Lexical analyzer: divides program text into “words” or tokens
A token is a syntactic category. A lexeme is the substring corresponding to the token.
Parsing: Diagramming Sentences
A parser takes a sequence of tokens as input and produces a parse tree (or derivation) if the input is valid.
The state of a program is all of the current variable and heap values.
Advantages of Functional Languages:
1. Tractable program semantics
2. referential transparency
3. No side-effects
Disadvantages of Functional Languages:
1. Efficiency
2. Compiler implementation
3. Not appropriate for Operating System programs.
Functions and type inferences are polymorphic: operate on more than one type.

```
let rec fold f acc lst = match lst with
| [] -> acc
| hd :: tl -> fold f (f acc hd) tl
let rec map f lst = match lst with
| [] -> []
| hd :: tl -> f hd :: map f tl
let map myfun lst = fold (fun acc elt -> (myfun elt) :: acc) [] lst
```

ML and Haskell have strong static typing and type inference while most others have “strong” dynamic typing.
Currying: “if you fix some arguments, you get a function of the remaining arguments.”
Regular expressions are a way to specify sets of strings. We use them to describe tokens.

```
fold_left f a [1;2;...;n] == f (...(f (f a 1) 2)) n
fold_left (fun a e -> e :: a) [] [1;2;3] = [3;2;1]
fold_left (fun a e -> a @ [e]) [] [1;2;3] = [1;2;3]
```

```
fold_right f [1;2;...;n] == f 1 (f 2 (... (f n b)))
fold_left (fun a e -> e :: a) [] [1;2;3] = [1;2;3]
fold_left (fun a e -> a @ [e]) [] [1;2;3] = [3;2;1]
```

Languages definition: Let Sigma be a set of characters. A language over Sigma is a set of strings of characters drawn from Sigma. Sigma is called the alphabet.
Each regular expression is a notation for a regular language.
Maximal Munch rule: Pick the longest possible substring that matches R. In general, use the rule listed first.
Finite automata are formal models of computation that can accept regular languages corresponding to regular expressions.
Nondeterministic finite automata feature epsilon transitions and multiple outgoing edges for the same input symbol.
Deterministic Finite Automata: One transition per input per state. No epsilon-moves.
NFA and DFAs have the same expressive power.
Context-free grammars are a notation for specifying formal languages. They contain terminals, non-terminals and productions. (rewrite rules)
Regular languages are the weakest formal languages while context-free languages are stronger type of formal language.
Limitations of regular languages:
1. A finite automaton that runs long enough must repeat states.
2. A finite automaton can’t remember how often it has visited a particular state.
3. Language of balanced parentheses is not regular.
Many grammars generate the same language.
A grammar is ambiguous if it has more than one parse tree for some string.
Instead of rewriting ambiguous grammar, most tools allow precedence and associativity declaration to solve the problem.
A top-down parser starts to work from the initial grammar rules (instead of the first token.) A recursive descent parser exhaustively tries all productions in the grammar. Earley parsers are top-down.
A recursive descent parser exhaustively tries all productions in the grammar. It does not work as it cannot solve left recursion.Main disadvantage is backtracking.
Speedcoding has an extensive set of operations to make the job of programming as easy as possible.
Reasons for speedcoding:
1. Some computing groups are working against time and the elapsed time for solving a problem may often be reduced by minimizing the time for programming and checking out the problem even though the running time is increased.
2. It is a matter of economy. Machine time spent checking out problems is frequently a very appreciable percentage of the total machine time.

Speedcoding is an interpretive system. Data and instructions have completely different forms and are treated differently.
Advantages and disadvantages:
1. It is simple and it makes programming easy as it provides convenient input and output operations.
2. Data and instructions are handled using different forms which may seem a little counter intuitive.
3. The performance is another weakness of the system.
Functional programming features:
1. First-class function values and higher-order functions
2. Extensive polymorphism
3. List types and operators
4. Structured function returns
5. Constructors for structured objects
6. Garbage collection.
The notion for context-free grammars is sometimes called Backus-Naur Form. The Kleene star and meta-level parentheses of regular expressions are not allowed in BNF.
LL stands for “Left to right, left most derivation”. LR stands for “Left to right, right most derivation”.
LL parsers are also called “top-down” or “predictive” parsers. They construct a parse tree from the root down, predicting at each step which production will be used to expand the current node.
LR parsers are also called “bottom-up” parsers. They construct a parse tree from the leaves up, recognizing when a collection of leaves or other nodes can be joined together as the children of a single parent. (shift reduce parsers)
The number inside the parentheses after LL or LALR indicates how many tokens of look-ahead are required in order to parse.
Functional Tips:

```
type btree =
| Node of btree * string * btree
| Leaf of string
let rec height tree = match tree with
| Leaf _ -> 1
| Node(x,_,y) -> 1 + max (height x) (height y)
let rec mem tree elt = match tree with
| Leaf str -> str = elt
| Node(x,str,y) -> str = elt || mem x elet || mem y elt
length lst = fold (fun acc elt -> acc + 1) 0 lst
sum lst = fold (fun acc elt -> acc + elt) 0 lst
product lst = fold (fun acc elt -> acc * elt) 1 lst
reverse lst = fold (fun acc e -> acc @ [e]) [] lst
filter keep_it lst = fold (fun acc elt -> if keep_it elt then elt :: acc else acc) [] lst
let map myfun lst = fold (fun acc elt -> (myfun elt) :: acc) [] lst
```

Insertion_sort in OCaml:

```
let rec insert_sort cmp lst = match lst with
| [] -> []
| hd :: tl -> insert cmp hd (insert_sort cmp tl)
let insert cmp elt lst = match lst with
| [] -> [elt]
| hd :: tl when cmp hd elt -> hd :: (insert cmp elt tl)
| _ -> elt :: lst
```