# Projet de programmation fonctionnelle et de traduction des langages

Année 2024/2025

Le but du projet de programmation fonctionnelle et de traduction des langages est d'étendre le compilateur du langage RAT réalisé en TP de traduction des langages pour traiter de nouvelles constructions : les **pointeurs**, les **variables globales**, les **variables statiques locales** et les **paramètres par défaut**.

Le compilateur sera écrit en OCaml et devra respecter les principes de la programmation fonctionnelle étudiés lors des cours, TD et TP de programmation fonctionnelle.

## Table des matières



## Preambule

- Le projet est à réaliser en binôme (même binôme qu'en TP).
- L'échange de code entre binômes est interdit.
- Les sources fournies doivent compiler sur les machines des salles de TP.
- Les sources et le rapport sont à déposer sous Moodle avant le **jeudi 16 Janvier - 23h**. **Aucun report ne sera accepté** : anticipez !
- Les sources seront déposées sous forme d'une unique archive `<rat_xxx_yyy>.tar` où `xxx` et `yyy` sont les noms du binôme. Cette archive devra créer un répertoire `rat_xxx_yyy` (pensez à renommer le répertoire nommé `sourceEtu` donné en TP) contenant tous vos fichiers.
- le rapport (`rapport.pdf`) doit être dans le même répertoire, à la racine. Il n'est pas nécessairement long, mais doit expliquer les évolutions apportées au compilateur (voir section sur les critères d'évaluation) et les jugements de typages liés aux nouvelles constructions du langage.

1. $PROG' \rightarrow PROG\ \$$
2. $PROG \rightarrow VAR\star\ FUN\star\ id\ \ BLOC$
3. $VAR \rightarrow static\ TYPE\ \ id\ \ =\ \ E\ ;$
4. $FUN \rightarrow TYPE\ id\ \ (\ DP\ \ )\ BLOC$
5. $BLOC \rightarrow\ \{\ I\star\ \}$
6. $I \rightarrow\ TYPE\ \ id\ \ =\ \ E\ ;$
7. $|static\ TYPE\ id\ \ =\ \ E\ ;$
   ~~$|\ id\ \ =\ \ E\ ;$~~
8. $|\ A\ \ =\ \ E\ ;$
9. $|\ const\ \ id\ \ =\ \ entier\ ;$
10. $|\ print\ \ E\ ;$
11. $|\ if\ \ E\ \ BLOC\ \ else\ \ BLOC$
12. $|\ while\ \ E\ \ BLOC$
13. $|\ return\ \ E\ ;$
14. $A \rightarrow\ id$
15. $|\ (\ *\ A\ )$
16. $DP \rightarrow\ \Lambda$
17. $|\ \ TYPE\ id\ \langle\ D\ \rangle ?\langle, TYPE\ id\ \langle\ D\ \rangle ?\rangle\star$
18. $D \rightarrow = E$
19. $TYPE \rightarrow\ bool$
20. $|\ int$
21. $|\ rat$
22. $|\ TYPE\ *$
23. $E \rightarrow\ id\ \ (\ CP\ )$
24. $|\ [\ E\ /\ E\ ]$
25. $|\ num\ \ E$
26. $|\ denom\ \ E$
    ~~$|\ id$~~
27. $|\ A$
28. $|\ true$
29. $|\ false$
30. $|\ entier$
31. $|\ (\ E\ \ +\ \ E\ )$
32. $|\ (\ E\ \ *\ \ E\ )$
33. $|\ (\ E\ \ =\ \ E\ )$
34. $|\ (\ E\ \ <\ \ E\ )$
35. $|\ (\ E\ )$
36. $|\ null$
37. $|\ (\ new\ TYPE\ )$
38. $|\ \&\ id$
39. $CP \rightarrow\ \Lambda$
40. $|\ E\ \langle,\ E\rangle\star$

FIGURE 1 – Grammaire (EBNF) du langage RAT étendu

# 1 Extension du langage RAT

Le compilateur demandé doit être capable de traiter le langage RAT étendu comme spécifié dans la figure 1. Pour simplifier la lecture, les ' ' sont omises autour des caractères, la répétition est noté $\star$ à distinguer du caractère '*' et le parenthésage EBNF est réalisé avec $\langle$ et $\rangle$.

Le nouveau langage permet de manipuler :

1. des pointeurs ;
2. des variables globales ;
3. des variables statiques locales ;
4. des paramètres par défaut.

## 1.1 Les pointeurs

RAT étendu permet de manipuler les pointeurs à l'aide d'une notation proche de celle de C :
— $A \rightarrow ( * A )$ : déréférencement : accès en lecture ou écriture à la valeur pointée par A ;
— $TYPE \rightarrow TYPE *$ : type des pointeurs sur un type TYPE ;
— $E \rightarrow null$ : pointeur null ;
— $E \rightarrow ( new\ TYPE )$ : initialisation d'un pointeur de type TYPE ;
— $E \rightarrow \& id$ : accès à l'adresse d'une variable.

Le traitement des pointeurs a été étudié lors du dernier TD, il s'agit ici de coder le comportement défini en TD. La libération de la mémoire n'est pas demandée.

### Exemple de programme valide

```
main{
  int * px = (new int);
  int x = 423;
  px = &x;
  int y = (*px);
  print y;
}
```

Ce programme affiche 423.

## 1.2 Les variables globales

Une variable globale est une variable accessible depuis n'importe quelle partie du programme, peu importe où elle est définie. Contrairement aux variables locales, qui sont limitées à la portée d'une fonction ou d'un bloc de code particulier, les variables globales peuvent être utilisées et modifiées dans différentes fonctions ou parties du programme. Dans le cadre du projet, et afin de ne pas compliquer la grammaire de manière excessive, nous limiterons leur déclaration au tout début du fichier.

RAT étendu permet de manipuler les variables globales à l'aide d'une notation à la RUST :
— $PROG \rightarrow VAR\star\ FUN\star\ id\ \ BLOC$
— $VAR \rightarrow static\ TYPE\ \ id\ \ =\ \ E\ ;$

### Exemple de programme valide

Ce programme affiche 13574.

```
static int x = 0;

int inc (int y){
  x = (x+1);
  return (x+y);
}

main {
  int a = inc (0);
  print a;
  a = inc (1);
  print a;
  a = inc (2);
  print a;
  a = inc (3);
  print a;
  print x;
}
```

## 1.3 Les variables statiques locales

Nous souhaitons pouvoir traiter les variables statiques locales *à la C*. Une variable statique locale à une fonction est une variable dont la portée est limitée à la fonction, qui est initialisée uniquement lors du premier appel à cette fonction et qui conserve la valeur qu'elle avait lors de l'appel précédent en cas de plusieurs appels successifs.

Il faut ajouter la règle : $I \rightarrow static\ TYPE\ id\ =\ E\ ;$ à la grammaire de RAT.

**Exemple de programme valide**

```
int f () {
  static int i = 0 ;
  i = (i+1);
  return i;
}
```

```
main{
  print (f ());
  print (f ());
  int pourlexercice = 0 ;
  print (f ());
  print (f ());
}
```

Doit s'exécuter correctement et afficher 1234.

## 1.4 Les paramètres par défaut

Certains langages de programmation, comme JavaScript, Python ou encore C++, permettent de définir des paramètres par défaut.

Vous pouvez définir une valeur par défaut pour un paramètre en l'assignant avec l'opérateur d'assignation d'égalité (=). Lorsque la fonction est invoquée sans argument pour un paramètre qui a une valeur par défaut, elle utilisera cette valeur par défaut.

Vous pouvez utiliser autant de paramètres par défaut que vous le souhaitez dans une fonction. La sémantique de ces paramètres n'est définie que lorsque tous les paramètres par défaut sont placés à la fin de la liste des paramètres. Cela permet de facilement ignorer les valeurs facultatives. La grammaire n'impose pas cette contrainte ; elle devra être vérifiée lors de la phase d'analyse sémantique.

Il faut modifier, dans la grammaire de RAT, les règles définisant les paramètres :

16 $DP \rightarrow\ \Lambda$

17 $\quad\quad |\ TYPE\ id\ \langle\ D\ \rangle ?\langle, TYPE\ id\ \langle\ D\ \rangle ?\rangle\star$

1. $D \rightarrow = E$

Aide à l'écriture du parser : il faut utiliser le mot clé `option`[1] dans Menhir. `option(X)` signifie `X` ou mot vide. L'attribut associé est une option de l'attribut de X.

1. `https://gallium.inria.fr/~fpottier/menhir/manual.html#sec38`

**Exemple de programme valide**

```
int f (int a, int b=2, int c=3 , int d=(2+2)){
  print a;
  print b;
  print c;
  print d;
  return a;
}

main{
  int f1 = f(10,20,30,40);
  int f2 = f(10,20,30);
  int f3 = f(10,20);
  int f4 = f(10);
}
```

Doit s'exécuter correctement et afficher 10203040102030410203410234.

### 1.5 Combinaisons des différentes constructions

Bien sûr ces différentes constructions peuvent être utilisées conjointement.

**Exemple de programme valide**

```
static int * g = (new int) ;

int f (int a, int b = (*g)) {
  static int c = 1 ;
  int r = ((a+b)+c);
  c = (c+1);
  return r;
}

main {
  *g = 3;
  print (f(1,2));
  print (f(1,2));
  print (f(1));
  print (f(1));
  *g = 4;
  print (f(1));
}
```

Ce programme doit afficher `457810`.

## 2 Travail demandé

Vous devez compléter le compilateur écrit en TP pour qu'il traite le langage RAT étendu. Vous devrez donc compléter les passes de gestion des identifiants, de typage, de placement mémoire et de génération de code.

Attention, il est indispensable de bien respecter la grammaire de la figure 1 pour que les tests automatiques qui seront réalisés sur votre projet fonctionnent.

D'un point de vue contrôle d'erreur, seules les vérifications de bonne utilisation des identifiants et de typage sont demandés. Les autres vérifications, par exemple déréférencement du pointeur null, ne sont pas demandées.

# 3 Conseil d'organisation du travail

Il est conseillé d'attendre la fin des TP pour commencer à coder le projet (le dernier TD porte sur les pointeurs), néanmoins le sujet est donné au début des TP pour que vous commenciez à réfléchir à la façon dont vous traiterez les extensions et que vous puissiez commencer à poser des questions aux enseignants lors des TD / TP.

Il est conseillé de finir la partie demandée en TP et que tous les tests unitaires fournis passent avant de commencer le projet, afin de partir sur des bases solides et saines.

Il est conseillé d'ajouter les fonctionnalités les unes après.

1. Introduction des affectables et **uniquement** des affectables
   (a) Remplacer les règles $I \to id$ et $E \to id$ pas les règles $I \to A$, $E \to A$ et $A \to id$.
   (b) Modifier les AST et toutes les passes (introduction des méthodes d'analyse des affectables).
   (c) Vérifier que les tests des TP passent.
2. Pour chaque nouvelle fonctionnalité de procéder par étape :
   (a) compléter la structure de l'arbre abstrait issu de l'analyse syntaxique ;
   (b) modifier la grammaire et construire l'arbre abstrait ;
   (c) tester avec le compilateur qui utilise des "passes NOP" ;
   (d) écrire les tests de la passe de gestion d'identifiant ;
   (e) compléter la structure de l'arbre abstrait issu de la passe de gestion des identifiants ;
   (f) modifier la passe de gestion des identifiants ;
   (g) tester avec le compilateur qui ne réalise que la passe de gestion de identifiants ;
   (h) écrire les tests de la passe de typage ;
   (i) compléter la structure de l'arbre abstrait issu de la passe de typage ;
   (j) modifier la passe de typage ;
   (k) tester avec le compilateur qui réalise la passe de gestion de identifiants et celle de typage ;
   (l) écrire les tests de la passe de placement mémoire ;
   (m) compléter la structure de l'arbre abstrait issu de la passe de placement mémoire ;
   (n) modifier la passe de placement mémoire ;
   (o) tester avec le compilateur qui réalise la passe de gestion de identifiants, celle de typage et de placement mémoire ;
   (p) écrire les tests de la passe de génération de code ;
   (q) modifier la passe de génération de code ;
   (r) tester avec le compilateur complet et itam.

# 4 Critères d'évaluation

Une grille critériée sera utilisée pour évaluer votre projet. Elle décrit, pour le style de programmation, le compilateur réalisé et le rapport, les critères évalués. Pour chacun d'eux est précisé ce qui est inacceptable, ce qui est insuffisant, ce qui est attendu et ce qui est au-delà des attentes.

| **Programmation fonctionnelle (40%)** | | | | | |
|---|---|---|---|---|---|
| | | Inacceptable | Insuffisant | Attendu | Au-delà |
| Compilation | | Ne compile pas | Compile avec des warnings | Compile sans warning | |
| Style de programmation fonctionnelle | | Le code est dans un style impératif | Il y a des fonctions auxiliaires avec accumulateurs non nécessaires | Les effets de bords ne sont que sur les structures de données et il n'y a pas de fonctions auxiliaires avec accumulateur non nécessaire | |
| Représentation des données | | Type non adapté à la représentation des données | Type partiellement adapté à la représentation des données | Type adapté à la représentation des données | Monade |
| Lisibilité | Code source documenté | Aucun commentaire n'est donné | Seuls des contrats succincts sont donnés | Des contrats complets sont donnés | Des contrats complets sont donnés et des commentaires explicatifs ajoutés dans les fonctions complexes |
| | Architecture claire | Mauvaise utilisation des modules / foncteurs et fonctions trop complexes | Mauvaise utilisation des modules / foncteurs ou fonctions trop complexes | Bonne utilisation des modules / foncteurs. Bon découpage en fonctions auxiliaires | Introduction, à bon escient, de nouveaux modules / foncteurs |
| | Utilisation des itérateurs | Aucun itérateur n'est utilisé | Seul List.map est utilisé | Une variété d'itérateurs est utilisé à bon escient dans la majorité des cas où c'est possible | Une variété d'itérateurs est utilisé à bon escient dans la totalité des cas où c'est possible |
| | Respects des bonnes pratiques de programmation | Code mal écrit rendant sa lecture et sa maintenabilité impossible (par exemple : failwith au lieu d'exceptions significatives, mauvaise manipulation des booléens, mauvais choix d'identifiants, mauvaise utilisation du filtrage...) | Code partiellement mal écrit rendant sa lecture et sa maintenabilité difficile | Code bien écrit facilitant sa lecture et sa maintenabilité | Code limpide |
| Fiabilité | Tests unitaires | Aucun test unitaire | Tests unitaires des fonctions hors *analyse_xxx*, ne couvrant pas tous les cas de bases et les cas généraux | Tests unitaires des fonctions hors *analyse_xxx*, couvrant les cas de bases et les cas généraux | Tests unitaires de toutes les fonctions, couvrant les cas de bases et les cas généraux |
| | Tests d'intégration | Aucun test d'intégration ou ne couvrant pas les quatre passes | Tests d'intégrations, des passes de gestion des identifiants, typage et génération de code, non complet | Tests d'intégration complets des passes de gestion des identifiants, typage et génération de code | Tests d'intégration complets des quatre passes |

| **Traduction des langages (40%)** | | | | |
|---|---|---|---|---|
| | Inacceptable | Insuffisant | Attendu | Au-delà |
| Grammaire | Non conforme à la grammaire du sujet | Partiellement conforme à la grammaire du sujet | Conforme à la grammaire du sujet | Plus complète que la grammaire du sujet |
| Fonctionnalités traitées intégralement | Aucune | Quelques-unes | Toutes celles du sujet | Fonctionnalités non demandées dans le sujet traitées correctement |
| Pointeurs | Non traité | Partiellement traité ou erroné | Complètement traité et correct | |
| Variables globales | Non traité | Partiellement traité ou erroné | Complètement traité et correct | |
| Variables statiques locales | Non traité | Partiellement traité ou erroné | Complètement traité et correct | |
| Paramètres par défaut | Non traité | Partiellement traité ou erroné | Complètement traité et correct | |

Le nombre de points accordés par fonctionnalité dépendra de la difficulté de celle-ci.

| **Rapport (20%)** | | | | |
|---|---|---|---|---|
| | Inacceptable | Insuffisant | Attendu | Au-delà |
| Forme | Beaucoup d'erreurs de syntaxe et d'orthographe et mise en page non soignée. | Beaucoup d'erreurs de syntaxe et d'orthographe ou mise en page non soignée | Peu d'erreurs de syntaxe ou d'orthographe et mise en page soignée | Pas d'erreur de syntaxe ou d'orthographe et mise en page soignée |
| Introduction | Non présente | Copier / coller du sujet | Bonne description du sujet et des points abordés dans la suite du rapport | |
| Types | Aucune justification sur l'évolution de la structure des AST | Justifications non pertinentes ou non complètes sur l'évolution de la structure des AST | Justifications pertinentes et complètes sur l'évolution de la structure des AST | Justifications pertinentes et complètes sur l'évolution de la structure des AST. Comparaison avec d'autres choix de conception. |
| Jugement de typage | Non donnés | Partiellement donnés ou erronés | Complètement donnés et corrects | |
| Pointeurs | Aucune explication sur leur traitement | Explications non pertinentes ou non complètes sur leur traitement | Explications pertinentes et complètes sur leur traitement (sans s'attarder sur les points déjà traités pour d'autres constructions) | Explications pertinentes et complètes sur leur traitement (sans...). Comparaison avec d'autres choix de conception. |
| Variables globales | Aucune explication sur leur traitement | Explications non pertinentes ou non complètes sur leur traitement | Explications pertinentes et complètes sur leur traitement (sans s'attarder sur les points déjà traités pour d'autres constructions) | Explications pertinentes et complètes sur leur traitement (sans...). Comparaison avec d'autres choix de conception. |
| Variables statiques locales | Aucune explication sur leur traitement | Explications non pertinentes ou non complètes sur leur traitement | Explications pertinentes et complètes sur leur traitement (sans s'attarder sur les points déjà traités pour d'autres constructions) | Explications pertinentes et complètes sur leur traitement (sans...). Comparaison avec d'autres choix de conception. |
| Paramètres par défaut | Aucune explication sur leur traitement | Explications non pertinentes ou non complètes sur leur traitement | Explications pertinentes et complètes sur leur traitement (sans s'attarder sur les points déjà traités pour d'autres constructions) | Explications pertinentes et complètes sur leur traitement (sans...). Comparaison avec d'autres choix de conception. |
| Conclusion | Non présente | Creuse | Bon recul sur les difficultés rencontrées | Bon recul sur les difficultés rencontrées et améliorations éventuelles |