Projet Génie Logiciel - Equipe GL 23

Manuel D’utilisateur Pour le Compilateur du langage Deca

Sommaire:

Guide D’utilisation:

Précisions générales:

L’utilisation de ce compilateur sur un fichier deca se fait principalement en deux étapes:

- **Une compilation** en utilisant la commande decac sur le fichier désiré:
  - **Exemple** : decac ./hello_world.deca
- **L’exécution** du fichier assembleur (.ass) obtenu, grâce à ima :
  - **Exemple** : ima ./hello_world.ass

Point sur les Options:

On a plusieurs possibilités d’options qu’on peut utiliser avec la commande decac, chacune avec son propre rôle:

- -b (banner) : affiche le nom de notre équipe
- -p (parse) : arrête decac après l’étape de construction de l’arbre, et affiche la décompilation de ce dernier
- -v (vérification) : arrête decac après l’étape de vérifications pour valider la syntaxe contextuelle du programme
- -n (no check) : ne plus générez plus le code assembleur explicite qui affiche des messages d'erreur à l'exécution par rapport à la sémantique dynamique du langage.
- -r X (registers) : limite les registres banalisés disponibles à X registres (X entre 4 et 16 nécessairement)
- -d (debug) : active les traces de debug.
- -P (parallel) : lance la compilation des fichiers en parallèle afin d’accélérer la compilation
- - arm : Lance la compilation en mode ARM, génère un fichier .s (plus de détails dans la partie Extension)

Les Messages D'erreur:

Les Messages D'erreurs du Compilateur

1/Les Erreurs dues à la compilation:

- "Failed to open output file : xxxx"

Ce message d'erreur s'affiche quand on n'arrive pas à trouver (...)

- "Failed to open input file : xxxx"

Ce message d'erreur s'affiche pour annoncer que le compilateur n'arrive pas à trouver le fichier qu'on lui avait demandé de compiler, et donc qu'il faut revoir le path et/ou le nom.

- "Stack overflow while compiling file xxxxx"

Ce message d'erreur s'affiche pour annoncer qu'il y avait un débordement de la pile.

- "Exception raised while compiling file xxxx"

Ce message d'erreur s'affiche quand le compilateur rencontre un problème lors de la compilation du fichier

- Assertion failed while compiling file xxxx"

Ce message d'erreur s'affiche lorsqu'il y a une erreur d'assertion d'une valeur, lors de la compilation du fichier

- Internal compiler error while compiling file xxxx, sorry"

Ce message d'erreur s'affiche à la suite des messages d'erreurs précédents

1/Les Erreurs dues à une mauvaise manipulation des options:

- "No Other Options nor files can be used with the option [-b]"

Ce message d'erreur s'affiche dans le cas où on écrit n'importe quelle autre option avec la "-b" ou lorsqu'on lui donne un fichier à compiler juste après. Cette option est à utiliser uniquement dans le format "decac -b"

- "The number you are giving is not born between 4 and 16"

Ce message d'erreur s'affiche quand l'utilisateur essaie de limiter le nombre des registres à utiliser en donnant un nombre de registres non acceptable. Le compilateur accepte uniquement les valeurs entre 4 et 16

- "Please precise an Integer right after the "-r". \n it should be born between 4 and 16"

Ce message d'erreur s'affiche lorsque l'utilisateur essaie de limiter le nombre des registres à utiliser et oublie de préciser combien de registres il veut utiliser.

- "xxxx is unrecognized. Please give a valid option before the name of your file, or give a deca filename"

Ce message s'affiche lorsque l'utilisateur utilise après "decac" une expression qu'on n'arrive pas à reconnaître comme une des options ou comme fichier deca.

- "You cannot use both -p and -v as options. Please choose one!"

Ce message s'affiche lorsque l'utilisateur essaie d'utiliser les options -p et -v au même temps. Ceci n'est pas accepté par le compilateur.

- " Here's the basic Syntax : decac [[-p | -v] [[- arm] | [-n] [-r X] [-d]* [-P] [-w]] ...] | [-b] "

Ce message s'affiche après tous les messages d'erreurs précédents afin de préciser à l'utilisateur la syntaxe correcte pour l'utilisation des options du compilateur.

Les Messages D'erreurs de L'étape A:

- "Token Recognition error at : xxxx"

Ce message d'erreur s'affiche quand le Lexer n'arrive pas à reconnaître un token vu qu'il ne rentre pas dans ce qu'il peut interpréter lexicalement.

- "No viable alternative at input xxxx"

Ce message d'erreur s'affiche quand le Lexer n'arrive pas à comprendre ce qu'on lui a donné comme programme déca vu qu'il ne correspond à aucune des possibilités de grammaire… Il déclare donc le dernier token qu'il arrive à reconnaître et annonce qu'il n'arrive pas à déchiffrer ce qui vient après.

➔ Les tests invalides associés à cet erreur :

-Method_without_type.deca
-late_decl.deca
-simple_lex.deca
-uncamelcase.deca

- " Circular include for file xxxx"

Ce message d'erreur s'affiche quand on appelle include sur le fichier lui-même, ce qui fait une inclusion circulaire.

- "missing 'xxxx' at xxxx"

Ce message d'erreur s'affiche lorsqu'il attend un caractère précis à un endroit donné (";" à la fin d'une instruction, ou '{' en EOF) ... et il précisera donc l'endroit où on devrait le retrouver.

- "mismatched input 'xxx' at 'xxxx' expecting {...} "

Ce message d'erreur s'affiche quand le compilateur ne reçoit pas ce qu'il attendait suivant la grammaire (par exemple ne pas donner d'expression après le signe '=')... Il précisera donc ce qu'il attend de l'utilisateur entre crochets.

- "extraneous input ‘xxx’ at ‘xxxx’ expecting {...} ”

Comme son nom l’indique, ce message d’erreur s’affiche quand le compilateur retrouve un élément de plus de ce qu’il attendait (par exemple boolean a = && true;)... Il précisera donc ce qu'il attend de l’utilisateur entre crochets.

- "left-hand side of assignment is not an lvalue ”

Le côté gauche d’une affectation doit nécessairement être un lvalue, c'est-à-dire soit un “identifier” ou une sélection.

Les Messages D’erreurs de L’étape B:

La Partie Sans Objet:

1/ Déclaration des variables :

- "The type xxxx is not authorized "

Ce message d’erreur s’affiche à la déclaration d’une variable avec un type qui n’est ni un type prédéfini (int ,float ,boolean,void,Object) dans le langage Deca ni un type défini par le programmeur (classes).

➔ Les tests invalides associés à cet erreur :

-return_type_undefined.deca
-undeclared_type_inField.deca
-undeclared_type_inParam.deca

- "we can't declare an identifier with type void "

Ce message s’affiche quand un programmeur Deca tente de déclarer une variable avec le type void .

➔ Les tests invalides associés à cet erreur :

-init-void-boolean.deca
-init-void-float.deca
-init-void-int.deca
-void-declaration.deca

- "Redeclaration of the variable"

On interdit la redéclaration d’une même variable que ce soit dans le programme principal Main ou dans les corps des méthodes d’une classe.

➔ Les tests invalides associés à cet erreur :

-double-declaration.deca

-RedeclarationvarInMethodBody.deca

2/ Initialisation et affectation des variables et d'expressions :

- "Non compatible assignment of expression"

Une opération d'affectation exige que les deux opérandes soient de même type ou le premier de type float et le second de type entier dans le cas contraire un message de non compatibilité de type s'affiche .

➔ Les tests invalides associés à cet erreur :

-return_type_wrong.deca

les affectations sans objet (*.deca)	**les affectations avec objet (*.deca)**	**les initialisations (*.deca)**
affect-boolean-float	assign_Classes	init-boolean-float
affect-boolean-int	assign_extended	init-boolean-int
affect-float-boolean	assign_incompatible_ inMethod	init-float-boolean
affect-incompatible	assign_incompatible_ ofClass	init-int-boolean
affect-int-boolean		init-int-float

- "The two operands are not of the compatible types for the binary Operation"

Pour les opérations binaires booléennes ou arithmétiques les deux opérandes doivent être de même type avec l'autorisation du sous typage entre int et float dans les opérations arithmétiques .

➔ Les tests invalides associés à cet erreur :

-sum-incompatible0.deca

-sum-incompatible1.deca

- "The two operands must be boolean expression "

Un message d'erreur s'affiche si les opérandes dans une opération booléenne ne sont pas tous les deux de type boolean .

- "Not must be followed by a boolean expression"

L'expression booléenne qui suit la négation booléenne doit être aussi de type boolean sinon il s'affiche le message d'erreur ci dessus .

- "Incompatible comparaison"

La comparaison arithmétique exige aussi que les deux opérandes soient de type entier ou float .

3/ Boucles while et for :

- "Condition must be of type boolean"

Un message d'erreur s'affiche dans le cas où on tente d'implémenter une boucle avec une condition de type non boolean .

➔ Les tests invalides associés à cet erreur :

-non-boolean-condition.deca

4/ print :

- "type of argument must be either int, float or String"

Print n'accepte pas d'argument de type différent à un entier , un float ou un string. Dans le cas contraire, le message d'erreur ci-dessus s'affiche.

➔ Les tests invalides associés à cet erreur :

-print-boolean.deca

La Partie Objet:

1/ La première passe :

- "The class xxxx has already been declared"

Dans le cas où le programmeur Deca tente de définir une classe avec un identifiant d'une autre classe déjà existante, on lui affiche un message d'erreur disant que l'identificateur est déjà consommé.

➔ Les tests invalides associés à cet erreur :

-Class_repeted.deca

- "The extended class xxxx does not exist"

A la déclaration d'une classe qui étend une autre classe on vérifie tout d'abord l'existence de sa super classe et dans le cas de l'absence de celle-ci on affiche le message d'erreur ci dessus .

➔ Les tests invalides associés à cet erreur :
-extends_nonexist.deca
-inter_extends.deca

- "The extended class xxxx must be of type class "

Ce message d'erreur s'affiche quand le programmeur tente d'étendre un type prédéfini (int ,void ,float,boolean).
exemple : class A extends int {}

➔ Les tests invalides associés à cet erreur :
-extends_type.deca

2/ La deuxième passe :

Passe sur les champs :

- "we can't declare a field with type void "

Si le programmeur Deca définit un champ d'une classe avec le type void la compilation s'arrête avec le message d'erreur ci-dessus .

➔ Les tests invalides associés à cet erreur :
-protected_void.deca

- "The field xxxx is already defined in a superclass but with a different type"

Ce message d'erreur s'affiche quand le programmeur redéfinit un champ qui existe déjà dans une super class mais avec un type différent .

- " The identifier xxxx is already used for a method or a field in the same class "

A la déclaration des champs on vérifie si l'identificateur est déjà consommé par un autre champ ou une méthode déjà définis dans la classe .Dans le cas échéant on affiche le message d'erreur ci-dessus .

➔ Les tests invalides associés à cet erreur :
-Field_repeted.deca
-redeclarationMethod.deca
-sameName_field_class.deca
-sameName_field_Method1.deca
-sameName_Method_field.deca

Passe sur les méthodes :

- "The type xxxx is not authorized "

A la déclaration des méthodes d'une classe on affiche ce message d'erreur si le type de retour de la fonction n'est ni dans les types prédéfinis (int ,float ,boolean,void,Object) du langage Deca ni dans les types définis par le programmeur (classes).

➔ Les tests invalides associés à cet erreur :

-return_type_undefined.deca
-undeclared_type_inField.deca
-undeclared_type_inParam.deca

- " The identifier xxxx is already used for a field in the same superclass"

Si l'identificateur qu'on utilise pour déclarer une méthode est déjà consommé par un champ dans la même classe on affiche le message ci-dessus .

- "The method is already defined in a superclass but with a different return type"

Ce message d'erreur s'affiche si le programmeur tente de redéfinir une méthode déjà définie dans une superClass mais avec un type de retour différent .

➔ Les tests invalides associés à cet erreur :

-redeclarationMEthodReturnType.deca
-redef_diff_type.deca
-type_redef_changed1.deca
-type_redef_changed2.deca
-wrong_redef_type.deca

- "The method is already defined in a superclass but with a different signature"

Ce message d'erreur s'affiche si le programmeur tente de redéfinir une méthode déjà définie dans une superClass mais avec une signature différente .

➔ Les tests invalides associés à cet erreur :

-redeclarationMethodSignature.deca
-redef_diff_signature.deca
-redef_order_signature.deca
-test_extends1.deca

- " The identifier xxxx is already used for a method or a field in the same class"

A la déclaration des méthodes on vérifie si l'identificateur est déjà consommé par une autre méthode ou un champ déjà définis dans la classe .Dans le cas échéant on affiche le message d'erreur ci-dessus .

➔ Les tests invalides associés à cet erreur :

-Field_repeted.deca
-redeclarationMethod.deca
-sameName_field_class.deca
-sameName_field_Method1.deca
-sameName_Method_field.deca

- "The type of an argument in the input of a function must not be Void "

Si le programmeur Deca tente de définir une méthode avec un argument de type void on lui affiche le message d'erreur ci-dessus .

➔ Les tests invalides associés à cet erreur :

-void_in_parameter.deca

- "Arguments in a function must be different "

A la déclaration d'une méthode si ses arguments sont identiques on arrête la compilation avec les message d'erreur ci dessus.

➔ Les tests invalides associés à cet erreur :

-same_param_repeted1.deca
-same_param_repeted2.deca
-sameArguments.deca

3/ La troisième passe :

- "The method is called with a wrong parameters "

A l'appel des méthodes on vérifie si les arguments de l'appel sont de même type (ou sous type) que les arguments dans la signature de déclaration de méthode appelée . Dans le cas contraire on affiche le message d'erreur ci-dessus

Remarque :

On applique la conversion (convFloat) dans le cas où l'argument dans la signature de déclaration est de type float alors qu'on fait l'appelle avec le type int .

➔ Les tests invalides associés à cet erreur :

-wrong_type_parameter.deca

- "Too much parameters while calling the method"

Si le programmeur Deca tente d'appeler une méthode avec un nombre d'arguments supérieur à celui de la déclaration de la méthode, on lui affiche le message d'erreur ci-dessus.

➔ Les tests invalides associés à cet erreur :

-TooMuch_parameters1.deca
-TooMuch_parameters2.deca

- "this is only used inside class to refer the object"

Si le programmeur utilise this dans le programme principal,on lui affiche le message ci-dessus disant que this est utilisé juste au sein d'une classe pour référencer une instance propre .

➔ Les tests invalides associés à cet erreur :

-this_noClass.deca

- "the selection must relate to an object of type class"

Si le programmeur tente d'appliquer une sélection sur une instance qui n'est pas objet d'une classe, on lui affiche le message ci-dessus .

Exemple :

```
{  int x ;
   int y = x.t;
}
```

➔ Les tests invalides associés à cet erreur :

-invalidSelection.deca

- "The field xxxx is not defined for the selected class"

Dans la sélection on cherche la classe de l'expression sur laquelle la sélection est appliquée et on vérifie si le champ de la sélection est défini pour cette classe dans le cas contraire on affiche le message d'erreur ci-dessus .

➔ Les tests invalides associés à cet erreur :

-indefinedField.deca

- "The field xxxx is declared protected and you try to get it out from the class "

Ce message d'erreur s'affiche si le programme est dans le programme principal et tente d'appeler le champ d'une classe sur un objet de cette classe alors que ce champ est déclaré protégé.

➔ Les tests invalides associés à cet erreur :

-conditon2_selection.deca

- "The field xxxx is declared protected and you try to get it out from the class or a subclasse"

Le type de la classe dans laquelle la sélection est faite doit être un sous-type de la classe où le champ protégé est déclaré.Dans le cas contraire on affiche le message d'erreur ci-dessus .

➔ Les tests invalides associés à cet erreur :

-protect_interclass_select2.deca

-protect_interclass_select1.deca

- "The type of the expression to which the selection is applied must be a subtype of the current class"

Le type de l'expression sur laquelle la sélection est appliquée doit être un sous-type de la classe courante .Dans le cas contraire, on affiche le message d'erreur ci-dessus.

Exemple :

```
class A {
        protected int x; }
class B extends A {
      int getX(A a){
            return a.x;

}
```

=> Erreur contextuelle : le type de 'a' (A) n'est pas un sous type de B .

➔ Les tests invalides associés à cet erreur :

-condition1_selection.deca

-protected_selection2.deca

- "New must be followed by a class Type "

L''identificateur qui suit le new doit avoir une définition d'une classe dans le cas contraire on affiche le message d'erreur ci dessus.

Exemple : { new int(); }

➔ Les tests invalides associés à cet erreur :

-newClass.deca

- "InstanceOf must be preceded by a class type or type Null "

L'expression qui précède instanceof doit être un objet d'une classe ou null sinon le message d'erreur ci-dessus arrête la compilation .

Exemple

```
class A {}
{ int x ;
  boolean b = ( x instanceof A);
}
```

➔ Les tests invalides associés à cet erreur :

-instanceOf_previous.deca

- "InstanceOf must be followed by a class type"

L'identificateur qui suit instanceof doit être de type class sinon le message d'erreur ci-dessus arrête la compilation .

➔ Les tests invalides associés à cet erreur :

-instanceOf_following.deca

- "The type of the expression we want to cast must not be void "

Dans le cast l'expression qu'on veut convertir ne doit pas être de type void

- "non compatible types for casting "

Dans le cast le type de l'expression qu'on veut convertir doit être un sous type du type auquel on veut aboutir .

➔ Les tests invalides associés à cet erreur :

-invalidCast.deca

- "Arithmetic operations are not allowed for type class "

Si le programmeur tente de faire une opération arithmétique entre deux objets des classes . Le message d'erreur ci-dessus arrête la compilation.

➔ Les tests invalides associés à cet erreur :

-operation_onClass.deca

- "The return of the method is void "

L'instruction return ne doit pas être de type void .

```
class A{
        void fct1(){}
        void fct2(){
             return fct1() ;  }
}
```

➔ Les tests invalides associés à cet erreur :

-invalidReturnType.deca

-returnvoid.deca

=> plus les messages d'erreur traités dans le sans objet

Les Messages D'erreurs à l'exécution:

- "Erreur: overflow pendant le calcul"

Cette erreur apparaît lorsqu'une instruction arithmétique ou bien une instruction de lecture telle que readInt() détecte un débordement dans le calcul. Notant que le calcul est tout de même fait et est celui de l'opération modulo 2^(32).

➔ Les tests invalides associés à cet erreur :

-division_by_zero.deca

- "Error; Input/Output error"

Ce message d'erreur surgit lors d'une interaction avec une instruction de lecture sur le terminal, c.à.d readInt() et readFloat(). Cela signifie que la valeur entrée ne correspond pas au type attendu par la fonction ou alors il s'agit d'un débordement.

➔ Les tests invalides associés à cet erreur :

-readint.deca
-readfloat.deca

- "Erreur: overflow de la pile"

Ce message signifie que le programme a détecté un débordement de la pile lors de l'exécution. Cela surgit lorsque le programme est susceptible d'écrire de l'information à l'extérieur de l'espace alloué à la pile ce qui risque un éventuel écrasement de données nécessaire pour son exécution.

- "Erreur: déréférencement null"

Cette erreur est provoquée lors d'une tentative d'utilisation d'un objet sans pour autant l'avoir initié.

➔ Les tests invalides associés à cet erreur :

-Dereferenced_null1.deca
-Dereferenced_null2.deca

- "Erreur method xxx should not return void"

Cette erreur est détectée lorsqu'une méthode qui devait retourné un résultat de type différent de void a fini son exécution sans avoir retourné un valeur.

- "Erreur : Invalid Cast "

L'exécution s'arrête avec le message d'erreur "Erreur : Invalid Cast "
si la conversion n'est pas faite : d'un int à float , d'un float à int , de null vers n'importe quelle classe ,d'une classe vers n'importe quelle super classe ,d'un type vers le même type.

➔ Les tests invalides associés à cet erreur : **invalidCast.deca**

Guide de travail en mode ARM:

Installation de la toolchain :

L'extension ARM porte sur la compilation de l'arbre syntaxique abstrait issu du programme Deca en code assembleur ARM aarch32. Pour cela, il faudra avoir en main la toolchain nécessaire; on a choisi de travailler avec la librairie GNU ARM-32bits qu'on installera dans un environnement Debian (à adapter pour les autres environnements Linux) en suivant les étapes ci-dessous. Les commandes shell sont précédées d'un $:

- S'assurer d'avoir l'accès root sur le profil utilisé pour l'installation de la toolchain.
- Mettre à jour le gestionnaire de paquets apt:
  $ sudo apt update -y && sudo apt upgrade -y

- Télécharger et installer les librairies et utilitaires GNU:
  $ sudo apt install qemu-user qemu-user-static build-essential
  $ sudo apt install gcc-arm-linux-gnueabihf binutils-arm-linux-gnueabihf binutils-arm-linux-gnueabihf-dbg

Routine de compilation et exécution :

- Pour générer un fichier ARM assembly .s:
  - $ decac -arm hello.deca

- Toutes les compilations des fichiers.s se font en mode statique avec la commande suivante:
  - $ arm-linux-gnueabihf-gcc -static -o hello hello.s

- Pour exécuter l'exécutable hello généré :
  - $ ./hello

Commandes optionnelles:

- Pour désassembler un exécutable généré en .s:
  $ arm-linux-gnueabihf-objdump -d hello

- Pour compiler du C en assembleur .s:
  $ arm-linux-gnueabihf-gcc -S hello.c

Capacités et limitations du compilateur :

A la différence du compilateur decac en mode IMA, le mode ARM ne supporte pas le calcul sur les flottants. Les deux modes restent cependant semblables sur le reste des spécifications. Voici une liste de ce que le compilateur ARM supporte:

- Déclaration de variables entières et booléennes.
- Affectation de valeurs à des variables déclarées.
- Calcul et comparaisons sur les entiers.
- Opérations sur les booléens.
- Lecture d'un entier entré par l'utilisateur (avec readInt()).
- Affichage d'entiers et de chaînes de caractères (print() / println()).
- Structures conditionnelles (if then else / while).
- Déclaration d'un flottant initialisé par un immédiat flottant et son affichage en décimal (pas grand intérêt).

Le compilateur ne supporte pas (en plus de ce qui est spécifié en mode IMA) :

- Les opérations sur les flottants immédiats et variables.
- La déclaration de classes et de méthodes.
- L'allocation dans le tas avec new.
- La déclarations de variables entières plus grandes que 2**31 -1 ou plus petites que -(2**31 -1).
- Les opérations sur les entiers faisant intervenir des immédiats plus grands que 2**16 ou plus petits que -2**16.

Erreurs rencontrées :

Les erreurs fréquemment rencontrées sont de plusieurs types, levées:

1. à la compilation vers assembleur ARM.
2. à la compilation vers exécutable ARM.
3. à l'exécution.

La 1ère catégorie d'erreurs provient généralement d'un programme Deca fournit ne respectant pas la spécification du compilateur ARM ou bien d'une erreur de ligne de commandes.
Cas notables:

- Les erreurs relevées par le compilateur en mode IMA.
- Intervention des flottants dans le calcul d'une expression ou comme immédiats.
- Présence d'Instructions/Déclarations relatives à l'Objet (new, this class..).
- Déclarations d'entiers signés ne pouvant pas tenir sur 32 bits.

Les erreurs de la catégorie 2 sont plus rares et concernent généralement une erreur sur le nom de fichier ou options de la commande; parfois même une mauvaise installation de la toolchain.

Pour la catégorie 3, les erreurs sont celles intervenant au sein des instructions dans l'exécutable. Ils peuvent intervenir dans le cas de:

- Pile pleine.
- Intervention d'immédiats entiers ne tenant pas sur 16 bits (spécification des registres ARM). Le message d'erreur suivant est affiché: Error: invalid constant (xxxx) after mixup.

D'autres erreurs concernant les définitions des labels, la syntaxe des instructions ARM.. ne doivent normalement pas intervenir à l'exécution.

Remarques:

- printx et print sont équivalentes pour les variables flottantes.
- La lecture d'une entrée de l'utilisateur peut donner l'impression qu'elle affiche en même temps cette entrée, alors que ce n'est que sa trace. En effet, elle ne fait pas partie de l'output de l'exécution mais est juste superposée au reste de l'affichage du terminal.
- Le compilateur ARM ne gérant pas les overflows des opérations, il faudra faire attention aux opérandes donnés. Plus précisément, si le résultat d'une opération ne tient pas sur 32 bits, alors il reçoit la valeur 2**31 -1. Et si une opération sur deux opérandes de même signes donne un résultat de signe contraire, le résultat est quand même

retourné (C'est le cas quand le résultat de l'opération est supérieur à 2**31 -1).
- éviter de print des chaînes composées seulement de "n" et des caractères spéciaux ou contenant "/*".

Comprendre un fichier .s*:*

Forme générale d'un fichier .s généré par decac -arm:

Un programme assembleur ARM en .s se compose principalement de deux parties: la section "text" qui contient les instructions et "data" qui contient les variables déclarées. L'entrée du programme est dénotée par le label "main:". La sortie se fait par l'appel système SVC #0.
Des variables passe-partout sont déclarées au début de chaque .data et seront utilisées ou pas dans le code généré.

- n: contient le caractère de saut de ligne.
- printfloatxxxxxxxx: chaîne d'affichage des flottants.
- printintxxxxxxxx: chaîne d'affichage des entiers.
- scannerxxxxxxxx: chaîne de lecture d'entiers.

Quand une variable est déclarée, un label contenant son nom concaténé à un nombre est créé pour définir le type et le contenu de l'espace mémoire occupé par la variable.
Le programme est parsemé de commentaires marqués par un @ en début de ligne.

Directives rencontrées:

.arch : spécifie l'architecture de sur laquelle sera exécuté le code compilé (ici armv8-a).
.extern : déclare des fonctions qui seront appelées des bibliothèques standards GNU (on appellera printf et scanf systématiquement à la génération de code même s'ils ne seront pas toujours utilisées).
.section : déclare une section.
.**text** : déclare la section contenant les instructions.
.global : annonce le label pour l'accès global depuis la bibliothèque standard GNU.
.data : déclare la section contenant les variables.

.align : indique à l'assembleur qu'il doit ajouter une quantité de padding pour aligner l'espace mémoire suivant sur une adresse multiple de 32 bits.
.word : déclare un mot de 32 bits en mémoire.
.double : déclare un mot de 64 bits en mémoire.
.asciz : déclare une chaîne de caractères ascii en mémoire se terminant par le caractère de fin "\0".
.- "label" : indique le taille de la zone mémoire vers laquelle pointe "label".

Listes d'instructions ARM utilisées:

Les Ri dénotent des registres, [Ri, #imm] leurs offsets, #imm des immédiats, =label l'adresse marqué par un label.
Les opérations unaires sont de la forme: **INST OP**
Les opérations binaires sont de la forme: **INST OP1 OP2**
Les opérations tertiaires sont de la forme: **INST OP1 OP2 OP3**

MOV Ri OP2 : Transfère le contenu de OP2 dans Ri avec OP2 soit Rj ou #imm.

MVN Ri OP2 : Similairement à MOV transfère le contenu de OP2 dans Ri mais en performant une négation bit à bit sur OP2.

LDR Ri OP2 : Charge le contenu de l'espace mémoire référencé par OP2 dans Ri. OP2 peut être [Rj, #imm] ou =label.

STR Ri OP2 : Opération inverse de LDR, stocke le contenu de Ri à l'adresse d'OP2.

SVC #IMM : Provoque une interruption système de type #IMM. Par exemple si IMM = 0, c'est l'interruption de sortie (exit).

ADD Ri Rj OP3 : Stocke le résultat de Op3 + Rj dans Ri. Positionne les bits V d'overflow signé et C de carry. Le carry signifie que le résultat ne tient pas dans le registre. OP3 peut être Rj ou #IMM.

SUB Ri Rj OP3 : Stocke le résultat de Rj - OP3 dans Ri. Positionne les bits d'overflow signé et de carry.

MUL Ri Rj Rk : Stocke le résultat de Rj * Rk dans Ri. Ne provoque pas d'overflow.

SDIV Ri Rj Rk : Stocke le résultat du quotient entier de Rj par Rk dans Ri. Ne provoque pas d'overflow.

CMP Ri OP2 : Compare OP2 au contenu de Ri (effectue Ri-OP2) et positionne les flags N (négatif), Z (zéro) V et C. OP2 est soit Rj ou #IMM.

B OP : branchement inconditionnel vers l'adresse à laquelle pointe OP (label ou autre).

Bcc OP : branchement si cc est vrai vers l'adresse à laquelle pointe OP (label ou autre). CC appartient à {EQ, NE, GT, LT, GE, LE, VS, VC, CC, CS…}.

BL OP : branchement inconditionnel vers l'adresse à laquelle pointe OP (label ou autre) avec sauvegarde de l'adresse de retour. Utile pour l'appel de fonctions, on l'utilisera pour afficher et lire avec printf et scanf.