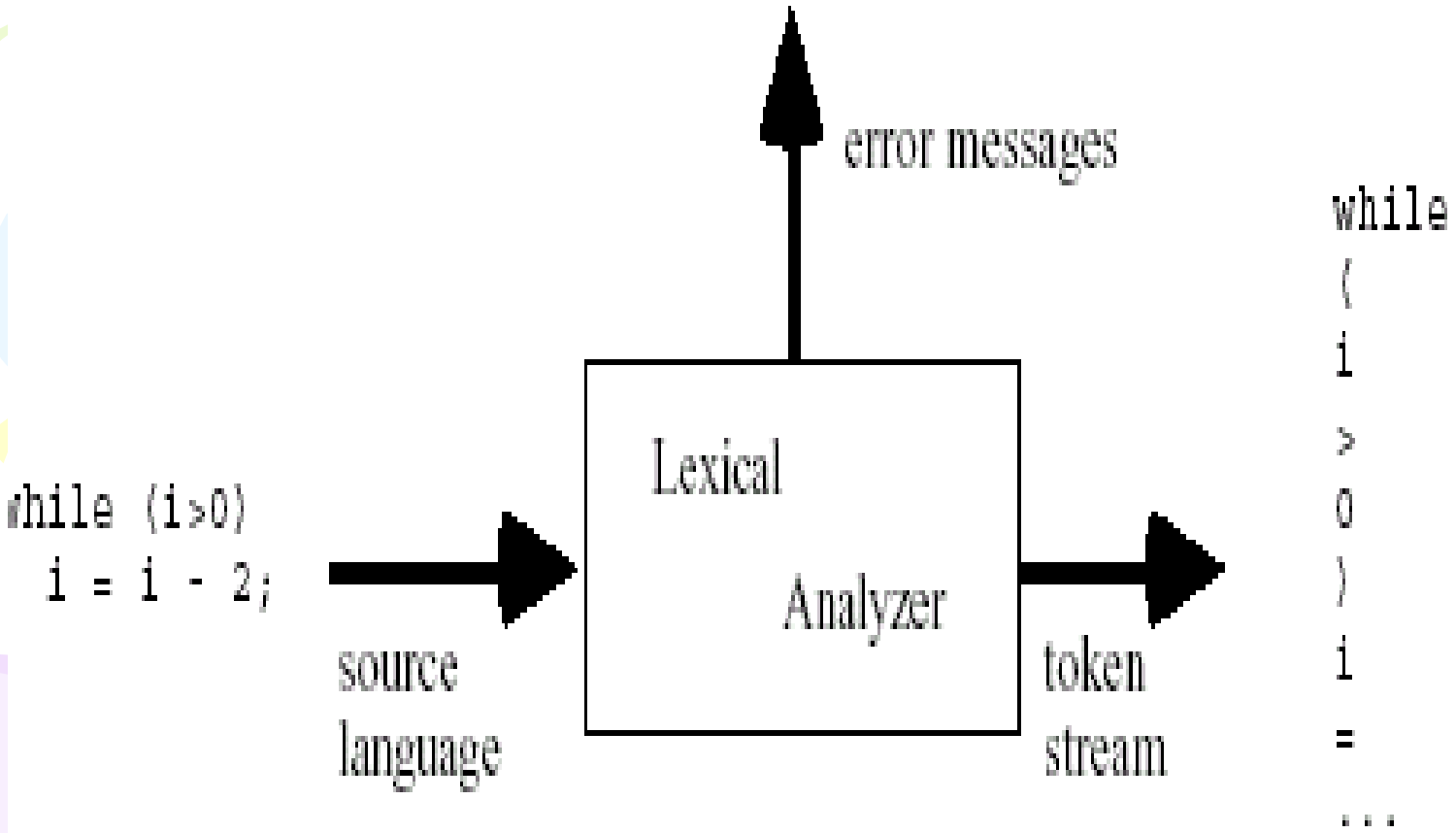
The Scanner (Lexical Analyzer)

The Scanner (Lexical Analyzer)

- **The basics**
- *Lexical analysis* or *scanning* is the process where the stream of characters making up the source program is read from left-to-right and grouped into tokens.
- *Tokens* are sequences of characters with a collective meaning.
- There are usually only a small number of tokens for a programming language: constants (integer, double, char, string, etc.), operators (arithmetic, relational, logical), punctuation, and reserved words.

lexical analysis



Notes on lexical analysis

- The lexical analyzer takes a source program as input, and produces a stream of tokens as output.
- lexical analyzer might recognize particular instances of tokens, such specific instances are called *lexemes*.
- A lexeme is the actual character sequence forming a token, the token is the general class that a lexeme belongs to.
- Some tokens have exactly one lexeme (e.g. the > character); for others, there are an infinite number of lexemes (e.g. integer constants).

Notes on lexical analysis

- There are two primary methods for implementing a scanner.
- The first is a program that is hard coded to perform the scanning tasks. It is .. **Scanner implementation #1: loop & switch**
- The second is the use of regular expressions and finite automata to model the scanning process. It is..**Scanner implementation #2: regular expressions & finite automata.**

lexical analysis

- ***التعرف علي ال recognition of ..tokens***

tokens خلال ذلك سيتم الاعتماد علي هذا النحو كمثال للشرح .

- مثال: افترض هذا الجزء من النحو :

- Stmt → **if** expr **then** stmt
- | **if** expr **then** stmt **else** stmt
- | ε
- expr →term **relop** term
- | term
- term → **id | num**

lexical analysis

- حيث النهايات (terminals) **if , then , else , relop , id , num . ...**
- متكونة من فئات من التسلسلات معطاه بواسطة التعريفات المنتظمة الآتية:

- **if → if**
- **then → then**
- **else → else**
- **relop→ < |<= | > | >= | <> | =**
- **id → letter (letter | letter |digit)***

num→digit+(.Digit+)?(E(+|-)?digit+)?

- **letter** → A|B….|Z|a|b|….|z |A – za – z|
- **digit** → 0|1|……|9 → |0 -9 |

lexical analysis

- وفي هذا الجزء من اللغة فإن المحلل الإملائي Lexical analyzer سيقوم بالتعرف علي الكلمات الحاكمة keywords بالإضافة لل lexeme -- ، **if , then , else relop , id , num**
- ولاختصار الحالة سنفترض:

* أن الكلمات الحاكمة keywords كلمات محجوزة أي لا يمكن استخدامها ك

- identifier
- ** num سيمثل عدد بدون اشارة أو عدد حقيقي real .
- هذا بالإضافة إلي إننا سنفترض lexemes سيتم فصلها بواسطة
- White Space
- nonnull sequence of newline , tabs , (ws) تحتوي علي blanks
- وسيقوم المحلل الإملائي بفصل white space وسيقوم بذلك بمقارنة التتابع بالتعريف المنتظم

Lexical analysis review
ملخص للمحلل اللغوي

- Lexical analysis: the process by which a sequence of characters (input stream) is broken up into its elementary words (tokens)
  - أنه العملية التي بها يتم تقطيع التسلسلات الداخلة إلي الكلمات الأولية
- Our goal: come up with an algorithm to scan an input stream and identify the tokens
  - إن الهدف هو عمل خوارزم يقوم بمسح التسلسلات الداخلة والتعرف للمعرفاتtokens
- What we know: a general description of the tokens
  - وما نعرفه هو توصيف عام للعلامات

Lexical analysis review
ملخص للمحلل اللغوي

- Step 1: Use regular expressions to formally describe the tokens
  - استخدم التعبير المنتظم في الوصف المحدد للعلامة token
- Step 2: Build a finite automaton out of the regular expressions, i.e. a machine (algorithm) that can recognize words described by those regular expressions.
  - قم ببناء الآلة ذاتية الحركة من التعبير المنتظم أي خوارزم الآلة التي تتعرف علي الكلمات الموصفة بذلك التعبير المنتظم
- Step 3: Add some additional functionality: error handling, symbol table access.
  - أضف بعض الدوال اللازمة لذلك الحلل مثل كيفية التعامل مع الأخطاء علي جدول الرموز

Lexical analysis review
ملخص للمحلل اللغوي

- Result: a program (machine, algorithm) that can read the input stream and return tokens.
- النتيجة: المحلل الإملائي هو عبارة عن برنامج <آلة- خوارزم> والذي يستطيع قراءة التتابع الداخل حيث يقوم بإعطاء أو أخراج علامات Tokens
- All this can be done automatically by a scanner generator (such as Lex) as long as we provide the regular definitions and specify the "additional" actions.
- كل هذا يمكن عمله آليا بإستخدام "المولد الماسح" مثل (Lex) طالما تم عمل التعريفات المنتظمة وأيضا إضافة بعض العمليات الإضافية

Syntax analysis
المحلل النحوي

- During this phase:
- Verify that the program is syntactically correct
- وخلال تلك المرحلة يتم الآتي :-
- *التحقق من أن البرنامج صحيح اعرابيا
- Build an internal representation of the source code
- *بناء تمثيل داخلي لبرنامج (كود) المنبع
- Handle errors
- *إعطاء وطباعة الأخطاء

Context-Free Grammars (CFGs)

- A scanner recognizes words of the language
- إن الماسح (المحلل الإملائي) يتعرف علي كلمات اللغة
- A parser recognizes sequences (strings) of tokens.
- بينما المعرب يتعرف علي تتابعات العلامات Tokens مثل :-
- e.g. (**id**+**num**)***id**
- is a sequence of tokens that "builds" an expression.
- -هذا التعبير عبارة عن تسلسلات من العلامات والتي من خلالها يتم بناء تعبير
- We need a way to formally describe such patterns of tokens.
- لذا يلزمنا لعمل المحلل الإعرابي طريقة معروفة لتوصيف تلك النمازج لل Tokens

Context-Free Grammars (CFGs)

- Regular expressions are not powerful enough (for example, they cannot be used to describe balanced parentheses or block structure)
- إن التعبيرات المنتظمة ليست بالقوة الكافية لعمل المعرب –فمثلا لا تستطيع عمل الأقواس وإتزانها- أي الفتح والقفل للاقواس
- Context-free grammars, however, are!
- بينما تستطيع تقنية CFG عمل ذلك
- كذلك معظم لغات البرمجة بناؤها يحتوي علي تكرير للبناء مما يمكن تعريفها بواسطة CFG

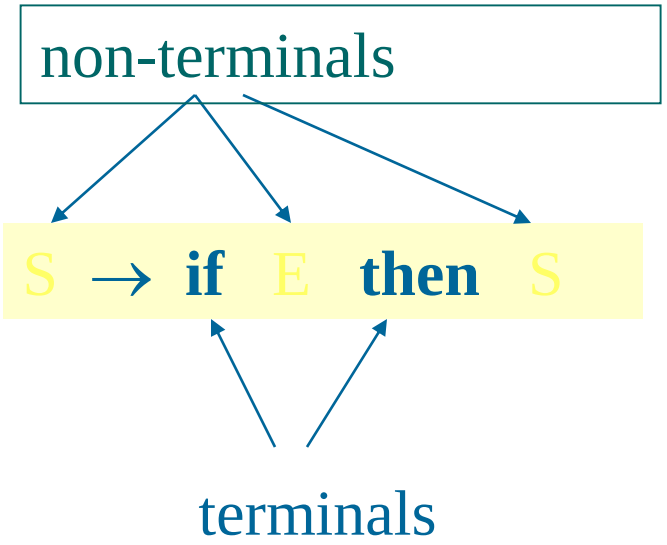
- **مثال** إذا كان عندنا تعبير مشروط conditional statement معرفة بالقاعدة الآتية :
- If S1 and S2 are statements and E is an expression , then :
- **" If E then S1 else S2" is a statement. (1)**
- وهذا التعبير لا يمكن وضعه في صورة تعبير منتظم (التعبير المنتظم يتم استخدامه في البناء الإملائي كما درسنا)
- لذا فإنه يمكن استخدام نحو المنتجات
- grammar production في (١) ليكون:-
- *Stmt* → **if** expr **then** *stmt* **else** *stmt*

Context-Free Grammars (CFGs)

- A CFG is a language generator that uses "rules" to generate valid strings of a language.
- إن تقنية CFG عبارة عن مولد للغة والذي يستخدم قواعد لتوليد تتابعات محققة من اللغة
- Intermediate symbols are called **non-terminals** الرموز المتوسطة تسمي نهايات غير طرفية
- Atomic symbols are called **terminals** الرموز الأساسية تسمي نهايات طرفية

Context-Free Grammars (CFGs)

- Example:



CFGs : definition

- A CFG is a quadruple (V, Σ, R, S) where
  - V is an alphabet consisting of terminals and nonterminals
  - $\Sigma \subseteq V$ is the set of terminals
  - $S \in V - \Sigma$ is the starting symbol
  - $R \subseteq (V - \Sigma) \times V^*$ is a set of rules

CFGs : definition

- Example: Consider (V, Σ, R, S) where

$V = \{S, (,)\}$–

$\Sigma = \{ (,) \}$–

$R = \{ S \rightarrow (S) , S \rightarrow SS, S \rightarrow \varepsilon \}$–

CONTEXT –FREE GRAMMERS (CFG)

- CFG consists of terminals , non-terminals , a start symbol , and productions.
- CFG تتكون من نهايات طرفية-نهايات غير طرفية ورمز البداية ومنتجات
- Terminals are the basic symbols from which strings are formed , ex. Keywords are terminals.
- النهايات الطرفية هي الرموز الأساسية والتي منها يتم تكوين التتابعات مثل الكلمات الحاكمة if – then –else...
- Non-terminals are syntactic variables like *expr , stmt.*
- *النهايات الغير طرفية هي المتغيرات النحوية مثل expr , stmt*

CONTEXT –FREE GRAMMERS (CFG)

- In a grammar , one non- terminal is distinguished as start symbol and the set of strings it denotes is the language defined by the grammar.
- وفي النحو فإن أحد هذه النهايات الغير طرفية يتم أختياره كرمز البداية ثم التتابعات التي يشير اليها هي اللغة التي تخضع لتلك القواعد
- The production of a grammar specify the manner in which the terminals and non-terminals can be combined

CONTEXT –FREE GRAMMERS (CFG)

- ومنتجات النحو عبارة عن الطريقة التي بها يتم ربط النهايات الطرفية وغير الطرفية لتكوين تتابعات النحو
- Each production consists of a non-terminal, followed by an arrow (sometimes the symbol ::= is used in place of the arrow) , followed by a string of non-terminals and terminals.
- حيث أن كل منتج عبارة عن نهاية غير طرفية متبوعة بسهم أو = أو ::= متبوعا بتسلسلات من النهايات الطرفية وغير الطرفية

CONTEXT –FREE GRAMMERS (CFG)

- Ex.
- *expr → expr op expr*
- *Expr → (expr)*
- *Expr →* *- expr*
- *Expr →* **id**
- *Op* *→ +*
- *Op → -*
- *Op →* $*$
- *Op →* $/$
- *Op →* $\uparrow$
- In this grammar , the terminal symbols are :-
- id ,+, - ,* ,$\uparrow$, (,)
- The non-terminals are *expr* and *op* and

Notational Conventions
مصطلحات رمزية

- ولتفادي ذكر دائما أن هذا نهاية طرفية terminal وهذا نهاية غير طرفية non-terminal.فأننا سنقوم بعمل اصطلاحات رمزية متعارف عليها وذلك خلال ما تبقي من دراسة هذه المادة
- **1-These symbols are terminals:**
- **هذه الرموز نهاية طرفية**
- i)Lower –case letters early in the alphabet such as a, b, c… الحروف الأنجليزية الصغيرة والتي في البداية مثل
- ii)Operator symbols such as +, - ,*..... علامات العمليات الرياضية
- iii)Punctuation symbols such as parentheses (,),comma, ..etc الرموز التي تستخدم بين الكلمات والجمل مثل
- iv)The digits 0 ,1 ,2 ,3 ,4 ,…..,9 الأعداد
- Boldface strings such as **id** or **if** (For us may be id or if)

Notational Conventions
مصطلحات رمزية

- **2-These symbols are non- terminals:**
- i)Upper – case letters early in the alphabet such as A , B, C, …
- ii)The letter S , which, when it appears , is usually the start symbol.
- iii)Lower –case italic names such as *expr* or *stmt*
- 3-Upper – case letters late in the alphabet , such as X ,Y,Z represent grammar symbols , that is , either non-terminals or terminals

Notational Conventions
مصطلحات رمزية

- 4-Lower – case letters late in the alphabet , chiefly u ,v , ….z , represent strings of terminals
- 5- Lower – case Greek letters , α , β , γ, for example , represents strings of grammar symbols , i.e:
- A → α , • معناها
- If A→ α1 , A → α2 , A→ α3 are all production with A on the left (we call them A-productions) we may write
- A→ α1| α2 | α3…..| αk. We call α1, α2,α3…., αk the alternatives for A
- Unless otherwise stated , the left side of the first production is the start symbol

Notational Conventions
مصطلحات رمزية

- Example :-
- وبإستخدام هذه المصطلحات سيتم التعبير عن النحو الآتي وكما هو موضح
- *Expr → expr op expr*
  Expr → (expr)
  Expr → - expr
  Expr → id
  Op → +
- *Op → -*
- *Op → /*
  *Op → *
  Op → ↑
- **E→ E A E | (E) | -E | id**
- **A→ + | - | * | ↑**
- وهكذا فإن المصلحات الرمزية تخبرنا أن **non-terminals are E , A**
- **And E start symbol . The remaining**

Derivations
إثباتات

- ولإثبات أن النحو يتم بواسطته تعريف اللغة
- وكمثال علي ذلك فإن هذا النحو والذي يتم إستخدامه للتعبير عن التعبيرات الرياضية حيث E تمثل تعبير Expression
- **E→E+E | E*E | (E) | - E | id**
- **E→ - E**
- **معناها أن تعبير مسبوقا بعلامة – أيضا تعبير ويمكن إختصاره كالآتي:-**
- **E ⇒ -E Which means "E derives –E"**
- **أي ينتج عنها**
- **وهكذا فإن E ⇒ (E)**
- **Which means we could also replace one instance of an E in any string of grammar symbols by (E) as:-**

Derivations
إثباتات

- النحو E→E+E | E*E | (E) | - E | id المستخدم
- E ⇒ E *E ⇒ E * (E) أو
- E ⇒ E *E ⇒ (E) * E
- وهكذا فيمكن أخذ E واحدة ثم تكرارها ومعوضين منتجات بأي ترتيب وذلك لعمل تتابع ما
- فمثلا للوصول الي -(id)
- E ⇒ -E ⇒ -(E) ⇒ -(id)
- وتسمي هذه إثباتات لإنتاج -(id) من E

Derivations
إثباتات

- النحو E→E+E | E*E | (E) | - E | id المستخدم
- مثال التسلسل (id+id)-عبارة عن جملة في النحو السابق
- هذا لانه يمكن إيجاد أثبات كالأتي
- E ⇒ -E ⇒ -(E) ⇒ -(E+E) ⇒ ⇒ -(id +E) ⇒ -(id+id)

CFGs : derivations

- Example: Consider (V, Σ, R, S) where
  Σ ={ (,) }- V = {S, (,)} –
  R ={ S→ (S) , S→SS, S→ε } –

$S \Rightarrow (S) \Rightarrow (SS) \Rightarrow ((S)S) \Rightarrow (( )S) \Rightarrow (( )(S)) \Rightarrow (( )((S))) \Rightarrow (( )(( )))$

sentential form

derivation

This example demonstrates a **leftmost derivation** : one where we always expand the leftmost non-terminal in the sentential form.