

Chapter 2

□ AN INTRODUCTION TO PASCAL

2.1 INTRODUCTION

A Pascal program is usually represented as a plain *text* (ASCII) file, i.e., a sequence of (ASCII or keyboard) characters. Such a file is called a Pascal **source** file. This file is then transformed by a Pascal *compiler* into an executable file which is a set machine instructions that are executed or run by the operating system of the computer. The general process of creating, compiling, and running a program is as follows :

1. Create a source program file using a plain text *editor*
2. Transform or *compile* the source program into an executable program file using a *compiler*.
3. *Run* the executable program using the operating system.

Let us look at a simple Pascal program to find the area of a rectangle whose width is w and whose height is h .

```
program AreaRect;
  var
    h, w, area : REAL;
begin
  h := 12.5;
  w := 5.0;
  area : h * w;
  Writeln('The height of the rectangle is : ', h);
  Writeln('The width of the rectangle is : ', w);
  Writeln('The area of the rectangle is : ', area);
end.
```

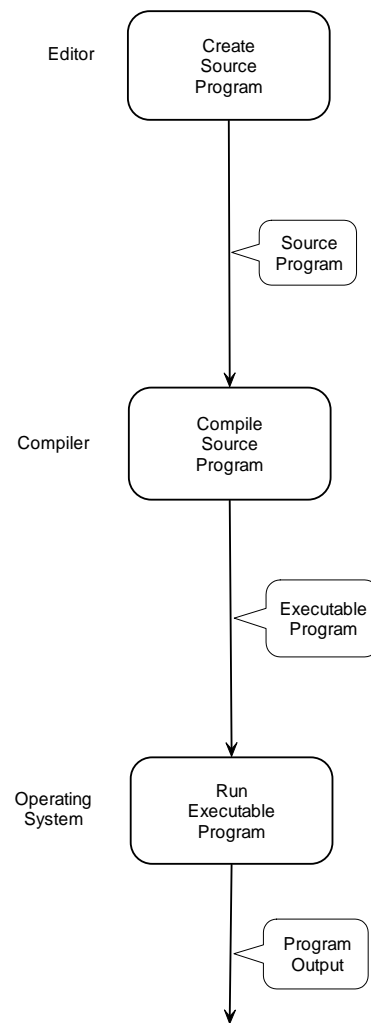


Figure 2.1. Creating a Program.

Although it is obvious what the program does, it is not obvious how the program is created. The program must be created according to strict rules of *syntax* (grammar) that specify how various elements of the language are put together to form a valid program. All languages have such rules, to a greater or lesser extent, but programming languages have strict, unambiguous rules that the compiler uses when translating a source program into an executable program. Hence, if a compiler detects something in a source program that breaks a rule it will stop translating and issue an error message — programmers must stick to the rules whether they like it or not.

2.2 Lexical Elements

The ASCII characters of a Pascal source file are grouped into 4 categories :

1. Letters : a,b,...,z,A,B,...,Z.
2. Digits : 0,1,2,3,4,5,6,7,8,9
3. Special : _ - * () % \$ + = } etc.
4. Control : non-printable

2.2.1 Tokens

The characters of a source file are grouped into **tokens** which are separated by at least one space (blank). No spaces are allowed within tokens except for string constants (see below). Thus tokens are strings of characters with no spaces.

The tokens of a Pascal program are classified into 6 categories :

1. one and two character symbols.
2. reserved words
3. identifiers
4. numbers
5. character and string literals

Symbols

One-Character +, - , *, /, =, <, >, { } () [] , ; : ^

and

Two-Character :=, <=, >=, <>, ..

Reserved Words Strings of alphabetic characters that have special meaning within Pascal. The following are some of the reserved words

**array, begin, const, do, end, for ,if, of,
procedure, then, to, type, var, while**

Identifiers These are tokens that 'name' various elements in Pascal. They are strings of letters and digits that must begin with a letter. Examples are : **Alpha, x, i, X2x, Three3Four4**. We note that Pascal does not distinguish between upper-case and lower-case letters,i.e., the identifier **alpha** is the same as **ALPHA**.

Numbers

123, -45877, 23.45, -0.001

Character and String Literals

Anything enclosed with quotes.

'This is a literal', '2334', 'A'. Note that a blank is significant in a sting literal

Comments Anything enclosed by { and }.

2.3 Syntax

These are the rules by which tokens are combined to form larger elements and ultimately complete programs. We will examine the syntax of the various parts of a Pascal program in detail later. For the moment we consider the general layout of a program only.

Every Pascal program has the following syntactical structure.

```
program <name> ;
  <CONSTant definition part>; optional
  <TYPE definition part>; optional
  <VARiable declaration part>; optional
  <PROCEDURE definition part>; optional
begin
  <statement part> optional
end.
```

2.4 Standard Types

The *type* of a variable describes the values that a variable may have and the operations that can be performed on the variable.

Pascal has types that are built into the language. Other types can be defined by the programmer using type definitions and previously-defined types. The standard types are as follows :

1. **Scalar** : INTEGER, CHAR, BOOLEAN, REAL, along with subranges of the first three and enumerated types.
2. **Structured** : ARRAY, FILE, RECORD, SET, STRING.

2.5 Statements

1. Assignment
2. **if**
3. **while**
4. **for**
5. Procedure and Function Call
6. Compound

We will go into the details of all these elements in later chapters. For the moment we will just look at examples to see how these elements are used in programs.

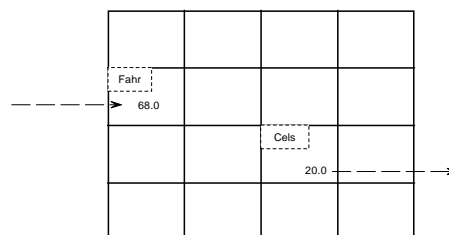
2.6 Examples

The following are complete Pascal programs

```
program Null ;  
  {This program does absolutely nothing}  
begin  
end.
```

```
program Message ;  
begin  
  Writeln('This is a program with no input');  
end.
```

```
program FarhToCels;  
  {This program converts Fahrenheit to Celsius}  
var Fahr, Cels : real;  
begin  
  Write('Type in degrees Fahrenheit : ')  
  Readln(Fahr);  
  Cels := (Fahr-32)*(5.0/9.0);  
  Writeln(Fahr,'degrees F = ', Cels, 'degrees C.');
```



Memory

Figure 2.2. Memory when running a Program.

```

program PayCalc;
{This program calculates the necessary information to print a payslip}

const
    PayRate = 11.5;
    TaxRate = 0.4;

var
    Name : STRING;
    Hours : REAL;
    Gross : REAL;
    Net : REAL;
    Tax : REAL;

begin
    Write('Type in name : ');
    Readln(Name);
    Write('Type in hours worked : ');
    Readln(Hours);

    Gross := PayRate * Hours;
    Tax := TaxRate * Gross;
    Net := Gross - Tax;

    Writeln('Payslip for :', Name);
    Writeln('Gross Pay = ', Gross);
    Writeln('Tax      = ', Tax);
    Writeln('Net Pay  = ', Net);
end.

```

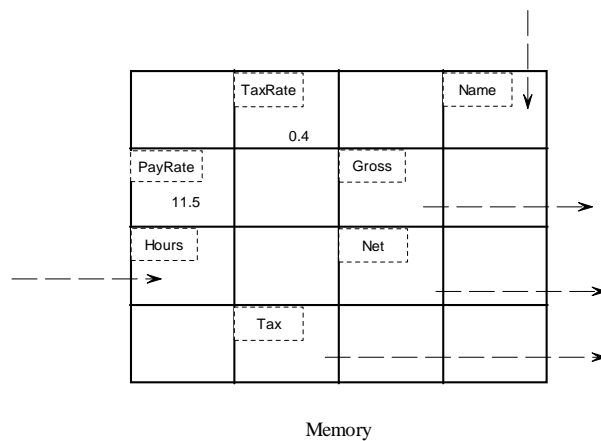


Figure 2.3. Memory when running a Program.

```
PROGRAM PayCalcRepeat ;
  {This program repeatedly calculates the necessary information to print payslips}

CONST
  PayRate = 11.5;
  TaxRate = 0.4;

VAR
  Name   : STRING;
  Hours  : REAL;
  Gross  : REAL;
  Net    : REAL;
  Tax    : REAL;
  Answer : CHAR;

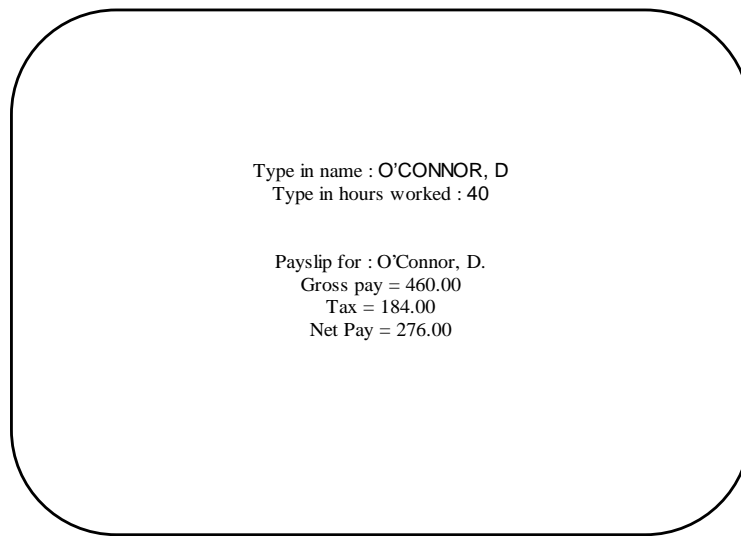
BEGIN
  Answer := 'Y';
  WHILE (Answer = 'Y') DO BEGIN
    Write('Type in name : ');
    Readln(Name);
    Write('Type in hours worked : ');
    Readln(Hours);

    Gross := PayRate * Hours;
    Tax   := TaxRate * Gross;
    Net   := Gross - Tax;

    Writeln('Payslip for :', Name);
    Writeln('Gross Pay = ', Gross);
    Writeln('Tax      = ', Tax);
    Writeln('Net Pay  = ', Net);

    Write('Do you want to continue (Y/N)?');
    Readln(Answer);
  END; { while }
END. { } PROG PayCalcRepeat
```

The output of this program will look as follows :



```
Type in name : O'CONNOR, D
Type in hours worked : 40

Payslip for : O'Connor, D.
Gross pay = 460.00
Tax = 184.00
Net Pay = 276.00
```

Figure 2.4. Output of Program.

Laboratory Exercise No. 2: *Payslip Calculation..*

Type in the program `PayCalcRepeat` above and make sure it runs correctly. Then perform the following alterations :

1. Get rid of the `PayRate` constant and read it in as a variable so that any pay rate can be handled by the program.
2. Do the same for the tax rate.

Make sure that you get the program running properly after the first modification before you attempt the second modification.
