Documentation utilisateur du compilateur deca

I/Limitation et spécificité du compilateur

Le compilateur en soi - hormis les extensions qui ne sont pas requises dans la spécification du langage - répond exactement aux exigences des langages sans objet, objet et complet, pour ce que nos tests, que nous espérons fiables (couverture à plus de 80%), ont révélé du reste.
Les options ont aussi été implémentées, pour rappel :

- -b : affiche la bannière de l'équipe (nos noms).
- -p : décompile- affiche l'arbre en sortie du *parser*
- -v : s'arrête après la décoration de l'arbre (étape B). Pas de sortie si succès.
- -n : supprime les erreur 11.1 et 11.3
- -r X : restreint le nombre de registres à X
- -d : affiche les traces de debug (chaque répétition ajoute un niveau)
- -P : lance la compilation en parallèle des fichiers

II/Messages d'erreurs retournés par le compilateur

Dans cette rubrique, vous trouverez une liste la plus exhaustive possible des erreurs que vous pourrez rencontrer en utilisant notre compilateur. Nous avons accompagné chaque erreur d'un exemple afin de faciliter votre lecture. Ces exemples ont pour but d'être simples, courts et efficaces, ils ne représentent donc pas toutes les configurations possibles qui génèrent les erreurs.

II.A/Message d’erreur lexicographique

Les erreurs lexicographiques arrivent lorsque ce que les mots que vous écrivez ne correspondent pas aux token référencés dans la spécification. Le message est alors :

`<nom-fichier>:<n°-ligne>:<n°-colonne>: token recognition error at: <caractère non reconnu>`

Voici un exemple qui vous donnera ce type d’erreur : `` `@à%$+=& ``

II.A/Message d’erreur de mauvaise syntaxe (hors contexte)

Les erreurs contextuelles sont affichées sous la forme:

`<nom-fichier>:<n°-ligne>:<n°-colonne>: <description>`

La description prend une des formes suivantes:

type d’erreur	exemple

`no viable alternative at input ‘<input>’`	/* un commentaire */ */
	Un caractère inattendu:
	{
	    1e20
	}

`missing ‘<token1>’ at ‘<token2>’`	{
	    if () {
	}

`mismatched input ‘<token1>’ expecting ‘<token2>’`	{
	    if {}
	}

`<entier> n'est pas un entier codable sur 32 bits (valeur max : <max_value>, valeur min : <min_value>)`	{ int a = 200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000;

	3 }
<flottant> n'est pas un flottant codable sur 32 bits (valeur extrêmes : +/- <max_value>)	1 { 2 float a = 1.0e99; 3 }
<flottant> est plus petit que la précision minimale pour les flottants qui vaut : <min_value>	1 { 2 float a = 1.00000e-99; 3 }

II.B/Message d'erreur de mauvaise syntaxe (contextuelle)

Les erreurs contextuelles sont affichées sous la forme:

`<nom-fichier>:<n°-ligne>:<n°-colonne>: <description>`

La description elle-même est sous la forme:

`(<n°-règle>)<message>`

où `n°-règle` est le numéro de la règle enfreinte.

Ci-dessous les tableaux des erreurs pour les langages sans objet et objet:

Liste spécifique au langage sans objet

Règle	**Message d'erreur**	**exemple :**
0.1	Expression non définie	1 { 2 a; 3 }
0.2	Type non reconnu	1 { 2 typeInconnu a; 3 }
3.17	Une variable ne peut être de type 'void'	1 { 2 void a; 3 }
3.17	La variable <nomVar> est déjà déclarée.	1 { 2 int a;

		3 float a; 4 }
3.28	Les types <type1> et <type2> sont incompatibles pour l'affectation	1 { 2 float a; 3 a = true; 4 }
3.29	Type de l'expression : <type>, attendu : 'boolean' pour une condition	1 { 2 if (0.5) {} 3 }
3.31	Les instances affichables peuvent être de type 'int', 'float', 'string'	1 { 2 print(true); 3 }
3.33	Types des opérandes : <typeG>, <typeD>. Attendu : <liste de types> pour l'opérateur <nom_op>	1 { 2 true + 1 ; 3 false && 3 ; 4 }
3.35	La valeur saisie doit être de type 'int'	1 { 2 ReadInt(); 3 } [~]$ hello
3.36	La valeur saisie doit être de type 'float'	1 { 2 ReadFloat(); 3 } [~]$ hello
3.37 - not, minus	Type opérande : <type>, attendu : <typeattendu> pour l'opérateur <nom_op>	1 { 2 !2; 3 - true; 4 }

Liste spécifique au langage objet

Règle	Message	exemple
1.3	Identificateur de classe attendu	1 class int {}

	OU Identificateur <superclasse> non déclaré OU La classe est déjà définie OU <ident> n'est pas un identificateur de classe	1 class A extends X {} 1 class A {} 2 class A {} 1 class A extends int {}
2.4	L'identificateur <ident> est déjà défini	1 class A { 2 int x; 3 float x; 4 }
2.4	<nom> définit déjà une méthode	1 class A { 2 void met() {} 3 int met; 4 }
2.5	Un attribut ne peut être de type 'void'	1 class A { 2 void x; 3 }

2.6	L'identificateur <method> est déjà défini	1 class A {
		2 int meth() {
		3 return 1;
		4 }
		5 void meth() {}
		6 }

2.6	<méthode> définit un champ	1 class A { 2 int x; 3 void x() {} 4 }

2.7	La signature diffère de la méthode redéfinie
	OU
	
	La méthode retourne le type <type1> mais devrait retourner le type de sa super méthode <type2>
	
	OU
	Type de retour effectif : <type1>, déclaré : <type2>
		1 class A {void meth(){}}
		2 class B extends A{
		3 void meth(int a){}
		4 }
		
		1 class A {void meth(){}}
		2 class B extends A{
		3 int meth(){}
		4 }
		
		1 class A {
		2 void meth(){
		3 return 1;
		4 }
		5 }

3.8	<classe> n'est ni la même classe ni une sous-classe de <classe attendue>	1 class A {} 2 class B {} 3 { 4 A a = new B(); 5 }

3.24	Le type de retour ne peut être 'void'	1 class A {
		2 void meth() {}
		3 int other() {
		4 return this.meth();
		5 }
		6 }

3.39	Cast impossible de <type1> vers <type2>	1 { 2 (int)(true); 3 (A) (true); 4 }
3.40	Type gauche : <type>, attendu : 'class' ou 'null'	1 class B {} 2 { 3 B b = new B(); 4 0.5 instanceof b ; 5 }
3.40	Type droit : <type>, attendu : 'class'	1 { 2 Object a = null; 3 a instanceof 0.5; 4 }

3.42	`<type> n'est pas un type de classe`	1 { 2 new boolean(); 3 }
3.43	`this ne peut être utilisé à dans le main`	1 { 2 this; 3 this.x; 4 this.x(); 5 }

3.65	`<nom de champ> n'est pas un membre de la classe <classe>`
	
	OU
	`Le membre gauche n'est pas de type 'classe'`
	
	OU
	`L'attribut est protégé`
	
	OU
	`<nom> est une méthode et non un champ`
		1 class A {}
		2 {
		3 A a = new A();
		4 a.x;
		5 }
		
		1 {
		2 true.x;
		3 }
		
		1 class A { protected int x;}
		2 {
		3 A a = new A();
		4 a.x;
		5 }
		
		1 class A { int x;}
		2 {
		3 A a = new A();
		4 a.x();
		5 }

3.71	`Le membre gauche n'est pas de type 'class'`
	
	OU
	`<method> n'est pas un membre de la classe`
	
	OU
	`Le champ n'est pas une méthode`
		1 {
		2 int a;
		3 a.random();
		4 }
		
		1 class A {}
		2 {
		3 A a = new A();
		4 a.x();
		5 }
		
		1 class A {
		2 int x;
		3 }

		4 { 5 A.x(); 6 }

3.74	La signature ne correspond pas aux paramètres
	
	OU
	Le nombre de paramètres ne correspond pas à celui de la méthode
		1 class A {
		2 void met(float b){}
		3 }
		4 {
		5 A a = new A();
		6 a.met(true);
		7 }
		
		1 class A {
		2 void met(){}
		3 }
		4 {
		5 A a = new A();
		6 a.met(true);
		7 }

II.C/ Message d'erreur d'exécution du code

Les erreurs à l'exécution sont de la forme `Erreur : <nom de l'erreur>`

Les tests des erreurs de type 11.1 et 11.3 sont supprimées par l'option -n du compilateur

Type	Nom	Signification	exemple
11.1	division_par_zero	division entière (et reste de la division entière) par 0	1 { 2 1/0; 3 }
11.1	debordement_flottant	débordement arithmétique sur les flottants (inclut la division flottante par 0.0)	1 { 2 1/0.0; 3 }

11.1	sortie de la méthode sans return	Absence de return lors de l'exécution d'une méthode dont le	1 class A {
			2 int met(){}
			3 }
			4 {

		type de retour n'est pas void	5 A a = new A(); 6 a.met(); 7 }
11.1	`cast_impossible`	Conversion de type impossible	1 class A {} 2 class B extends A 3 {} 4 { 5 B b = (B)(new A()); 6 }
11.1	`dereferencement.null`	Déréférencement de null	1 class A{int x;} 2 { 3 A a = null; 4 a.x; 5 }
11.2	`io_error`	Erreur de lecture (valeur entrée pas dans le type attendu)	1 { 2 ReadFloat(); 3 } [~]$ hello
11.3	`pile_pleine`	Le programme doit empiler plus de mots que ne peut contenir la pile	utilisez les options, pour boucher la pile

11.3	`tas_plein`	Une allocation dans le tas est impossible par manque de place	1 class A {
			2 B b = new B();}
			3 class B {
			4 A a = new A();}
			5 { A a = new A();}

III/ Utilisation de l'extension trigo

L'utilisation de l'extension trigo se fait exactement comme spécifié dans le pdf du sujet. C'est à dire qu'il faut inclure le fichier Math.decah avec `#include "Math.decah"` puis créer un nouvel objet math avec new Math(). Enfin on peut se servir d'une des méthodes sin, cos, asin , atan, ulp avec la syntaxe suivante `<nom de l'objet Math>.<nom de la méthode>(<argument>)`. Toutes ces méthodes renvoient un float.

Ci dessous vous pouvez voir des exemples pour chacunes des fonctions trigonométriques ainsi que pour ulp:

Ulp

```
#include "Math.decah"
{
    float f;
    Math m = new Math();
    f = m.ulp(0.0);
    println(f);
}
```

Sin

```
#include "Math.decah"
{
    float f;
    Math m = new Math();
    f = m.sin(0.0);
    println(f);
}
```

Cos

```
#include "Math.decah"
{
    float f;
    Math m = new Math();
    f = m.cos(0.0);
    println(f);
}
```

ArcSin

```
#include "Math.decah"
{
    float f;
    Math m = new Math();
    f = m.asin(0.0);
    println(f);
}
```

ArcTan

```
#include "Math.decah"
{
    float f;
    Math m = new Math();
    f = m.atan(0.0);
    println(f);
}
```

IV/ Limite de l'extension trigo

La précision est le critère le plus important de notre extension, étant donné que nous avons choisi TRIGO.
Les algorithmes CORDIC et ANTI CORDIC présents sur internet sont ceux utilisés par les premiers ordinateurs et les calculatrices. Ces algorithmes réalisent des approximations du cos et du sin de l'angle passé en argument. Notre problème principal était de mettre en œuvre des solutions permettant d'avoir la meilleure précision possible pour les fonctions cos, sin, asin, atan et ulp. Le temps d'exécution est aussi un facteur important de l'extension TRIGO mais il demeure moins important que la précision.
Les algorithmes pour cos et sin sont assez similaires et renvoient des résultats avec la même précision.

1. D'après nos tests, les résultats de ulp correspondent à ceux de Math.ulp() de java exactement.

2. Sur le cercle trigonométrique, (ex: $cos(1)$ ou $sin(1)$) les algorithmes vont afficher les valeurs attendues avec une précision allant de 10^{-7} à 10^{-8}.

3. Si on sort du cercle trigonométrique, une nouvelle problématique apparaît : on doit se ramener à $[-\pi, \pi]$ ce qui peut entraîner des imprécisions ou des erreurs.
Cependant, notre version donne des résultats corrects pour des angles inférieurs à 10^{6}, au-delà de cette valeur, la précision va diminuer progressivement (par exemple 10^{-3} pour 10^{6}).

4. Pour asin et atan nous obtenons une précision de 10^{-6} à 10^{-8} sur leurs intervalles respectifs.

En pratique, on fera rarement appel à $cos(10^n)$ *avec n grand* , nous vous conseillons de ne pas utiliser de nombres d'une taille trop importante. Par exemple, lorsque l'angle passé en argument est trop élevé, les résultats donnés ne sont plus acceptables. Par exemple pour $cos(10^{10})$, l'écart entre la véritable valeur et celle de notre algorithme est de **1.5** .

Pour vérifier ces valeurs, nous avons pris comme référence les valeurs de cos et de sin sur Python, Java ainsi que Wolfram|Alpha.

En résumé:

Fonction	Précision	Intervalle
ulp	**exact**	[min; max][1.]
sin	10^{-6} **à** 10^{-8}	$[-\pi, \pi]$
cos	10^{-6} **à** 10^{-8}	$[-\pi, \pi]$
arcsin	10^{-6} **à** 10^{-8}	[-1; 1]
arctan	10^{-6} **à** 10^{-8}	[min ; max][1.]

1.min et max correspondent aux valeurs maximales disponibles pour les flottants