Compilation : théorie, techniques et outils

Thème 4 : Table des symboles

Habib ABDULRAB (Institut National des Sciences Appliquées de Rouen)
Claude MOULIN (Université de Technologie de Compiègne)
Sid TOUATI (Université de Versailles Saint-Quentin en Yvelines)

Table des matières

Objectifs

Structure de données centrale dans un compilateur. Implémentations efficaces, interfaces, etc.

Lectures conseillées :

- Aho, Sethi et Ullman : chapitre 7.6
- Compilateurs : chapitres 2.1.11 et 6.1

Table des symboles

I

A. Table des symboles

- Elle centralise les informations sémantiques des identificateurs d'un programme :
  - pour une variable : nom, type, taille, etc.
  - pour un type : nom, types de base, etc.
  - pour un tableau : nom, dimension, etc.
  - pour une fonction : type, nb de paramètres, etc.
- Rien n'interdit d'avoir plusieurs tables de symboles
  - en fonction de la nature des ids : table des types, etc.
  - en fonction des blocs du programme (portée des déclarations).

Identification

II

A.Identification

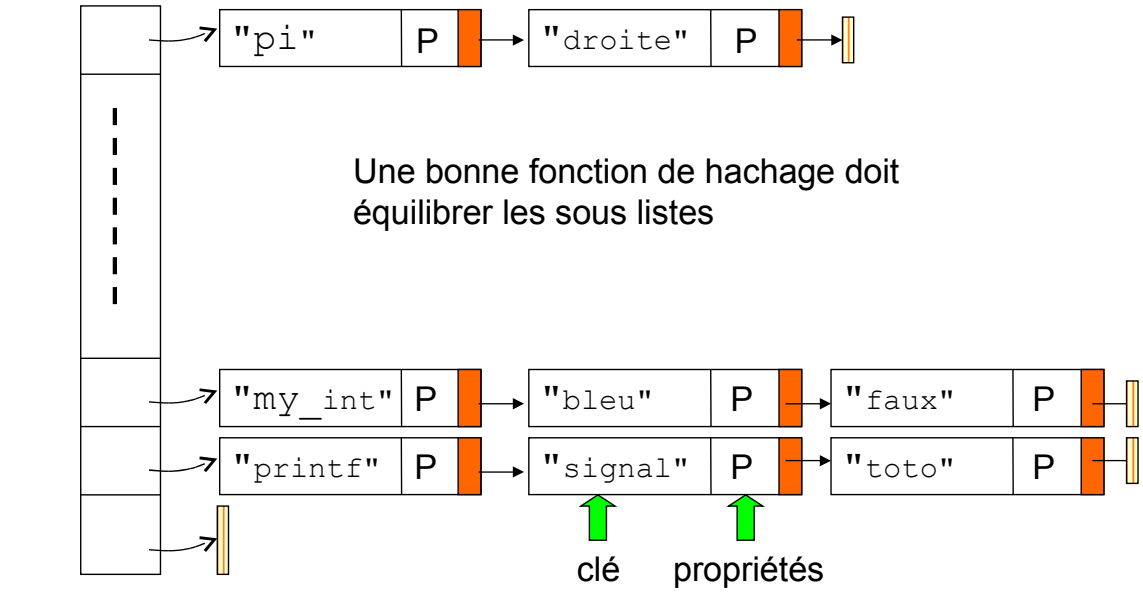
- C'est le processus de retrouver la définition (dans la table de symbole) d'un identificateur donné.
- La TS fournit les informations sur un identifiant: sa nature, son type, valeur initiale, propriétés d'allocation, etc.
- L'accès à la TS doit être rapide, d'où l'utilisation des tables de hachage en général. Elle est indexée par le nom de l'id (chaîne de car).

Complément

Dans certains compilateurs, les mots clés du langage ne font pas partie de la syntaxe, mais ils sont insérés dans la table des symboles.

L'utilisation des tables de hachage permet d'avoir un accès avec une complexité presque constante en fonction du nombre des symboles (à condition que la fonction de hachage soit bien choisie). On peut également utiliser d'autres structures : listes, arbres, tableaux, etc.

B.Structure de hachage



Graphique 1 Structure de hachage

Complément

Le nom d'un id est une chaîne de caractère qui sert à indexer l'accès. Si f est la fonction de hachage, et str est le nom de l'id, alors f(str) donne le numéro de la liste où doit résider l'entrée de l'id.

Les propriétés d'un id peuvent être des structures de données très complexes (arbres, tables, listes, champs, etc.) !

C.Identification

- Lors d'une définition d'un id, on insère une entrée dans la TS : la déclaration d'un id est unique en générale (on étudiera la surcharge par la suite).
- Lors d'une utilisation d'un id dans le programme, on cherche l'entrée correspondante dans la TS.

Complément

Dans certains compilateurs, les mots clés du langage ne font pas partie de la syntaxe, mais ils sont insérés dans la table des symboles.

D.Interface simple

- Comme toute structure de données, il faut d'abord concevoir la structure d'une entrée dans la TS: les informations attachées à un id.
- Deux fonctions de base: insertion, recherche d'un id.
- Autres : modification, suppression, ...

Complément

La modification d'un nœud dans la TS s'effectue pendant l'analyse du programme pour compléter au fur et à mesure les informations sur un symbole.

La suppression d'un id est rarement utilisée. Si un id n'est plus utilisé dans un programme, rien n'empêche de le laisser dans la TS. S'il est redéfini (re-déclaré), on peut utiliser des mécanismes de surcharge pour le modifier dans la TS.

E.Identification

- Tous les id ne sont pas recherchés dans les mêmes **espaces de noms**.
- En général, la position syntaxique d'un id détermine son espace
  - variable, nom de fonction : espace général
  - nom d'un champ (struct) : espaces spéciaux

```
struct un_entier {
int i ; ⟵ recherché dans l'espace des noms de i
} i ; ⟵ recherché dans l'espace général
```

Complément

Les espaces de noms peuvent être vues comme des tables de symboles distinctes.

Le langage définit en général les différents espaces de noms possible. Par ex, le langage C définit 3 espaces principaux de noms:

- espace des noms des types énumérés : typedef, union
- étiquettes (labels)
- le reste : variables, fonctions, types, ids des valeurs d'énumération.

En plus, à chaque type struct, il lui attache un espace de nom spécial qui ne contient que les identificateurs des champs.

Portée d'un identificateur

III

A.La portée d'une déclaration

- Les quatre identifiants `i` référencent-ils la même variable ?
- Comment gérer la portée des noms dans un compilateur ?

```
int i(){
  int i;
  if(…){
    int i;
    i=…
    {
      int i;
      …
      i=…;
    }
    x=i;
  }
  i=…
}
```

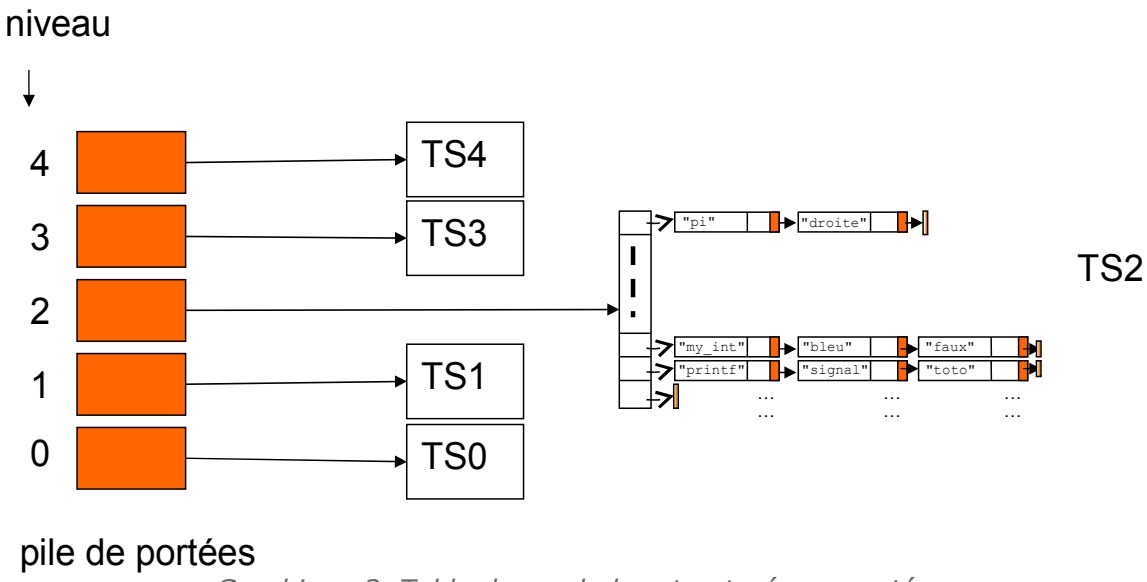
Graphique 2 Portée d'une déclaration

Complément

La déclaration visible d'un identificateur est celle qui se trouve dans le bloc englobant le plus interne.

B.TS structurée en portée

- On utilise une pile de TS : lors de l'analyse d'un bloc dans un programme source, la TS du bloc est mise au sommet de la pile.
- L'identification d'un id se fait en parcourant les TS du sommet de la pile jusqu'à sa base.



Graphique 3 Table de symboles structurée en portée

Complément

En C :

niveau 0 : espace des bibliothèques

niveau 1 : programme (déclaration des fonctions, variables globales)

niveau 2 : paramètres formels

niveau 3 : variables locales

niveau 4 : bloc

- Avantage : simplicité
  - Lorsque le compilateur analyse un nouveau bloc de programme, une nouvelle TS est empilée.
  - La TS du bloc courant est dépilée lorsque le compilateur finit d'analyser le bloc.
- Inconvénient:
  - L'identification d'un nom peut être lente (accès à plusieurs TS successives).
  - La complexité du temps d'accès dépend du niveau d'imbrication.

C.TS structurée en portée avec table associative

- Une table associative centrale indexée par les noms des ids.
- Chaque entrée pointe sur une pile de définitions, le sommet de chaque pile est la définition la plus récente.

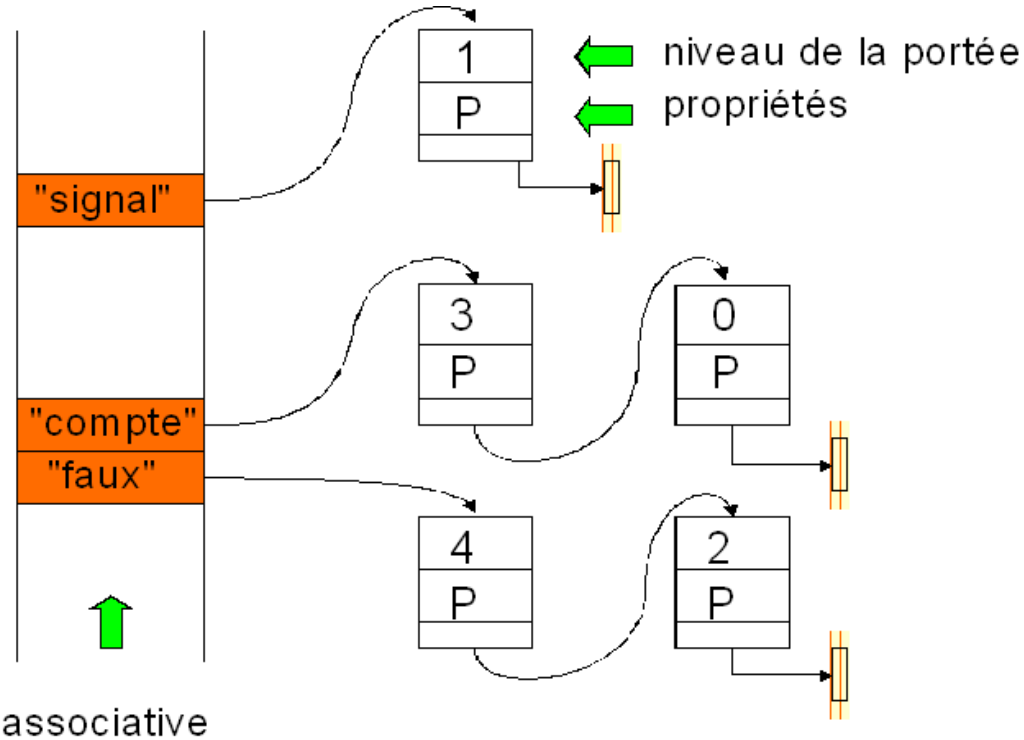
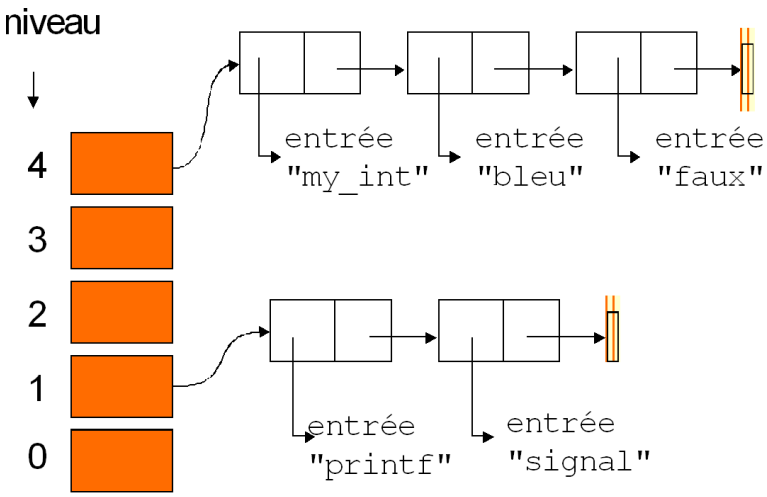


Table associative
indexé par les noms

- Avantage
  - Identifications rapides.
- Inconvénient
  - Gestion des suppressions assez complexe. Elle nécessite une autre structure de données

Complément

Quand on sort d'un bloc, le compilateur a besoin d'actualiser les déclarations visibles (portée des noms). Si on n'utilise que la table associative, on serait obligé de parcourir toutes les entrées de la table, car on manque d'information qui permet de retrouver la liste des ids déclarés à l'intérieur d'un bloc.



Grâce à cette structure auxiliaire, on peut retrouver les ids du même bloc

pile de portées

Complément

Pour accélérer la suppression des entrées (définitions), on utilise une pile de portées qui pointe sur toutes les déclarations d'une portée. Ainsi, à la sortie d'un bloc, il est facile de retrouver toutes les déclarations : la suppression des entrées dans la TS est accélérée.

Surcharge d'un identificateur

IV

A.Surcharge

- La surcharge permet de déclarer plusieurs fois le **même** identificateur.
- C'est une propriété que certains langages autorisent.
- Deux problèmes :
  - Le processus d'identification renvoie un ensemble de définitions au lieu d'une seule.
  - Comment résoudre l'ambiguïté (réduction de l'ensemble des définitions à une seule).

B.Identification avec surcharge

- Il suffit simplement d'inclure toutes les déclarations (définitions) dans la TS.
- Une portée peut contenir donc plusieurs définitions du même ids.
- Évidemment, on parle de surcharge uniquement entre les ids du même espace de noms et de la même portée.

C.Réduction des définitions

- Les règles de réduction des surcharges est dictée par le langage
- Il faut pouvoir analyser le contexte pour se retrouver dans le cas où une seule définition est possible.
- Exemple : en C++ et en ada, les noms des fonctions peuvent être surchargées. La réduction à une seule définition est possible. Elle se fait grâce à la comparaison entre les listes des paramètres.

Complément

Si le processus de réduction ne se termine pas avec une et une seule définition, le

compilateur doit émettre une erreur.

Portées importées

V

A.Spécification d'une portée

- La complexité du génie logiciel a fait exploser le nombre d'ids dans un programme.
- Pour éviter le risque d'avoir des collisions de noms très fréquemment, beaucoup de langages permettent de spécifier de nouvelles portées .
- Le C++ utilise `x::toto` et `using namespace`
- etc.

B.Importation des portées

- La sémantique dépend du langage.
- C++ : `x::toto`
  - si plusieurs `toto` sont visible dans la portée courante, `x::toto` permet de sélectionner un seul.
- C++ : `using namespace` *`nom_espace`*
  - Permet d'inclure une nouvelle portée.
  - Similaire dans pascal et modula 2 : `WITH` *`nom_espace`* `DO ... END`
- Modula 2 : `FROM module IMPORT`
  - fusionne les nouveaux noms dans la portée courante.
  - Similaire : `use` de ADA

Complément

Ne pas confondre le mot clé namespace de C++, et le concept d'espace de noms en compilation.

C.Implémentation de l'importation des portées

- Si une nouvelle portée est déclarée, on l'inclut dans la TS, comme si le compilateur entre dans un nouveau bloc.
- En revanche, si on insère de nouveaux noms à la portée courante, on peut provoquer des surcharges qu'il faut résoudre. Tout dépend du langage.

Conclusion

Résumé :

- Une table de symbole est une centralisation des informations rattachées aux identificateurs d'un programme.
- Elle doit être bien conçue pour des accès rapides.
- Elle peut être très compliquée et devient très vite une machine à gaz.
- Rien n'interdit d'avoir plusieurs tables de symboles, il faut simplement savoir les gérer.