12 ## Pascal- : A subset of Pascal

Pascal- has only
two simple types `integer` and `Boolean`
two structured types `array` and `record`

Type definition: A type definition always creates a new type; it can never rename an existing type

```
type
 table = array [1..100] of integer;
 stack = record
           contents: table;
           size: integer end;
```

13 ## Pascal- : A subset of Pascal

Variable definition: A type name must be used in a variable definition

```
var
A: table;
x, y: integer;
```

All constants have simple types:

Predefined constants: `true, false`

Constant definition:

```
const max = 100; on = true;
```

14 ## Pascal- : A subset of Pascal

Statements:

| | |
|---|---|
| -assignment | `x := y;` |
| -if-statement | `if x = y then x := 1;` |
| -while-statement | `while I <10 do I := I+1;` |
| -compound statement | `begin x := y; y := z end` |
| -procedure call | |
| -recursion | |

15 ## A Complete Pascal- Program

```
Program ProgramExample;
const n=100;
type table=array[1..n] of integer;
var A: table; i,x: integer; yes: Boolean;

procedure search(value: integer; var found: Boolean; var index: integer);
  var limit: integer;
```

```
begin
  index:=1; limit:=n;
  while index<limit do
    if A[index]=value then
        limit := index
    else
        index := index+1;
  found := A[index] = value
end;
```

## 16 A Complete Pascal- Program

```
begin {input table}
   i:=1;
   while i<=n do
      begin
        read(A[i]);
        i:=i+1
      end;
{test search}
   read(x);
   while x<>0 do
   begin
    search(x,yes,i);
    write(x);
    if yes then
      write(i);
    read(x);
   end
end. {program}
```

## 17 Pascal- Vocabulary

The vocabulary of a programming language is made up of *basic symbols* and *comments*.

Basic Symbols:

a) Identifiers: In Pascal-, an identifier is called a `Name,` and consists of a letter that may be followed by any sequence of letters and digit (Identifiers are case insensitive)

b) Denotations: Denotations represent specific values, according to conventions laid down by the language designer. In Pascal- a `Numeral` is the only denotation in the vocabulary.

## 18 Pascal- Vocabulary

c) There are two kinds of delimiters in Pascal-, *word symbols* and *special symbols*:

Word symbols: `and array begin const div do else end if mod not of or procedure program record then type var while`

Special symbols: `+ - * < = > <= <> >= := ( ) [ ]`
`, . : ; ..`

Comments: A comment in Pascal- is an arbitrary sequence of characters enclosed in braces `{ }.` Comments may extend over several lines and may be nested to arbitrary depth.

## 19 Pascal- Vocabulary

White space (spaces, tabs and new lines) and comments are called *separators*. Any basic symbol may be preceded by one or more separators, and the program may be followed by zero or more separators

Example:

```
{Incorrect}
  ifx>0thenx:=10divx-1;
{Correct}
  if x>0
```

```
then{Can divide}x:=10   div x-1;
```

## 20 Pascal- Grammar

```
Program --> 'program' ProgramName ';' BlockBody '.'
BlockBody --> [ConstantDefinitionPart] [TypeDefinitionPart] [VariableDefinitionPart]
  {ProcedureDefinition} CompoundStatement .
```

Constant, Type, and Variable definition grammar:

## 21 Pascal- Grammar

Constant, Type, and Variable definition grammar

```
ConstantDefinitionPart --> 'const' ConstantDefinition {ConstantDefinition}
ConstantDefinition --> ConstantNameDef '=' Constant ';'
Constant -> Numeral | ConstantNameUse

TypeDefinitionPart --> 'type' TypeDefinition {TypeDefinition}
TypeDefinition -->  TypeNameDef '=' NewType ';'
NewType --> NewArrayType | NewRecordType

NewArrayType --> 'array' '[' IndexRange ']' 'of' TypeNameUse .
IndexRange --> Constant '..' Constant
```

## 22 Pascal- Grammar

Constant, Type, and Variable definition grammar

```
NewRecordType --> 'record' FieldList 'end'
FieldList --> RecordSection {';' RecordSection}
RecordSection -->  FieldNameDefList ':' TypeNameUse
FieldNameDefList --> FieldNameDef {',' FieldNameDef}

VariableDefinitionPart --> 'var' VariableDefinition {VariableDefinition}
VariableDefinition --> VariableNameDefList ':' TypeNameUse ';'
VariableNameDefList --> VariableNameDef {',' VariableNameDef}
```

## 23 Pascal- Grammar

Expression grammar

```
Expression --> SimpleExpression [RelationalOperator SimpleExpression]

RelationalOperator --> '<' | '=' | '>' | '<=' | '<>' | '>='

SimpleExpression --> [SignOperator] Term | SimpleExpression AddingOperator Term

SignOperator --> '+' | '-'
AddingOperator --> '+' | '-' | 'or'

Term --> Factor | Term MultiplyingOperator Factor
MultiplyingOperator: '*' | 'div' | 'mod' | 'and'
```

## 24 Pascal- Grammar

Expression grammar

```
Factor -->
  Numeral |
  VariableAccess |
  '(' Expression ')' |
  NotOperator Factor

NotOperator --> 'not' .

VariableAccess -->
  VariableNameUse |
  VariableAccess '[' Expression ']' |
  VariableAccess '.' FieldNameUse
```

25

## Pascal- Grammar

### Statement grammar

```
Statement -->
  AssignmentStatement |
  ProcedureStatement |
  IfStatement |
  WhileStatement |
  CompoundStatement |
  Empty

AssignmentStatement --> VariableAccess ':=' Expression

ProcedureStatement --> ProcedureNameUse ActualParameterList
```

26

## Pascal- Grammar

### Statement grammar

```
ActualParameterList: --> '(' ActualParameters ')'
ActualParameters --> ActualParameter {',' ActualParameter}
ActualParameter --> Expression

IfStatement -->
  'if' Expression 'then' Statement |
  'if' Expression 'then' Statement 'else' Statement

WhileStatement --> 'while' Expression 'do' Statement

CompoundStatement: 'begin' Statement {';' Statement} 'end' .
```

27

## Pascal- Grammar

### Procedure grammar

```
ProcedureDefinition --> 'procedure' ProcedureNameDef ProcedureBlock ';'

ProcedureBlock --> FormalParameterList ';' BlockBody
FormalParameterList --> |'(' ParameterDefinitions ')'
ParameterDefinitions --> ParameterDefinition {';' ParameterDefinition}
ParameterDefinition -->
  'var' ParameterNameDefList ':' TypeNameUse |
  ParameterNameDefList ':' TypeNameUse

ParameterNameDefList -->
  ParameterNameDef | ParameterNameDefList ',' ParameterNameDef
```