Traduction et Sémantique
La table des symboles
Gestion de l'espace des noms

Luigi Santocanale
LIF, Université de Provence
Marseille, FRANCE

22 avril 2010

Plan

La table des symboles

- Un premier approche
- . . . et plus en profondeur
- Aspects algorithmiques

Gestion de l'espace des noms

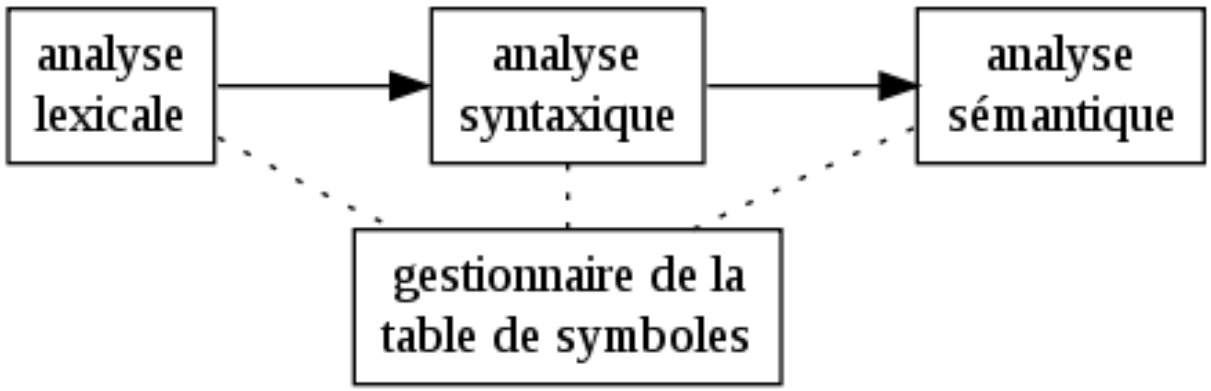
- Contexte
- Langages à blocs
- Principes d'implantation d'une table avec blocs

Plan

La table des symboles
- Un premier approche
- . . . et plus en profondeur
- Aspects algorithmiques

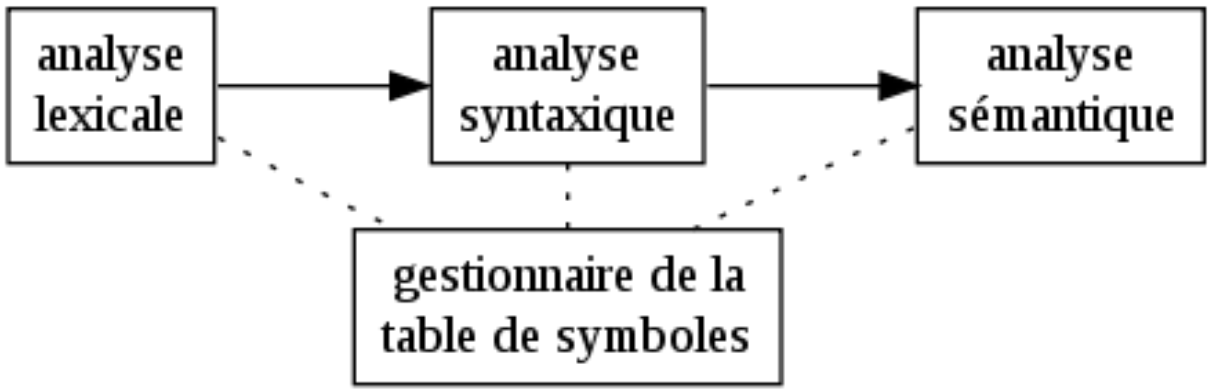
Gestion de l'espace des noms
- Contexte
- Langages à blocs
- Principes d'implantation d'une table avec blocs

Interaction entre les différentes phases et la table



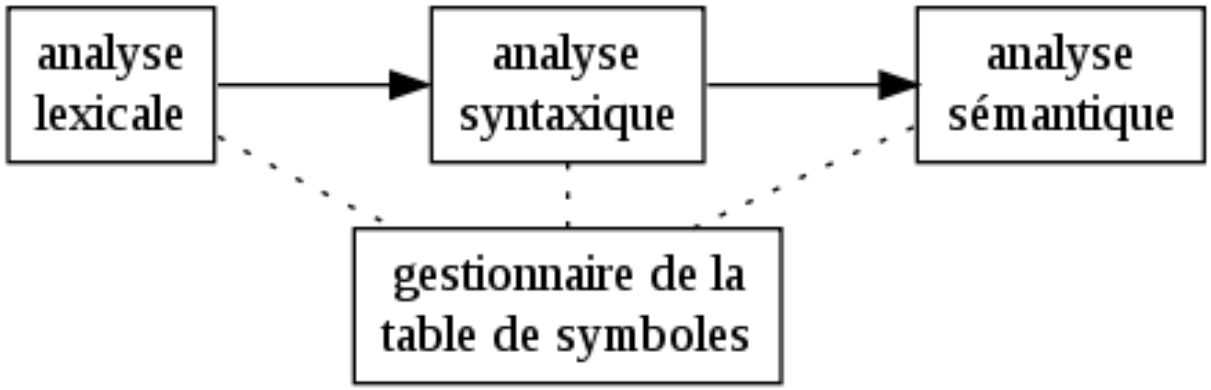
- Analyse lexicale : insertion de noms dans la table,
- Analyses syntaxique et sémantique : ajout d'attributs ;

Interaction entre les différentes phases et la table



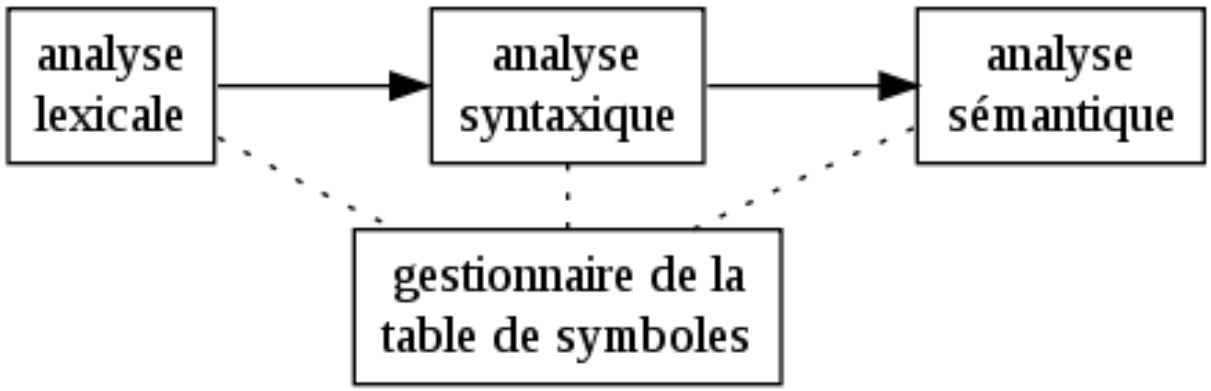
- Analyse lexicale : insertion de noms dans la table,
- Analyses syntaxique et sémantique : ajout d'attributs :
  - type,
  - utilisation (procédure, variable, étiquette),
  - paramètres formels,
  - adresse en mémoire,
  - ...

Interaction entre les différentes phases et la table



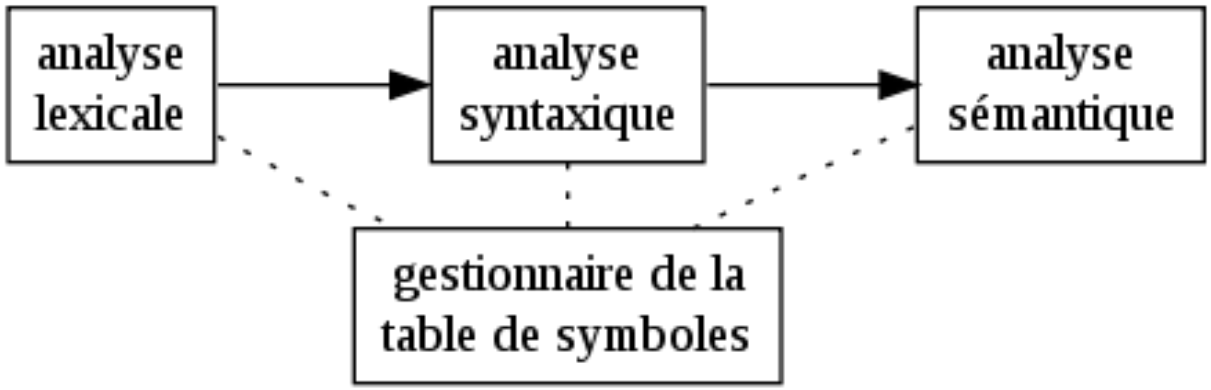
- Analyse lexicale : insertion de noms dans la table,
- Analyses syntaxique et sémantique : ajout d'attributs :
  - type,
  - utilisation (procédure, variable, étiquette),
  - paramètres formels,
  - adresse en mémoire,
  - ...

Interaction entre les différentes phases et la table



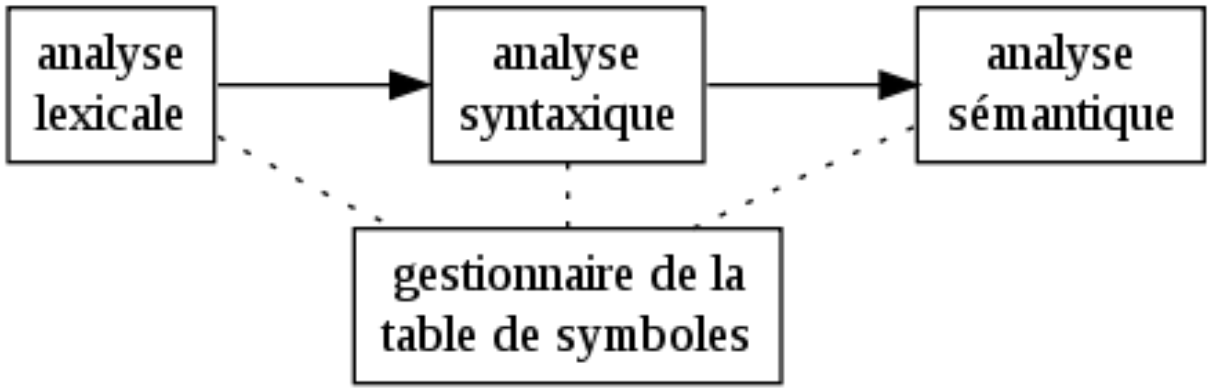
- Analyse lexicale : insertion de noms dans la table,
- Analyses syntaxique et sémantique : ajout d'attributs :
  - type,
  - utilisation (procédure, variable, étiquette),
  - paramètres formels,
  - adresse en mémoire,
  - ...

Interaction entre les différentes phases et la table



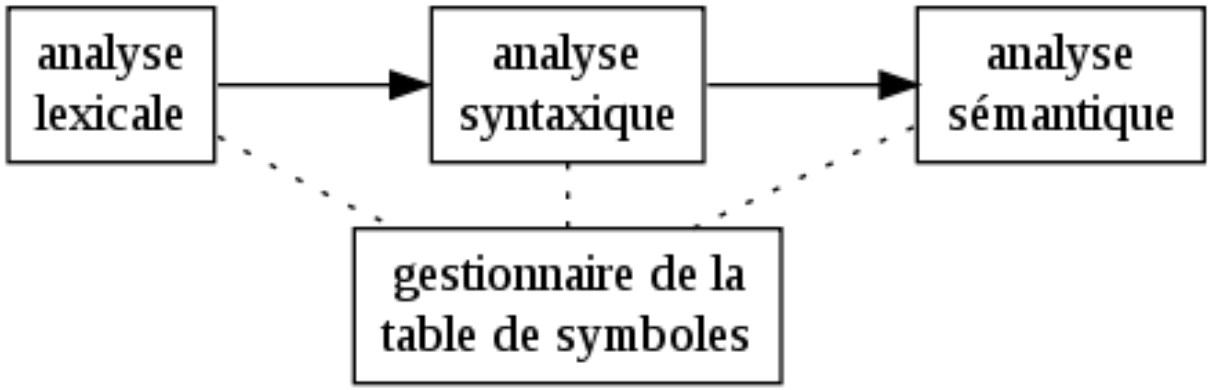
- Analyse lexicale : insertion de noms dans la table,
- Analyses syntaxique et sémantique : ajout d'attributs :
  - type,
  - utilisation (procédure, variable, étiquette),
  - paramètres formels,
  - adresse en mémoire,
  - ...

Interaction entre les différentes phases et la table



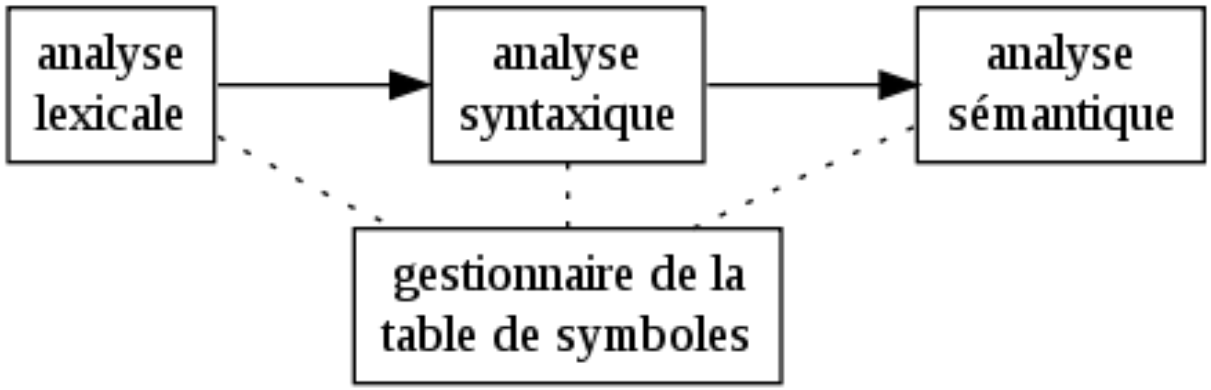
- Analyse lexicale : insertion de noms dans la table,
- Analyses syntaxique et sémantique : ajout d'attributs :
  - type,
  - utilisation (procédure, variable, étiquette),
  - paramètres formels,
  - adresse en mémoire,
  - ...

Interaction entre les différentes phases et la table



- Analyse lexicale : insertion de noms dans la table,
- Analyses syntaxique et sémantique : ajout d'attributs :
  - type,
  - utilisation (procédure, variable, étiquette),
  - paramètres formels,
  - adresse en mémoire,
  - ...

Interaction entre les différentes phases et la table

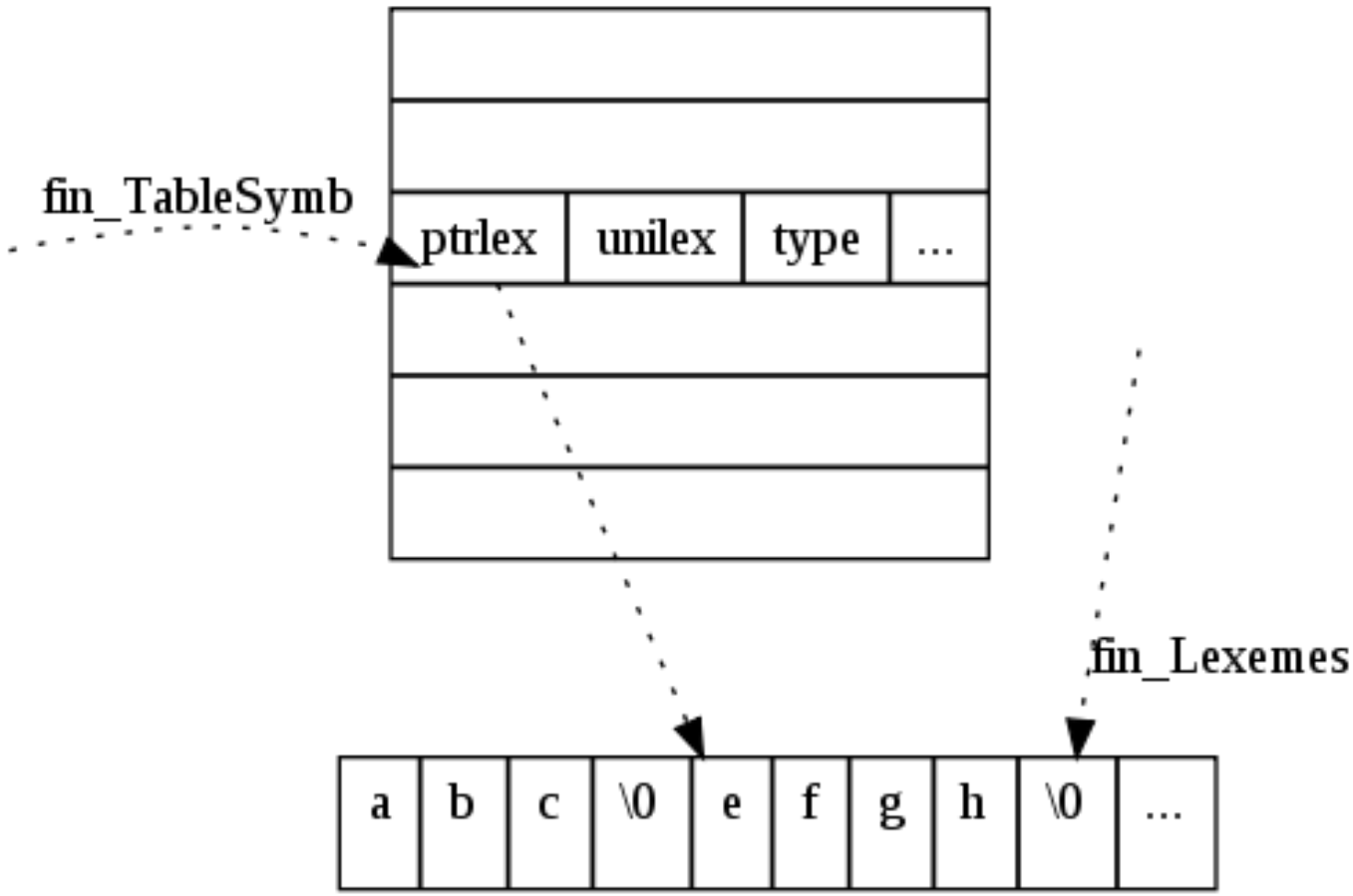


- Analyse lexicale : insertion de noms dans la table,
- Analyses syntaxique et sémantique : ajout d'attributs :
  - type,
  - utilisation (procédure, variable, étiquette),
  - paramètres formels,
  - adresse en mémoire,
  - ...

L'interface

- `Inserer(s,t)` :
  rend l'indice de la nouvelle entrée pour
  la chaîne `s` et l'unité lexicale `t`.
- `Chercher(s)` :
  rend l'indice de l'entrée pour la chaîne `s`,
  ou `0` si `s` n'est pas dans la table.

Une implantation



Un exemple : l'`Entree` de la table

```
#ifndef SYMBOLE_H
#define SYMBOLE_H

struct Entree {
  char *ptrlex;
  int unilex;
  /* Utilisés par l'analyseur lexicale */

  int type;
  /* Utilisé par l'analyseur sémantique */
};

#endif /* SYMBOLE_H */
```

Un exemple : chercher

```
#define T_LEXEMES 999    /* taille de Lexemes */
#define T_TABLESYMB 100 /* taille de la TableSymb */

char Lexemes[T_LEXEMES];
int fin_Lexemes = -1;
/* Denière position utilisée dans Lexemes */

struct Entree TableSymb[T_TABLESYMB];
int fin_TableSymb = 0;
/* Denière position utilisée dans TableSymb */

int chercher(char *symb)
{
  int p;

  for(p = fin_TableSymb; p>0;p--)
    if(strcmp(TableSymb[p].ptrlex,symb) == 0)
      return p;

  return 0;
}
```

Un exemple : inserer

```
int inserer(char *symb,int Lex)
{
  int long_symb = strlen(symb);

  /* Vérifier qu'on a assez de place */
  if(fin_TableSymb > T_TABLESYMB)
    Erreur("Table des symboles pleine");
  if(fin_Lexemes + long_symb + 1 > T_LEXEMES)
    Erreur("Table des lexemes pleine");

  /* Faire l'insertion et incrémenter */
  fin_TableSymb++;
  TableSymb[fin_TableSymb].unilex = Lex;
  TableSymb[fin_TableSymb].ptrlex = &Lexemes[fin_Lexemes +1];
  fin_Lexemes += long_symb + 1;
  strcpy(TableSymb[fin_TableSymb].ptrlex,symb);

  return fin_TableSymb;
}
```

L'entrée de la table

- Entrée $\leftrightarrow$ déclaration (explicite ou implicite)
- Création de l'entrée :
  - pendant l'analyse lexicale
  - avant l'analyse lexicale :
    si le mots clès ne sont pas reservées,
  - pendant l'analyse syntaxique :

Remarque : les noms ne sont pas nécessairement uniques.

L'entrée de la table

- Entrée $\leftrightarrow$ déclaration (explicite ou implicite)
- Création de l'entrée :
  - pendant l'analyse lexicale
  - avant l'analyse lexicale :

    si le mots clès ne sont pas reservées,
  - pendant l'analyse syntaxique :

```
int i;
struct i {int j; float k;};
```

Remarque : les noms ne sont pas nécessairement uniques.

L'entrée de la table

- Entrée $\leftrightarrow$ déclaration (explicite ou implicite)
- Création de l'entrée :
  - pendant l'analyse lexicale
  - avant l'analyse lexicale :

    si le mots clès ne sont pas reservées,
  - pendant l'analyse syntaxique :

```
int i;
struct i {int j; float k;};
```

Voir identificateurs/constructeurs en C++.

Remarque : les noms ne sont pas nécessairement uniques.

L'entrée de la table

- Entrée $\leftrightarrow$ déclaration (explicite ou implicite)
- Création de l'entrée :
  - pendant l'analyse lexicale
  - avant l'analyse lexicale :
    si le mots clès ne sont pas reservées,
  - pendant l'analyse syntaxique :

```
int i;
struct i {int j; float k;};
```

Voir identificateurs/constructeurs en `tic`.

Remarque : les noms ne sont pas nécessairement uniques.

L'entrée de la table

- Entrée $\leftrightarrow$ déclaration (explicite ou implicite)
- Création de l'entrée :
  - pendant l'analyse lexicale
  - avant l'analyse lexicale :
    si le mots clès ne sont pas reservées,
  - pendant l'analyse syntaxique :

```
int i;
struct i {int j; float k;};
```

Voir identificateurs/constructeurs en `tic`.

Remarque : les noms ne sont pas nécessairement uniques.

L'entrée de la table

- Entrée $\leftrightarrow$ déclaration (explicite ou implicite)
- Création de l'entrée :
  - pendant l'analyse lexicale
  - avant l'analyse lexicale :
    si le mots clès ne sont pas reservées,
  - pendant l'analyse syntaxique :

```
int i;
struct i {int j; float k;};
```

Voir identificateurs/constructeurs en `tic`.

Remarque : les noms ne sont pas nécessairement uniques.

L'entrée de la table

- Entrée $\leftrightarrow$ déclaration (explicite ou implicite)
- Création de l'entrée :
  - pendant l'analyse lexicale
  - avant l'analyse lexicale :
    si le mots clès ne sont pas reservées,
  - pendant l'analyse syntaxique :

```
int i;
struct i {int j; float k;};
```

    Voir identificateurs/constructeurs en `tic`.

Remarque : les noms ne sont pas nécessairement uniques.

Contenu de l'entrée

- Le nom,
- un pointeur vers la chaîne de caractères,
  - la table des chaînes,
- type, utilisation,
- informations sur la mémoire.

Contenu de l'entrée

- Le nom,
- la chaîne de caractères,
  - la table des chaînes,
- type, utilisation,
- informations sur la mémoire.

Contenu de l'entrée

- Le nom,
- un pointeur vers la chaîne de caractères,
  - la table des chaînes,
- type, utilisation,
- informations sur la mémoire.

Contenu de l'entrée

- Le nom,
- un pointeur vers la chaîne de caractères,
  - la table des chaînes,
- type, utilisation,
- informations sur la mémoire.

Contenu de l'entrée

- Le nom,
- un pointeur vers la chaîne de caractères,
  - la table des chaînes,
- type, utilisation,
- informations sur la mémoire.

Informations sur la mémoire

Liaison :
association entre un nom et une location en mémoire.

À ne pas confondre avec :
État :
association entre locations de mémoire et leur contenu.

Pour les données statiques,
si la traduction est vers le code machine :

- position de la donnée par rapport à une origine fixe.

Informations sur la mémoire

Liaison :
association entre un nom et une location en mémoire.

À ne pas confondre avec :
État :
association entre locations de mémoire et leur contenu.

Pour les données statiques,
si la traduction est vers le code machine :

- position de la donnée par rapport à une origine fixe.

Informations sur la mémoire

Liaison :
association entre un nom et une location en mémoire.

À ne pas confondre avec :
État :
association entre locations de mémoire et leur contenu.

Pour les données statiques,
si la traduction est vers le code machine :

- position de la donnée par rapport à une origine fixe.

Informations sur la mémoire

Liaison :
association entre un nom et une location en mémoire.

À ne pas confondre avec :
État :
association entre locations de mémoire et leur contenu.

Pour les données statiques,
si la traduction est vers le code machine :

- position de la donnée par rapport à une origine fixe.

Informations sur la mémoire

Liaison :
association entre un nom et une location en mémoire.

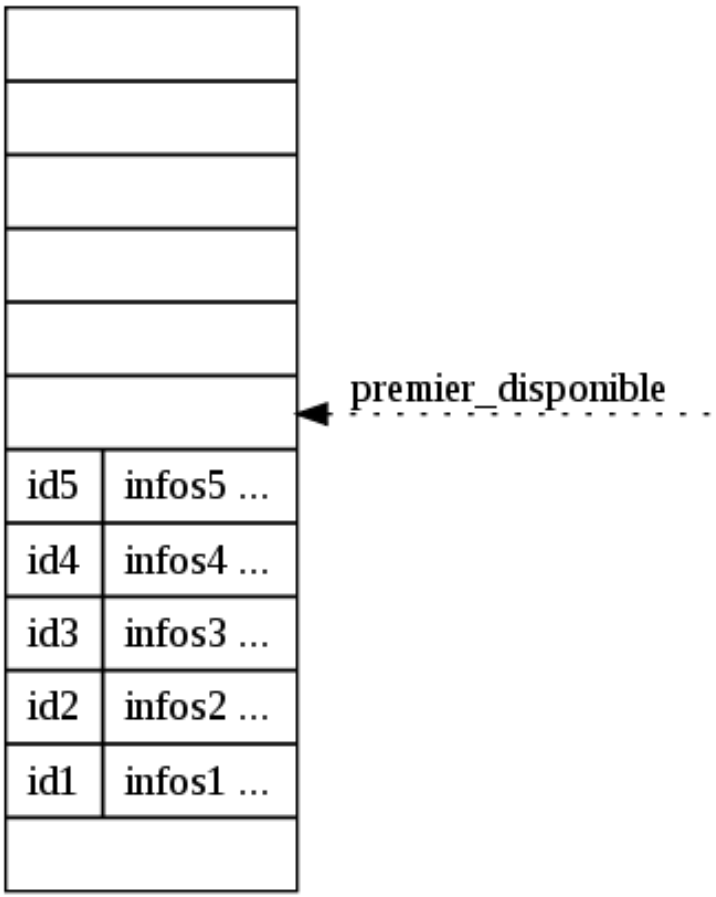
À ne pas confondre avec :
État :
association entre locations de mémoire et leur contenu.

Pour les données statiques,
si la traduction est vers le code machine :

- position de la donnée par rapport à une origine fixe.

Structure de donnée : liste linéaire

Vue auparavant :



- un (simple) mécanisme d'allocation d'entrée,
- les noms se trouvent dans la liste dans l'ordre inverse : denier noms = premier dans la liste.

Analyse de la performance

Hypothèses :

- les noms dupliqués ne sont pas permis :
  à l'insertion, on parcourt toute la liste.
- n symboles à insérer,
- r recherches.

Temps :

$$t \sim c_0 \frac{n}{2} n + c_1 \frac{n}{2} r = c \frac{n}{2} (n + r)$$

Analyse de la performance

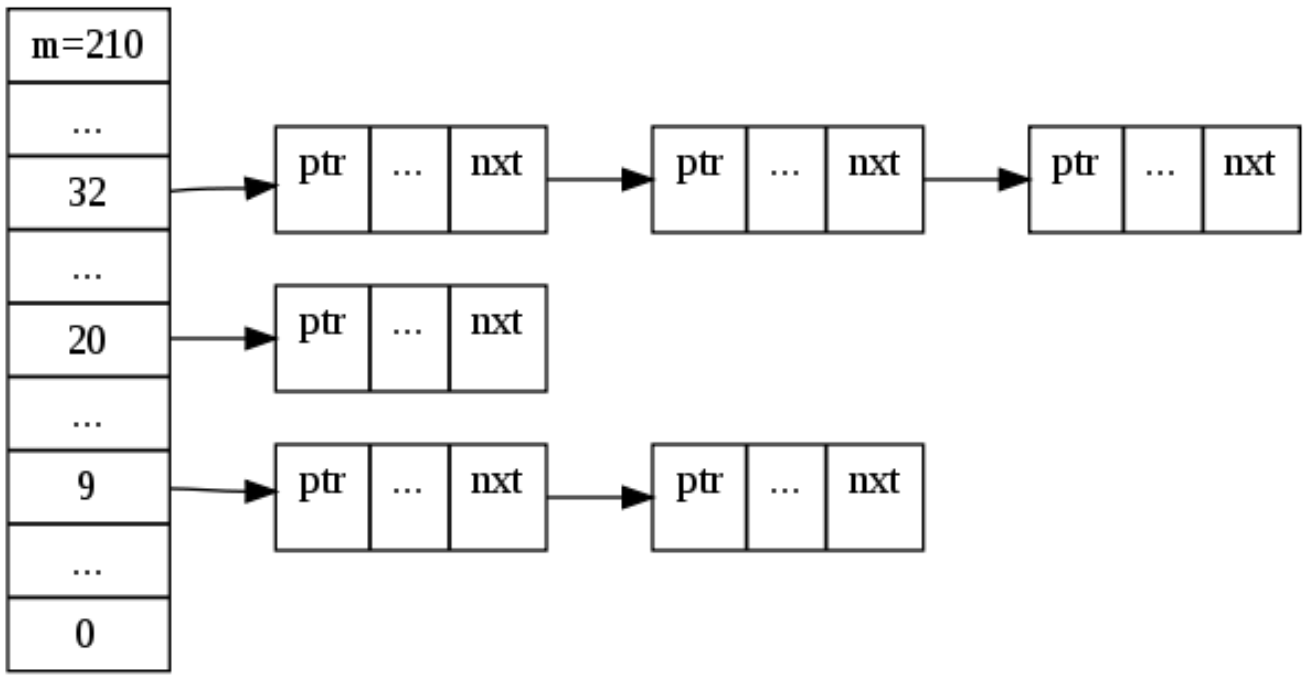
Hypothèses :

- les noms dupliqués ne sont pas permis :
  à l'insertion, on parcourt toute la liste.
- n symboles à insérer,
- r recherches.

Temps :

$$t \sim c_0 \frac{n}{2} n + c_1 \frac{n}{2} r = c \frac{n}{2} (n + r)$$

Structure de donnée : table d'adressage dispersé



Ingrédients

- Tableau d'adresses de taille m,
  (avec m possiblement un nombre premier)
- m listes chaînées,
- une fonction d'hachage

$$h : NOMS \longrightarrow \{0, \ldots, m-1\}$$

telle que

$$Prob(h(X) == i)$$

est uniforme.

Important : le nom s est inséré et cherché dans la liste $h(s)$.

Ingrédients

- Tableau d'adresses de taille m,
  (avec m possiblement un nombre premier)
- m listes chaînées,
- une fonction d'hachage

$$h : NOMS \longrightarrow \{0, \ldots, m-1\}$$

telle que

$$Prob(h(X) == i)$$

est uniforme.

Important : le nom s est inséré et cherché dans la liste $h(s)$.

Ingrédients

- Tableau d'adresses de taille m,
  (avec m possiblement un nombre premier)
- m listes chaînées,
- une fonction d'hachage

$$h : NOMS \longrightarrow \{0, \ldots, m-1\}$$

telle que

$$Prob(h(X) == i)$$

est uniforme.

Important : le nom s est inséré et cherché dans la liste $h(s)$.

Ingrédients

- Tableau d'adresses de taille m,
  (avec m possiblement un nombre premier)
- m listes chaînées,
- une fonction d'hachage

$$h : NOMS \longrightarrow \{0, \ldots, m-1\}$$

telle que

$$Prob(h(X) == i)$$

est uniforme.

Important : le nom s est inséré et cherché dans la liste $h(s)$.

Ingrédients

- Tableau d'adresses de taille m,
  (avec m possiblement un nombre premier)
- m listes chaînées,
- une fonction d'hachage

$$h : NOMS \longrightarrow \{0, \ldots, m-1\}$$

telle que

$$Prob(h(X) == i)$$

est uniforme.

Important : le nom s est inséré et cherché dans la liste $h(s)$.

La fonction d'hachage, un exemple

Si

$$nom = c_1 c_2 \ldots c_n ,$$

on peut calculer h par

$$\begin{aligned} h_0 &= 0 \\ h_{i+1} &= h_i + \alpha c_{i+1} \\ h(nom) &= h_n \mod m , \end{aligned}$$

où α est une constante entière
(possiblement un nombre premier).

Exercice :
implémentez une table de symboles avec une table d'adressage – et ce h – dans le projet `tic`.

La fonction d'hachage, un exemple

Si

$$nom = c_1 c_2 \ldots c_n ,$$

on peut calculer h par

$$\begin{aligned} h_0 &= 0 \\ h_{i+1} &= h_i + \alpha c_{i+1} \\ h(nom) &= h_n \mod m , \end{aligned}$$

où α est une constante entière
(possiblement un nombre premier).

Exercice :
implémentez une table de symboles avec une table d'adressage – et ce h – dans le projet `tic`.

Analyse

Insérer/Chercher un nom se fait en temps

$$t \sim c_{n,m} = \frac{n}{m},$$

si $Prob(h(X) = j)$ est uniforme.

Pour m proche de n,

- on peut considérer $c_{n,m} = \frac{n}{m}$ une petite constante :
  gain en temps,
- il faut stocker un tableau de m adresses :
  perte en espace.

Analyse

Insérer/Chercher un nom se fait en temps

$$t \sim c_{n,m} = \frac{n}{m},$$

si $Prob(h(X) = j)$ est uniforme.

Pour m proche de n,

- on peut considérer $c_{n,m} = \frac{n}{m}$ une petite constante :
  gain en temps,
- il faut stocker un tableu de m adresses :
  perte en espace.

Analyse

Insérer/Chercher un nom se fait en temps

$$t \sim c_{n,m} = \frac{n}{m},$$

si $Prob(h(X) = j)$ est uniforme.

Pour m proche de n,

- on peut considérer $c_{n,m} = \frac{n}{m}$ une petite constante : gain en temps,
- il faut stocker un tableu de m adresses : perte en espace.

A la recherche d'un bon hachage

Une meilleure évaluation :
insérer n noms $\{ nom_0, \ldots, nom_{n-1} \}$ se fait en temps

$$t = \sum_{j=0}^{m-1} \frac{b_j(b_j+1)}{2}$$

où

$$b_j = \text{longueur de la liste } j = \#\{ k \mid h(nom_k) = j \}$$

L'optimum théorique est :

$$t = \frac{n(n+2m-1)}{2m}.$$

A la recherche d'un bon hachage

Une meilleure évaluation :
insérer n noms $\{ nom_0, \ldots, nom_{n-1} \}$ se fait en temps

$$t = \sum_{j=0}^{m-1} \frac{b_j(b_j+1)}{2}$$

où

$$b_j = \text{longueur de la liste } j = \#\{ k \mid h(nom_k) = j \}$$

L'optimum théorique est :

$$t = \frac{n(n+2m-1)}{2m}.$$

Plan

La table des symboles
- Un premier approche
- . . . et plus en profondeur
- Aspects algorithmiques

Gestion de l'espace des noms
- Contexte
- Langages à blocs
- Principes d'implantation d'une table avec blocs

Un nom peut dénoter plusieurs données

Le même nom peut denoter données différentes

- selon l'utilisation :

```
struct a {int i;};
int main(void){
  struct a b;
  int i=0;
  b.i=i;
  goto i;
  return;
 i:
  printf("i est une etiquette, "
         "un entieur, "
         "un champ de structure");
}
```

- dans un appel de fonction récursive :

```
int factoriel(int n)
{
  int i;
  if(n<=0) return 1;
  i= factoriel(n-1);
  return n * i;
}
```

La notion de bloc

Un bloc de code :

- en C :

$$\{ \ldots \}$$

- en Pascal :

```
begin ... end
```

Remarque :
une procédure (fonction) peut se considérer comme un bloc.

Deux blocs peuvent être :

- disjoints,
- imbriques l'un dans l'autre

La notion de bloc

Un bloc de code :

- en C :

$$\{ \ldots \}$$

- en Pascal :

```
begin ... end
```

Remarque :
une procédure (fonction) peut se considérer comme un bloc.

Deux blocs peuvent être :

- disjoints,
- imbriques l'un dans l'autre

La notion de bloc

Un bloc de code :

- en C :

$$\{ \ldots \}$$

- en Pascal :

```
begin ... end
```

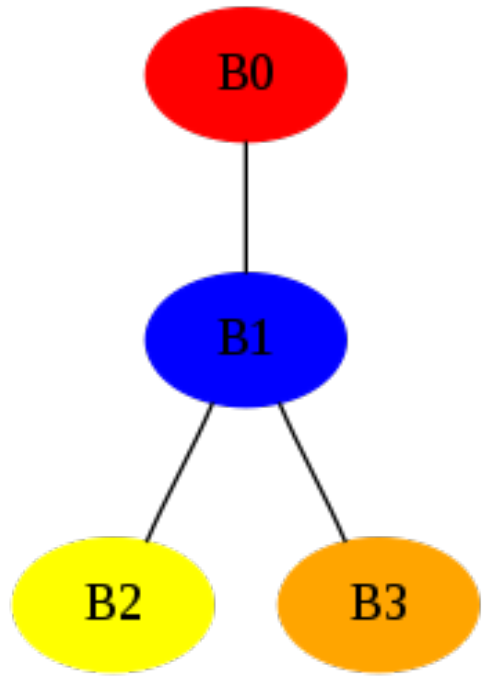
Remarque :
une procédure (fonction) peut se considérer comme un bloc.

Deux blocs peuvent être :

- disjoints,
- imbriques l'un dans l'autre

Structure arborescente

```
 1  main()
 2  {B0:
 3    int a=0;
 4    int b=0;
 5    {B1:
 6      int b=1;
 7      {B2:
 8        int a = 2;
 9        printf("B2 : %d %d\n",a,b);
10      }
11      {B3:
12        int b = 3;
13        printf("B3 : %d %d\n",a,b);
14      }
15      printf("B1 : %d %d\n",a,b);
16    }
17    printf("B0 : %d %d\n",a,b);
18  }
```

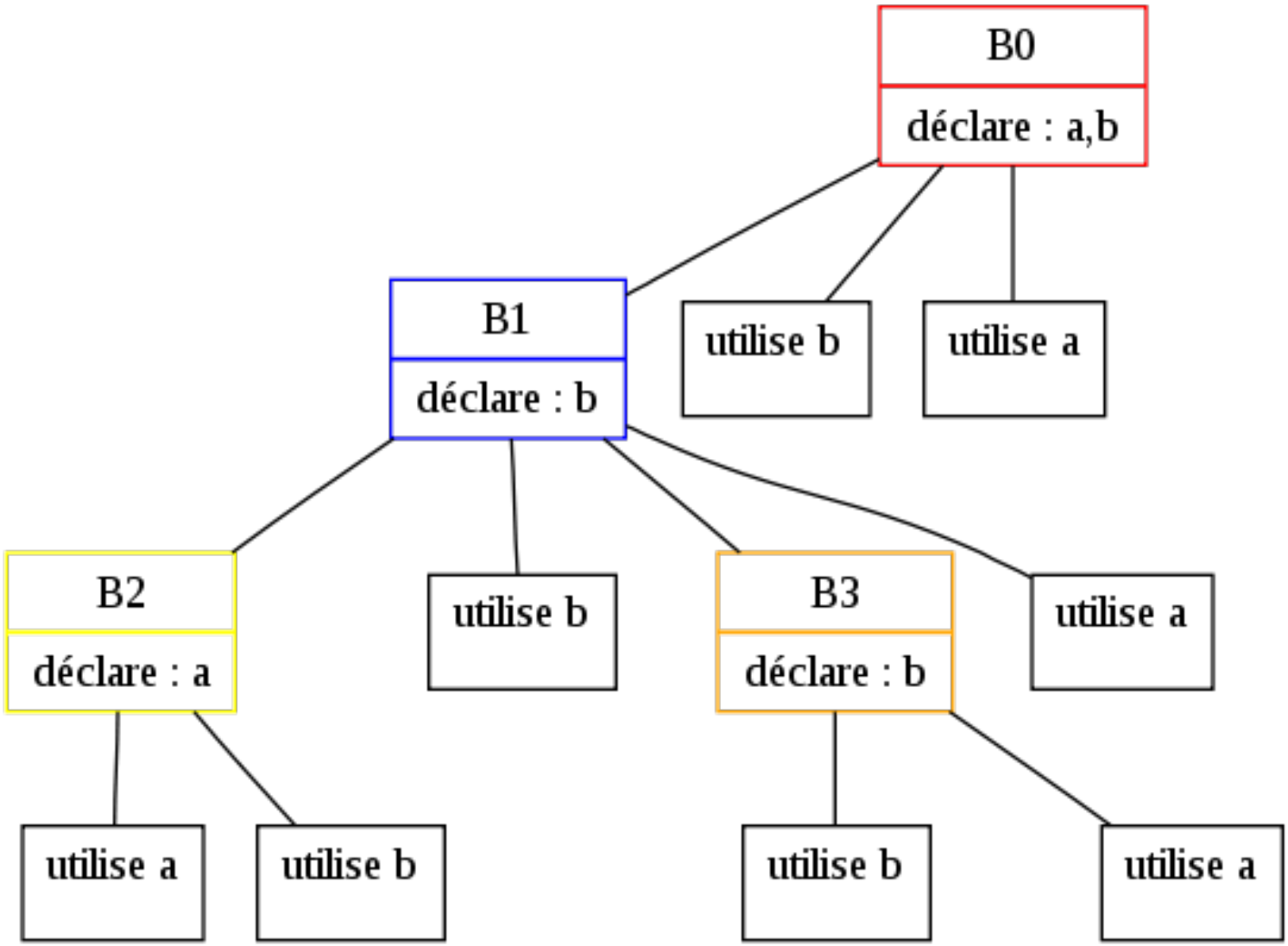


Qualité des noms

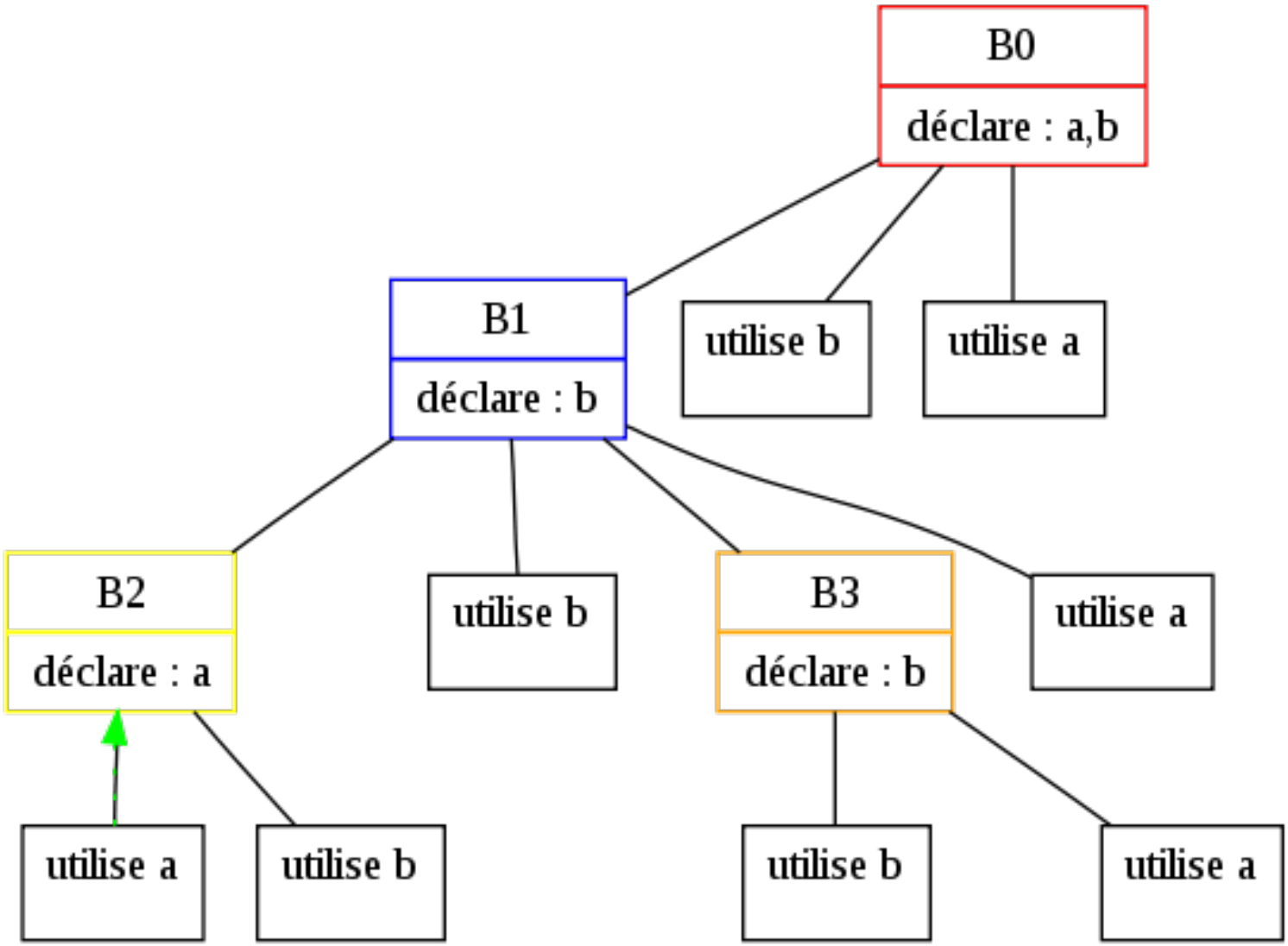
Un nom :

- declaré dans un bloc est locale à ce bloc,
- si utilisé dans un bloc et non déclaré dans le bloc,
  alors il doit être declaré dans un bloc inglobant.

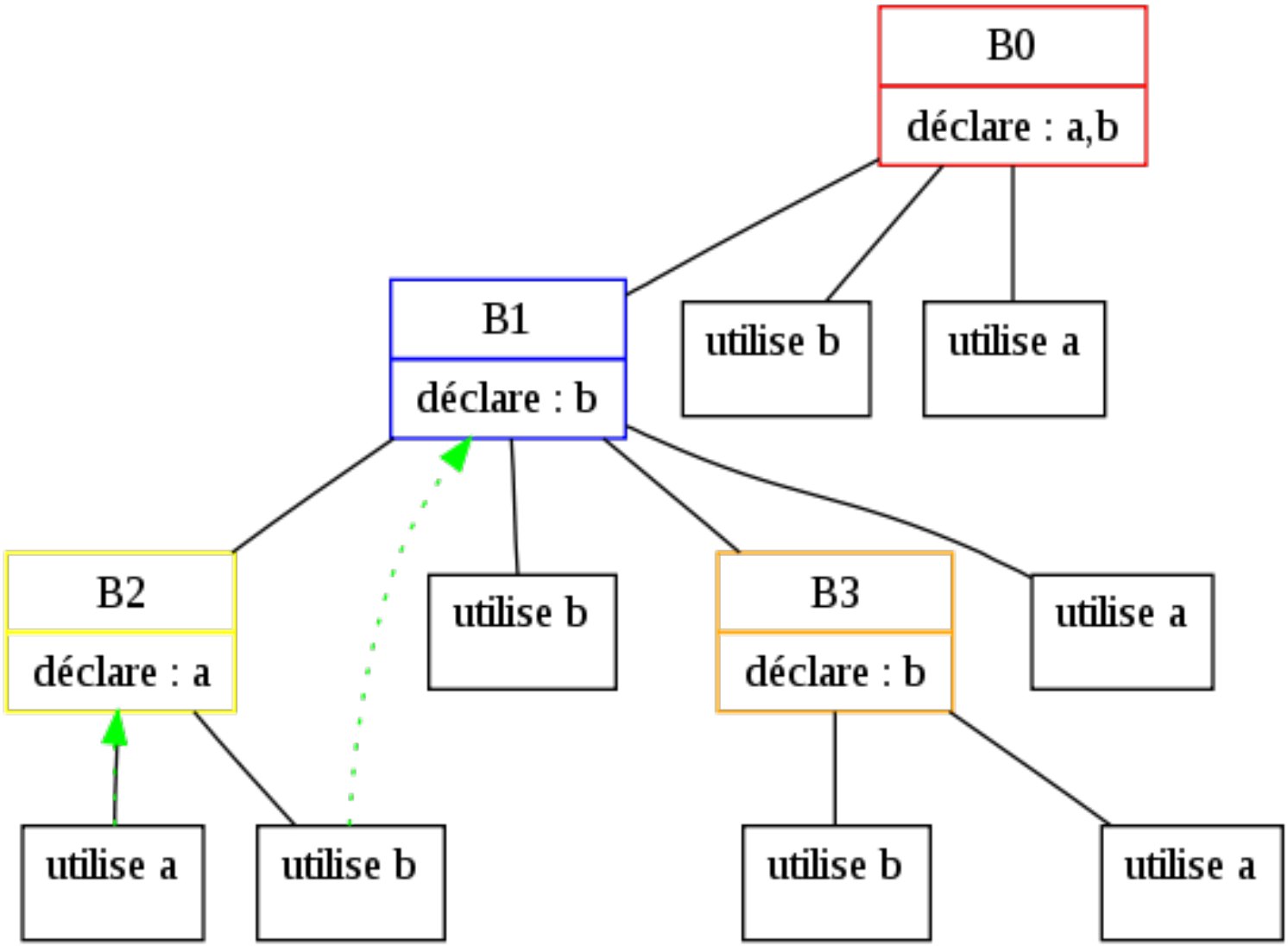
Portée des identificateurs : la règle de l'englobant le plus imbriqué



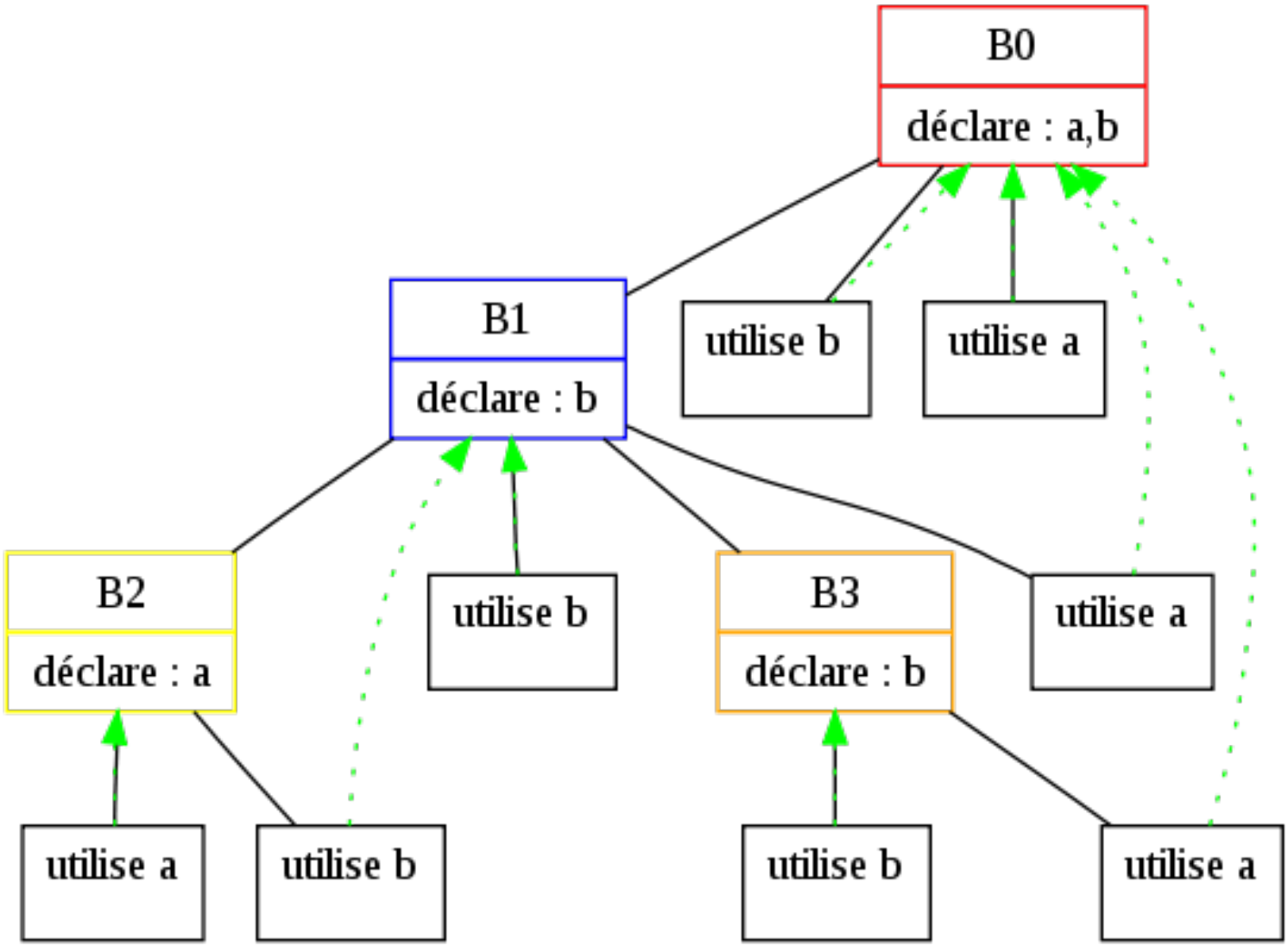
Portée des identificateurs :
la règle de l'englobant le plus imbriqué



Portée des identificateurs : la règle de l'englobant le plus imbriqué



Portée des identificateurs : la règle de l'englobant le plus imbriqué



Remarque :

règle de liaison pour les formules logiques

Considérez :

$$\forall a.\forall b.(\ \forall b.(\ \forall a.R(a,b) \vee \forall b.S(a,b) \vee Q(a,b)\ ) \wedge P(a,b)\ )$$

Remarque :

règle de liaison pour les formules logiques

Considérez :

$$\forall a.\forall b.(\ \forall b.(\ \forall a.R(a,b) \vee \forall b.S(a,b) \vee Q(a,b)\ ) \wedge P(a,b)\ )$$

Remarque :

règle de liaison pour les formules logiques

Considérez :

$$\forall a.\forall b.(\ \forall b.(\ \forall a.R(a,b) \vee \forall b.S(a,b) \vee Q(a,b)\ ) \wedge P(a,b)\ )$$

Remarque :

règle de liaison pour les formules logiques

Considérez :

$$\forall a.\forall b.(\ \forall b.(\ \forall a.R(a,b) \vee \forall b.S(a,b) \vee Q(a,b)\ ) \wedge P(a,b)\ )$$

Remarque :

règle de liaison pour les formules logiques

Considérez :

$$\forall a.\forall b.(\ \forall b.(\ \forall a.R(a,b) \vee \forall b.S(a,b) \vee Q(a,b)\ ) \wedge P(a,b)\ )$$

Remarque :

règle de liaison pour les formules logiques

Considérez :

$$\forall a.\forall b.(\ \forall b.(\ \forall a.R(a,b) \vee \forall b.S(a,b) \vee Q(a,b)\ ) \wedge P(a,b)\ )$$

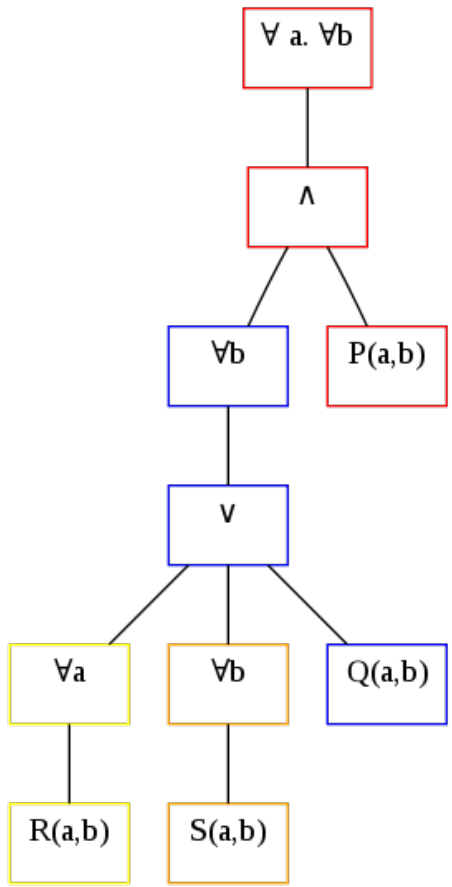


Table de symboles : modification de l'interface

- `Inserer(s,t)` : ...
- `Chercher(s)` : ...
- `Supprimer` :
  à la sortie d'un bloc
  on supprime tous les noms locaux à ce bloc.

Remarque :
on supprime un nom local de la table des symboles actives,
non pas de la mémoire (utilisation dans un deuxième passe).

Listes linéaires

Première solution :

- on marque une entrée comme active (ou non),
- comme début de bloc (ou non).

- `Supprimer` :
  les entrées
  du début de la liste à la première entrée « début du bloc »
  deviennent non-actives.

Exemple

```
{
   int a,b,c;
   {
      int a,b;
   }
   {
      int b,c;
   }
}
```

b	act	N
a	act	debut
c	act	N
b	act	N
a	act	debut

b	N	N
a	N	debut
c	act	N
b	act	N
a	act	debut

c	act	N
b	act	debut
b	N	N
a	N	debut
c	act	N
b	act	N
a	act	debut

Remarque :

Les recherches se font par un parcours de toute la table (pas seulement les éntrées actives).

Amélioration : des vraies listes

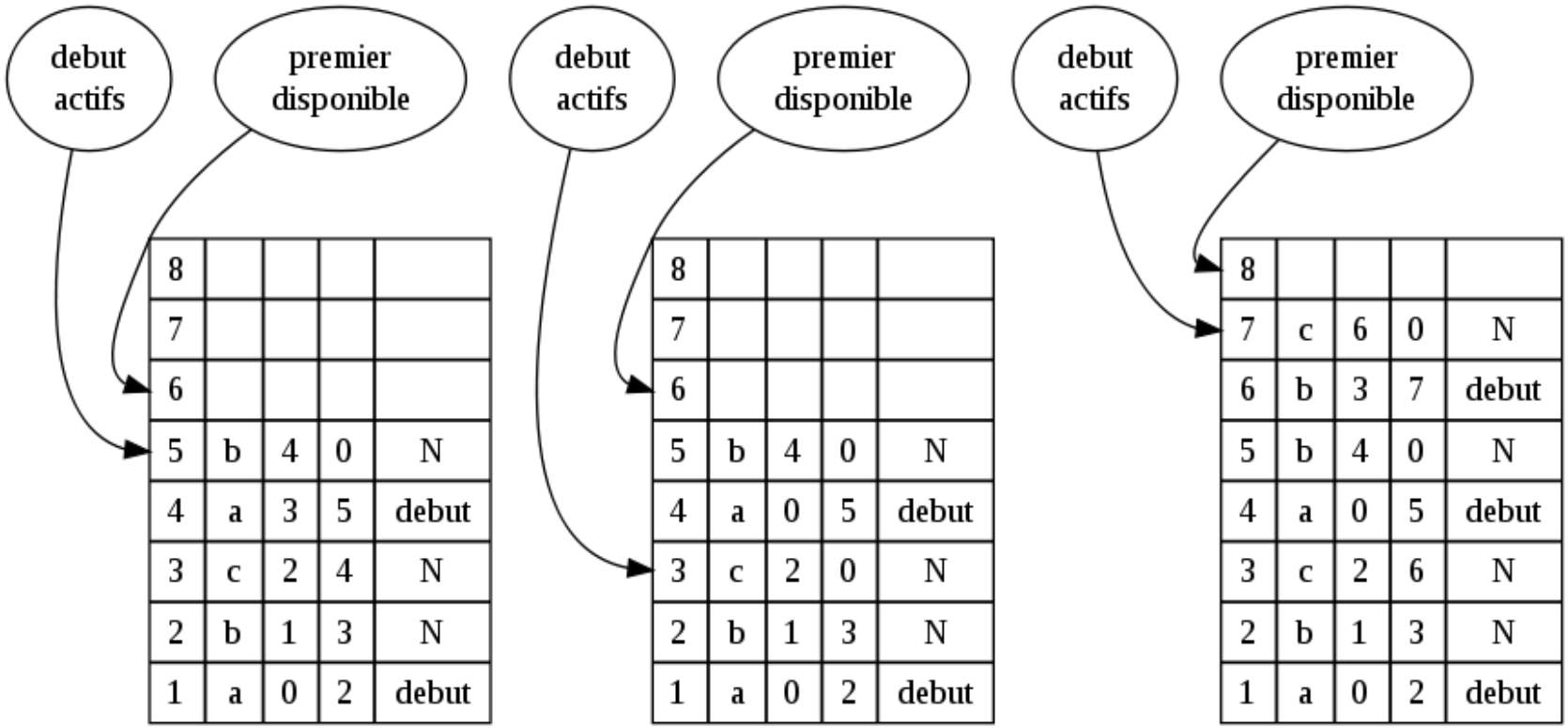
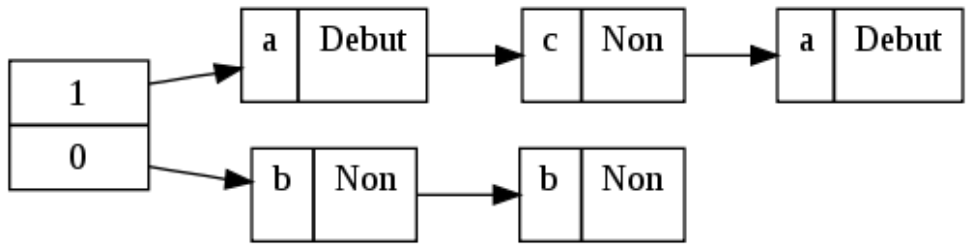


Table d'adressage

Problème :

- les entrées à supprimer peuvent se trouver sur les listes d'hachage différentes.



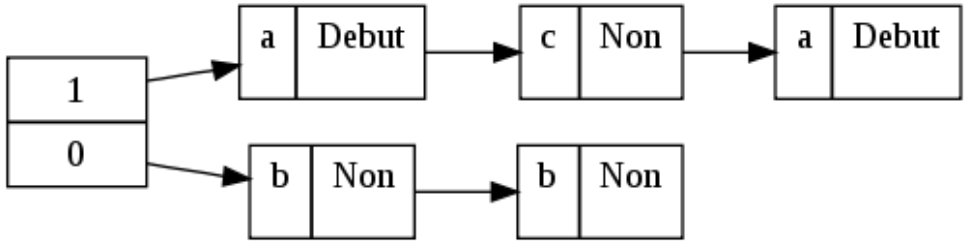
Solution :

- le champs bloc, la pile des blocs :

Table d'adressage

Problème :

- les entrées à supprimer peuvent se trouver sur les listes d'hachage différentes.



Solution :

- le champs bloc, la pile des blocs :