# Documentation Utilisateur du compilateur Deca

Gonthier Florentin, Bergeron Matthieu, Beaupère Matthias,
Fischman Adrien, Geoffroy Germain

---

Cette documentation destinée à un utilisateur du compilateur deca facilite la prise en main du compilateur. Elle explique les différentes limites imposées au langage soumit au compilateur ainsi que la signification des différentes erreurs de compilation et d'exécution. Elle propose enfin des exemples d'utilisation en exposant le code assembleur généré par le compilateur.

---

# Table des matières

# 1 Limitations du compilateur

- Taille maximum des nombres entiers et flottants :
  Les nombres entiers étants reconnus avec la fonction de l'api Java Integer.parseInt(), ils doivent être compris entre $-2^{-31}$ et $2^{31}-1$.
  Pareil pour les nombres flottants, qui sont reconnus avec la fonction Float.parsefloat(). Ils doivent êtres compris entre $2^{-149}$et $(2-2^{-23})2^{127}$.

- Les classes sont soumises à plusieurs limitations. Tout d'abord il est impossible de déclarer des champs et des méthodes ayant le même nom. Les champs peuvent être définis comme protégés ou publiques, mais ils ne peuvent pas être restreints à la classe elle-même. Les méthodes peuvent être redéfinis dans les sous-classe mais uniquement avec la même signature et un type de retour identique ou un sous-type de celle définie dans la superclasse. Lorsqu'elle sont appelée, ces méthode peuvent avoir comme paramètre des sous-types de ceux définis.

- Le compilateur ne gère pas les récursions très profondes. A partir de plusieurs dizaines d'appels, un dépassement de pile est indiqué.

- Les casts implicites ne sont pas toujours interprétés. Dans certains cas comme les paramètres d'appel de fonction, il est préférable d'expliciter les cast pour de meilleurs résultats.

# 2 Limitations de la bibliothèque math

La classe Math contient différentes méthodes :

- float ulp(float f)
- float sin(float f)
- float cos(float f)
- float asin(float f)
- float atan(float f)

Chaque méthode a été testée grâce aux valeurs retournées par les fonctions prédéfinies de Java.Math. Ces méthodes sont des approximations de valeurs

mathématiques, dont on mesure la précision grâce aux ULP (Unit in the Last Place).
Pour une analyse complète et détaillée des algorithmes et de leurs précisions respectives, il faut se reporter à la **documentation de la classe Math**.
On présente ici, sous forme de tableau, les différents résultats obtenus.

## 2.1 Méthode ulp.

La méthode ulp donne le résultat exacte sans approximation.

| | intervalle | nb erreur($> 1ulp$) | erreur max(ulp) | pas | nb tests |
|---|---|---|---|---|---|
| ulp | $[0; 1\ 000]$ | 0 | 0 | $2^{-14}$ | 16 384 000 |
| ulp | $[10\ 000; 100\ 000]$ | 0 | 0 | $2^{-7}$ | 11 520 000 |
| ulp | $[2^{30}; 2^{31}]$ | 0 | 0 | 128 | 8 388 608 |

## 2.2 Méthode sin.

La méthode sin est implémenté grâce au développement en série entière de sinus et d'une réduction de Cody and Waite sur $[-\pi/8; \pi/8]$.
La méthode sin donne une approximation à 5 ulp d'écart au maximum.
Pour des petits angles ($< \pi/8$), la précision est de 1 ulp dans le pire cas.

| | intervalle | nb erreur($> 1ulp$) | erreur max(ulp) | erreur moyenne(ulp) | pas | nb tests |
|---|---|---|---|---|---|---|
| sin | $[0; \pi/8]$ | 0 | 1 | null | $2^{-23}$ | 3 294 199 |
| sin | $[\pi/8; 2\pi]$ | 1117838 | 5 | 2.13 | $2^{-20}$ | 6 176 623 |
| sin | $[1\ 000; 1\ 000 + 2\pi]$ | 17777 | 5 | 2.13 | $2^{-14}$ | 102 944 |
| sin | $[100\ 000; 100\ 000 + 2\pi]$ | 127 | 4 | 2.20 | $2^{-7}$ | 804 |

## 2.3 Méthode cos.

La méthode cos est implémenté grâce au développement en série entière de cosinus et d'une réduction de Cody and Waite sur $[-\pi/8; \pi/8]$.
La méthode cos donne une approximation à 6 ulp d'écart au maximum.
Pour des petits angles ($< \pi/8$), la précision est de 2 ulp dans le pire cas.

| | intervalle | nb erreur($> 1ulp$) | erreur max(ulp) | erreur moyenne(ulp) | pas | nb tests |
|---|---|---|---|---|---|---|
| cos | $[0; \pi/8]$ | 365 | 2 | 2.0 | $2^{-23}$ | 3 294 199 |
| cos | $[\pi/8; 2\pi]$ | 1335192 | 6 | 2.18 | $2^{-20}$ | 6 176 623 |
| cos | $[1\ 000; 1\ 000 + 2\pi]$ | 20987 | 5 | 2.19 | $2^{-14}$ | 102 944 |
| cos | $[100\ 000; 100\ 000 + 2\pi]$ | 162 | 4 | 2.12 | $2^{-7}$ | 804 |

## 2.4 Méthode asin.

La méthode asin est implémenté grâce au développement en série entière de la fonction arc sin.
Par imparité de la fonction arcsinus, on ne donne le résultat que pour des nombres positifs.
La méthode asin donne une approximation à 4 ulp d'écart au maximum.

| | intervalle | nb erreur($> 1ulp$) | erreur moyenne | erreur max(ulp) | pas | nb tests |
|---|---|---|---|---|---|---|
| asin | $[0; 0.5]$ | 0 | 0 | 1 | $2^{-20}$ | 524288 |
| asin | $[0.5; 0.72]$ | 1974 | 2 | 2 | $2^{-20}$ | 230687 |
| asin | $[0.72; 0.98]$ | 22660 | 2.12 | 4 | $2^{-20}$ | 272630 |
| asin | $[0.98; 1]$ | 1687 | 2.43 | 7 | $2^{-20}$ | 20972 |

## 2.5 Méthode atan.

La méthode atan est implémenté grâce aux polynômes de Hermite.
Par imparité de la fonction arctangente, on ne donne le résultat que pour des nombres positifs.
La méthode atan donne une approximation à 2 ulp d'écart au maximum.

| | intervalle | nb erreur($> 1ulp$) | erreur moyenne | erreur max(ulp) | pas | nb tests |
|---|---|---|---|---|---|---|
| atan | $[0; 0.6875]$ | 0 | 0 | 1 | $2^{-20}$ | 720 896 |
| atan | $[0.6875; 1.1875]$ | 0 | 0 | 1 | $2^{-20}$ | 524 288 |
| atan | $[0.1875; 2]$ | 43 006 | 2.0 | 2.0 | $2^{-23}$ | 6 815 744 |
| atan | $[2; 100]$ | 0 | 0 | 1.0 | $2^{-12}$ | 401 408 |
| atan | $[100; 10\ 000]$ | 0 | 0 | 1.0 | $2^{-7}$ | 1 267 200 |

# 3 Recensement des erreurs

## 3.1 Erreur de lexicographie

| Description de l'erreur | Exemple | Résultat |
|---|---|---|
| Mauvais token | #include "é" | Token recognition error at '#include' |
| Inclusion circulaire | | Circular include for file file.deca |

## 3.2 Erreur de syntaxe hors-contexte

En plus des erreurs générés par ANTLR4, le parseur retourne une exeption "**Number too large**" pour les nombres trop grands (entiers ou flottants).

## 3.3 Erreur de syntaxe contextuelle

**Cannot override method <method name> with a different signature.**
Tentative de sucharge d'une méthode de superclasse avec une signature différente.

```
class A {
   void m() {}
}
class B exends A {
   void m(int a) {}
}
```

**Type returned by method <method name> is not a subtype of overrided method.**
Une méthode est redéfinie mais le type renvoyé n'est pas un sous-type de la méthode initiale.

```
class A {
   int m() {}
}
class B exends A {
   boolean m() {}
}
```

**Double declaration of method <method name>**
Un méthode est déclarée deux fois dans la même classe.

```
class A {
    int m() {}
    int m() {}
}
```

**Condition must be a boolean.**
La condition donnée n'est pas un booléen.

```
{
    while(5) {}
}
```

**Arithmetic operation on expressions which types are differents.**
Les deux opérande de l'opération ne sont pas de même type, et il ne s'agit pas d'un entier et d'un flottant.

```
{
    int a = 5 + false;
}
```

**Boolean operation on non-boolean operands.**
L'un des opérande de l'opération booléenne n'est pas un booléen.

```
{
    boolean x = false || 5;
}
```

**Inequality on non-numbers.**
L'un des opérande de l'inégalité n'est pas un nombre.

```
{
    boolean x = 5 == true;
}
```

**Cannot assign <right type> to <left type>**

**Incompatible cast from <operand type> to <cast type>**
Impossible d'effectuer le cast de la variable de type <cast type> en une variable de type <cast type>.

```
{
    int a = (int)5.2;
}
```

**Class <name> twice declared.**
La classe de nom <name> a été déclaré deux fois.

```
class A {}
class A {}
```

**Name <name> already used in the class.**
Le nom donnée <name> a déjà été utilisé, pour une méthode ou pour un champs dans la classe.

```
class A {
    int a;
    void a() {}
}
```

**"Double definition of variable <name>**
Le paramètre de nom <name> ne peut pas être déclaré car un paramètre ou une variable de même nom à déja été déclaré.

```
class A {
    int a(int a, boolean a) {}
}
```

**Multiple declaration of variable <name>**
La variable de nom <name> a été défini deux fois.

```
{
    int a = 5;
    int a;
}
```

**A variable can not be declared as void.**
Une variable est définie de type void. Cela est impossible.

```
{
    void a;
}
```

**Undefinded variable <name>**
La variable <name> n'a pas été défini.

```
{
    int a;
    a = unknownVar;
}
```

**Type <name> undefinded.**
Les type de nom <name> n'a pas été défini. Ou alors il ne s'agit pas d'un type.

```
{
    UnknownType a;
}
```

**Class <name> undefinded.**
La classe <name> n'a pas été définit, ou alors il ne s'agit pas d'une classe.

```
{
    int a;
    boolean x = a instanceof UndefinedClass;
}
```

**Method <name> undefinded.**
La méthode indiquée n'a pas été définie.

```
class A {}
{
    int a = a.undefMethod();
}
```

**Invalid signature for method <name>**
Les paramètres de la méthode appelée <name> ne possède pas les type ou les sous-type correspondant à la signature de la fonction.

```
class A {
    int m(void b) {}
}
{
    A a = new A();
    a.m(false);
}
```

**Incompatible type initialization.**
L'expression donnée pour l'initialisation de la variable ou du champ n'est pas du type de la variable ou du champ.

```
{
    int a = false;
}
```

**Wrong argument number.**
La méthode a été appelée avec un mauvais nombre d'arguments.

```
class A {
    int m() {}
}
{
    A a = new A();
    a.m(int a);
}
```

**Wrong type for param <number>**
Le paramètre numéro <number> de la fonction n'est pas de type correspondant à la signature de la méthode.

```
class A {
    int m(int a) {}
}
{
    A a = new ();
    a.m(false);
}
```

**Not and instance of a class given for method call.**
L'objet indiqué por l'appel de méthode n'est pas une instance de classe.

```
{
    int a;
    a.m();
}
```

**Modulo must be used with two int.**
Un des opérande de l'opérateur modulo n'est pas un entier.

```
{
    int a;
    boolean b;
    int c = a % c;
}
```

**Must apply 'not' operator to a boolean.**
L'opérande de l'opérateur 'non' n'est pas un booléen.

```
{
    int a;
    boolean b = !a;
}
```

**Return type does not match definition.**
Le type de l'expression retournée par la fonction n'est pas celui indiqué par la défninition de la fonction.

```
class A {
    int m() {
       return false;
    }
}
```

**Not and instance of a class given for selection.**
L'objet indiqué por la selection n'est pas une instance de classe.

```
{
    int a;
    a.m;
}
```

**Cannot call a protected field outside its class or a subclass.**
Tentative de sélection d'un champ protégé hors de sa classe ou d'une de ses sous-classe.

```
class A {
    protected int b;
}
{
    a A = new A();
    a.b = 5;
}
```

**Cannot use 'this' in main.**
Le mot 'this' a été utilisé dans le main. C'est impossible.

```
{
    int a = this;
}
```

**Must apply unary minus to an int or float.**
Le mois doit être appliqué à un nombre.

```
{
    int a = -false;
}
```

### 3.4 Erreur à l'éxécution

| Description de l'erreur | Exemple | Résultat |
|---|---|---|
| Débordement arithmétique | int a = 5.4/0.0; | Error : overflow during arithmetic operation |
| Débordement du tas | Trop d'allocation mémoire avec new() | Error : heap overflow |
| Débordement de la pile | Trop d'appel de fonctions récursives | Error : stack overflow |
| Déréférencement d'un pointeur NULL | Object a = null; a.equals(a); | Error : dereferencing null pointer |

## 4 Exemples

### 4.1 Hello World

Un hello-world basique.

**Code deca :**

```
{
        println("Hello");
}
```

**Code assembleur :**

```
; start main program
TSTO #4
BOV stack_overflow
ADDSP #4
LOAD #null, R0
STORE R0, 1(GB)
```

```
LOAD code.Object.equals, R0
STORE R0, 2(GB)
; Main program
; Beginning of main instructions:
WSTR "Hello"
WNL
HALT
init.Object:
RTS
code.Object.equals:
TSTO #2
BOV stack_overflow
PUSH R2
PUSH R3
LOAD -2(LB), R2
LOAD -3(LB), R3
CMP R2, R3
SEQ R0
POP R3
POP R2
RTS
; end main program
arith_overflow:
WSTR "Error : overflow during arithmetic operation"
ERROR
stack_overflow:
WSTR "Error : stack overflow"
ERROR
heap_overflow:
WSTR "Error : heap overflow"
ERROR
dereferencement.null:
WSTR "Error : dereferencing null pointer"
ERROR
```

## 4.2 Fibonacci itératif

Implémentation de la fonction de fibonacci en itératif. Le code ci dessous calcule fibo(25).

**Code deca :**

```
{
    int n = 25;
    int i = 1;
    int first = 0;
    int second = 1;
    int tmp;
    while(i <= n){
        tmp = first + second;
        first = second;
        second = tmp;
        i = i + 1;
    }
    print(first);
}
```

**Code assembleur :**

```
; start main program
TSTO #9
BOV stack_overflow
ADDSP #9
LOAD #null, R0
STORE R0, 1(GB)
LOAD code.Object.equals, R0
STORE R0, 2(GB)
; Main program
LOAD #25, R2
STORE R2, 3(GB)
LOAD #1, R3
STORE R3, 4(GB)
LOAD #0, R4
STORE R4, 5(GB)
```

```
LOAD #1, R5
STORE R5, 6(GB)
; Beginning of main instructions:
DebutWhile0:
LOAD 4(GB), R0
LOAD 3(GB), R1
CMP R1, R0
BGT EndWhile0
LOAD 5(GB), R6
ADD 6(GB), R6
STORE R6, 7(GB)
LOAD 6(GB), R2
STORE R2, 5(GB)
LOAD 7(GB), R2
STORE R2, 6(GB)
LOAD 4(GB), R2
ADD #1, R2
STORE R2, 4(GB)
BRA DebutWhile0
EndWhile0:
LOAD 5(GB), R1
WINT
HALT
init.Object:
RTS
code.Object.equals:
TSTO #2
BOV stack_overflow
PUSH R2
PUSH R3
LOAD -2(LB), R2
LOAD -3(LB), R3
CMP R2, R3
SEQ R0
POP R3
POP R2
RTS
; end main program
```

```
arith_overflow:
WSTR "Error : overflow during arithmetic operation"
ERROR
stack_overflow:
WSTR "Error : stack overflow"
ERROR
heap_overflow:
WSTR "Error : heap overflow"
ERROR
dereferencement.null:
WSTR "Error : dereferencing null pointer"
ERROR
```

## 4.3 Grosse expression et option -r 4

Nous testons ici le bon fonctionnement de l'option -r qui permet de limiter le nombre de registres disponibles. On remarque que l'option fonctionne correctement, puisqu'aucun registre supérieur à R3 n'est utilisé. L'expression est aussi correctement évaluée à 94.

**Code deca :**

```
{
    int a = ( ( (5 + (3 % 1)) - 2 + ((2 - 2 + 3)*5 + 2 - 2) + (1 - 1) - (1/3 + 54)
    - ( 2 + 3*2*3*(9*10/11)) + 1 - 8) % 4
    *
    ( (5 + (3 % 1)) - 2 + ((2 - 2 + 3)*5 + 2 - 2) + (1 - 1) - (1/3 + 54) + 2
    - ( 2 + 3*2*3*(9*10/11)) + 1 - 8) % 4
    -
    ( (5 + (3 % 1)) - 2 + ((2 - 2 + 3)*5 + 2 - 2) + (1 - 1) - (1/3 + 54) + 2
    - ( 2 + 3*2*3*(9*10/11)) + 1 - 8) % 4
    + (2*23*3 - 1)
    ) / ( (23%2) + 2*2*(2-(3*(4-3+1-1))))
    *
     (( (5 + (3 % 1)) - 1 + ((2 - 2 + 3)*5 + 2 - 2) + (1 - 1) - (1/3 + 54) + 2
    - ( 2 + 3*2*3*(9*10/11)) + 1 - 8) % 4) ;
    print(a);
}
```

**Code assembleur :**

```
; start main program
TSTO #5
BOV stack_overflow
ADDSP #5
LOAD #null, R0
STORE R0, 1(GB)
LOAD code.Object.equals, R0
STORE R0, 2(GB)
; Main program
LOAD #5, R2
LOAD #3, R3
REM #1, R3
BOV arith_overflow
ADD R3, R2
SUB #2, R2
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #2, R3
SUB #2, R3
ADD #3, R3
MUL #5, R3
ADD #2, R3
SUB #2, R3
ADD R3, R2
POP R3
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #1, R3
SUB #1, R3
ADD R3, R2
POP R3
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #1, R3
QUO #3, R3
```

```
BOV arith_overflow
ADD #54, R3
SUB R3, R2
POP R3
ADD #2, R2
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #2, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #3, R2
MUL #2, R2
MUL #3, R2
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #9, R3
MUL #10, R3
QUO #11, R3
BOV arith_overflow
MUL R3, R2
POP R3
ADD R2, R3
POP R2
SUB R3, R2
POP R3
ADD #1, R2
SUB #8, R2
REM #4, R2
BOV arith_overflow
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #5, R3
TSTO #1
BOV stack_overflow
```

```
PUSH R2
LOAD #3, R2
REM #1, R2
BOV arith_overflow
ADD R2, R3
POP R2
SUB #2, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #2, R2
SUB #2, R2
ADD #3, R2
MUL #5, R2
ADD #2, R2
SUB #2, R2
ADD R2, R3
POP R2
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #1, R2
SUB #1, R2
ADD R2, R3
POP R2
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #1, R2
QUO #3, R2
BOV arith_overflow
ADD #54, R2
SUB R2, R3
POP R2
ADD #2, R3
TSTO #1
BOV stack_overflow
PUSH R2
```

```
LOAD #2, R2
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #3, R3
MUL #2, R3
MUL #3, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #9, R2
MUL #10, R2
QUO #11, R2
BOV arith_overflow
MUL R2, R3
POP R2
ADD R3, R2
POP R3
SUB R2, R3
POP R2
ADD #1, R3
SUB #8, R3
MUL R3, R2
POP R3
REM #4, R2
BOV arith_overflow
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #5, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #3, R2
REM #1, R2
BOV arith_overflow
ADD R2, R3
POP R2
```

```
SUB #2, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #2, R2
SUB #2, R2
ADD #3, R2
MUL #5, R2
ADD #2, R2
SUB #2, R2
ADD R2, R3
POP R2
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #1, R2
SUB #1, R2
ADD R2, R3
POP R2
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #1, R2
QUO #3, R2
BOV arith_overflow
ADD #54, R2
SUB R2, R3
POP R2
ADD #2, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #2, R2
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #3, R3
MUL #2, R3
```

```
MUL #3, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #9, R2
MUL #10, R2
QUO #11, R2
BOV arith_overflow
MUL R2, R3
POP R2
ADD R3, R2
POP R3
SUB R2, R3
POP R2
ADD #1, R3
SUB #8, R3
REM #4, R3
BOV arith_overflow
SUB R3, R2
POP R3
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #2, R3
MUL #23, R3
MUL #3, R3
SUB #1, R3
ADD R3, R2
POP R3
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #23, R3
REM #2, R3
BOV arith_overflow
TSTO #1
BOV stack_overflow
PUSH R2
```

```
LOAD #2, R2
MUL #2, R2
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #2, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #3, R2
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #4, R3
SUB #3, R3
ADD #1, R3
SUB #1, R3
MUL R3, R2
POP R3
SUB R2, R3
POP R2
MUL R3, R2
POP R3
ADD R2, R3
POP R2
QUO R3, R2
BOV arith_overflow
POP R3
TSTO #1
BOV stack_overflow
PUSH R3
LOAD #5, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #3, R2
REM #1, R2
BOV arith_overflow
```

```
ADD R2, R3
POP R2
SUB #1, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #2, R2
SUB #2, R2
ADD #3, R2
MUL #5, R2
ADD #2, R2
SUB #2, R2
ADD R2, R3
POP R2
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #1, R2
SUB #1, R2
ADD R2, R3
POP R2
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #1, R2
QUO #3, R2
BOV arith_overflow
ADD #54, R2
SUB R2, R3
POP R2
ADD #2, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #2, R2
TSTO #1
BOV stack_overflow
PUSH R3
```

```
LOAD #3, R3
MUL #2, R3
MUL #3, R3
TSTO #1
BOV stack_overflow
PUSH R2
LOAD #9, R2
MUL #10, R2
QUO #11, R2
BOV arith_overflow
MUL R2, R3
POP R2
ADD R3, R2
POP R3
SUB R2, R3
POP R2
ADD #1, R3
SUB #8, R3
REM #4, R3
BOV arith_overflow
MUL R3, R2
POP R3
STORE R2, 3(GB)
; Beginning of main instructions:
LOAD 3(GB), R1
WINT
HALT
init.Object:
RTS
code.Object.equals:
TSTO #2
BOV stack_overflow
PUSH R2
PUSH R3
LOAD -2(LB), R2
LOAD -3(LB), R3
CMP R2, R3
SEQ R0
```

```
POP R3
POP R2
RTS
; end main program
arith_overflow:
WSTR "Error : overflow during arithmetic operation"
ERROR
stack_overflow:
WSTR "Error : stack overflow"
ERROR
heap_overflow:
WSTR "Error : heap overflow"
ERROR
dereferencement.null:
WSTR "Error : dereferencing null pointer"
ERROR
```

## 4.4 Un exemple de code objet

Voici un exemple pour un code orienté objet. Le résultat obtenu par le calcul ci-dessous est bon.

**Code deca :**

```
class Calcul {
   int carre(int a){
        return a*a;
   }

   int moinsTrois(int a){
        return a - 3;
   }

   int grosseExpression(int a,int b,int c,int d){
        return a*(c + d) + b*(c - a % c) % 3;
   }
}
```

```
{
    Calcul c = new Calcul();
    int res = c.carre(c.moinsTrois(3) + c.grosseExpression(1,2,3,4)*c.carre(3-4+8)
    print(res);

}
```

**Code assembleur :**

```
; start main program
TSTO #11
BOV stack_overflow
ADDSP #11
LOAD #null, R0
STORE R0, 1(GB)
LOAD code.Object.equals, R0
STORE R0, 2(GB)
LEA 1(GB), R0
STORE R0, 3(GB)
LOAD code.Object.equals, R0
STORE R0, 4(GB)
LOAD code.Calcul.carre, R0
STORE R0, 5(GB)
LOAD code.Calcul.moinsTrois, R0
STORE R0, 6(GB)
LOAD code.Calcul.grosseExpression, R0
STORE R0, 7(GB)
; Main program
NEW #1, R1
BOV heap_overflow
LEA 3(GB), R0
STORE R0, 0(R1)
TSTO #3
BOV stack_overflow
ADDSP #1
STORE R1, 0(SP)
BSR init.Calcul
SUBSP #1
LOAD R1, R2
```

```
STORE R2, 8(GB)
TSTO #4
BOV stack_overflow
ADDSP #2
LOAD 8(GB), R3
CMP #null, R3
BEQ dereferencement.null
STORE R3, 0(SP)
TSTO #4
BOV stack_overflow
ADDSP #2
LOAD 8(GB), R3
CMP #null, R3
BEQ dereferencement.null
STORE R3, 0(SP)
LOAD #3, R3
STORE R3, -1(SP)
LOAD 0(SP), R3
LOAD 0(R3), R3
BSR 3(R3)
LOAD R0, R3
SUBSP #2
TSTO #7
BOV stack_overflow
ADDSP #5
LOAD 8(GB), R4
CMP #null, R4
BEQ dereferencement.null
STORE R4, 0(SP)
LOAD #1, R4
STORE R4, -1(SP)
LOAD #2, R4
STORE R4, -2(SP)
LOAD #3, R4
STORE R4, -3(SP)
LOAD #4, R4
STORE R4, -4(SP)
LOAD 0(SP), R4
```

```
LOAD 0(R4), R4
BSR 4(R4)
LOAD R0, R4
SUBSP #5
TSTO #4
BOV stack_overflow
ADDSP #2
LOAD 8(GB), R5
CMP #null, R5
BEQ dereferencement.null
STORE R5, 0(SP)
LOAD #3, R5
SUB #4, R5
ADD #8, R5
STORE R5, -1(SP)
LOAD 0(SP), R5
LOAD 0(R5), R5
BSR 2(R5)
LOAD R0, R5
SUBSP #2
MUL R5, R4
ADD R4, R3
STORE R3, -1(SP)
LOAD 0(SP), R3
LOAD 0(R3), R3
BSR 2(R3)
LOAD R0, R3
SUBSP #2
STORE R3, 9(GB)
; Beginning of main instructions:
LOAD 9(GB), R1
WINT
HALT
init.Object:
RTS
code.Object.equals:
TSTO #2
BOV stack_overflow
```

```
PUSH R2
PUSH R3
LOAD -2(LB), R2
LOAD -3(LB), R3
CMP R2, R3
SEQ R0
POP R3
POP R2
RTS
init.Calcul:
RTS
code.Calcul.carre:
TSTO #0
BOV stack_overflow
LOAD -3(LB), R0
MUL -3(LB), R0
BRA fin.code.Calcul.carre
WSTR "Erreur : sortie de la methode carre sans return"
WNL
ERROR
fin.code.Calcul.carre:
RTS
code.Calcul.moinsTrois:
TSTO #0
BOV stack_overflow
LOAD -3(LB), R0
SUB #3, R0
BRA fin.code.Calcul.moinsTrois
WSTR "Erreur : sortie de la methode moinsTrois sans return"
WNL
ERROR
fin.code.Calcul.moinsTrois:
RTS
code.Calcul.grosseExpression:
TSTO #0
BOV stack_overflow
LOAD -3(LB), R0
LOAD -5(LB), R6
```

```
ADD -6(LB), R6
MUL R6, R0
LOAD -4(LB), R7
LOAD -5(LB), R8
LOAD -3(LB), R9
REM -5(LB), R9
BOV arith_overflow
SUB R9, R8
MUL R8, R7
REM #3, R7
BOV arith_overflow
ADD R7, R0
BRA fin.code.Calcul.grosseExpression
WSTR "Erreur : sortie de la methode grosseExpression sans return"
WNL
ERROR
fin.code.Calcul.grosseExpression:
RTS
; end main program
arith_overflow:
WSTR "Error : overflow during arithmetic operation"
ERROR
stack_overflow:
WSTR "Error : stack overflow"
ERROR
heap_overflow:
WSTR "Error : heap overflow"
ERROR
dereferencement.null:
WSTR "Error : dereferencing null pointer"
ERROR
```