Institut International de Technologie
Cours Techniques de compilation

Introduction à la compilation

Soufiene.Lajmi@gmail.com

Année universitaire 2020-2021

Sommaire

- Généralités sur le cours
- L'utilité de l'étude des compilateurs
- Introduction aux compilateurs
- Langages de programmation
- Structure d'un compilateur

Généralités sur le cours

- Connaissances préalables
  - Culture informatique
  - Théorie des langages
  - Au moins un langage de programmation
- Bibliographie
  - Compilateurs: Principes, techniques et outils. A. Aho, M. Lam, R. Sethi et J. Ullman
- Évaluation
  - % CC + % Examen Final

Plan

- Analyse Lexicale
- Analyse Syntaxique
- Analyse sémantique
- Génération de code

L'utilité de l'étude des compilateurs

- Construire de gros logiciels et des applications intéressantes et de grandes tailles
- Utilisation de la théorie dans un domaine d'application tel que le compilateur
- Apprendre comment construire des langages de programmation
- Savoir comme les langages de programmation fonctionne
- Apprendre des compromis pour la conception de langages

Introduction aux compilateurs

- Compilateurs
- Interpréteurs
- Une brève histoire des compilateurs
  - Premières étapes
  - La programmation coûte très chère
  - 1954- 1957: Apparition du premier compilateur
  - 1958: 50 % des programmes sont en FORTRAN

Introduction aux compilateurs

- Ce premier compilateur a un impact important sur la science de l'informatique
  - Théorie + Pratique
  - Les compilateurs modernes suivent la silhouette de FORTRAN

Langages de programmation

- Plusieurs langages de programmation
- Apparition de nouvelles langages de programmation
- Quel est le meilleur langage?
- **Langages de programmation** :
  - Langage machine
  - Langage assembleur
  - Langage de haut niveau

Langages de programmation

- **Styles de programmation :**
  - **Programmation structurée**
    - Exemples : C, Pascal, Fortran, Basic, Cobol, Algol.
  - **Programmation fonctionnelle**
    - Exemples : Lisp, Scheme, Caml, Haskel.
  - **Programmation logique**
    - Exemples : Prolog, Python.
  - **Programmation orientée objet**
    - Exemples : Java, C++, OCaml, Objective-C, Ada, Simula, Smalltalk.

Langages de programmation

- Étudions l'exemple du calcul de la factorielle :
  - En programmation structurée (C) :

```
#include <stdio.h>
main()
{
   int n, k, res;
   scanf("%d", &n);
   res = 1;
   for(k = 2; k <= n; k++)
        res *= k;
   printf("%d", res);
}
```

Langages de programmation

- En programmation logique (ProLog) :

```
predicates
    factorielle(integer, integer)
clauses
   factorielle(1, 1).
   factorielle(n, res) :- n > 1,
                                  k = n - 1,
                                 factorielle(k, resk),
                                  res = resk * n.
Goal : factorielle(-1, X)
No solution
Goal : factorielle(3, X)
X = 6 Yes
```

Langages de programmation

- **Remarque** : le seul langage qui peut être directement exécutable par l'ordinateur est le langage machine (code binaire).

- **Conclusion** : l'existence des divers langages et styles de programmation suppose l'existence d'un outil pour convertir n'importe quel langage en code binaire : cet outil s'appelle un **compilateur**.

Phases d'un compilateur

- **Modèle de base d'un compilateur**
- Il y a deux parties dans le processus de compilation :
  - Partie frontale (***frontend***).
  - Partie arrière (***backend***).

- **La partie analyse**
- **La partie synthèse**

Phases d'un compilateur

- **Modules de la partie analyse d'un compilateur (Analyseurs lexical, syntaxique, sémantique et générateur de code intermédiaire)**
- L'optimisation de code intermédiaire.
- **Modules de la partie synthèse d'un compilateur (Générateur de code cible)**

L’analyseur lexical

- L’analyseur lexical va générer des unités lexicales.
- Les caractères espaces et le point virgule « ; » sont ignorés par l’analyseur lexical.

L'analyseur lexical

- **Détection des erreurs lexicales**
- **Exemples d'erreurs lexicales (du langage C)**
  - Un « main » écrit en majuscule (Main, mAin, MAIN).
  - Un identificateur qui commence par un chiffre (comme 1position).
  - Un identificateur qui contient un caractère qui n'est pas alphanumérique ou différent du caractère souligné (comme position@, posi(tion, posi tion).

L'analyseur lexical

- **Exemple**
- Soit le texte à analyser : « =+*>a-125 ».
  - Erreurs lexicales.
  - Unités lexicales délivrées.

L'analyseur syntaxique

- Les règles suivants permettent de reconnaître la structure grammaticale de cette expression :

  - (R1) Tout identificateur est une ***expression***;
  - (R2) Tout nombre est une ***expression***;
  - (R3) Si $Expr_1$ et $Expr_2$ sont des ***expressions***, alors :
    - $Expr_1 + Expr_2$
    - $Expr_1 * Expr_2$
    - $(Expr_1)$

    sont des ***expressions***.
  - (R4) Si « Ident » est un identificateur et « Expr » est une expression alors « Ident = Expr » est une ***instruction d'affectation***.

L'analyseur syntaxique

- **Remarques** :
  - L'analyse lexical est une analyse linéaire du fichier source.
  - L'analyse syntaxique est une analyse hiérarchique des unités lexicales.
  - Lorsque l'analyseur syntaxique rencontre une structure qui n'est pas définie par les règles syntaxiques (les règles grammaticales), il signale une erreur syntaxique.
  - Cela se traduit par le fait que l'analyseur syntaxique n'arrive pas à construire l'arbre de dérivation en utilisant les règles syntaxiques dont il dispose.

L'analyseur syntaxique

- Les analyseurs syntaxiques utilisent généralement des **arbres syntaxiques abstraits** (**AST** pour *Abstract Syntax Tree*) à la place des **arbres de dérivation**.
  - Un arbre de dérivation utilise comme étiquettes des nœuds de niveaux supérieurs les noms des entités qui décrivent les règles.
  - Un arbre syntaxique abstrait est une représentation compacte d'un arbre de dérivation. Il n'utilise que les opérateurs et les opérandes :
    - les opérateurs sont les nœuds d'un AST.
    - les opérandes sont les feuilles d'un AST.

L'analyseur sémantique

- L'analyse sémantique contrôle si le programme source contient des erreurs sémantiques et collecte des informations de type destinées à la phase de production de code qui la suit.

  - L'analyseur sémantique prend en entrée l'arbre syntaxique délivré par l'analyseur syntaxique pour identifier les opérateurs et les opérandes des expressions, ainsi que les instructions.

L'analyseur sémantique

- La tâche importante de l'analyseur sémantique est la **vérification de type**.
- **Conversion implicite de type**.
- Détecter les **erreurs d'incompatibilité de types** : opération sur deux types incompatibles
- Détecter certaines **erreurs de calcul**

L'analyseur sémantique

- Détecter les **erreurs de portée des identificateurs** :
- Détecter l'**unicité des identificateurs** : par exemple :
  - une variable doit être déclarée une seule fois dans un même espace de visibilité (ou portée).
  - les étiquettes dans une instruction case (switch) doivent être distinctes.
  - les éléments d'un type « énumération » ne peuvent pas être répétés.
- Détecter certaines **erreurs de flot d'exécution** : un break utilisé sans while (for, do, ou switch).

Le générateur de code intermédiaire

- Après l'analyse sémantique, certains compilateurs construisent explicitement une représentation intermédiaire du programme source.

L'optimiseur de code intermédiaire

- La conversion « entier vers réel » peut être effectuée une fois pour toute lors de la compilation.

Le générateur de code cible

- La phase finale dans la chaîne de compilation est la production de code cible (code en langage assembleur ou en langage machine).

Caractéristiques d'un bon compilateur

- Le code produit est correct (équivalence entre code source et code cible).
- Le code produit est rapide.
- Le temps de réponse du compilateur est rapide.
- Les messages d'erreurs sont précis (nature d'erreur, indication de la place de l'erreur dans le fichier).
- Débogueur offert.
- Le code produit est optimisé (en temps et en espace).

Questions ??