Introduction

Notions supposées connues

- Notion de programme
- Notion de variable, de type, de déclaration
- Instruction conditionnelle

```
if (...) {...} else {...}
```

- Instructions itératives

```
while (...) {...}
do {...} while (...)
for (...;...;...) {...}
```

- Fonctions et appels

Objectifs du cours

Sémantique

- Différences entre la syntaxe et la sémantique
- Comprendre le *sens* des programmes

Mémoire

- Comment l'information est mémorisée
- Transfert et partage de variables

Algorithmique

- Structures de données de base
- Notion d'algorithme
- Récursivité

Sémantique

Définitions

La syntaxe d'un langage est la description des expressions correctes du langage indépendamment du sens.
La sémantique d'un langage est la description du sens des construction du langage.

Syntaxe correcte $\not\Rightarrow$ Sémantique correcte

Les *phrases* syntaxiquement correctes d'un langage n'ont pas toutes un sens.

Exemple: Le chat est analytique à l'origine.

Le compilateur traduit un programme source (ex: en C++) en un programme exécutable sous forme binaire. (en langage machine)

Définition

Ce qui se produit pendant la compilation est dit statique.

Exemples:

- allocation d'une variable globale
- addition d'une variable entière et d'un variable de chaine de caractères

Le processeur exécute le programme binaire afin de réaliser ce qui est décris dans le programme source.

Exécution

Ce qui se produit pendant l'exécution est dit dynamique.

Exemples:

- allocation d'une variable locale
- division par une variable contenant la valeur 0

Erreurs de sématique statique

```
 1 #include <iostream>
 2 #include <vecto>
 3 using namespace std;
 4
 5 void pp(float z) {
 6     int i;
 7     i = 2;
 8     z = i;
 9 }
10
11 int main() {
12     int x;
13     vector<int> t(10);
14     cin >> i;
15     x = t;
16     pp(x, t);
17     return 0;
18 }
```

Erreurs de sématique statique

```
 1 #include <iostream>
 2 #include <vecto>
 3 using namespace std;
 4
 5 void pp(float z) {
 6     int i;
 7     i = 2;
 8     z = i;
 9 }
10
11 int main() {
12     int x;
13     vector<int> t(10);
14     cin >> i;
15     x = t;
16     pp(x, t);
17     return 0;
18 }
```

L14: `i` n'est déclarée que dans pp.

Erreurs de sématique statique

```
 1 #include <iostream>
 2 #include <vecto>
 3 using namespace std;
 4
 5 void pp(float z) {
 6     int i;
 7     i = 2;
 8     z = i;
 9 }
10
11 int main() {
12     int x;
13     vector<int> t(10);
14     cin >> i;
15     x = t;
16     pp(x, t);
17     return 0;
18 }
```

L14: `i` n'est déclarée que dans pp.

L15: affectation d'un vecteur à un entier

Erreurs de sématique statique

```
 1 #include <iostream>
 2 #include <vecto>
 3 using namespace std;
 4
 5 void pp(float z) {
 6     int i;
 7     i = 2;
 8     z = i;
 9 }
10
11 int main() {
12     int x;
13     vector<int> t(10);
14     cin >> i;
15     x = t;
16     pp(x, t);
17     return 0;
18 }
```

L14: `i` n'est déclarée que dans pp.

L15: affectation d'un vecteur à un entier

L16: appel à pp avec deux paramètres au lieu d'un

Erreurs de sémantique statique

Principe

Incohérences entre la déclaration d'un objet et son utilisation : il faut déclarer les objets avant de les utiliser afin de vérifier que l'utilisation est confirme.

- Erreurs sur la portée des variables
- Erreurs sur le contrôle des types
- Erreurs lors de l'appel de fonctions

Nous allons voir cela en détail...

Sémantique statique

Définition

La sémantique statique est la partie de la sémantique qui peut être vérifiée sans exécuter le programme.

Retenir

La sémantique statique est contrôlée par le compilateur.
La *compilation* et l'*exécution* d'un programme ne sont pas possible tant que la sémantique statique n'est pas correcte.

On ne peut pas traduire et a fortiori exécuter un programme qui n'a pas de sens !

Sémantique dynamique

```
 1 int f(int n) {
 2     int res = n;
 3     if (n > 2)
 4         res = f(n-1) + f(n-2);
 5     return res;
 6 }
 7
 8 int main() {
 9     int i;
10     vector<int> t(10);
11     i = 1;
12     // ... initialisation de t
13     while (f(i) < 1000) {
14         cout << t[i] << endl;
15         i++;
16     }
17     return 0;
18 }
```

Sémantique dynamique

```
 1 int f(int n) {
 2     int res = n;
 3     if (n > 2)
 4         res = f(n-1) + f(n-2);
 5     return res;
 6 }
 7
 8 int main() {
 9     int i;
10     vector<int> t(10);
11     i = 1;
12     // ... initialisation de t
13     while (f(i) < 1000) {
14         cout << t[i] << endl;
15         i++;
16     }
17     return 0;
18 }
```

L14: Accès en dehors du tableau.

Problème

On ne peut pas décrire toute la sémantique d'un langage uniquement par sa sémantique statique !

Des erreurs peuvent apparaître à l'exécution, par exemple :

- division par zéro
- accès en dehors des bornes d'un tableau
- plus généralement, mauvais accès à la mémoire
- ressources (fichier, réseau...) indisponible
- ...

Définition

La sémantique dynamique est la partie de la sémantique qui ne peut être contrôlée que lors de l'exécution du programme.

Avantages des langages compilés :

- plus petite proportion de sémantique dynamique
- plus d'erreurs sémantique détectées à la compilation
- moins d'erreurs d'exécution plus difficiles à corriger

Décrire la sémantique

La mémoire

La mémoire est composée de :

- $\sim 10^{11}$ condensateurs chargés ou déchargés
- un réseau de fils et de transistors

Retenir

On groupe les condensateurs par mots. (typiquement 8, 16, 32 ou 64) On repère un mot en mémoire par un nombre appelé adresse.

L'opérateur `&`, retourne l'adresse d'un variable :

```
int a = 5;
cout << a << " " << &a;
```

affiche quelquechose comme `5 0x7ffe94f3a8bc`.

Mémoire simplifiée

Dans ce cours

Pour simplifier, on suppose que toutes les valeurs de base (int, float, char...) occupent exactement une case en mémoire.

On représente la mémoire par un tableau :

4	?
3	?
2	'a'
1	?
0	4

?: Une valeur inconnue, imprévisible, non définie, aléatoire.

Retenir

La première écriture dans un emplacement mémoire s'appelle l'initialisation.
Sans initialisation, l'emplacement contient quand même une valeur mais elle est imprévisible.

?: *Une valeur inconnue, imprévisible, non définie, aléatoire.*

Attention

Certains langages ont décidés de tout initialiser à 0.
Ce n'est pas le cas du C/C++.

Notion de variable

- **But :** stocker des informations en mémoire pendant l'exécution
- On ne veut pas manipuler directement les adresses; on manipule les *variables*
- Le programmeur nomme les variables (*identificateur*)

Définition

Une variable est un espace de stockage où le programme peut mémoriser une donnée.

La variables désignent une ou plusieurs cases mémoires contenant une suite de 0 et 1 codant leur valeur.

Retenir

Une variable en cours d'utilisation possède quatre propriétés :

- un nom
- une adresse
- un type
- une valeur

Définition

réserver de la mémoire s'appelle l'allocation

libérer de la mémoire s'appelle la désallocation

L'allocation peut être :

- **statique** ou **dynamique**
- **automatique** ou **manuelle**

Définition

La mémoire est découpée en zones souvent appelées segments. Dans chaque segment la mémoire est allouée différemment.

Exemples :

- code: contient le code binaire du programme
- données statiques: contient les données globales
- pile: contient les variable automatique et les informations pour les appels de fonctions
- tas: contient les variables dynamiques manuelles

Portée

Définition

La portée d'une déclaration d'une variable indique dans quelle portion du programme source la variable est visible.
C'est une question de sémantique statique.

Définition

La durée de vie d'une variable indique la portion d'exécution pendant laquelle la place mémoire associée à la variable est réservée.
C'est une question de sémantique dynamique.

Définition

- occurence de liaison : déclaration (**relie** un nom de variable à un emplacement mémoire).
- occurence d'utilisation : toutes les autres apparitions. On utilise une variable
  - ◇ pour lire son contenu
  - ◇ pour le changer
  - ◇ pour connaître son adresse

Attention: il peut y avoir plusieurs occurences de liaison correspondant à un même nom de variable.

Retenir

Les règles de portée permettent de déterminer si une occurence d'utilisation correspond bien à une occurence de liaison et, le cas échéant, à laquelle.
Dans le cas contraire, une erreur de sémantique a été commise !

Règles de portée du C++ (simplifiées)

Variable locale

Déclarée à l'intérieur d'une fonction

⇒ La portée s'étend du point de déclaration jusqu'à la fin du bloc (paire d'accolades) où se trouve cette déclaration

Variable globale

Déclarée en dehors d'une fonction, d'une classe...

⇒ La portée s'étend du point de déclaration à la fin du fichier.

Des variables déclarées dans des fonctions différentes sont différentes, même si elles portent le même nom !

Cas particulier : le for

Retenir

Dans le cas des boucles (en particulier `for`), ce qui est entre parenthèse fait partie du bloc.

```
1 for (int i = 0; i < 10; i++) {
2     cout << i << endl; // valide
3 }
4 cout << i << endl; // erreur
```

Exemple

```
 1 double power(double a, int n) {
 2     double res = 1;
 3     int i;
 4     for (i = 0; i < n; i++) {
 5         res = res * a;
 6     }
 7     return res;
 8 }
 9 double evalpoly(vector<double> v, double x) {
10     double res = 0;
11     for (int i = 0; i < v.size(); i++) {
12         res = res + power(x, i) * v[i];
13     }
14     return res;
15 }
16 void main() {
17     vector<double> pol = {1.5, 2.5, 2.5, 4.0};
18     cout << "adresse = " << &pol << endl;
19     cout << evalpoly(pol, 3.1) << endl;
20 }
```

Retenir

Une déclaration d'un nom dans un bloc peut en masquer une autre située dans le bloc conteneur.

```
1  int x; // x global
2  void f() {
3      int x; // x local masque le x global
4      x = 1; // affectation au x local
5      {
6          int x; // masque le premier x local
7          x = 2; // affectation au secon x local
8      }
9      x = 3; // affectation au premier x local
10 }
```

Variables

Variables globales

Retenir

Les variables globales (valables dans tout le programme) sont statiques, c'est-à-dire :

- elles ont pour durée de vie tout le temps d'exécution du programme
- elles sont allouées statiquement dans le segment de données statiques

Le compilateur décide de l'emplacement mémoire de la variable. Cet emplacement est réservé au chargement du programme en mémoire et est libéré à la fin du programme.

Retenir

Les variables locales (déclarées dans un bloc) sont (par défaut) automatiques, c'est-à-dire :

- elles ont pour durée de vie le temps d'exécution du bloc
- sont allouées dynamiquement dans le segment de pile

L'emplacement de mémoire est décidé et réservé uand on exécute le début du bloc. Il est libéré à la fin du bloc.

Adresses des variables

```
1 int g = 3; // Variable globale, allocation statique
2
3 void main() {
4     cout << "Globale " << g << " " << &g << endl;
5
6     int l = 5; // Variable locale, allocation dynamique
7     cout << "Locale " << l << " " << &l << endl;
8 }
```

Retenir

- L'adresse de g est fixée à la compilation
- L'adresse de h change d'une exécution à l'autre