Introduction à l'informatique
Compléments: détection et correction des erreurs
INFO0205-1

X. Baumans
(xavier.baumans@ulg.ac.be)

Contenu

- Erreurs et débogage d'un programme
  $\rightarrow$ comment repérer et corriger les erreurs d'un programme

  - Erreurs syntaxiques
  - Erreurs sémantiques
  - Utilisation du débogueur

Erreurs et débogages

Lors de l'implémentation d'un code, différents types d'erreurs peuvent survenir. Il y a les

- erreurs **syntaxiques** : elles sont détectées par le compilateur (erreurs de compilation) car elles ne respectent pas la syntaxe (manière d'écrire les instructions) prévue par le langage $\rightarrow$ Ce sont les erreurs de déclaration, de notations des instructions (points-virgules),...
- erreurs **sémantiques** : ces erreurs ne sont **pas** détectées par le compilateur ! Elles correspondent à des erreurs **logiques** dans la signification de la suite des instructions (ce que fait le programme). $\rightarrow$ Le compilateur ne connaît pas l'objectif du programme et ne peut donc pas les détecter. Deux possibilités alors :
  - Le programme s'arrête avec un message d'erreur du type "Segmentation Fault" ;
  - Le programme ne fait pas ce qu'il devrait (**ATTENTION**).

Erreurs courantes

- Diviser deux entiers :
  quand on divise deux **int** l'un par l'autre, on obtient un nombre entier, même si l'on place le résultat dans un **double**.
  Pour obtenir un résultat non-entier, il faut écrire

```
1 int a;
2 int b;
3 double c = (double)a/b;
```

  → erreur sémantique

- Expression de comparaison :
  utiliser = (affectation) à la place de == (comparaison)
  → erreur sémantique

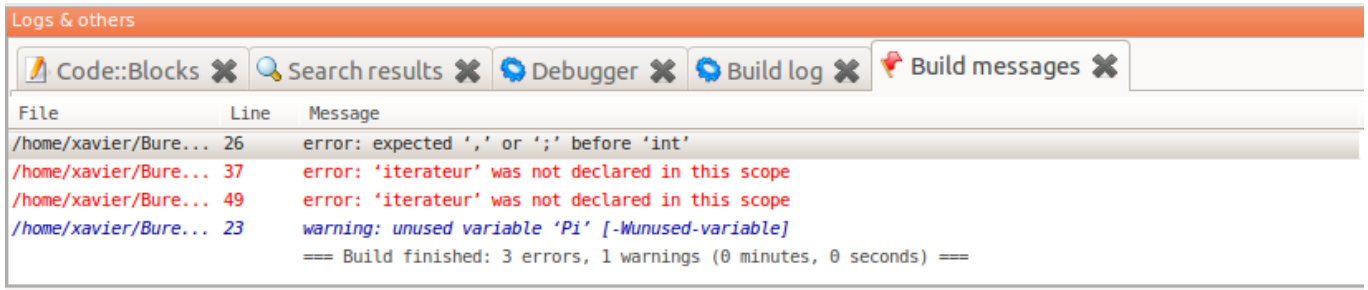
- Oublier un point-virgule :
  toutes les instructions se terminent par un point-virgule
  → erreur syntaxique

Erreurs syntaxiques

Lorsque des erreurs **syntaxiques** sont présentes, le compilateur

- les énumère en fournissant le numéro de la ligne correspondante
- fournit également un bref descriptif du problème rencontré

Il est important de commencer par lire et résoudre les **premières erreurs**, car les suivantes peuvent être provoquées par celles qui les précèdent.



Le compilateur fournit également des messages d'alerte ("warnings"). Ceux-ci n'empêche pas la compilation mais mettent en lumière des pratiques non recommandées. Il est donc fortement conseillé de les corriger car ils pourraient mener à des erreurs.

Erreurs sémantiques et débogage

Les erreurs sémantiques ne sont pas directement visibles et sont donc **difficiles à détecter**. Personne n'est à l'abris d'en commettre, l'important est de pouvoir les repérer et les corriger. Conseils :

- Approcher petit-à-petit de l'erreur en vérifiant d'abord les grandes étapes du programme puis en réduisant progressivement la zone de vérification ;
- Tester le programme avec des valeurs simples pour lesquelles la solution est connue. De cette manière, il est possible de contrôler efficacement les affichages des variables car les valeurs attendues sont connues.
- Structurer son code dès le début, avec des **noms de variables explicites** et des **commentaires**, ce qui permet de contrôler aisément le programme étape par étape.

Déboguer un programme

Deux manières principales de procéder :

1. Afficher les valeurs de certaines variables dans la console

   Simple et rapide pour contrôler le déroulement de certains points clefs

   - Etablir un affichage clair et précis
   - Ne pas multiplier les informations inutiles
   - Technique limitée à de petits programmes et aux erreurs simples

2. Utiliser le débogueur

   Code : :Blocks dispose d'une interface intégrée avec un débogueur. Il permet :

   - d'exécuter le code source étape par étape
   - de contrôler directement les valeurs des variables du programme à chaque instant
   - d'interrompre le programme à un moment précis pour vérifier son état

Exemple de détection par affichage

- Erreurs de comparaison : afficher la variable avant et après le test car celle-ci ne doit pas être modifiée par le test de comparaison (==) mais en cas d'utilisation d'un mauvais opérateur (=), elle le sera ;
- Erreur de type de variable : le plus souvent un nombre à virgule est attendu mais le programme retourne un nombre entier ou zéro.

  Par exemple : 1/2=0, 5/2 = 2 mais 5./2 = 2.5
- Les boucles infinies : afficher la condition de contrôle de la boucle (et les éléments qui la constituent si nécessaire) à chaque itération. De cette manière, il est possible de rechercher à quel moment la condition aurait dû faire aboutir la boucle.
- Les boucles jamais exécutées : vérifier si la boucle est exécutée en ajoutant l'affichage d'un message à chaque itération de la boucle. Si le message n'apparaît jamais, c'est que la boucle n'est pas exécutée.

Utilisation du débogueur avec Code : :Blocks (1/3)

Pour déboguer un programme :

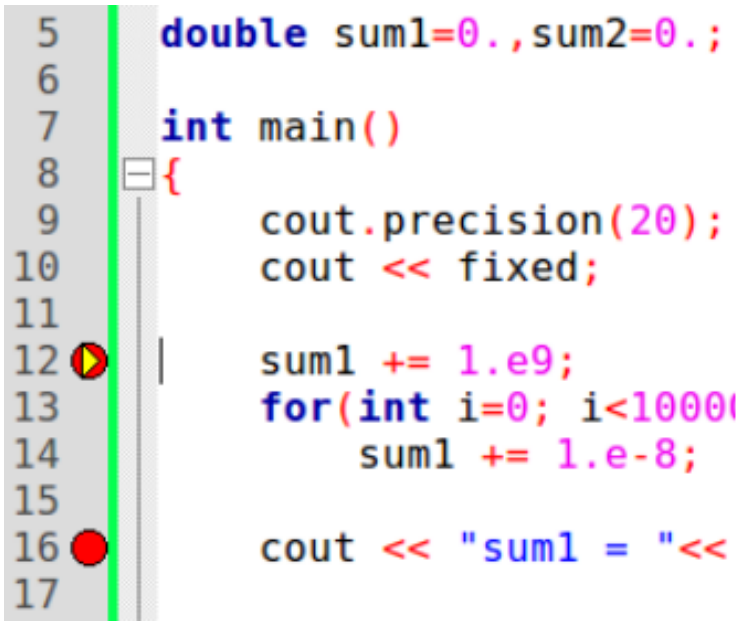
- Placer le projet en configuration "Debug"

- Compiler le programme normalement, mais ne pas l'exécuter de la manière habituelle. la barre d'outil *Debug* sera utilisée à la place :

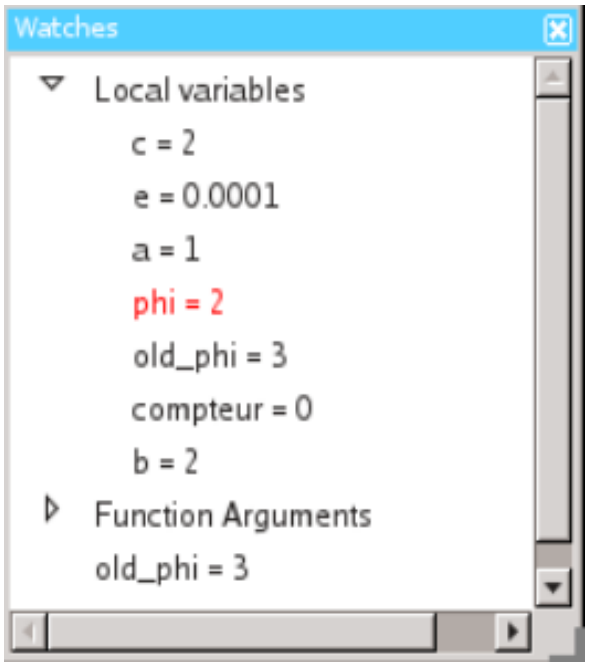
  - Le premier bouton sert à exécuter le programme jusqu'au point d'arrêt suivant
  - Le deuxième exécute le programme jusqu'à la position actuelle du curseur
  - Le troisième exécute la ligne de code suivante ;
  - etc...

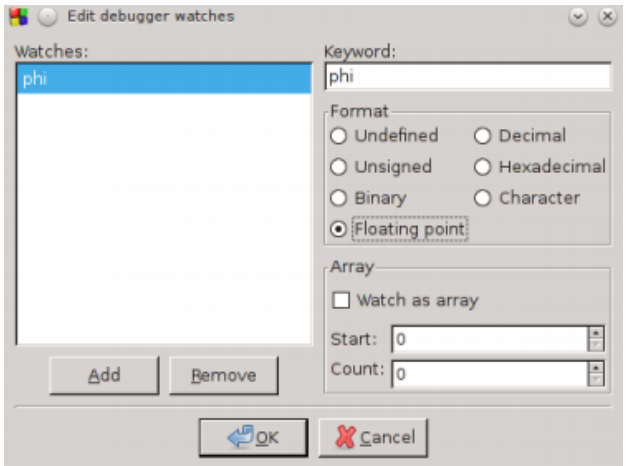
Utilisation du débogueur avec Code : :Blocks (2/3)

- Pour ajouter un *point d'arrêt*, il suffit de cliquer dans la gouttière (entre le code et le numéro de la ligne) à côté de la ligne souhaitée ;
- En cours de débogage, Code : :Blocks indique la ligne en cours d'exécution par une petite flèche jaune.



Utilisation du débogueur avec Code : :Blocks (3/3)





- La fenêtre *Watches* permet d'afficher les valeurs actuelles de toutes les variables locales.
- Lorsque la valeur d'une variable a été modifiée lors de la dernière instruction, elle s'affiche en rouge.
- Il est possible de suivre la valeur d'une variable spécifique non affichée par défaut en ajoutant un 'espion'. Cela signifie que Code : :Blocks affichera constamment sa valeur dans la fenêtre *Watches*.

Exercices

1. Débogage d'un programme :
   - Charger le fichier main.cpp fourni ;
   - L'inclure dans un nouveau projet ;
   - Lire et s'attacher à comprendre ce que le programme tente de faire ;
   - Corriger les erreurs à l'aide des informations données par le compilateur et le débogueur.

Correction exercice 1 (débogage)

```
21  int main()
22
23      double Pi = 0; // contient la valeur actuelle de Pi
24      double terme = 0; // Le terme de la série qu'on va ajouter
25      int seuil = 0 // Si le terme actuel est plus petit que seuil, fin du programme
26      int iterateur = 0; // Compteur du nombre d'itérations
27
28      cout << "Calcul de Pi" << endl;
29
30      cout << "Quel seuil voulez-vous choisir ?";
31      cin >> seuil;
32      cout << "Début du calcul de pi avec une precision de " << seuil << endl;
33      cout.precision(15);
34
35      do
36      {
37          terme = 4/(iterateur*2+1); // Calcul du terme actuel: le numérateur vaut toujours 4 et le dénominateur
38          // prend les valeurs 1, 3, 5, 7, 9, 11,...
39          if(iterateur%2 = 1) //Si iterateur est pair, alors 2*iterateur+1 = 1, 5, 9,... et il faut ajouter le terme
40              Pi += terme;
41          else // sinon, il faut le soustraire
42              Pi -= terme;
43
44
45          iterateur++; // on incrémente la valeur de l'itérateur à chaque passage dans la boucle
46
47      }while(terme > seuil); // on calcule tant que le terme est plus grand que le seuil fixé
48
49      cout << "Pi = " << Pi << " (" << iterateur << " iterations)" << endl;
50
51      return 0;
52  }
```

Correction exercice 1 (débogage)

```
21  int main()
22  {
23      double Pi = 0; // contient la valeur actuelle de Pi
24      double terme = 0; // Le terme de la série qu'on va ajouter
25      int seuil = 0 // Si le terme actuel est plus petit que seuil, fin du programme
26      int iterateur = 0; // Compteur du nombre d'itérations
27
28      cout << "Calcul de Pi" << endl;
29
30      cout << "Quel seuil voulez-vous choisir ?";
31      cin >> seuil;
32      cout << "Début du calcul de pi avec une precision de " << seuil << endl;
33      cout.precision(15);
34
35      do
36      {
37          terme = 4/(iterateur*2+1); // Calcul du terme actuel: le numérateur vaut toujours 4 et le dénominateur
38          // prend les valeurs 1, 3, 5, 7, 9, 11,...
39          if(iterateur%2 = 1) //Si iterateur est pair, alors 2*iterateur+1 = 1, 5, 9,... et il faut ajouter le terme
40              Pi += terme;
41          else // sinon, il faut le soustraire
42              Pi -= terme;
43
44
45          iterateur++; // on incrémente la valeur de l'itérateur à chaque passage dans la boucle
46
47      }while(terme > seuil); // on calcule tant que le terme est plus grand que le seuil fixé
48
49      cout << "Pi = " << Pi << " (" << iterateur << " iterations)" << endl;
50
51      return 0;
52  }
```

Correction exercice 1 (débogage)

```
21  int main()
22  {
23      double Pi = 0; // contient la valeur actuelle de Pi
24      double terme = 0; // Le terme de la série qu'on va ajouter
25      int seuil = 0 // Si le terme actuel est plus petit que seuil, fin du programme
26      int iterateur = 0; // Compteur du nombre d'itérations
27
28      cout << "Calcul de Pi" << endl;
29
30      cout << "Quel seuil voulez-vous choisir ?";
31      cin >> seuil;
32      cout << "Début du calcul de pi avec une precision de " << seuil << endl;
33      cout.precision(15);
34
35      do
36      {
37          terme = 4/(iterateur*2+1); // Calcul du terme actuel: le numérateur vaut toujours 4 et le dénominateur
38          // prend les valeurs 1, 3, 5, 7, 9, 11,...
39          if(iterateur%2 = 1) //Si iterateur est pair, alors 2*iterateur+1 = 1, 5, 9,... et il faut ajouter le terme
40              Pi += terme;
41          else // sinon, il faut le soustraire
42              Pi -= terme;
43
44
45          iterateur++; // on incrémente la valeur de l'itérateur à chaque passage dans la boucle
46
47      }while(terme > seuil); // on calcule tant que le terme est plus grand que le seuil fixé
48
49      cout << "Pi = " << Pi << " (" << iterateur << " iterations)" << endl;
50
51      return 0;
52  }
```

point-virgule !!!

Correction exercice 1 (débogage)

```
21  int main()
22  {
23      double Pi = 0; // contient la valeur actuelle de Pi
24      double terme = 0; // Le terme de la série qu'on va ajouter
25      int seuil = 0 // Si le terme actuel est plus petit que seuil, fin du programme
26      int iterateur = 0; // Compteur du nombre d'itérations
27
28      cout << "Calcul de Pi" << endl;
29
30      cout << "Quel seuil voulez-vous choisir ?";
31      cin >> seuil;
32      cout << "Début du calcul de pi avec une precision de " << seuil << endl;
33      cout.precision(15);
34
35      do
36      {
37          terme = 4/(iterateur*2+1); // Calcul du terme actuel: le numérateur vaut toujours 4 et le dénominateur
38          // prend les valeurs 1, 3, 5, 7, 9, 11,...
39          if(iterateur%2 = 1) //Si iterateur est pair, alors 2*iterateur+1 = 1, 5, 9,... et il faut ajouter le terme
40              Pi += terme;
41          else // sinon, il faut le soustraire
42              Pi -= terme;
43
44
45          iterateur++; // on incrémente la valeur de l'itérateur à chaque passage dans la boucle
46
47      }while(terme > seuil); // on calcule tant que le terme est plus grand que le seuil fixé
48
49      cout << "Pi = " << Pi << " (" << iterateur << " iterations)" << endl;
50
51      return 0;
52  }
```

point-virgule !!!

== (opérateur de comparaison !)

Correction exercice 1 (débogage)

```
21  int main()
22  {
23      double Pi = 0; // contient la valeur actuelle de Pi
24      double terme = 0; // Le terme de la série qu'on va ajouter
25      int seuil = 0 // Si le terme actuel est plus petit que seuil, fin du programme
26      int iterateur = 0; // Compteur du nombre d'itérations
27
28      cout << "Calcul de Pi" << endl;
29
30      cout << "Quel seuil voulez-vous choisir ?";
31      cin >> seuil;
32      cout << "Début du calcul de pi avec une precision de " << seuil << endl;
33      cout.precision(15);
34
35      do
36      {
37          terme = 4/(iterateur*2+1); // Calcul du terme actuel: le numérateur vaut toujours 4 et le dénominateur
38          // prend les valeurs 1, 3, 5, 7, 9, 11,...
39          if(iterateur%2 = 1) //Si iterateur est pair, alors 2*iterateur+1 = 1, 5, 9,... et il faut ajouter le terme
40              Pi += terme;
41          else // sinon, il faut le soustraire
42              Pi -= terme;
43
44
45          iterateur++; // on incrémente la valeur de l'itérateur à chaque passage dans la boucle
46
47      }while(terme > seuil); // on calcule tant que le terme est plus grand que le seuil fixé
48
49      cout << "Pi = " << Pi << " (" << iterateur << " iterations)" << endl;
50
51      return 0;
52  }
```

double !!!

point-virgule !!!

== (opérateur de comparaison !)

Correction exercice 1 (débogage)

```
21  int main()
22  {
23      double Pi = 0; // contient la valeur actuelle de Pi
24      double terme = 0; // Le terme de la série qu'on va ajouter
25      int seuil = 0 // Si le terme actuel est plus petit que seuil, fin du programme
26      int iterateur = 0; // Compteur du nombre d'itérations
27
28      cout << "Calcul de Pi" << endl;
29
30      cout << "Quel seuil voulez-vous choisir ?";
31      cin >> seuil;
32      cout << "Début du calcul de pi avec une precision de " << seuil << endl;
33      cout.precision(15);
34
35      do
36      {
37          terme = 4/(iterateur*2+1); // Calcul du terme actuel: le numérateur vaut toujours 4 et le dénominateur
38          // prend les valeurs 1, 3, 5, 7, 9, 11,...
39          if(iterateur%2 = 1) //Si iterateur est pair, alors 2*iterateur+1 = 1, 5, 9,... et il faut ajouter le terme
40              Pi += terme;
41          else // sinon, il faut le soustraire
42              Pi -= terme;
43
44
45          iterateur++; // on incrémente la valeur de l'itérateur à chaque passage dans la boucle
46
47      }while(terme > seuil); // on calcule tant que le terme est plus grand que le seuil fixé
48
49      cout << "Pi = " << Pi << " (" << iterateur << " iterations)" << endl;
50
51      return 0;
52  }
```

double !!!

point-virgule !!!

== (opérateur de comparaison !) 0 (le reste de la division par 2 doit être nul si on veut que le nombre soit pair!)

Correction exercice 1 (débogage)

```
21  int main()
22  {
23      double Pi = 0; // contient la valeur actuelle de Pi
24      double terme = 0; // Le terme de la série qu'on va ajouter
25      int seuil = 0 // Si le terme actuel est plus petit que seuil, fin du programme
26      int iterateur = 0; // Compteur du nombre d'itérations
27
28      cout << "Calcul de Pi" << endl;
29
30      cout << "Quel seuil voulez-vous choisir ?";
31      cin >> seuil;
32      cout << "Début du calcul de pi avec une precision de " << seuil << endl;
33      cout.precision(15);
34
35      do
36      {
37          terme = 4/(iterateur*2+1); // Calcul du terme actuel: le numérateur vaut toujours 4 et le dénominateur
38          // prend les valeurs 1, 3, 5, 7, 9, 11,...
39          if(iterateur%2 = 1) //Si iterateur est pair, alors 2*iterateur+1 = 1, 5, 9,... et il faut ajouter le terme
40              Pi += terme;
41          else // sinon, il faut le soustraire
42              Pi -= terme;
43
44
45          iterateur++; // on incrémente la valeur de l'itérateur à chaque passage dans la boucle
46
47      }while(terme > seuil); // on calcule tant que le terme est plus grand que le seuil fixé
48
49      cout << "Pi = " << Pi << " (" << iterateur << " iterations)" << endl;
50
51      return 0;
52  }
```

double !!!

point-virgule !!!

4. (sinon division entière!)

== (opérateur de comparaison !) 0 (le reste de la division par 2 doit être nul si on veut que le nombre soit pair!)