = 135;
5      int rep;
6      do {
7          printf("Ta proposition : ");
8          scanf("%d",&rep);
9          if(rep < secret) printf("\tplus grand !\n");
10         if(rep > secret)printf("\tplus petit !\n");
11     }
12     while (rep!=secret);
13     printf("Bravo, tu as trouve !\n");
14     return 0;
15 }
```

Plan

Structures de contrôle : erreur classique

L'erreur classique avec les structures de contrôle est l'oubli d'accolades pour définir un bloc d'instructions :

```
1  ...
2     if (a > b){
3         max = a;
4         min = b;
5     }
6     else
7         max = b;
8         min = a; // ATTENTION: n'appartient pas au else
9
10    printf("le min est %d, le max est %d\n",min, max);
11 ...
```

Structures de contrôle : erreur classique

L'erreur classique avec les structures de contrôle est l'oubli d'accolades pour définir un bloc d'instructions :

```
1  ...
2     if (a > b)
3         max = a;
4         min = b;
5     else {
6         max = b;
7         min = a; // grace aux accolades appartient au else
8     }
9     printf("le min est %d, le max est %d\n",min, max);
10 ...
```

mais l'oubli des accolades sur le bloc du `if` génère une erreur de compilation car le `else` se retrouve isolé.

Structures de contrôle : erreur classique

Avec les accolades, tout va bien :

```
1  if (a > b) {
2      max = a;
3      min = b;
4  }
5  else {
6      max = b;
7      min = a;
8  }
9  printf("le min est %d, le max est %d\n",min, max);
```

On peut toujours mettre des accolades même quand on a une seule instruction à faire :

```
1  if (a < 0) {
2      a = -a;
3  }
```

Structures de contrôle : erreur classique

L'erreur classique avec les structures de contrôle est l'oubli d'accolades pour définir un bloc d'instruction :

```
1  ...
2    int i,s;
3    s = 0;
4    i = 1;
5    while (i<11)
6        s = s+i;
7        i++; // n'appartient pas a la boucle
8
9    printf("la somme est %d\n",s);
10 ...
```

Le programme ne s'arrête pas $\Rightarrow$ boucle infinie !

Structures de contrôle : erreur classique

L'erreur classique avec les structures de contrôle est l'oubli d'accolades pour définir un bloc d'instruction :

```
1  ...
2     int i,s;
3     s = 0;
4     i = 1;
5     while (i<11) {
6         s = s+i;
7         i++; // grace aux accolades appartient a la boucle
8     }
9     printf("la somme est %d\n",s);
10 ...
```

Récapitulatif

- les variables ont un type (ex : int, float...)
- on peut calculer grâce aux opérateurs (+,*,%,...)
- on modifie un programme par des affectations (ex : `a=6`)
- faiblement typé → conversions de type implicite
- on affiche les valeurs à l'écran avec `printf`
- on saisit les valeurs au clavier avec `scanf`
- on peut écrire des algorithmes avec les structures de contrôle classiques : `if else`, `for`, `while`, `do...while`, `switch`