ENSIMAG

Projet Génie Logiciel

Groupe 13

Manuel Utilisateur du Compilateur Deca

Réalisé par :

Benjelloun Otman
Birée Thomas
Boulouz Amine
El Goumiri Rida
Loginova Aleksandra

Année Universitaire : 2024/2025

Contents

1 Introduction

Ce manuel utilisateur est destiné aux utilisateurs du compilateur Deca ayant une connaissance préalable des spécifications du langage. Il décrit les limitations de l'implémentation, les messages d'erreur possibles, ainsi que les instructions d'utilisation.

En cas d'erreur à n'importe quelle étape, le compilateur s'arrête et affiche un message d'erreur précis indiquant la nature du problème rencontré.

2 Utilisation du Compilateur

Pour utiliser le compilateur Deca, utilisez la commande suivante :

```
decac [options] fichier.deca
```

et plus précisément :

```
decac [[-p | -v] [-n] [-r X] <source files>...] | [-b]
```

2.1 Options Disponibles

Le compilateur Deca accepte les options suivantes :

- `-b` (banner) : Affiche une bannière indiquant le nom de l'équipe
- `-p` (parse) : Arrête la compilation après l'étape de construction de l'arbre abstrait et affiche sa décompilation. Si un seul fichier source est compilé, la sortie doit être un programme Deca syntaxiquement correct.
- `-v` (verification) : Arrête la compilation après l'étape de vérifications contextuelles. Ne produit aucune sortie en l'absence d'erreur.
- `-n` (no check) : Supprime les tests à l'exécution suivants :
  - Division par zéro
  - Débordement arithmétique sur les flottants
  - Déréférencement de null
  - Débordement mémoire
- `-r X` (registers) : Limite les registres banalisés disponibles à R0 ... R{X-1}, avec : $4 \leq X \leq 16$
- `-d` (debug) : Active les traces de debug. L'option peut être répétée plusieurs fois pour avoir plus de traces.
- `-P` (parallel) : Lance la compilation des fichiers en parallèle lorsque plusieurs fichiers sources sont fournis.

Notes importantes :

- Les options `-p` et `-v` sont mutuellement exclusives.

- En l'absence des options `-b`, `-p`, une exécution réussie ne produit aucun affichage.
- L'option `-b` ne peut être utilisée qu'en l'absence d'autres options et sans fichier source.
- Si un fichier apparaît plusieurs fois sur la ligne de commande, il n'est compilé qu'une seule fois.

2.2 Exemples d'Utilisation

Voici quelques exemples d'utilisation typiques du compilateur :

```
Afficher la bannière
decac -b
Compiler un fichier avec 4 registres
decac -r 4 programme.deca
Vérifier la syntaxe et afficher l'arbre
decac -p programme.deca
Compiler plusieurs fichiers en parallèle
decac -P prog1.deca prog2.deca prog3.deca
Compiler sans vérification à l'exécution
decac -n programme.deca
Activer les traces de debug
decac -d programme.deca
```

3 Catégories de messages d'erreur

Lorsque le compilateur ne parvient pas à compiler le fichier source, il génère des messages d'erreur précis pour aider l'utilisateur à comprendre et corriger le problème. Les erreurs peuvent survenir à différentes étapes de la compilation :

- **Erreurs de syntaxe** : surviennent lors des phases d'analyse lexicale et syntaxique
  - Les erreurs lexicales sont détectées par ANTLR4 directement.
  - Les erreurs syntaxiques indiquent une violation de la grammaire Deca
- **Erreurs contextuelles** : apparaissent lors de la vérification des règles sémantiques du langage
- **Erreurs d'exécution** : se produisent lors de l'exécution du code assembleur généré

Les sections suivantes détaillent les différents types d'erreurs possibles, leurs causes et les messages associés, classés par catégorie.

Les messages d'erreur indiquent le numéro de ligne et la colonne exacte où l'erreur a été détectée.

4 Messages d'erreurs contextuelles

4.1 Erreurs Liées aux Déclarations

- `"Invalid type for variable declaration:  " + t1`
  Type non valide pour la déclaration de la variable : nom du type.
- `"Invalid type for field declaration:  " + fieldType`
  Type non valide pour la déclaration du champ : nom du type.
- `"The identifier's name cannot be null."`
  Le nom de l'identifiant ne peut pas être nul.
- `"The identifier '%s' is already in use in its context."`
  Une variable a été défini plusieurs fois dans le même contexte.

4.2 Erreurs Liées aux Types et Conversions

- `"Expression of type '" + exprType + "' cannot be assigned to an '" + t + "'"`
  Une expression de type type de l'expression ne peut pas être assignée à un type type cible.
- `"The type '" + this.name + "' is not defined in the environment of types"`
  Le type nom du type n'est pas défini dans l'environnement des types.
- `"The type definition for '" + this.name + "' is invalid or incomplete."`
  La définition du type nom du type est invalide ou incomplète.

- `"Unary 'Not' is not compatible with type '"`
  L'opérateur unaire `Not` *n'est pas compatible avec le type type concerné.*

- `"Variable '%s' must be initialized before cast"`
  La variable spécifiée par %s doit être initialisée avant d'effectuer un cast (conversion de type).

- `"Cannot cast to booleans"`
  Le type ne peut être converti en boolean.

- `"Cannot assign expression of type '%s' to a variable of type '%s'"`
  Une valeur d'un type incompatible est assignée à une variable d'un autre type.

- `"Invalid type for variable declaration:  " + t1`
  Un type invalide a été utilisé pour déclarer une variable.

4.3 Erreurs liées aux Classes

- `"Class %s is already defined"`
  Une classe portant le nom spécifié par %s a déjà été définie dans le même contexte

- `"Class '" + objectType.getName() + "' not found."`
  Classe nom de la classe introuvable.

4.4 Erreurs Liées aux Méthodes

- `"Incorrect number of arguments for method '" + method.getName() + "'"`
  Nombre incorrect d'arguments pour la méthode nom de la méthode.

- `"Method call must be performed on an object of a class type."`
  L'appel de méthode doit être effectué sur un objet de type classe.

- `"%s method override is invalid:  mismatch in return type, expected:  '%s', got '%s'"`
  Incompatibilité entre le type de retour attendu et le type de retour réel fourni dans la méthode redéfinie.

- `"%s method override is invalid:  mismatch in signature"`
  La méthode redéfinie dans la sous-classe a une signature (nom, types et nombre de paramètres) différente de celle de la méthode de la classe parente.

- `"Method %s is already defined in current environment."`
  Une méthode avec le même nom et la même signature a déjà été définie dans le même contexte.

- `"Method call must be performed on an object of a class type."`
  Une méthode a été appelée sur quelque chose qui n'est pas un objet d'une classe (par exemple, une variable de type primitif).

- `"Class '" + objectType.getName() + "' not found."`
  La classe spécifiée par le nom objectType.getName() n'a pas été trouvée dans l'environnement d'exécution.

- `"Incorrect number of arguments for method '" + method.getName() + "'. Expected: " + methodSignature.size() + ", Found: " + args.size()`
  Le nombre d'arguments fournis lors de l'appel de la méthode ne correspond pas au nombre attendu par la signature de la méthode

- `"Argument " + (i + 1) + " of method '" + method.getName() + "' does not match the expected type. Expected: " + expectedType + ", Found: " + argType`
  L'argument à la position spécifiée (i+1) dans l'appel de la méthode n'a pas le bon type.

- `"Invalid type for parameter declaration: " + paramType`
  Le type spécifié pour un paramètre de méthode n'est pas valide.

- `"Identifier %s appears twice in method definition parameters", paramName.getName`
  Le paramètre a été défini deux fois dans la liste des paramètres d'une même méthode.

4.5 Erreurs Liées aux Opérations

- `"Incompatible types for binary operation: "`
  Types incompatibles pour l'opération binaire.

- `"Unary minus is not compatible with type '"`
  L'opérateur unaire ***`Moins`*** *n'est pas compatible avec le type type concerné.*

- `"Operands for modulus operator '%' must both be integers, but got: " + leftType + " and " + rightType`
  Les deux opérandes du modulus (%) doivent être des entiers.

- `"Unary 'Not' is not compatible with type '" + operandType.getName() + "'"`
  L'opérateur unaire "Not" ne peut pas être appliqué au type spécifié.

- `"Invalid operands for 'instanceof' binary operator"`
  L'opérateur binaire 'instanceof' a été utilisé avec des opérandes incorrects.

4.6 Erreurs Spécifiques aux Instructions

- `"Selection must be performed on an object of a class type."`
  La sélection doit être effectuée sur un objet de type classe.

- `"Field or method '" + field.getName() + "' does not exist in class '" + objectType.getName() + "'"`
  Le champ ou la méthode nom du champ ou de la méthode n'existe pas dans la classe nom de la classe.

5 Message d'erreurs d'exécution

Le compilateur génère du code assembleur permettant de détecter et de gérer certains types d'erreurs pouvant survenir à l'exécution. Ces erreurs sont décrites ci-dessous :

5.1 Erreur de dépassement de pile (Stack Overflow Error)

Cette erreur est levée lorsque l'exécution du programme dépasse la capacité de la pile. Elle peut survenir dans des cas de récursions excessives ou de besoins mémoire dépassant les limites définies.

Message d'erreur : `"ERROR: Stack overflow detected, terminating program execution with error."`

Condition de déclenchement : La vérification est activée uniquement si un dépassement de pile est possible.

5.2 Erreur de dépassement (Overflow Error)

Cette erreur est levée lorsqu'une opération arithmétique provoque un dépassement des limites des types de données.

Message d'erreur : `"ERROR: Overflow detected, terminating program execution with error."`

Condition de déclenchement : La vérification est activée uniquement si un dépassement arithmétique est possible.

Note : Les divisions par zéro sont traitées comme des erreurs de dépassement.

5.3 Erreur d'entrée/sortie (IO Error)

Cette erreur est levée lorsque des opérations d'entrée/sortie comme `readInt` ou `readFloat` échouent.

Message d'erreur : `"ERROR: IO error detected, terminating program execution with error."`

Condition de déclenchement : La vérification est activée uniquement si une erreur d'IO est possible.

5.4 Erreur de déréférencement de pointeur nul (Null Dereference Error)

Cette erreur est levée lorsqu'un programme tente d'accéder à un pointeur nul.

Message d'erreur : `"ERROR: null dereference detected, terminating program execution with error."`

Condition de déclenchement : La vérification est activée uniquement si un déréférencement nul est possible.

5.5 Erreur de tas plein (Full Heap Error)

Cette erreur est levée lorsqu'il n'est plus possible d'allouer de la mémoire dans le tas.

Message d'erreur : `"ERROR: heap is full, cannot allocate, terminating program execution with error."`

Condition de déclenchement : La vérification est activée uniquement si un dépassement de la capacité du tas est possible.

6 Limitations du Compilateur

Notre compilateur Deca implémente l'ensemble des fonctionnalités requises, avec quelques limitations et points d'amélioration :

- **Gestion des registres** :
  - Pour la partie sans objet, la gestion du débordement de registres disponibles est complètement implémentée
  - Pour la partie objet, cette gestion est partiellement implémentée. Bien que notre implémentation actuelle semble suffisante pour la plupart des cas d'utilisation, des tests plus approfondis seraient nécessaires pour garantir sa robustesse dans tous les scénarios possibles
- **Tests et validation** :
  - Toutes les fonctionnalités du langage Deca sont implémentées
  - Des tests supplémentaires seraient bénéfiques pour garantir l'absence totale de bugs et assurer une meilleure couverture des cas d'utilisation
- **Optimisations possibles** :
  - Le code généré est fonctionnel mais pourrait être optimisé davantage
  - Des améliorations potentielles incluent l'optimisation de l'utilisation des registres et la réduction de la taille du code généré
  - Ces optimisations n'affectent pas la fonctionnalité du compilateur mais pourraient améliorer les performances des programmes compilés
  - Dans cette optique d'optimisation, nous avons choisi l'extension ARM, largement utilisée dans les systèmes embarqués. Cette extension permettrait d'optimiser la consommation énergétique des programmes compilés, un enjeu crucial pour les applications embarquées modernes

7 Extension ARM pour Deca

L'extension ARM pour Deca permet de compiler des programmes Deca vers l'architecture ARMv7-A, une architecture 32 bits largement utilisée dans les systèmes embarqués. Cette extension vise à offrir une alternative optimisée pour les plateformes ARM, tout en respectant les conventions d'appel et en intégrant des fonctionnalités spécifiques à ARM.

7.1 Objectifs de l'extension

- **Support pour ARMv7-A (32 bits)** : Prise en charge de 16 registres généraux (`R0-R15`) ainsi que des registres flottants (`s0-s31`) et doubles (`d0-d15`).
- **Compilation optimisée** : Respect des spécifications ABI d'ARM pour garantir une exécution efficace et conforme.
- **Compatibilité avec le langage Deca** : Assurer que les fonctionnalités de Deca (instructions, types, conditions, boucles, etc.) soient correctement traduites en assembleur ARM.

7.2 Utilisation de l'option `--arm`

Pour activer la compilation ARM, ajoutez l'option `--arm` lors de l'utilisation du compilateur Deca. Par exemple :

```
decac --arm fichier.deca
```

Cela générera un fichier assembleur ARM avec l'extension `.s`.

Pour exécuter ce fichier avec Ubuntu sur un système ARM ou simulateur compatible, utilisez le script suivant :

```
src/test/script/launchers/run_arm_ubuntu.sh fichier.s
```

Ce script prend en charge la génération du binaire et son exécution sur un simulateur ou une plateforme ARM réelle.

7.3 Fonctionnalités disponibles

- **Support des registres ARM** :
  - Gestion des registres généraux `R0-R15`, y compris des registres spéciaux comme `SP` (pointeur de pile) et `FP` (pointeur de base).
  - Utilisation des registres flottants `s0-s31` et doubles `d0-d15` pour des opérations sur les nombres flottants.
- **Instructions prises en charge** :
  - Génération de code pour les boucles, conditions, affectations et appels de méthodes.
  - Prise en charge des instructions spécifiques à ARM pour les opérations arithmétiques, la gestion des constantes et les impressions avec `printf`.

- **Tests préliminaires** :
  - Simulation des programmes générés à l'aide de `QEMU`.
  - Possibilité d'exécution sur un **Raspberry Pi** pour valider le code sur du matériel ARM réel.

7.4 Limites actuelles

- Les fonctionnalités avancées comme la gestion complète des registres flottants et doubles nécessitent des ajustements supplémentaires.
- Certaines optimisations spécifiques à ARMv7-A restent à implémenter.
- Les tests sont encore en cours d'extension sur du matériel ARM réel, bien que le Raspberry Pi soit déjà intégré pour des validations de base.

7.5 Prochaines étapes

- Finaliser les méthodes de génération de code ARM pour toutes les structures du langage Deca.
- Étendre les tests et valider les performances sur des plateformes ARM réelles.
- Documenter et publier un guide d'utilisation détaillé pour l'exécution sur du matériel ARM.