Analyse syntaxique (parsing) Ast type Simplification (aptimisation middle - end ation - (angage internicionie (batt code) code bjet (assembleur) Exercise 2: Differe problème que pour le nombre de parenthèses : (e n'eot pas possible) X X On peut écrice n'importe que code declars (* { c = 1, BEGIN(c); } < c (* { c + ; } < c > 14 Inimporte quoi Exercise 3:	pourroit pas vérifier qu'il my ait autant de parenthèses d'un côté de l'autre. Rappel: Expression régulière N -> t N -> tN
Analyse syntaxique (passing) Ast type Analyse sémantique Ast type Simplification/optimisation middle-end code Simplification/optimisation middle-end code Emission de code code objet (assemblesser) Exercise 2: Ditême problème que pour le nombre de parenthises: ce n'est pas possit (* {c=1, BEGIN(c); } <c> *) {c; if (c==0) BEGIN(o); } <c> (* {c++;} <c> 14</c></c></c>	autant de parenthèses d'un côté de l'autre. Rappel: Expression régulière N -> t N -> tN
Analyse syntaxique (paesing) Ast type Simplification l'optimisation middle - end cation = (angage intermédiaire (baste code) code tenission de code code objet (assemblesser) Exercice 2: Differe problème que pour le nombre de parenthèses : ce n'est pas possit 2) // code dedans (* {c=1, BEGIN(c); } <c> *) {c; if (c==0) BEGIN(o); } <c> 4 Code quoi</c></c>	pourroit pas vérifier qu'il y ait autant de parenthèses d'un côté de l'autre. Rappel: Expression régulière N -> t N -> tN
Ast type Ast type Simplification/optimisation middle-end ation (anguage intermidiaire (buyte code) code Code object (assemblence) Exercise 2: Diffine problème que pour le nombre de parenthises : ce n'est pas possible (* {c=1, BEGIN(c); } <c> *) {c; if (c==0) BEGIN(o); } <c> (* {c++;} <c> 1/2 A importe quoi</c></c></c>	autant de parenthèses d'un côté de l'autre. Rappel: Expression régulière N -> t N -> tN
Analyse sémantique Ast type Simplification/aptimisation middle - end ation - (angage intermédiaire (byte code) code Emission de code back-end code objet (assembleur) Exercise 2: On peut écrise n'importe que code dedans (* {c=1, BEGIN(c); } <c> *) {c; if (c==0) BEGIN(o); } <c> (* {c++;} <c> 14</c></c></c>	Rappel: Expression régulière N -> t N -> tN
Ast type Simplification (aptimisation middle - end ation - Cangage intermédiaire (bute code) code trission de code back-end code objet (assembleur) Exercise 2: Tième problème que pour le nombre de parenthèses : ce n'est pas possib 1 / / 3	Rappel: Expression regulière N → E N → tN
Simplification/aptimisation middle-end ation - language intermidiaire (butte code) code Emission de code back-end code objet (assembleur) Exercise 2: Differe problème que pour le nombre de parenthises: ce n'est pas possit) // on peut écree n'importe quel code dedans (* {c=1, BEGIN(c); } <c> *) {c; if (c==0) BEGIN(o); } <c> (* {c++;} <c> 1/2 importe quoi</c></c></c>	N → F
code l'assemblement (byte code) code objet (assemblement) Exercice 2: D'Ilême problème que pour le nombre de parenthèses : ce n'est pas possit 1 / / San peut écrice n'importe que code dedans (* {c=1, BEGIN(C); } <c> *) {c; if (c==0) BEGIN(0); } <c> (* {c++;} <c> 1/9 n'importe quoi</c></c></c>	N → F
Exercise 2: Differe problème que pour le nombre de parenthises : ce n'est pas possible de la code de d	n→ fn
traission de code code objet (assembleur) Exercice 2: Tiême problème que pour le nombre de parenthises : ce n'est pas possible (*	
code objet (assembleur) Exercise 2: Différme problème que pour le nombre de parenthises : ce n'eot pas possible 1	ble avec une regex.
D'Ilême problème que pour le nombre de parenthèses : ce n'est pas possible) // // On peut écrise n'importe quel code dedans (* {c=1, BEGIN(c); } <c> *) {c; if (c==0) BEGIN(o); } <c> (* {c++;} <c> 1/9</c></c></c>	ble avec mre reger.
Thême problème que pour le nombre de parenthèses : ce n'est pas possible) // // c=1, BEGIN(c); } c> (* {c=-; if (c==0) BEGIN(o); } c> (* {c++;} c> (*)	ble avec me regex.
Thême problème que pour le nombre de parenthèses : ce n'est pas possible) // // c=1, BEGIN(c); } c> (* {c=-; if (c==0) BEGIN(o); } c> (* {c++;} c> (*)	ble avec une reger.
) //	ble avec me reger.
) //	ble avec me reger.
) //	
(* {c=1, BEGIN(c); } <c> *) {c; if (c==0) BEGIN(0); } <c> (* {c++;} <c></c></c></c>	
(* {c=1, BEGIN(c); } <c> *) {c; if (c==0) BEGIN(0); } <c> (* {c++;} <c></c></c></c>	
<pre><c> *) {c; if (c==0) BEGIN(0); } <c> (* {c++;} <c> 14</c></c></c></pre>	
<c> (* {c++;}</c>	
<c> 14</c>	
s n'importe quoi	
xercile 3:	
. Na we o.	
) - lex . ml = générateur de lexer ast, ml	[t
	lex. ml
- parse mly = généraleur de parser type ast =	rule tokens = parse '+' { Plus }
- ast, ml Add of ast x ast	
- eval. ml of ast * ast	\\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\
Int of int	[0-9] as i TINT (int_of_string i)
	l eol {Fol}
parse. My 1. start main 1. left Plus 1. left DUL	(CPAR)
1. token Fol main: expr Fol (\$1);	1')' { RPAR }
1. token PLUS expr: premier elem & straisième	
de la com	
\$4 \$2 \$3	
1. token (int) INT expr 17 ML (\$1,\$3) 1 1. token (PAR INT { Int (\$4)}	splus priotaire à la fin

^{1.} token RPAR | LPAR
1. type <ast> main } les deux règles
1. type <ast> expr renvoient un ast

(2) <u>Forctionnel:</u> pratique pour l'ast, pattern matching. <u>Orienté-objet:</u> compliqué pour l'ast (hiérarchie complexe), compliqué	pour le typage	
3) - analyse sémantique - générateur de code (pour le moment, on n'a fait que la portie interprétation)		
Exercice 4:		
1. left Plus nin) 1. left nul Div		
(4) expr:	(1)0	
	(2))On veut pas de expr à ganche dans	
expr Plus expr {Add (\$1,\$3)}	notre grammaire.	
expr 114L expr \ \[\lambda (\\$ 1, \\$ 3) \]		
	skbc :	
LPAR expr RPAR \$\$2} Sil fandrait Evidenment modifier	INT exper	
lexpr nin expr les autres fichiers pour tout	ID expc1	
expr DiV expr \\ faire correspondre	INT	
ID -> identificant \}	ID	
ID LPAR expr RPAR {}		
sappel de fanction		
	expr.A:	
	PLUS expr	
Exercise 5:		
(1) if::= if expr then expr[else expr] if A then if B then		
2 ast possible		
if	if	
A if D	A if	
в с	BCD	
(2) ifther: - if expr then expr		
if expr then if then else expr		
, ,		
iftherelse ::= (expr sans if)		
ATTIONNE CEXIL SING IT		
e::=		