Information, Calcul et Communication

Composante Pratique: Programmation C++

MOOC semaine 3: boucles

Fiche: les boucles conditionnelles et l'itération

Une boucle particulière : la lecture

Branchements inconditionnels avec break et continue

Bonnes pratiques en matière de portée d'une variable

Bonnes pratiques en matière de calculs numériques

Boucles conditionnelles :

```
while (condition)
   une seule instruction
   ou un bloc
```

1) Utiliser un bloc {…} pour contrôler plus d'une instruction

- l'indentation ne suffit PAS !!
- Si on dispose d'espace, toujours utiliser un bloc même avec une seule instruction contrôlée

2) Comme pour if et else, ne PAS ajouter de point virgule en fin de ligne du while ou du for

```
do
   une seule instruction
   ou un bloc
while (condition);
```

3) Pour do-while, NE PAS OUBLIER le point virgule en fin de ligne du while

Itérations :

```
for (expr_init ; condition ; expr_post)
   une seule instruction
   ou un bloc
```

4) quand on connaît a priori le nombre de passages, privilégier l'itération for plutôt que while et do-while

```
for( init ; condition ; post )
    une seule instruction contrôlée ;
```

Les expressions **`init`** et **`post`** peuvent contenir plusieurs sous-expressions séparées par une virgule.

```
for (int i(0),j(1) ; i<10 ; ++i, j*=2)
{
    cout << j << endl;
}
```

Quel est le but de ce code ?

Sem4_MOOC3_SpeakUp3: what is displayed by this code

```
…
double t(1.0);
double delta_t(0.1);
double fin(10.);

for(double x(t) ; x != fin ; x+= delta_t)
{
      cout << x << endl;
}

cout << "the end" << endl;
…
```

A. Toutes les valeurs de 1.0 à 9.9 par pas de 0.1 et affiche **the end**

B. Entre dans une boucle d'affichage infinie

C. Seulement la valeur 10. puis affiche **the end**

D. N'entre pas dans la boucle et affiche seulement **the end**

Une boucle «système» particulière: la lecture

La suite de **caractères alphanumériques** qui est fournie au clavier et validée avec *Enter* est *mémorisée* par le système dans un tableau de caractères (type **char**).

Si le programme a besoin de lire plusieurs valeurs, elles doivent être séparées par un **séparateur** : espace, tabulation, passage à la ligne.

Restriction selon le type: le programme extrait **seulement les caractères autorisés** pour le type de la variable lue ; il stoppe dès le premier caractère non-autorisé ou dès le premier séparateur:

- *Types entiers*: seulement le signe et les dix chiffres
- *Types à virgule flottante*: signe, les dix chiffres et le point décimal
- ***bool***: seulement 0 ou 1
- ***char***: tout sauf les séparateurs ; utiliser **cin.get(c)** pour lire tout

Une boucle «système» particulière: la lecture (2)

Exemple: code de **lecture_entier.cc**

1) Vérifier comment se comporte le programme lorsqu'on fournit des caractères autorisés pour les types **double** et **int**.

Ex: `3.7 99 0.2`

2) Constater que la lecture d'un entier n'est **PAS** faite en prenant la partie entière d'un nombre à virgule. **Ex: `3.7 9.9 0.2`**

Comment le programme consomme-t-il les caractères fournis en entrée en cas de valeur non-autorisée pour l'entier **b**?

```
…
double a(0.);
cin  >> a;
cout << a << endl;

int b(0);
cin  >> b;
cout << b;

double c(0.);
cin  >> c;
cout << c << endl;
…
```

```
3.7 9.9 0.2
```

Complément: branchements inconditionnels: `break` et `continue`

`break` permet de **quitter** la boucle dans laquelle il se trouve

En cas de boucles imbriquées : quitte seulement la boucle interne

Analogue à l'instruction «**Sortir**» utilisée pour décrire un algorithme

Ne pas en abuser **car le flot de contrôle devient moins lisible**

⇒ Augmente le nombre de points de sortie de la boucle
au lieu d'un avoir un seul dans le while ou le for

⇒ Il est préférable d'enrichir l'unique condition de la boucle

`continue` **ne quitte PAS la boucle ;**

effectue seulement un saut à la fin des instructions contrôlées

ex: code
break_continue.cc

Quitte la boucle
dès que i == v

Filtre les valeurs
paires de i

```
#include <iostream>
using namespace std;

int main()
{
	int v(0), total(0);
	cin >> v;

	for(int i(1) ; i != 10 ; ++i)
	{
		if(i%2 ==0) continue; // passe directement à l'incrémentation

		if(i == v) break;

		total += i;
	}

	cout << "total = " << total << endl;

	return 0;
}
```

Bonnes pratiques en matière de portée des variables

1) Déclarer une variable au plus proche de son utilisation

2) **Déclarer une variable le plus localement possible**
⇒ seulement dans le bloc où elle est utile (bloc d'instructions contrôlées)

3) ATTENTION: **déclarer une variable globale en dehors de tout bloc**
=> EST UNE TRES MAUVAISE PRATIQUE (interdite en projet)

4) Règle de masquage des noms de variable: c'est «légal» de déclarer deux variables de même nom dans des blocs imbriqués. Dans ce cas, la variable du bloc le plus interne masque la variable du bloc externe

(ex: prédire l'affichage du code **masquage.cc**)

```
#include <iostream>
using namespace std;

// variable globale => this is pretty bad
int x(100);

int main()
{
	int x(33); // variable locale à main()

	for(int i(1) ; i != 6 ; ++i)
	{
		int x(i%3); // variable locale au bloc

		cout << "A = " << x << endl;
	}
	cout << "B = " << x << endl;

	return 0;
}
```