L’analyse LR : Left to right, Rightmost derivation, analyseur pour les grammaires non contextuelles qui lit l'entrée de gauche à droite et produit une dérivation droite. On parle aussi d'analyseur LR(k) où k représente le nombre de symboles « anticipés » et non consommés qui sont utilisés pour prendre des décisions d'analyse syntaxique.

L'analyse LL est une analyse descendante pour un sous-ensemble de grammaire non contextuelle. Il analyse l'entrée de gauche à droite (Left to right) et en construit une dérivation à gauche (Leftmost derivation). Une analyse LL est appelée analyse LL(k) lorsqu'elle utilise k lexèmes d'avance.

L’analyse LALR (Look-Ahead Left Recursive) permet d'améliorer la sélectivité d'un analyseur syntaxique LR. Il est utilisé lorsque le traitement des données doit répondre à de multiples cas.

L’analyse GLR (Generalized LR) est une extension de l’analyseur LR, pour gérer les grammaires non déterministe et ambigues.

Flyweight : Lorsque de nombreux (petits) objets doivent être manipulés, mais qu'il serait trop coûteux en mémoire s'il fallait instancier tous ces objets, il est judicieux d'implémenter le poids-mouche.
Dans le cas d'une classe représentant des données, il est parfois possible de réduire le nombre d'objets à instancier si tous ces objets sont semblables et se différencient sur quelques paramètres. Si ces quelques paramètres peuvent être extraits de la classe et passés ensuite via des paramètres des méthodes, on peut réduire grandement le nombre d'objets à instancier.

Composite : Manipuler un groupe d’objet de la même façon que s’il s’agissait d’un seul objet
Visitor : Manière de séparer un algo d’une structure de données
Command : Encapsule la notion d’invocation. Séparer complètement le code initiateur de l’action du code de l’action elle-même.

TC-1 Code to Write

src/parse/scantiger.ll
: The scanner must be completed to read strings, identifiers etc. and track locations. Symbols (i.e., identifiers) must be returned as misc::symbol objects, not strings. The locations are tracked. The class Location to use is produced by Bison: src/parse/location.hh.

To track of locations, adjust your scanner, use YY_USER_ACTION and the yylex prologue:

src/parse/parsetiger.yy
- The grammar must be complete but without actions.
- Complete %printer to implement --parse-trace support (see TC-1 Samples). Pay special attention to the display of strings and identifiers.
- Use %destructor to reclaim the memory bound to symbols thrown away during error recovery.

src/parse/tiger-parser.cc
: The class TigerParser drives the lexing and parsing of input file. Its implementation in src/parse/tiger-parser.cc is incomplete.

lib/misc/symbol.*
lib/misc/unique.*
: The class misc::symbol keeps a single copy of identifiers, see lib/misc. Its implementation in lib/misc/symbol.hxx and lib/misc/symbol.cc is incomplete. Note that running ‘make check’ in lib/misc exercises lib/misc/test-symbol.cc: having this unit test pass should be a goal by itself. As a matter of fact, unit tests were left to help you: once they pass successfully you may proceed to the rest of the compiler. misc::symbol's implementation is based on misc::unique, a generic class implementing the Flyweight design pattern. The definition of this class, lib/misc/unique.hxx, is also to be completed.

lib/misc/variant.*
: The implementation of the class template misc::variant<T0, T1> lacks a couple of conversion operators that you have to supply.

Finding prelude.tih

When run, the compiler needs the file prelude.tih that includes the signature of all the primitives.

TC-2 Goals

Understanding the use of a glr Parser : The parser should now use all the possibilities of a glr parser.
Error recovery with Bison : Using the error token, and building usable asts in spite of lexical/syntax errors.
Using stl containers : The ast uses std::list, misc::symbol uses std::set.
Inheritance : The ast hierarchy is typical example of a proper use of inheritance, together with...
Inclusion polymorphism : An intense use of inclusion polymorphism for accept.
Use of constructors and destructors
: In particular using the destructors to reclaim memory bound to components. Virtual Dynamic and static bindings.
misc::indent
: misc::indent extends std::ostream with indentation features. Use it in the PrettyPrinter to pretty-print. Understanding how misc::indent works will be checked later, see TC-3 Goals.
The Composite design pattern : The ast hierarchy is an implementation of the Composite pattern.
The Visitor design pattern : The PrettyPrinter is an implementation of the Visitor pattern.