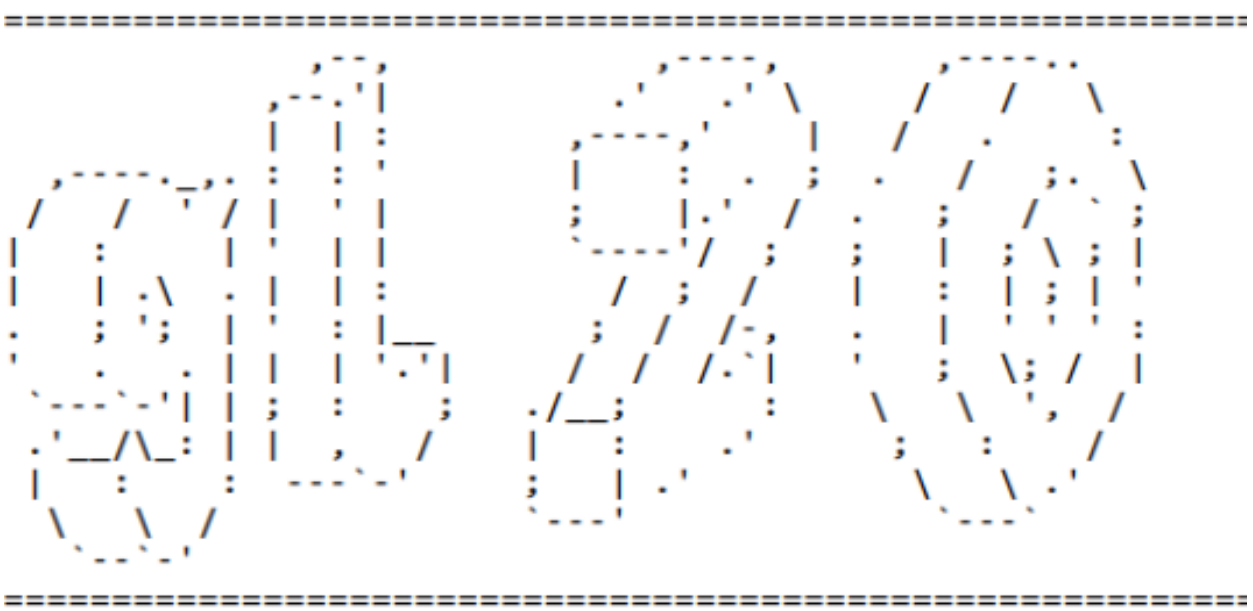
Documentation utilisateur du compilateur Deca
- Groupe GL20



Melvin Bonamour
Steve Lening
Mathieu Meunier
Paul Grassi
Emmanuel Loroux

Table des matières

I - Comportements propres au compilateur et limitations

Le compilateur s'utilise avec la commande **decac**.

Le fichier généré sera un fichier assembleur nommé "**<nom_fichier_source>.ass**" dans le même répertoire que le fichier source deca.

Sa syntaxe est la suivante :

decac [[[-p | -v] [-n] [-r X] [-d]* [-P] [-w] | [-B] [-n] [-d]* [-P] [-w]] <fichier deca>...] | [-b]

Liste des options :

-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 vérification (ne produit aucune sortie en l'absence d'erreur).

-B (BYTECODE) : Compile le programme en Bytecode Java au lieu de le compiler en assembleur. (spécification dans la partie Extension).

-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.

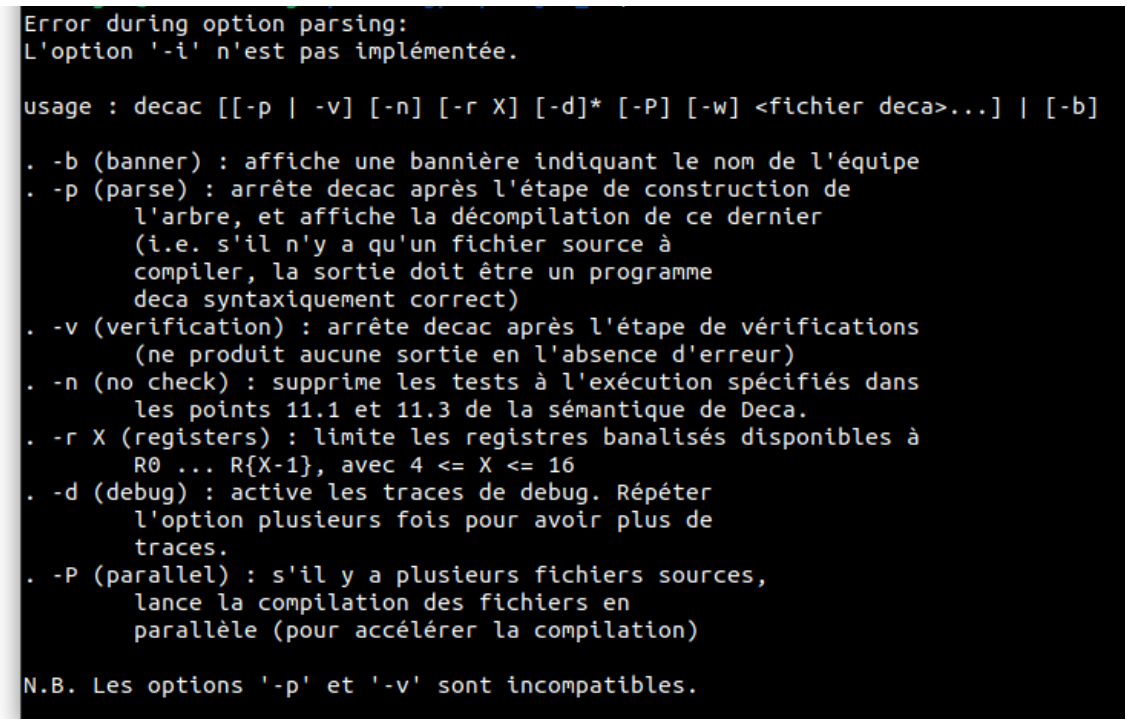
-P (parallel) : S'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation).

De plus, les options -p et -v -B sont incompatibles deux a deux et les options -r et - B sont incompatibles.

Limitations :

- **Le display usage ne s'affiche pas quand on lance decac tout seul** :

  La spécification du projet voudrait que la commande **decac** sans option et sans paramètre affiche le display usage, ce que nous avons pas ajouté, mais par contre, si l'utilisateur renseigne une option qui ne figure pas dans la liste des options précédentes (Exemple : **decac -i**), on lui affiche une mini-doc de la commande **decac** qui explique comment cette commande fonctionne. Voici ce que l'utilisateur obtient dans son terminal lorsqu'il exécute la commande **decac** avec une option erronée (commande qui n'existe pas dans notre compilateur), à l'instar de **-i** :



- **Des erreurs de lexer affichent un message ‘null’ :**
  Normalement les erreurs de lexer doivent afficher un message parlant qui permet à l'utilisateur de savoir à quel niveau se trouve l’erreur dans son code Deca. En l'état, notre compilateur affiche null mais le message que nous souhaitions afficher est : “Erreur de syntaxe : Un token inconnu à la position indiquée empêche la compilation”.
- **Cast et instanceof ne sont pas implémentés :**
  Il n’est pas possible pour le moment de faire des conversions de type (**cast**) et de tester qu’un objet est une instance d’une classe (**instanceof**). Le compilateur en l'état ne prend en compte que le langage Deca essentiel, mais nous envisageons d'implémenter les parties manquantes à savoir le cast et le instanceof afin d’offrir à l'utilisateur un compilateur complet qui puisse prendre en compte le langage Deca complet.

II - Liste des messages d'erreur

Préambule :

Les messages d'erreurs sont tous formatés de la même manière :
« <chemin/vers/source.deca>:<ligne>:<colonne>: <Message d'erreur> »

Messages d'erreur :

Les différentes erreurs possibles sont listées ci-dessous, triées par type. Si vous rencontrez une erreur, cherchez le message que vous obtenez pour avoir une explication et une solution.

- ***Erreurs de syntaxe***

Un message d'erreur de syntaxe s'affiche lorsqu'un problème est repéré dans les symboles utilisés dans le fichier source (caractères utilisés ou association de ceux-ci).

- **null (devrait afficher un message spécifique, voir I-Limitations)**

Cause : Un caractère non supporté (voir liste des caractères supportés) apparaît dans le programme à la position indiquée, ou bien une chaîne de caractères n'est pas fermée.
Exemple : int ß;
Correction : Si le caractère est un guillemet : ", vérifier que la chaîne de caractères est bien délimitée à son début et à sa fin par des guillemets. Sinon il faut changer ou retirer le caractère désigné.

- **no viable alternative at input '<identifiant>'**

Cause : Le programme présente un mot inconnu du langage.
Exemple : int x; *en dehors du bloc principal.*
Correction : Si le mot désigné est une déclaration de variable (ou de champ), vérifiez qu'il se trouve bien à l'intérieur d'une classe ou du bloc principal, si c'est un mot réservé (voir liste des mots réservés) du langage, vérifiez qu'il n'est pas utilisé pour autre chose que son usage réservé et utilisé au bon endroit.

- **missing <symbole> at <symbole>**
- **extraneous input <symbole> expecting <symbole>**

Cause : Il manque sûrement un point virgule ou une parenthèse ouvrante ou fermante pour une déclaration, appel de méthode …

Exemple : print(“hello”;
Correction : Il faut vérifier la présence d’un point virgule à la fin de la ligne, le bon nombre de parenthèses fermantes, …

- **mismatched input <symbole> expecting …**

Cause : Il manque une expression, pour une assignation ou un opérateur arithmétique par exemple
Exemple : print(3 +); *Il manque l’expression à droite du +*
Correction : Vérifier que l’opérateur est appliqué au bon nombre d'expressions.

- **left hand side of assignment is not an lvalue**

Cause : Tentative d’assignation d’une valeur à un objet auquel on ne peut pas assigner, comme un appel de méthode par exemple;
Exemple : print(x) = 4;
Correction : retirer l’assignation.

- **include file not found**

Cause : le nom de fichier est mal orthographié, n’est ni dans le même répertoire que le fichier source ni dans la bibliothèque standard ou n’existe pas.
Correction : Vérifier l’orthographe et l’emplacement du fichier.

● ***Erreurs de contexte***

Un message d’erreur contextuelle est dû à un problème de typage des expressions du programme, par exemple une mauvaise assignation, un opérateur arithmétique mal utilisé, une variable non déclarée.

- **L’opérande de gauche/droite de [opérateur] doit être de type ‘int’.**

Causes : utilisation de l’opérateur modulo avec un des deux opérandes qui n’est pas un int.
Correction : Il peut être corrigé en vérifiant le type des opérandes.

- **L’opérande gauche/droit de [opérateur] doit être de type ‘int’ ou ‘float’.**

Causes : utilisation d’un opérateur arithmétique ou de comparaison avec un des deux opérandes qui n’est pas un int/float.
Correction : Il peut être corrigé en vérifiant le type des opérandes.

- **L’opérateur [opérateur] doit être appliqué à un type ‘int’ ou ‘float’.**

Causes : utilisation de l'opérateur unaryMinus avec le mauvais type.
Correction : Il peut être corrigé en vérifiant le type de son opérande.

- **L'opérateur new doit être appliqué à une classe.**

Causes : utilisation de l'opérateur new sur un élément qui n'est pas une classe.
Correction : Il peut être corrigé en vérifiant le type de son opérande.

- **Types non compatibles pour l'assignation, l'opérande droit n'est pas un sous-type de l'opérande gauche.**

Causes : utilisation d'une assignation pour les classes.
Correction : Il peut être corrigé en vérifiant le type des deux classes.

- **Types non compatibles pour l'assignation.**

Causes : une assignation entre deux variables de types différents (pas float).
Correction : Il peut être corrigé en vérifiant le type des variables.

- **Types incompatibles pour l'assignation, on ne peut pas assigner null à un int, float ou boolean.**

Causes : assignation de null. Une méthode retourne peut être void.
Correction : Il peut être corrigé en retirant l'assignation à null

- **Une condition doit être de type booléen.**

Causes : utilisation d'une condition avec une valeur autre que booléenne. Une méthode retourne peut être le mauvais type.
Correction : Il peut être corrigé en vérifiant le type de la condition.

- **Les arguments de print doivent être de type 'int', 'float' ou 'string'**

Causes : utilisation de print pour une variable d'un autre type.
Correction : Il peut être corrigé en vérifiant le type de cette variable.

- **Impossible de convertir l'expression en float.**

Causes : on tente de convertir autre chose qu'un int en float. Potentiellement lors d'une assignation(ex: float a = true).
Correction : Il peut être corrigé en vérifiant le type de l'opérande droit de l'assignation.

- **La classe [super classe] n'existe pas..**

Causes : La classe étend une classe qui n'existe pas ou qui est défini après elle-même

Correction : Il peut être corrigé en vérifiant l'orthographe de la super classe ou en la créant avant.

- **Déclaration de [champ/paramètre] invalide (type void).**

Causes : tentative de déclaration d'un paramètre ou d'un champ de type void.(ex void x; ou void a(void x){})
Correction : Il peut être corrigé en changeant l'implémentation de vos méthodes et champs.

- **[champ] override un identifiant qui n'est pas un champ.**

Causes : Une classe étend une autre en définissant à nouveau un de ses champs mais en le changeant en méthode.
Correction : Il peut être corrigé en changeant le nom de la méthode ou du champ.

- **[champ/méthode/paramètre] déjà déclaré dans le contexte courant.**

Causes : Double définition d'un identifier.
Correction : Il peut être corrigé en supprimant une des deux définitions.

- **Impossible de déclarer une variable de type void.**

Causes : tentatives de déclaration d'une variable de type void.
Correction : Il peut être corrigé en changeant l'implémentation de cette variable

- **Identifiant déjà déclaré dans le contexte courant.**

Causes : Double définition d'un identifier.
Correction : Il peut être corrigé en supprimant une des deux définitions.

- **Cet identifier n'a pas de définition dans l'environnement, a-t-il bien été déclaré?**

Causes : utilisation d'un identifier qui n'a pas été déclaré.
Correction : Il peut être corrigé en déclarant cet identifier ou en vérifiant l'orthographe.

- **Cet identifier n'est pas un type de l'environnement. Vérifier l'orthographe.**

Causes : utilisation d'un type(probablement de classe) qui n'a pas été déclaré.
Correction : Il peut être corrigé en déclarant cette classe ou en vérifiant l'orthographe

- **Ce type à deux définitions incompatibles.**

La grammaire spécifie cette condition mais nous n'avons compris ce qui peut la provoquer

- **[method] n'existe pas.**

Causes : la méthode n'est pas déclarée.

Correction : Il peut être corrigé en vérifiant l'orthographe de la méthode ou de la classe, ou en déclarant la méthode.

- **trop/pas assez d'arguments pour la méthode.**

Causes : mauvais nombre d'arguments pour la méthode.

Correction : Il peut être corrigé en vérifiant votre implémentation de la méthode.

- **On ne peut pas retourner un type void.**

Causes : on tente de retourner un objet de type void.

Correction : Il peut être corrigé en vérifiant l'implémentation de la méthode ou de l'objet en question.

- **[objet] n'est pas une classe. On ne peut récupérer le champ que sur une classe.**

Causes : on tente de récupérer un champ sur autre chose qu'une classe.

Correction : Il peut être corrigé en vérifiant l'orthographe de l'opérande.

- **On ne peut pas accéder à ce champ dans le main. (champ protégé)**

Causes : On tente d'accéder à un champ protégé dans le main.

Correction : Il peut être corrigé en modifiant la visibilité du champ ou en implémentant un getter.

- **Le type de l'objet concerné par le champ doit être une sous-classe de la classe courante.**

Causes : on tente d'accéder à un champ protégé d'un objet d'une classe qui n'est pas une sous classe de la classe courante.

Correction : Il peut être corrigé en vérifiant l'appel de ce champ.

- **Impossible d'accéder à ce champ : la classe courante n'est pas une sous-classe de la classe propriétaire du champ protégé.**

Causes : on tente d'accéder à un champ protégé d'un objet dont on hérite pas.

Correction : Il peut être corrigé en vérifiant l'appel de ce champ.

- **Impossible d'utiliser this dans le programme principal.**

Causes : utilisation de this dans le programme principal.

Correction : Il peut être corrigé en enlevant this.a

III - Extension du compilateur Deca - Génération de bytecode JAVA

● Fonctionnement de l'extension

La génération de Bytecode Java par le compilateur decac - gl20 se base sur la librairie ASM, ce qui le rend plus avantageux en termes de performance que d'autres compilateurs qui se baseraient sur BCEL ou Javassist. Elle peut être activée grâce à l'option "**decac -B"** du compilateur. Les fichiers générés seront générés dans le répertoire courant (où a été lancée la commande **decac -B**). Le bytecode est généré dans un fichier "**Main.class**", il peut aussi être récupéré sous forme de suite d'instructions JVM dans le fichier "**<nom_fichier_source>_class**".

Vitesse et coût de la génération de bytecode:

La génération de bytecode grâce au compilateur decac -gl20 se base sur la Core API de la librairie ASM. Elle est donc beaucoup plus rapide et a un coût en mémoire beaucoup plus faible qu'une autre qui utiliserait la Tree API d'ASM.

● Limitations

Le Bytecode Java peut être généré pour l'ensemble du compilateur Deca sans objet. Il ne peut pas être généré pour le langage Deca avec objet. Aussi, l'option "**-B**" ne peut être utilisée que pour un fichier deca à la fois.

Note sur la vitesse:

Bien que l'utilisation de la Core API d'ASM rende la génération du bytecode beaucoup plus rapide et moins coûteuse en mémoire, le calcul des frames ayant été généré automatiquement par la librairie, le coût de la génération de bytecode avec le compilateur decac -gl 20 est environ deux fois plus lente qu'une génération avec un compilateur qui calculerait les frames de manière manuelle tout en utilisant la Core API.

IV - Annexes

Liste des caractères supportés

Les caractères supportés sont les suivants :

- Les chiffres et les lettres de l'alphabet latin sans accent : **0-9 a-z A-Z**
- Les opérateurs arithmétiques suivants : **+ - / * = < > %**
- Les symboles de ponctuation suivants : **! . : ; , ( ) { } $ _**
- Le caractère d'espace et les caractères de formatage défini par la norme ASCII par : **\n \r \t**

Liste des mots réservés

asm	*class*	*extends*	*else*	*false*
if	*instanceof*	*new*	*null*	*readInt*
readFloat	*print*	*println*	*printx*	*printlnx*
protected	*return*	*this*	*true*	*while*

Bien que l' opération *instanceof* ne soit pas gérée par le compilateur (voir limitations), il s'agit quand même d'un mot réservé du langage deca, il ne pourra pas être utilisé dans le programme.