Manuel Utilisateur

Compilateur Deca

Projet Génie Logiciel

Equipe 12

Baptiste Le Duc - Malo Nicolas - Ryan El Aroud - Mathéo Dupiat - Théo Giovinazzi

1. Utilisation du compilateur

Le programme principal « decac » est un compilateur Deca complet. On permettra de désigner le fichier d'entrée par des chemins de la forme <répertoires/nom.deca> (le suffixe .deca est obligatoire) ; Le résultat sera sous la forme <répertoires/nom.ass> situé dans le même répertoire que le fichier source.

La syntaxe d'utilisation de l'exécutable decac est :
decac [[-p | -v] [-n] [-r X] [-d]* [-P] [-w] <fichier deca>...] | [-b]

La commande decac, sans argument, affichera les options disponibles. On peut appeler la commande decac avec un ou plusieurs fichiers sources Deca. On définira les options suivantes à la commande decac :

- -b (banner) : affiche une bannière indiquant le nom de l'équipe
- -p (parse) : arrête decac après l'étape de construction de l'arbre, et affiche la décompilation de ce dernier (i.e. s'il n'y a qu'un fichier source à compiler, la sortie doit être un programme deca syntaxiquement correct)
- -v (verification) : arrête decac après l'étape de vérifications (ne produit aucune sortie en l'absence d'erreur)
- -n (no check) : supprime les tests à l'exécution spécifiés dans les points 11.1 et 11.3 de la sémantique de Deca.
- -r X (registers) : limite les registres banalisés disponibles à R0 … R{X-1}, avec 4 <= X <= 16
- -d (debug) : active les traces de debug. Répéter l'option plusieurs fois pour avoir plus de traces.
- -P (parallel) : s'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation.

N.B. Les options '-p' et '-v' sont incompatibles.
L'option '-b' ne peut être utilisée que sans autre option, et sans fichier source. Dans ce cas, decac termine après avoir affiché la bannière.
Si un fichier apparaît plusieurs fois sur la ligne de commande, il n'est compilé qu'une seule fois.

Le fichier assembleur généré est ensuite exécutable avec une machine virtuelle ima qui réalise le linkage et l'assemblage du fichier via la commande ima <répertoires/nom.ass>

2. Limitations du compilateur

Le compilateur que nous avons développé respecte fidèlement les spécifications établies dans le document de projet de Génie Logiciel. Chaque étape de son implémentation a été réalisée en stricte conformité avec les exigences décrites, notamment pour l'analyse lexicale, l'analyse syntaxique et la génération de code. Nous avons veillé à ce que l'ensemble des fonctionnalités définies dans la grammaire du langage Deca soit pris en charge de manière complète et précise.

Enfin, le compilateur respecte les spécifications en termes de messages d'erreur, en produisant des retours clairs et conformes aux exigences, notamment pour les erreurs de lexicographie, de syntaxe hors-contexte et contextuelle. Ce souci du détail reflète notre volonté de livrer un outil en parfait accord avec les attentes définies dans le cadre du projet.

S'il existe des portions de langages non ou mal implémentées, alors nous ne nous en sommes pas rendus compte à ce jour et nous nous en excusons.

Nos équipes sont impliquées et réactives et vous pouvez les contacter à tout moment pour signaler tout problème rencontré suite à l'utilisation de notre compilateur.

3. Messages d’erreur

Les messages d’erreur (lexicales, syntaxiques, contextuelles, et éventuelles limitations du compilateur) sont formatés de la manière suivante :
<nom de fichier.deca>:<ligne>:<colonne>: <description informelle du problème>

2.1 Erreurs lexicales

Les erreurs lexicales seront toutes retournées avec le message suivant : line X:X token recognition error at: '...'

Les erreurs lexicales détectables par le parseur sont :

a. Unrecognized character: Aucun token ne correspond.
    - Exemple d'entrée fautive : @ ou ~ ou #(sans include)

b. Unrecognized character: Absence de guillemet fermant.
    - Exemple d'entrée fautive : "Unclosed string

c. Invalid escape sequence in string: Échappement non supporté.
    - Exemple d'entrée fautive : "String with \q escape"

d. Invalid include directive: Nom de fichier sans guillemets.
    - Exemple d'entrée fautive : #include file.deca

2.2 Erreurs syntaxiques hors-contexte

En plus des messages d’erreurs pouvant être générés par ANTLR4 que l’on ne spécifie pas dans cette documentation, mais que l’on peut trouver dans la documentation ANTLR4 Java Runtime Errors.

Messages personnalisés dans la grammaire :

e. InvalidLValue : levée si une expression dans une affectation n'est pas un LVALUE.
    - Exemple d'entrée fautive : 3 = x;.

f. IntNotCodableException : levée si une valeur entière dépasse la plage supportée.
    - Exemple d'entrée fautive : int x = 999999999999999;.

g. IntNotCodableException :levée si un flottant n'est pas décodable.
    - Exemple d'entrée fautive : float x = 1.0e400;.

2.3 Erreurs contextuelles

1. Erreurs sur les opérations binaires, unaires.

h. "Var" + leftType.getName() + " can't be used for arithmetical operation"
   - Lorsque l'opérande n'est ni un entier (int) ni un flottant (float), et qu'une opération arithmétique est tentée.

i. "Vars " + rightType.getName() + " and " + leftType.getName() + " must be boolean values for logical operations (AND, OR)"
   - Lorsque au moins l'un des opérandes n'est pas de type booléen pour les opérations logiques AND ou OR.

j. "Vars " + rightType.getName() + " and " + leftType.getName() + " can't be compared"
   - Lorsque l'opération de comparaison n'est pas une comparaison exacte avec des types compatibles (bool|bool, class|null, null|class), ou lorsque l'un des opérandes n'est ni un entier (int) ni un flottant (float).

k. "Var" + rightType.getName() + " is not an int : modulo impossible"
   - Lorsque l'opérande droit de l'opérateur modulo n'est pas un entier (int).

l. "Var" + leftType.getName() + " is not an int : modulo impossible"
   - Lorsque l'opérande gauche de l'opérateur modulo n'est pas un entier (int).

m. "Var " + operandType.getName() + " can't be used for 'Not'"
   - Lorsque l'utilisateur tente d'appliquer l'opérateur NOT sur une expression qui n'est pas de type booléen.

n. "Var " + operandType.getName() + " can't be used for 'UnaryMinus'"
   - Lorsque l'utilisateur tente d'appliquer un moins unaire sur une expression qui n'est ni un entier (int) ni un flottant (float).

2. Erreurs sur les affectations

a. classTypeRvalue.getName() + " is not subclass of " + classTypeExpected.getName()
    - Lorsque les classes ne permettent pas de réaliser une assignation entre deux objets de type classe.

b. "Expected type " + expectedType + " but found type " + rvalueType
    - Lorsque l'utilisateur tente de réaliser une assignation entre deux objets dont les types ne sont pas compatibles.

c. "Var " + arg.getType().getName() + " can't be printed"
    - Lorsque l'utilisateur tente d'afficher une variable qui n'est pas de type entier (int), flottant (float) ou chaîne de caractères (string).

3. Erreurs sur les conversions de types

a. "Can't cast a void"
    - Lorsque l'utilisateur tente de convertir une expression de type void vers un autre type.

b. " can only cast from a class type or int, float"
    - Lorsque le type source pour une conversion n'est ni une classe, ni un entier (int), ni un flottant (float).

c. " can only cast to a class type or int, float"
    - Lorsque le type cible pour une conversion n'est ni une classe, ni un entier (int), ni un flottant (float).

d. "Can't cast those expressions"
    - Lorsque l'utilisateur tente de convertir entre deux classes qui ne sont pas dans une relation d'héritage.

4. Erreurs sur la déclaration de variable, classe, champ, méthode, paramètre

a. "Class with the same name already existing"
   - Lorsque l'utilisateur tente de déclarer une classe portant un nom déjà utilisé.

b. "Can't declare a field " + name.getName() + " with void type"
   - Lorsque l'utilisateur tente de déclarer un champ de type void.

c. name.getName() + " must be declared as a Field "
   - Lorsque l'utilisateur tente de déclarer un champ portant le nom d'une méthode existante dans une superclasse.

d. methodName.getName() + " can not be overloaded"
   - Lorsque l'utilisateur tente de surcharger une méthode avec une signature ou un type de retour incompatible.

e. "Can't declare a param with void type"
   - Lorsque l'utilisateur tente de déclarer un paramètre de type void.

f. varName.getName() + " : can't declare var with type void"
   - Quand on essaie de déclarer une variable de type void

g. "Variable " + varName.getName() + " already declared in this context"
   - iuqscfhuihqsuichqsui

h. "Method " + var.getName() + " already declared in this class"
   - Lorsque l'utilisateur tente de déclarer une méthode portant un nom déjà utilisé dans la classe.

i. "Field " + var.getName() + " already declared in this class"
    - Lorsque l'utilisateur tente de déclarer un champ portant un nom déjà utilisé dans la classe.

j. "Parameter " + e.getName() + " already declared as a parameter in this method"
    - Lorsque l'utilisateur tente de déclarer un paramètre portant un nom déjà utilisé dans la méthode.

k. "Must use a class, not a " + newType.getName() + " to create a new object"
    - Lorsque l'utilisateur tente de créer un objet avec un type qui n'est pas une classe.

l. "Type " + name.getName() + " is not defined"
    - Lorsque l'utilisateur déclare une variable avec un type inconnu.

m. "methodName.getName() + is already declared as a field in a parent class"
    - Lorsque l'utilisateur tente de déclarer une méthode portant un nom déjà utilisé dans un champ d'une classe parent.

n. "Method declared in field environment"
    - Lorsque l'utilisateur tente de déclarer une méthode portant un nom déjà utilisé dans un champ dans la même classe.

5. Autres erreurs

a. "Can't use instanceof with this type " + rightOperand.getName()
    - Lorsque l'utilisateur tente d'utiliser l'opérateur instanceof alors que le type de gauche n'est ni une classe ni null, ou lorsque le type de droite n'est pas une classe.

b. "method " + rightOperand.getName() + " can only be called on class types"
    - Lorsque l'utilisateur tente d'appeler une méthode sur un objet dont le type n'est pas une classe.

c. "can't return a void expression"
    - Lorsque l'utilisateur tente de retourner une expression de type void.

d. "Can not use this outside of a class"
    - Lorsque l'utilisateur tente d'utiliser l'expression this en dehors d'une classe.

e. "Variable " + name.getName() + " is not declared"
    - Lorsque l'utilisateur tente de manipuler une variable (affectation, sélection, appel de méthode, etc.) qui n'a pas été définie.

f. Condition must be booleanType
    - Lorsque l'utilisateur tente d'utiliser une expression non booléenne comme condition d'un if ou d'un while.

g. rightOperand.getName() + " method have " + sig.size() + " arguments but " + params.size() + " are given"
    - Lorsque l'utilisateur tente d'appeler une méthode avec un nombre de paramètres différent.

2.4 Erreurs dans l'exécution du code assembleur

a. Error: Stack Overflow
    - Déclenché lors d'un dépassement de la pile. Géré par generateStackOverflowError

b. Error: Input/Output error
    - Déclenché lors d'une erreur d'entrée/sortie. Géré par generateIOError.

c. Error: Overflow during arithmetic operation
    - Déclenché lors d'un dépassement arithmétique dans une opération arithmétique sur les flottants. Géré par generateOverflowError.

d. Error: Division by zero
    - Déclenché lors d'une division entière ou flottante par zéro. Géré par generateDivideByZeroError.

e. Error: Heap Overflow
    - Déclenché lors d'un dépassement du tas (heap). Géré par generateHeapOverflowError.

f. Error: Deferencing a null pointer
    - Déclenché lors de l'accès à un pointeur null. Géré par generateNullPointerError.

g. Error: Exit from method without return
    - Déclenché si on sort d'une fonction dont le type de retour n'est pas void sans être passé par une instruction return. Géré par generateNoReturnError.

h. Error: Cast invalid
    - Déclenchée lorsqu'un cast explicite est tenté entre deux types incompatibles ou lorsque l'objet à caster ne correspond pas au type cible attendu dans une hiérarchie de classes.. Géré par generateCastError.

4. Mode opératoire pour l'extension ARM

Au jour du rendu du décompilateur, le lundi 20 janvier 2025, l'extension ARM n'est pas opérationnelle. Elle le sera au moins pour le langage sans objet au moment du rendu le vendredi 24 janvier 2025. Cette partie de la documentation sera mise à jour à ce moment-là.

Pour générer un fichier assembleur pour arm, il suffira d'exécuter la commande :
decac –arm nomdufichier.deca

5. Limitations de l'extension ARM

De même, ce paragraphe sera complété d'ici la soutenance et le rendu de l'extension ARM afin d'être à jour sur notre livrable.