

Présentation du projet - Compilateur Nano Go

Yoan GERAN, Amélie LEDEIN et Louis LEMONNIER

Et le sujet du projet est...

Programmer en OCaml un compilateur d'un sous-ensemble du langage Go¹, nommé ici Nano Go, vers de l'assembleur x86-64.

1. Go est un langage inspiré de C et Pascal <https://golang.org/doc/>.

Vous avez dit un compilateur ?

- ▶ Un **compilateur** est un programme qui lit un flux d'entrée structuré par une grammaire et qui produit une sortie.

-
2. ou **lexing** en anglais
 3. ou **parsing** en anglais

Vous avez dit un compilateur ?

- ▶ Un **compilateur** est un programme qui lit un flux d'entrée structuré par une grammaire et qui produit une sortie.
- ▶ Il est capable de :
 - ▶ signaler des erreurs de syntaxe, de typage, de sémantique, etc ;
 - ▶ faire des optimisations (vitesse d'exécution, taille du code, utilisation de la mémoire, etc.).

Vous avez dit un compilateur ?

- ▶ Un **compilateur** est un programme qui lit un flux d'entrée structuré par une grammaire et qui produit une sortie.
- ▶ Il est capable de :
 - ▶ signaler des erreurs de syntaxe, de typage, de sémantique, etc ;
 - ▶ faire des optimisations (vitesse d'exécution, taille du code, utilisation de la mémoire, etc.).
- ▶ Les **étapes de compilation** sont :

Langage \mathcal{L}_{input}	\implies	Analyse lexicale	\implies	$\{(t_0, v_0), (t_1, v_1), \dots\}$
$\{(t_0, v_0), (t_1, v_1), \dots\}$	\implies	Analyse grammaticale	\implies	AST
AST	\implies	Analyse sémantique	\implies	Type abstrait
Type abstrait	\implies	Génération du code	\implies	Langage \mathcal{L}_{output}

- ▶ t_i = un **jeton** (ou **token** en anglais)
- ▶ v_i = une **valeur**
- ▶ AST : Abstract Syntax Tree, i.e. arbre de syntaxe abstraite
- ▶ Analyse syntaxique = Analyse lexicale² + Analyse grammaticale³
- ▶ ATTENTION : Certains ouvrages parlent d'analyse syntaxique au lieu d'analyse grammaticale.

-
2. ou **lexing** en anglais
 3. ou **parsing** en anglais

Plus précisément, le sujet est...

Code Nano Go	\implies	Analyse lexicale	\implies	$\{(t_0, v_0), (t_1, v_1), \dots\}$
$\{(t_0, v_0), (t_1, v_1), \dots\}$	\implies	Analyse grammaticale	\implies	AST
AST	\implies	Vérification du typage	\implies	TAST
TAST	\implies	Génération du code	\implies	Code Assembleur

- ▶ AST : Abstract Syntax Tree
- ▶ TAST : Typed Abstract Syntax Tree

- ▶ Analyse syntaxique déjà codée pour vous.
- ▶ Analyse sémantique⁴ = Ici, typage seulement
- ▶ Typage (Premier rendu - **5 décembre**)
- ▶ Génération de code (Deuxième rendu - **18 janvier**)

4. L'analyse sémantique étudie l'arbre de syntaxe abstraite produit par l'analyse syntaxique pour éliminer au maximum les programmes qui ne sont pas corrects du point de vue de la sémantique.

Sommaire

Prise en main du Nano Go

Prise en main des règles de typage

Prise en main du code source

Lancement du projet

Exercice 1 - Comprendre la grammaire du Nano Go

- ▶ Lisez l'exemple dans la partie "Syntaxe" du sujet.
- ▶ Lisez la grammaire disponible à l'annexe A.2 - Figure 1 du sujet.
- ▶ Ecrivez une fonction écrite dans le langage de Nano Go qui calcule la somme de tous les entiers de 1 à n .

3 volontaires pour le faire au tableau !

Sommaire

Prise en main du Nano Go

Prise en main des règles de typage

Prise en main du code source

Lancement du projet

Rappels de l'objectif

- ▶ **Objectif : S'assurer de la conformité du typage.**
- ▶ La grammaire du Nano Go est disponible dans le sujet.
- ▶ Elle est représentée en OCaml par le type `pfile`.
- ▶ Le système de types de Nano Go est un système très simple, sans relation de sous-typage entre les types.

Typage statique

- ▶ **Type** : $\tau ::= \text{int} \mid \text{bool} \mid \text{string} \mid S \mid * \tau$
où S désigne un nom de structure (introduit avec la déclaration `type`).
- ▶ **Déclaration de variable** : $\text{var_decl} ::= x : \tau$
- ▶ **Déclaration de fonction** :
 $\text{fct_decl} ::= f(\tau_1, \dots, \tau_n) \Rightarrow \tau'_1, \dots, \tau'_m$ avec $n \geq 0$ et $m \geq 0$
- ▶ **Déclaration de structure** :
 $\text{struct_decl} ::= S\{\text{var_decl}, \dots, \text{var_decl}\}$
- ▶ **Contexte de typage** :

$$\Gamma ::= \Gamma, \text{struct_decl} \mid \Gamma, \text{fct_decl} \mid \Gamma, \text{var_decl} \mid \emptyset$$

Le fichier `src/tast.mli` code pour le moment le contexte pour les variables.

Règles d'inférence

- ▶ Une **règle d'inférence** est de la forme :

$$(nom) \frac{premise(s)}{conclusion}$$

où la conclusion est un **jugement**.

Règles d'inférence

- ▶ Une **règle d'inférence** est de la forme :

$$(nom) \frac{premise(s)}{conclusion}$$

où la conclusion est un **jugement**.

- ▶ Jugements :

- ▶ $\Gamma \vdash \tau$ *bf* signifie

« le type τ est bien formé dans le contexte Γ »

- ▶ $\Gamma \vdash e : \tau$ signifie

« dans le contexte Γ , l'expression e est bien typée de type τ »

- ▶ $\Gamma \vdash_l e : \tau$ signifie

« e est une valeur gauche (*l-value*) bien typée et de type τ »

- ▶ $\Gamma \vdash f(e_1, \dots, e_n) \Rightarrow \tau_1, \dots, \tau_m$ signifie

« dans le contexte Γ , l'appel de fonction $f(e_1, \dots, e_n)$ est bien typé et renvoie m valeurs de types τ_1, \dots, τ_m »

- ▶ $\Gamma \vdash s$ signifie

« dans le contexte Γ , l'instruction s est bien typée »

Bonne formation d'un type

Rappel : $\tau ::= \text{int} \mid \text{bool} \mid \text{string} \mid S \mid * \tau$

Bonne formation d'un type

Rappel : $\tau ::= \text{int} \mid \text{bool} \mid \text{string} \mid S \mid * \tau$

$$(BF_{int}) \frac{}{\Gamma \vdash \text{int} \text{ } bf} \quad (BF_{bool}) \frac{}{\Gamma \vdash \text{bool} \text{ } bf}$$

$$(BF_{string}) \frac{}{\Gamma \vdash \text{string} \text{ } bf}$$

$$(BF_{struct}) \frac{S \in \Gamma}{\Gamma \vdash S \text{ } bf} \quad (BF_{pointeur}) \frac{\Gamma \vdash \tau \text{ } bf}{\Gamma \vdash * \tau \text{ } bf}$$

où le jugement $\Gamma \vdash \tau \text{ } bf$ signifie « le type τ est bien formé dans le contexte Γ ».

Typage d'une expression (1)

$$(TE_{const}) \frac{\text{Constante } c \in \Sigma \text{ de type } \tau}{\Gamma \vdash c : \tau}$$

$$(TE_{moins}) \frac{\Gamma \vdash e : \text{int}}{\Gamma \vdash - e : \text{int}} \quad (TE_{neg}) \frac{\Gamma \vdash e : \text{bool}}{\Gamma \vdash !e : \text{bool}}$$

$$(TE_{eq}) \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \quad op \in \{==, !=\} \quad e_1 \neq \text{nil} \vee e_2 \neq \text{nil}}{\Gamma \vdash e_1 \text{ op } e_2 : \text{bool}}$$

$$(TE_{lin}) \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \quad op \in \{<, <=, >, >=\}}{\Gamma \vdash e_1 \text{ op } e_2 : \text{bool}}$$

$$(TE_a) \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \quad op \in \{+, -, *, /, \%\}}{\Gamma \vdash e_1 \text{ op } e_2 : \text{int}}$$

$$(TE_b) \frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool} \quad op \in \{\&\&, ||\}}{\Gamma \vdash e_1 \text{ op } e_2 : \text{bool}}$$

où le jugement $\Gamma \vdash e : \tau$ signifie « dans le contexte Γ , l'expression e est bien typée de type τ ».

Typage d'une expression (1)

$$(TE_{const}) \frac{\text{Constante } c \in \Sigma \text{ de type } \tau}{\Gamma \vdash c : \tau}$$

$$(TE_{moins}) \frac{\Gamma \vdash e : \text{int}}{\Gamma \vdash - e : \text{int}} \quad (TE_{neg}) \frac{\Gamma \vdash e : \text{bool}}{\Gamma \vdash !e : \text{bool}}$$

$$(TE_{eq}) \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \quad op \in \{==, !=\} \quad e_1 \neq \text{nil} \vee e_2 \neq \text{nil}}{\Gamma \vdash e_1 \text{ op } e_2 : \text{bool}}$$

$$(TE_{lin}) \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \quad op \in \{<, <=, >, >=\}}{\Gamma \vdash e_1 \text{ op } e_2 : \text{bool}}$$

$$(TE_a) \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \quad op \in \{+, -, *, /, \%\}}{\Gamma \vdash e_1 \text{ op } e_2 : \text{int}}$$

$$(TE_b) \frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool} \quad op \in \{\&\&, ||\}}{\Gamma \vdash e_1 \text{ op } e_2 : \text{bool}}$$

Exercice 2 - Démontrer que l'expression suivante a bien le type attendu : `4 + 75 >= 100 || 3 == 42 && true`.

Typage d'une expression (2)

$$\frac{\Gamma \vdash_l e : \tau' \quad \Gamma \vdash e.x : \tau}{\Gamma \vdash_l e.x : \tau}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash_l x : \tau}$$

$$\frac{\Gamma \vdash_l e : \tau \quad e \neq _}{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \ni S\{x : \tau\}}{\Gamma \vdash e.x : \tau}$$

$$\frac{\Gamma \vdash e : *S \quad e \neq \text{nil} \quad \Gamma \ni S\{x : \tau\}}{\Gamma \vdash e.x : \tau}$$

$$\frac{\Gamma \vdash \tau \text{ bf}}{\Gamma \vdash \text{nil} : * \tau}$$

$$\frac{\Gamma \vdash_l e : \tau}{\Gamma \vdash \&e : * \tau}$$

$$\frac{\Gamma \vdash e : * \tau \quad e \neq \text{nil}}{\Gamma \vdash_l *e : \tau}$$

$$\frac{S \in \Gamma}{\Gamma \vdash \text{new}(S) : *S}$$

où le jugement $\Gamma \vdash_l e : \tau$ signifie « e est une valeur gauche (*l-value*) bien typée et de type τ ».

Typage d'une expression (2)

$$\frac{\Gamma \vdash_1 e : \tau' \quad \Gamma \vdash e.x : \tau}{\Gamma \vdash_1 e.x : \tau}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash_1 x : \tau}$$

$$\frac{\Gamma \vdash_1 e : \tau \quad e \neq _}{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \ni S\{x : \tau\}}{\Gamma \vdash e.x : \tau}$$

$$\frac{\Gamma \vdash e : *S \quad e \neq \text{nil} \quad \Gamma \ni S\{x : \tau\}}{\Gamma \vdash e.x : \tau}$$

$$\frac{\Gamma \vdash \tau \text{ bf}}{\Gamma \vdash \text{nil} : * \tau}$$

$$\frac{\Gamma \vdash_1 e : \tau}{\Gamma \vdash \&e : * \tau}$$

$$\frac{\Gamma \vdash e : * \tau \quad e \neq \text{nil}}{\Gamma \vdash_1 *e : \tau}$$

$$\frac{S \in \Gamma}{\Gamma \vdash \text{new}(S) : *S}$$

Exercice 3 - Démontrer que l'expression suivante a bien le type attendu, avec $\Gamma = S_{\text{coord}}\{x : \text{int}, y : \text{int}\} : e.x$

Typage d'un appel

$$\frac{\Gamma \vdash f(e_1, \dots, e_n) \Rightarrow \tau_1}{\Gamma \vdash f(e_1, \dots, e_n) : \tau_1}$$

$$\frac{\Gamma \ni f(\tau_1, \dots, \tau_n) \Rightarrow \tau'_1, \dots, \tau'_m \quad \forall i, \Gamma \vdash e_i : \tau_i}{\Gamma \vdash f(e_1, \dots, e_n) \Rightarrow \tau'_1, \dots, \tau'_m}$$

$$\frac{n \geq 2 \quad \Gamma \ni f(\tau_1, \dots, \tau_n) \Rightarrow \tau'_1, \dots, \tau'_m \quad \Gamma \vdash g(e_1, \dots, e_k) \Rightarrow \tau_1, \dots, \tau_n}{\Gamma \vdash f(g(e_1, \dots, e_k)) \Rightarrow \tau'_1, \dots, \tau'_m}$$

Cette dernière règle permet de passer directement les n résultats d'une fonction g en arguments d'une fonction f .

où le jugement $\Gamma \vdash f(e_1, \dots, e_n) \Rightarrow \tau_1, \dots, \tau_m$ signifie « dans le contexte Γ , l'appel de fonction $f(e_1, \dots, e_n)$ est bien typé et renvoie m valeurs de types τ_1, \dots, τ_m ».

Typage d'une instruction (1)

$$\frac{\Gamma \vdash_I e : \text{int}}{\Gamma \vdash e++}$$
$$\frac{\Gamma \vdash_I e : \text{int}}{\Gamma \vdash e--}$$
$$\frac{\forall i, \Gamma \vdash e_i : \tau_i}{\Gamma \vdash \text{fmt.Print}(e_1, \dots, e_n)}$$
$$\frac{n \geq 2 \quad \Gamma \vdash f(e_1, \dots, e_k) \Rightarrow \tau_1, \dots, \tau_n}{\Gamma \vdash \text{fmt.Print}(f(e_1, \dots, e_k))}$$
$$\frac{\forall i, \Gamma \vdash_I e_j : \tau_j \quad \forall i, \Gamma \vdash e'_i : \tau_i}{\Gamma \vdash e_1, \dots, e_n = e'_1, \dots, e'_n}$$
$$\frac{\forall i, \Gamma \vdash_I e_j : \tau_j \quad \Gamma \vdash f(e'_1, \dots, e'_m) \Rightarrow \tau_1, \dots, \tau_n}{\Gamma \vdash e_1, \dots, e_n = f(e'_1, \dots, e'_m)}$$
$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash b_1 \quad \Gamma \vdash b_2}{\Gamma \vdash \text{if } e \text{ } b_1 \text{ else } b_2}$$
$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash b}{\Gamma \vdash \text{for } e \text{ } b}$$
$$\frac{\forall i, \Gamma \vdash e_i : \tau_i}{\Gamma \vdash \text{return } e_1, \dots, e_n}$$
$$\frac{\Gamma \vdash f(e_1, \dots, e_k) \Rightarrow \tau_1, \dots, \tau_n}{\Gamma \vdash \text{return } f(e_1, \dots, e_k)}$$

Typeage d'une instruction (2)

$$\frac{\Gamma \vdash \tau \text{ bf} \quad \Gamma + x_1 : \tau, \dots, x_n : \tau \vdash \{s_2; \dots; s_m\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n \tau; s_2; \dots; s_m\}}$$
$$\frac{\Gamma \vdash \tau \text{ bf} \quad \forall i, \Gamma \vdash e_i : \tau \quad \Gamma + x_1 : \tau, \dots, x_n : \tau \vdash \{s_2; \dots; s_n\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n \tau = e_1, \dots, e_n; s_2; \dots; s_n\}}$$
$$\frac{\forall i, e_i \neq \text{nil} \quad \forall i, \Gamma \vdash e_i : \tau_i \quad \Gamma + x_1 : \tau_1, \dots, x_n : \tau_n \vdash \{s_2; \dots; s_m\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n = e_1, \dots, e_n; s_2; \dots; s_m\}}$$
$$\frac{\Gamma \vdash \tau \text{ bf} \quad \Gamma \vdash f(e_1, \dots, e_k) \Rightarrow \tau^n \quad \Gamma + x_1 : \tau, \dots, x_n : \tau \vdash \{s_2; \dots; s_m\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n \tau = f(e_1, \dots, e_k); s_2; \dots; s_m\}}$$
$$\frac{\Gamma \vdash f(e_1, \dots, e_k) \Rightarrow \tau_1, \dots, \tau_n \quad \Gamma + x_1 : \tau_1, \dots, x_n : \tau_n \vdash \{s_2; \dots; s_m\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n = f(e_1, \dots, e_k); s_2; \dots; s_m\}}$$
$$\frac{\Gamma \vdash s_1 \quad \Gamma \vdash \{s_2; \dots; s_n\}}{\Gamma \vdash \{s_1; \dots; s_n\}}$$
$$\frac{}{\Gamma \vdash \{\}}$$

Par ailleurs, toutes les variables introduites dans un *même* bloc doivent porter des noms différents. Une exception est faite pour la variable « $_$ ». où le jugement $\Gamma \vdash s$ signifie « dans le contexte Γ , instruction s est bien typée ».

Typage d'une instruction (2)

$$\frac{\Gamma \vdash \tau \text{ bf} \quad \Gamma + x_1 : \tau, \dots, x_n : \tau \vdash \{s_2; \dots; s_m\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n \tau; s_2; \dots; s_m\}}$$
$$\frac{\Gamma \vdash \tau \text{ bf} \quad \forall i, \Gamma \vdash e_i : \tau \quad \Gamma + x_1 : \tau, \dots, x_n : \tau \vdash \{s_2; \dots; s_n\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n \tau = e_1, \dots, e_n; s_2; \dots; s_n\}}$$
$$\frac{\forall i, e_i \neq \text{nil} \quad \forall i, \Gamma \vdash e_i : \tau_i \quad \Gamma + x_1 : \tau_1, \dots, x_n : \tau_n \vdash \{s_2; \dots; s_m\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n = e_1, \dots, e_n; s_2; \dots; s_m\}}$$
$$\frac{\Gamma \vdash \tau \text{ bf} \quad \Gamma \vdash f(e_1, \dots, e_k) \Rightarrow \tau^n \quad \Gamma + x_1 : \tau, \dots, x_n : \tau \vdash \{s_2; \dots; s_m\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n \tau = f(e_1, \dots, e_k); s_2; \dots; s_m\}}$$
$$\frac{\Gamma \vdash f(e_1, \dots, e_k) \Rightarrow \tau_1, \dots, \tau_n \quad \Gamma + x_1 : \tau_1, \dots, x_n : \tau_n \vdash \{s_2; \dots; s_m\}}{\Gamma \vdash \{\text{var } x_1, \dots, x_n = f(e_1, \dots, e_k); s_2; \dots; s_m\}}$$
$$\frac{\Gamma \vdash s_1 \quad \Gamma \vdash \{s_2; \dots; s_n\}}{\Gamma \vdash \{s_1; \dots; s_n\}}$$
$$\frac{}{\Gamma \vdash \{\}}$$

où le jugement $\Gamma \vdash s$ signifie « dans le contexte Γ , instruction s est bien typée ».

Exercice 4 : Démontrer que les 2 programmes écrits à l'exercice 1 ont le type attendu.

Sommaire

Prise en main du Nano Go

Prise en main des règles de typage

Prise en main du code source

Lancement du projet

Hiérarchie générale des fichiers

Code Nano Go	\implies	Analyse lexicale	\implies	$\{(t_0, v_0), (t_1, v_1), \dots\}$
$\{(t_0, v_0), (t_1, v_1), \dots\}$	\implies	Analyse grammaticale	\implies	AST
AST	\implies	Vérification du typage	\implies	TAST
TAST	\implies	Génération du code	\implies	AA

- ▶ AST : Abstract Syntax Tree
- ▶ TAST : Typed Abstract Syntax Tree
- ▶ AA : Abstract Assembly

Hiérarchie générale des fichiers

Code Nano Go	\implies	lexer.mll	\implies	$\{(t_0, v_0), (t_1, v_1), \dots\}$
$\{(t_0, v_0), (t_1, v_1), \dots\}$	\implies	parser.mly	\implies	AST
AST	\implies	typing.ml	\implies	TAST
TAST	\implies	compile.ml	\implies	AA

- ▶ AST : Abstract Syntax Tree
- ▶ TAST : Typed Abstract Syntax Tree
- ▶ AA : Abstract Assembly

Hiérarchie générale des fichiers

Code Nano Go	⇒	lexer.mll	⇒	$\{(t_0, v_0), (t_1, v_1), \dots\}$
$\{(t_0, v_0), (t_1, v_1), \dots\}$	⇒	parser.mly	⇒	AST
AST	⇒	typing.ml	⇒	TAST
TAST	⇒	compile.ml	⇒	AA

- ▶ AST : Abstract Syntax Tree → [ast.mli](#)
- ▶ TAST : Typed Abstract Syntax Tree → [tast.mli](#)
- ▶ AA : Abstract Assembly → [x86_64.ml](#) et [x86_64.mli](#)

Hiérarchie générale des fichiers

Code Nano Go	⇒	lexer.mll	⇒	$\{(t_0, v_0), (t_1, v_1), \dots\}$
$\{(t_0, v_0), (t_1, v_1), \dots\}$	⇒	parser.mly	⇒	AST
AST	⇒	typing.ml	⇒	TAST
TAST	⇒	compile.ml	⇒	AA

- ▶ AST : Abstract Syntax Tree → [ast.mli](#)
- ▶ TAST : Typed Abstract Syntax Tree → [tast.mli](#)
- ▶ AA : Abstract Assembly → [x86_64.ml](#) et [x86_64.mli](#)

- ▶ [lib.ml](#) : Extension de la bibliothèque standard de OCaml
- ▶ [pretty.ml](#) : Affichage des AST et des TAST
- ▶ [main.ml](#) : Programme général et parsing de la ligne de commande (option `--debug`, `--parse-only` et `--type-only`)

Hiérarchie générale des fichiers

Code Nano Go	⇒	<code>lexer.mll</code>	⇒	$\{(t_0, v_0), (t_1, v_1), \dots\}$
$\{(t_0, v_0), (t_1, v_1), \dots\}$	⇒	<code>parser.mly</code>	⇒	AST
AST	⇒	<code>typing.ml</code>	⇒	TAST
TAST	⇒	<code>compile.ml</code>	⇒	AA

- ▶ AST : Abstract Syntax Tree → `ast.mli`
- ▶ TAST : Typed Abstract Syntax Tree → `tast.mli`
- ▶ AA : Abstract Assembly → `x86_64.ml` et `x86_64.mli`

- ▶ `lib.ml` : Extension de la bibliothèque standard de OCaml
- ▶ `pretty.ml` : Affichage des AST et des TAST
- ▶ `main.ml` : Programme général et parsing de la ligne de commande (option `--debug`, `--parse-only` et `--type-only`)

- ▶ Pour lancer la compilation : `Makefile`
- ▶ Pour gérer la compilation : `dune` et `dune-project`

Quelques outils pour vous aider

Quelques outils pour vous aider

- ▶ Dune : <https://dune.readthedocs.io/>

Quelques outils pour vous aider

- ▶ Dune : <https://dune.readthedocs.io/>
- ▶ Makefile :
 - ▶ La commande `make` crée le compilateur nommé `pgoc`.
 - ▶ La commande `make clean` efface tous les fichiers que `make` a engendrés et ne laisse dans le répertoire que les fichiers sources.
 - ▶ Le fichier `Makefile` distribué utilise `dune`, mais vous pouvez le réécrire pour éviter l'installation de ce paquet.

Quelques outils pour vous aider

- ▶ Dune : <https://dune.readthedocs.io/>
- ▶ Makefile :
 - ▶ La commande `make` crée le compilateur nommé `pgoc`.
 - ▶ La commande `make clean` efface tous les fichiers que `make` a engendrés et ne laisse dans le répertoire que les fichiers sources.
 - ▶ Le fichier `Makefile` distribué utilise `dune`, mais vous pouvez le réécrire pour éviter l'installation de ce paquet.
- ▶ Git : Dites-le si vous rencontrez des difficultés avec Git !

Sommaire

Prise en main du Nano Go

Prise en main des règles de typage

Prise en main du code source

Lancement du projet

Consignes générales

Travail à faire :

- ▶ **seul.e.**
- ▶ à partir des sources initiales disponibles sur eCampus
- ▶ **en modifiant uniquement les TODO.**
TODO uncomment signifie qu'il faut simplement dé-commenter le code déjà écrit.

Travail à rendre :

- ▶ sur GitHub en invitant SuperProf-Prog1-2022
- ▶ contenant :
 - ▶ Les **sources**
 - ▶ Un **court rapport** (format ASCII, Markdown ou PDF) expliquant les différents choix techniques qui ont été faits, les difficultés rencontrées, les éléments réalisés et leur test, les éléments non réalisés et plus généralement toute différence par rapport à ce qui a été demandé.
 - ▶ Des **tests**.

Quelques conseils

- ▶ **Procéder construction par construction** et s'assurer à chaque étape que votre code compile et passe les tests donnés dans le répertoire tests.
- ▶ **Sauvegarder régulièrement** (Git ou support externe) afin de ne pas perdre des versions qui fonctionnent.
- ▶ Vous devez documenter votre code et écrire un journal de votre développement (le format Markdown est très facile pour cette tâche). Ces notes et commentaires vous permettront de générer rapidement le rapport du projet.
- ▶ **Respectez le calendrier** et les étapes proposées afin d'avancer régulièrement.
- ▶ **Et n'hésitez pas à poser des questions !**

Exercice 5 - A vous de jouer !

- ▶ Comparez la grammaire du sujet avec le code fourni.
- ▶ Écrivez la fonction `ast_file` (fichier `pretty.ml`) permettant de visualiser vos premiers AST.
- ▶ Testez-la sur des exemples simples.
- ▶ Ajoutez les vérifications concernant "main" et "fmt".
- ▶ Réussir à typer les fichiers `hello-world.c`, `testfile-print-1.go` et `return.go`