Manuel d'utilisateur

Compilateur du langage Deca

Equipe : gl07

Naima AMALOU Yidi ZHU Jiayun Po Zaineb Tiour An Xian

27 Janvier 2020

Table des matières

1 A propos de notre compilateur

Notre programme est un compilateur pour un sous langage de Java qui s'appelle DECA. A partir d'un fichier DECA, il génère un fichier .ass en assembleur qui est destiné à etre lu par une machine IMA.

Le document suivant est un manuel d'utilisateur pour vous faciliter la prise en main de ce compilateur en vous offrant une vue d'ensemble sur la totalité de ces caractérisques ainsi que plusieurs exemples concrets.

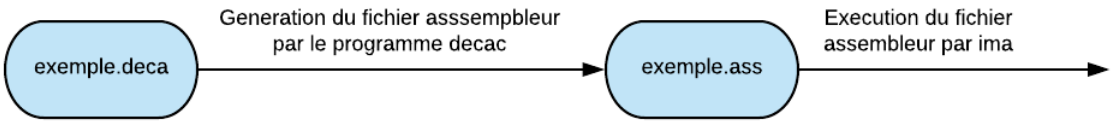
2 Configuration requise

Notre programme est destiné à être exécuter sous un systéme d'exploitation dérivé d'UNIX.

Il faudra avoir au minimum Java 1.8.

Le projet est compilé avec l'outil Maven (commande mvn sous Unix). Le fichier pom.xml présent dans le répertoire principale est le fichier de configuration de Maven.Cet outil utilise un ensemble de plugins (téléchargés automatiquement à la première exécution avec *mvn compile*) pour compiler (et tester avec *mvn test-compile*) les différents fichiers relatif à notre projet.

3 Commandes pour lancer l'éxécution de notre programme



Pour exécuter un fichier exemple.deca, il faut exécuter les commandes suivantes dans un terminal ouvert dans le répertoire src/main/bin/ :

- ./decac exemple.deca
- ima exemple.ass

La première commande sert à générer le fichier assembleur relatif au fichier .deca donné en entrée et la deuxiéme sert à l'éxecuter avec ima.

4 Les options du compilateur

La syntaxe d'utilisation de l'exécutable decac est :

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

-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 de débordement à l'exécution. - débordement arithmétique - débordement mémoire - déréférencement de null

-r X (registers) : limite les registres banalisés disponibles à R0 ... RX-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)

5 Aide

Pour afficher les options du compilateur et leurs rôles au cas où vous aviez besoin d'aide, placez vous dans le dossier bin et lancer l'exécutable ./decac.

6 Les limites du compilateur

Les limites au niveau syntaxique

Notre compilateur possède un nombre limite d'instructions qui peut gérer , cette limite n'a pas été géré par notre compilateur , le compilateur est incapable d'afficher un message d'erreur à l'utilisateur. Une exception de Java est affiché lorsque l'utilisateur produit un programme deca avec un très grand nombre d'instructions. Par exemple : un programme deca avec 1500 boucles imbriquées de IF .

Les limites au niveau de la générations de code

l'algorithme utilisé pour la déclaration des champs d'une classe et ses méthodes n'est pas très stable, ce qui peut générer des erreurs.

7 Les types d'erreurs gérés (avec des exemples)

Erreurs de syntaxe hors-contexte

- **left-hand side of assignment is not an lvalue** : Cette erreur est levée quand l'expression à gauche d'une affectation n'est pas une lvalue.
- [**ParseINTError**] **literal values cannot be infinite** : Cette erreur est levée quand le programme deca contient un entier trés grand.
- [**ParseFloatError**] **literal values cannot be infinite** : Cette erreur est levée quand le programme deca contient un entier très grand.
- **Circular include for file <fileName>.decah** : Une inclusion infinie de fichiers <fileName>.decah .
- **<fileName>.decah : include file not found** : Le fichier <fileName> n'est pas trouvé , cette erreur peut être générée si le fichier est mal placé ou l'utilisateur a oublié de relancer la commande mvn compile aprés la création de son fichier.

Erreurs de syntaxe contextuelle

— **[SymbolError] : <name> is not defined** : L'utilisation d'un symbole non définie.

— **[TypeError] : The type <name> is not predefined** : L'utilisateur utilise un type non-prédéfini , sachant que la liste des types prédéfinie est : void,int,float,boolean, Object et les classes définies par l'utilisateur.

— **[ClassDeclarationError] : This class <name> is already defined** : Double déclaration de classe <name>.

— **[ClassDeclarationError] : the upper class <super> is not predefined** : Cette erreur est levée lorsque l'utilisateur essaye de faire un extends avec une classe non definie précédement.

— **[DeclFieldError] Double definition of <fieldName>** : erreur relative à une double déclaration de champs ou bien quand un champ est déjà déclaré comme méthode.

— **[DeclFieldError] : <fieldName> is not defined as field in the superior class** : le champ déclaré n'est pas un champ dans la classe supérieur.

— **[DeclFieldError] : Field type must be different from void** : un champ déclaré doit être différent de void.

— **[DeclMethodError] Double definition of <methodName>** : Double déclaration de méthode ou une méthode déjà déclarée comme un champ.

— **[DeclMethodError] : <methodName> is not a method in the class <superClass>** : la méthode redéfinie n'est pas une méthode dans les classes supérieur.

— **[DeclMethodError] : The signature of the method <methodName> is different from that defined in the super class** : Signatures différentes lors d'une redéfinition d'une méthode.

— **[DeclMethodError] : The return type of <methodName> is not a subtype of the higher class method definition (<superTypeName>)** : le type de retour de le redéfinition de la méthode n'est pas un sous type de l'ancienne définition de la méthode.

— **[DeclParamError] : Parameter type must be different from void** : le type d'un paramètre déclaré doit être différent de void.

— **[DeclParamError] : Parameter Type is not predefined** : Le type de paramètre doit être prédéfini.

— **[DeclParamError] Double definition of <paramName>** : Double définition d'un paramètre d'une méthode.

— **[InitializationError] : <type2> is not a subtype of <type1>** Initialisation avec un type incompatible.

— **[InitializationError] : <name> is a method and not a variable** Initialisation d'une variable avec une définition de méthode.

— **[DeclVarError] : Variable type must be different from void** : Un type de variable déclarée doit être différent de void.

— **[DeclVarError] : Double definition of <name>** : Variable déja déclarée.

— **[ReturnError] : This method returns nothing** : Une méthode de type de retour Void ne retourne rien.

— **[AssignCompatibleError] : <type1> and <type2> are not compatible for assignment** : Un type de retour incompatible avec le type déclaré.

— **[ConditionError] : <type> must be a boolean** : Le type de retour d'une condition doit être un boolean.

— **[PrintError] : A method cannot be printed** Une méthode ne peut pas être imprimée.

— **[PrintError] : <type> cannot be printed** On ne peut afficher que des entiers, des flottants et des strings.

— **[ArithmeticOperationError] : An arithmetic operation cannot be performed between <type1> and <type2>** : Une opération arithmétique n'est pas pas possible entre les deux types.

— **[ReadIntError] : an integer is expected** : Un entier est attendu.

— **[ReadFloatError] : a float is expected** : Un float est attendu

— **[UnaryMinusError] : A unary minus cannot be done on <type>** L'opération - n'est pas possible sur la variable de type <type>.

— **[UnaryNotError] : Not cannot be done on <type>** : L'opération Not n'est pas possible sur la variable de type <type>.

— **[CastError] : Impossible to cast <type1> in <type2>** L'opération de Cast n'est pas possible entre les deux types.

— **[InstanceOfError] : The InstanceOf operation cannot be performed between <type1> and <type2>** : L'opération de InstanceOf n'est pas possible entre les deux types.

— **[MethdoCallError] : the class <name> is not defined** L'utilisateur essaye d'appeler une méthode que la classe <name> ne possède pas.

— **[MethdoCallError] : <name> is not a class** L'utilisateur essaye d'appeler une méthode d'un objet qui n'est pas une classe.

— **[NewError] : <className> is not defined** : Création d'une classe qui n'est pas prédéfinie.

— **[NewError] : <name> is not a class** : L'opération New est utiliser uniquement pour créer des objets de type class, les autres types ne sont pas possible.

— **[ThisError] : This must be applied inside a class** This est utilisé en dehors de la classe.

— **[SelectionError] : The class <name> is not defined** : L'opération de sélection nécessite un identifiant de class déjà prédéfinie.

— **[FieldAccessError] <fieldName> is protected in the class <className>**
[SelectionError] : <name> is not a class

— **[ExprError] : The type parameter type <type> is not compatible with the expected type** : Vérification de la signature de la méthode lors de son appel.

— **[BooleanOperationError] : Boolean operation cannot be performed between <type1> and <typ2>** : Erreur levée pendant les opérations binaires incompatibles.

— **[ComparisonOperationError] : Comparison operation cannot be performed between <type1> and <type2>**
— **[FieldCallError] : The <fieldName> is not declared neither in the current class nor in its super Classes**

— **[FieldCallError] : <fieldName> is not a field**

— **[MethodCallError] : <methodName> is not declared neither in the current class nor in its super Classes**

— **[MethdoCallError] : the class is not defined**

— **[MethodCallError] : <methodName> is not a method**

— **[MethodCallError] : The method called has a different signature than that of its call** : La signature utilisée pour appelé la méthode est différente de celle utilisé lors de la définition de la méthode.

— **[AsmMethodError] : the ASM method must only contains strings**

— **[ModuloOperationError] : Modulo operation can be performed only between int and int**

— **[UnaryOperationError] : <type> is not an integer or float**

— **[UnrechableCode] : Return must be the last instruction**

Erreurs pendant l'exécution du code assembleur

— **Stack_overflow :** Erreur levée quand il y'a un débordement de pile.

— **Arithmetic_overflow :** Erreur levée si il y'a un débordement arithmétique dans une opération dans le programme de l'utilisateur.

— **[RegisterError] There are not enough registers :** Erreur levée si l'utilisateur lance le compilateur avec l'option -r et lui donne un nombre plus que 15 registres.

— **Erreur : bad cast from <type1> to <type2> :** Erreur levée quand un cast est fait d'un type à un autre qu'il n'est pas son sous-type.

— **Dereferencement nul :** Erreur levée quand une action est faite sur une variable non intialisée ou un attribut qui est éventuellement Null.

— **Varaible_non_definie :** Erreur levée quand une variable utilisée est non définie.

— **[ReturnError] : the method <method> is expecting a return but has none. :** Erreur levée quand une méthode a un type de return bien définie mais qui n'a pas de return dans son body.

8 Extension

8.1 Utilisation d'une bibliothèque standard

Pour ajouter une bibliothèque standard, l'utilisateur devrait faire un include dans l'entête du fichier .deca. De plus, la bibliothèque standard doit être au format de deca et nom avec l'extension de fichier,. decah. Tous les fichiers sans le nom .decah seront refusée. Un fichier decah utilisé doit être impérativement dans le fichier src/main/resources/include.

Dans la bibliothèque standard, les fonctions sont écrites dans une classe nommée (voir l'exemple ci-dessous), et les fonctions sont nommés avec un tiret en bas comme premiére lettre. Les fonctions peuvent avoir un type de retour (soit float, int ou boolean) ou sans un type de retour (void), sachant qu'aucun changement sera fait sur les variables passées comme paramètres dans le programme principal.

Un exemple de decah :

```
Class Hello{
    void _kitty(int x){
        X=x+1;
    }
    float _superman(){
        return  22.3;
    }
}
```

Utilisation d'une bibliothèque standard dans un programme principale

```
#include " Bibliothque -standard. decah"
{
    //constructeur de la classe de bibliotheque standard
    Hello a=new Hello();
    float k ;
    //appelation une methode dans le blibliotheque standard
    a._ kitty(5);
    //une methode avec une valeur retournee
    k=a._ superman();
}
```

Utilisation des fonctions trigonométriques

Les fonctions de trigonométrique sont codées dans le fichier Math.decah. Pour pouvoir utiliser les fonctions cosinus, sinus, Arctangente, Arcsinus et ulp, l'utilisateur doit faire un include dans l'entête du fichier relatif à son programme.

include "Math.decah"

Et l'exemple de l'utilisation de différents fonctions est suivant :

```
# include "Math.decah"
{
    Math meth= new Math();
    int x=23;
    float y ;
    float k ;
    y=meth ._cos(x);
    y=meth ._sin(x);
    y=meth ._arcsin(x);
    y=meth ._arctan(x);
    y=meth ._ulp(k);
}
```

Les limites relatives à l'extension

Les fonctions définies dans le Math.decah sont sinus, cosinus, arcsinus, arctangente et ulp(unit in last place). Pour effectuer un calcul d'arccosinus, l'utilisateur peut l'obtenir en calculant la valeur de pi/2-arcsinus().

Limite relative à chaque fonction :

- **float _sin (float rad) :**
  La fonction sin fait le calcul de la valeur sinus d'un radian. La marge d'erreur est entre +1E-7 et -1E-7. mais la marge augmente quand le radian dépasse -2pi et 2pi.
- **float _cos (float rad) :**
  La fonction cos fait le calcul de la valeur cosinus d'un radian.. La marge d'erreur est entre +1E-7 et -1E-7 mais la marge augmente quand le radian dépasse -2pi et 2pi.
- **float _asin (float val) :**
  Calcul de l'arcsinus d'une valeur entre [-1 ;1]. La marge d'erreur est entre +1E-4et -1E-4.

- **float _atan (float val) :**
  Le résultat de la fonction atan est forcément entre [-pi/2 ;pi/2]. La marge d'erreur est entre +1E-3et -1E-3 avec la valeur d'entrée entre [-1 ;1], mais la marge d'erreur augment après dépassé 1.
- **float _ulp (float val) :** ULP est le plus petit pas que le compilateur arrive de s'ajouter. La valeur de sortie est entre 1E-6 et 1E-9 pour une entrée de [-100 ;100].

9 Exemples : (On fait un exemple pour chaque types de langages.)

9.1 Exemple : Langage Hello World

Le compilateur est capable de compiler des instructions simples, par exemple l'affichage sur l'écran une ligne simple "Hello, world.". En outre, l'affichage d'un entier ou un flottant est possible, mais "print" un boolean ou une méthode ou void n'est pas autorisé.

— Un 'print' vide est compilable, mais cela n'affiche rien en écran.
— Un 'println' fait un print et retourne à la ligne suivante.
— Un 'printlnx' ou un 'printx' affiche une valeur en héxadécimale.

A titre d'exemple :

```
{
    print("Hello, world");
    println("!");
    printlnx(2.353);
}
```

L'affichage en écran :

```
    Hello, world!
    0x1.2d2f1ap+1
```

9.2 Exemple : Langage Sans Objet

Le compilateur pourrait être utilisé pour effectuer différentes opérations dans un programme principal. Les opérations possibles sont listées ci-dessous :

— Déclaration et Initialisation des variables.
— Assignation une valeur à une variable.
— Affichage une valeur en écran, "Print".
— Condition 'if else'.
— Condition d'une boucle 'while'.
— Opération Arithmétique (+ - * / %).
— Opération Logique Ou, Et, Not (|| !).
— Opération comparaison supérieur à et inférieur à (> < <= >=).
— Conversion du type float et int.
— Lecture d'une valeur de type entière ou flottant en entrée standard avec les fonctions readInt() et readFloat().

La déclaration des variables doit être fait avant une instruction.

```
{
```

```
    boolean a,b;
    int k,c=22;
    float d=3.22;

    a=true;
    b=false;

    if(a || b){ println("Hello World!");}

    println(c+d);

    k=readInt();

    while( ((k > (c+d))) && (a==true))
    {
        a= false;
        print(k);
    }

}
```

L'affichage en écran si k=100 :

```
    Hello, World!
    25.22
    100
```

9.3 Exemple : Langage Objet

```
/**
  Description: Un exemple d' utilisation de compilateur avec le langage Objet
**/

class Vehicule{
  // Le nombre de chevaux qui dtermine la puissance de la vehicule
  int nombreChevaux;
  // La consommation en litre de la voiture
  int consommation;

  /**
     Initialisation de nombre de chavaux de la vehicule
  **/
  void initNombreChevaux(int n)
  {
    this.nombreChevaux = n;
  }

  /**
    Changer ou initialiser la consommation de le vehicule
  **/
  void setConsommation(int c)
```

```
  {
    this.consommation = c;
  }
  // Affiche une Vehicule
  void affichage(){
      println("Nombre de chevaux : ", this.nombreChevaux);
      println("Consommation : ", this.consommation, "l/h");
  }
}

class Utilitaire extends Vehicule{
      // L'hauteur de la vehicule
      float hauteur;
      void setHauteur(float f){
        this.hauteur = f;
      }
      // Affiche un Utilitaire
      void affichage(){
          println("Vehicule de type : Utilitaire ");
          println("Nombre de chevaux : ", this.nombreChevaux);
          println("Consommation : ", this.consommation, "l/h");
          println("Hauteur : ", this.hauteur, "m");
      }
}

class Voiture extends Vehicule {

  // Affiche une voiture
  void affichage(){
      println("Vehicule de type : Voiture ");
      println("Nombre de chevaux : ", this.nombreChevaux);
      println("Consommation : ", this.consommation, "l/h");
  }

}

class Camion extends Utilitaire {
    // Le poids de Camion
    float poids;

    // initialiser le poids de camion
    void setPoids(float f){
        this.poids = f;
    }
    // Affiche Un camion
    void affichage(){
        println("Vehicule de type : Camion ");
        println("Nombre de chevaux : ", this.nombreChevaux);
        println("Consommation : ", this.consommation, "l/h");
        println("Hauteur : ", this.hauteur, "m");
        println("Poids : ", this.poids, "Kg");
    }
}

{
```

```
  // Le programme principal

  /**
    Declaration de variable
  **/
  Vehicule v = new Vehicule();
  Camion c = new Camion();
  Voiture vo = new Voiture();

  /**
     Initialisation de variables
  **/
  v.initNombreChevaux(2000);
  v.setConsommation(20);
  c.initNombreChevaux(2500);
  c.setConsommation(15);
  c.setPoids(15000);
  c.setHauteur(5);
  vo.initNombreChevaux(1500);
  vo.setConsommation(10);

  /**
    Affichage
  **/
  v.affichage();
  c.affichage();
  vo.affichage();
}
```

A la sortie On a

```
/** Sortie **/
Nombre de chevaux : 2000
Consommation : 20 l/h
Vehicule de type : Camion
Nombre de chevaux : 2500
Consommation : 15 l/h
Hauteur : 5 m
Poids : 15000 Kg
Vehicule de type : Voiture
Nombre de chevaux : 1500
Consommation : 10 l/h
```

9.4 Exemple : Langage Complet

```
/**
  Description: Un exemple d' utilisation de compilateur avec le langage Complet (Cast et
instanceOf)
**/
class Animal{
  boolean equals(Object o){
    Animal a;
    if (o instanceof Animal){
```

```
      return true;
    }
    return false;
  }
}
class Humain extends Animal{

}
{
  Animal a = new Animal();
  Humain h = new Humain();
  if (a.equals(h)){
    println("Un humain est un Animal");
  }
}
```

A la sortie On a

```
Un humain est un Animal
```