CHAPTER III

DATA PRESENTATION AND ANALYSIS

PROGRAM LISTING

```
Program Example (Input, Output);
(* This program computes the midpoint between the
   freezing and boiling points of water                    *)

CONST
   Freeze = 32; (* Freezing point of water *)
   Boil   = 212;        (* Boiling point of water   *)

VAR
   AvgTemp:             (* Variable to hold the result    *)
      Real;             (* of averaging Freeze and Boil *)

(*  The executable statements begin here  *)

BEGIN  (* Example *)
   Writeln('Water freezes at ', Freeze);
   Writeln(' and boils at ', Boil, ' degrees. ');
   AvgTemp := Freeze + Boil;
   AvgTemp := AvgTemp / 2;
   Writeln('Halfway between is ', AvgTemp, ' degrees. ');
END.  (* Example *)

PROGRAM Payroll (Input, Output, PayFile);
(* This program computes each employee's wages and the total company payroll *)

CONST
   MaxHours = 40.0;          (* Maximum normal work hours *)
   OverTime = 1.5;           (* Overtime pay rate factor      *)

VAR
   PayRate,          (* Employee's pay rate          *)
   Hours,            (* Hours worked                      *)
   Wages,            (* Wages earned                      *)
   Total:            (* Total company payroll             *)
      Real;
   EmpNum:           (* Employee ID number           *)
```

```
      Integer;
   PayFile:              (* Company payroll file          *)
      Text;
(*************************************************************)

PROCEDURE CalcPay(        PayRate,       (* Employee's pay  rate *)
                          Hours:         (* Hours worked         *)
                             Real;
                    VAR   Wages:         (* Wages earned         *)
                             Real;
(* Calc Pay  computes wages wages from the employee's  pay rate
   and the hours worked, taking overtime into account          *)
BEGIN  (* CalcPay *)
   IF Hours > MaxHours          (* Check for overtime *)
      THEN                          (* Computes wages if overtime *)
         Wages := (MaxHours * PayRate)  +
                  (Hours - MaxHours) * PayRate * Overtime
      ELSE
         Wages := Hours * PayRate   (* Computes wages if no overtime *)
END;    (* CalcPay *)
(*************************************************************)
BEGIN (* Payroll *)
   Rewrite(PayFile);                                  (* Open File PayFile *)
   Total := 0.0;                                   (* Initialize Total to zero *)
   Write('Enter employee number:    ');                          (* Prompt *)
   Readln(EmpNum);                              (* Read employee ID number *)
   WHILE EmpNum <> 0 DO                       (* While employee number <> 0  *)
      BEGIN
         Write('Enter pay rate:   ');                              (* Prompt *)
         Readln(PayRate);                                (* Read hourly pay rate *)
         Write('Enter hours worked:  ');                           (* Prompt *)
         Readln(Hours);                                 (* Read Hours worked *)
         CalcPay(Payrate, Hours, Wages);                  (* Computes wages *)
         Total := Total + Wages;                        (* Add wages to total *)
         Writeln(PayFile, EmpNum, PayRate, Hours, Wages); (*Put result in PayFile*)
         Write('Enter employee number:   ');                       (* Prompt *)
         Readln(EmpNum);                                 (* Read ID number *)
      END;
   Writeln(' Total payroll is ', Total: 10:2);          (* Print total payroll on screen *)
END.    (* Payroll *)

PROGRAM HouseCost(Output);
(*  This program computes the cost per square  foot of     *)
(*  living space for  a house, given the dimensions  of    *)
(*  the  house, the number of stories, the size of the     *)
```

```
(* nonliving space, and the total cost less land                *)

CONST
   Width = 30.0;
   Length = 40.0;
   Stories = 2.5;
   NonLivingSpace = 825.0;
   Price = 150000.0;
VAR
   GrossFootage,
   LivingFootage,
   CostPerFoot:
      Real
BEGIN (* HouseCost *)
   GrossFootage := Length * Width * Stories;
   LivingFootage := GrossFootage - NonLivingSpace;
   CostPerFoot := Price / LivingFootage;
   Writeln(' Cost per square foot is ', CostPerFoot:6:2)
END.

PROGRAM Game (FileA, FileB, Output);
(* This program simulates the children's game ' rock, paper, and *)
(* scissors.' Each game consist of inputs from two players, coming *)
(* from FileA and FileB. A winner is determined for each          *)
(* individual game, and for the games overall.                    *)

TYPE
   PlayType = (Rock, Paper, Scissors);
VAR
   PlayerA,     (* Player A's play *)
   PlayerB:     (* Player B's play *)
      PlayType;
   WinsForA,    (* Number of games A wins *)
   WinsForB,    (* Number of games B wins *)
   GameNumber:  (* Number of games played *)
      Integer;
   FileA,               (* Player A's play *)
   FileB:               (* Player B's play *)
      Text;
   Legal:               (* True if play is legal *)
      Boolean;
(*************************************************************************)
PROCEDURE PlayerAWins (          GameNumber: (* Receives game number *)
                                 Integer;
                       VAR  WinsForA:     (* Rec's/returns win count *)
```

```
                                    Integer);
(* Message  that Player A has won the current game is written,   *)
(* and Player A's total is  updated   *)
BEGIN   (* PlayerAWins  *)
   Writeln('Player A has won game number ',  GameNumber:1, '.');
   WinsForA  := WinsForA + 1
END;    (* PlayerAWins *)
(*****************************************************************************)
PROCEDURE PlayerBWins  (        GameNumber: (* Receives game number *)
                                    Integer;
                          VAR  WinsForB:    (* Rec's/returns win count *)
                                    Integer);
(* Message  that Player B has won the current game is written,   *)
(* and Player B's total is  updated   *)
BEGIN   (* PlayerBWins  *)
   Writeln('Player B has won game number ',  GameNumber:1, '.');
   WinsForB  := WinsForB + 1
END;    (* PlayerBWins *)
(*****************************************************************************)
PROCEDURE ProcessPlays (        GameNumber:  (* Receives game number *)
                                    Integer;
                                PlayerA,    (* Receives A's play *)
                                PlayerB:    (* Receives B's play *)
                                    PlayType;
                          VAR  WinsForA, (* Rec's'returns A's wins *)
                                WinsForB: (* Rec's'returns B's wins *)
                                    Integer;
(* ProcessPlays determines  the winning play. If there is a tie, a   *)
(* message is written. Otherwise the number of wins of the winning  *)
(* player is incremented   *)
BEGIN   (* ProcessPlay *)
   IF PlayerA = PlayerB
      THEN
         Writeln('Game number   ', GameNumber:1, ' is a tie.');
      ELSE IF (PlayerA = Paper) AND (PlayerB = Rock)
         THEN
            PlayerAWins(GameNumber, WinsForA)
      ELSE IF (PlayerA = Scissors) AND (PlayerB = Paper)
         THEN
            PlayerAWins(GameNumber, WinsForA)
      ELSE IF (PlayerA = Rock) AND (PlayerB = Scissors)
         THEN
            PlayerAWins(GameNumber, WinsForA)
      ELSE
         PlayerBWins(GameNumber, WinsForB)
```

```
END; (* ProcessPlays *)
(*****************************************************************************)
PROCEDURE GetPlays (VAR  PlayerA,  (* Returns A's plays *)
                         PlayerB:  (* Returns B's plays *)
                           PlayType;
                    VAR Legal:     (* Returns True if legal plays *)
                           Boolean);
(* PlayerA's play is read from FileA, PlayerB's play is read from  *)
(* FileB. If both plays are legal, Legal is set to True and both plays  *)
(* are converted to corresponding values of  PlayType. Else Legal   *)
(* is False and PlayerA and PlayerB are undefined. FileA and FileB*)
(* are accessed globally   *)
VAR
   CharForA,   (* PlayerA's input *)
   CharForB:   (* PlayerB's input *)
      Char;
(*****************************************************************************)
FUNCTION Convert (Character:              (* Receives play character    *)
                  Char): PlayType;       (* Returns play literal       *)
(* Converts character into associated value in PlayType  *)
BEGIN (* Convert *)
   CASE Character OF
      'R' : Convert := Rock;
      'P' : Convert := Paper;
      'S' : Convert := Scissors
   END  (* Case *)
END;  (* Convert *)
(*****************************************************************************)
BEGIN   (* GetPlays  *)
   Readln(FileA, CharForA);
   Readln(FileB, CharForB);
   Legal  :=  (CharForA  IN  ['R',  'P',  'S']) AND
              (CharForB  IN  ['R',  'P',  'S']);
   IF Legal
      THEN
         BEGIN
            PlayerA := Convert(CharForA);
            PlayerB := Convert(CharForB)
         END
END;     (*  GetPlays *)
(*****************************************************************************)
PROCEDURE PrintBigWinner (WinsForA,   (* Receives A's win count     *)
                          WinsForB:   (* Receives B's win count     *)
                            Integer);
(* Prints number of wins for each player and the overall winner  *)
```

```
BEGIN  (* PrintBigWinner *)
   Writeln('Player A has won ', WinsForA:1, ' games,');
   Writeln('Player B has won ', WinsForB:1, ' games,');
   (* Determine and print winner  *)
   IF WinsForA > WinsForB
      THEN
         Writeln('Player  A has won the most games.')
      ELSE IF WinsForB > WinsForA
         THEN
            Writeln('Player B has won the most games.')
      ELSE
         Writeln('Player A and B have tied.')
END;   (* PrintBigWinner *)
(*************************************************************************)
BEGIN  (* Game *)
   Reset(FileA);
   Reset(FileB);
   WinsForA := 0;
   WinsForB := 0;
   GameNumber := 0;
   (* Play a series of games and keep track of who wins  *)
   WHILE NOT EOF(FileA) AND NOT EOF(FileB) DO
      BEGIN
         GameNumber := GameNumber + 1;
         GetPlays(PlayerA, PlayerB, Legal);
         IF Legal
            THEN
               ProcessPlay(GameNumber, PlayerA, PlayerB, WinsForA, WinsForB)
            ELSE
               Writeln('GameNumber ', GameNumber:1, 'contained aan illegal play.')
      END;
   (* Print overall winner *)
   PrintBigWinner(WinsForA, WinsForB)
END.       (* Game  *)

PROGRAM Contoh (Input, Output);
(* Program dengan deklarasi variabel menggunakan tipe pengenal *)
(* yang sudah dideklarasikan *)
TYPE
   Pecahan  = real;
   Logika   = boolean;
   Bulat    = integer;
   Huruf    = string[25];
VAR
   Total, Gaji, Tunjangan : Pecahan;
```

```
    Menikah : Logika;
    JumlahAnak : Bulat;
    Keterangan : Huruf;
BEGIN
    Gaji := 50000.00;
    Menikah := True;
    JumlahAnak := 3;
    Tunjangan := 0.25 * Gaji + JumlahAnak * 30000.00;
    Total := Gaji + Tunjangan;
    Keterangan := 'Karyawan Teladan';
    Writeln('Gaji bulanan      : Rp  ',Gaji);
    Writeln('Tunjangan         : Rp  ',Tunjangan);
    Writeln('Total Gaji        : Rp  ',Total);
    Writeln('Sudah menikah     :  ',Menikah);
    Writeln('Jumlah Anak       :  ',JumlahAnak);
    Writeln('Keterangan        :  ',Keterangan);
END.

PROGRAM Contoh_Fungsi (Layar);
FUNCTION Tambah(x,y : integer) : integer;
BEGIN
   Tambah := x + y;
END;

BEGIN
   Writeln('2 + 3 =  ',Tambah(2,3));
END.

PROGRAM RataRata (Input, Output);
(* Program untuk membaca  nomor, nama, dan nilai tes dari sejumlah      *)
(* siswa, dan menghitung nilai rata-ratanya serta menampilkan hasil     *)
(* pengolahan ke layar disusun berdasarkan nilai rata-rata yang tertinggi  *)

CONST
   JumlahTes = 5; (* Jumlah maksimal tes *)
   MaksSiswa = 20;       (* Jumlah maksimal siswa *)
TYPE
   RekamanSiswa =  Record
                      Nomor : Integer;
                      Nama : String[20]
                      Nilai : Array[1..JumlahTes]  OF Real;
                      Rerata : Real;
                   END;
   AraySiswa = Array[1..MaksSiswa] OF RekamanSiswa;
VAR
```

```
    Jumlah: 1..MaksSiswa;
    i, j : Integer;
    TotalNilai : Real;
    Siswa  : ArraySiswa;
    Temporer : RekamanSiswa;
BEGIN
    clrscr;
(* Pemasukan data *)
    Write(' Jumlah data :  ');
    Readln(Jumlah);
    Writeln;
    FOR i := 1 TO Jumlah DO
       BEGIN
          Write('Nomor siswa :  ');
          Readln(Siswa[i].Nomor);
          Write('Nama siswa :  ');
          Readln(Siswa[i].Nama);
          Writeln('NILAI :  ');
          FOR j :=  1 TO JumlahTes DO
             BEGIN
                Write(j:2. '. ');
                Readln(Siswa[i].Nilai[j]);
             END;
       Writeln;
       END;
(* Menghitung nilai  rata-rata masing-masing siswa *)
    FOR i := 1 TO MaksSiswa DO
       BEGIN
          TotalNilai := TotalNilai + Siswa[i].Nilai[j];
          Siswa[i].Rerata :=TotalNilai / JumlahTes;
       END;
(* Mengurutkan data berdasarkan nilai rata-rata tertinggi *);
    FOR i     := 1 TO Jumlah - 1 DO
    FOR j := i + 1 TO Jumlah DO
       IF Siswa[i].Rerata  < Siswa[j].Rerata THEN
          BEGIN
          (* Tukarkan isi record *)
             Temporer := Siswa[i];
             Siswa[i] := Siswa[j];
             Siswa[j] := Temporer;
          END;
(* Tampilkan data yang telah diurutkan *)
    Writeln;
    Writeln('=============================');
    Writeln('Nomor          Nama          Nilai');
```

```
    Writeln('siswa          siswa          rata-rata');
    Writeln'═══════════════════════════════════');
    FOR i := 1 TO Jumlah DO
       Writeln(Siswa[i].Nomor:6, Siswa[i].Nama:20,
               Siswa[i].Rerata:10:1);
    Writeln('═══════════════════════════════════');
END.

PROGRAM HapusFile
(* Program menghapus file dari disk yang sama dengan perintah Del *)
(* pada Prompt DOS *)
VAR
    FileHapus: File;      (* variabel file *)
    NamaFileHapus : String[139];
PROCEDURE Logo;
    BEGIN
       Writeln('Utility menghapus suatu file di disk');
       Writeln('Versi 1.0 (C) Jogiyanto H.M, 1988');
       Writeln;
    END;        (* Logo *)
BEGIN (* Program utama *)
    Write('Nama file dihapus ? ');
    Readln(NamaFileHapus);
    IF NamaFileHapus = ' ' THEN Halt;
    Assign(FileHapus,NamaFileHapus);
    Erase(FileHapus);
    IF IOResult <> 0 THEN
       Writeln('File ini TIDAK DITEMUKAN  di disk !!!',^G)
    ELSE
       Writeln('File ',NamaFileHapus:1,' sudah dihapus');
END.

PROGRAM GantiNama;
(* Program mengganti nama suatu file di  disk  yang *)
(* berfungsi sama dengan perintah Rename pada promt DOS *)
VAR
    FileGantiNama : File;  (* variabel file *)
    NamaFileLama,
    NamaFileBaru : Stirng[139];
PROCEDURE Logo;
    BEGIN
       Writeln('Utility mengganti nama suatu file di disk');
       Writeln('Versi 1.0 (C) Jogiyanto H.M, 1988');
       Writeln;
    END;        (* Logo *)
```

```
BEGIN       (* Program utama *)
    Writeln('Nama file yang diganti ?');
    Readln(NamaFileLama);
    Writeln('Nama file baru  ?');
    Readln(NamaFileBaru);
    IF (NamaFileLma = ' ') OR (NamaFileBaru = ' ') THEN Halt;
    Assign(FileGantiNama,NamaFileLama);
    Rename(FileGantiNama,NamaFileBaru);
    IF IOResult <> 0 THEN
       Writeln('Salah, file TIDAK DITEMUKAN atau nama baru TELAH ADA !!!');
    ELSE
       Writeln('File telah diganti namanya');
END.

PROGRAM ContohStatemen1 (Input, Output);
VAR
   Pilihan : byte
   R,L,T,Luas : real;
BEGIN
   Clrscr;
   GotoXY(10,2); Writeln('       <<<PILIHAN>>>');
   GotoXY(10,4); Writeln('1. Menghitung Luas Lingkaran');
   GotoXY(10,6); Writeln('2. Menghitung Luas Segitiga');
   GotoXY(10,8); Writeln('3. Menghitung Luas Bujursangkar');
   GotoXY(10,10); Writeln('0. S e l e s a i');
   Pilihan := 9;
   WHILE (Pilihan < 0) OR (Pilihan > 3) DO
      BEGIN
         GotoXY(10,20); Write('Pilih Nomor (0-3) ? '); Read(Pilihan);
      END;
   Clrscr;
   IF Pilihan = 1 THEN
      BEGIN
         Write('Jari-jari lingkaran ? '); Readln(R);
         Luas := Pi * R*R;
         Writeln('Luas lingkaran =  ',Lusa:9:2);
      END;
   IF Pilihan = 2 THEN
      BEGIN
         Write('Panjang sisi alas ?  '); Readln(L);
         Write('Tinggi segitiga ?  '); Readln(T);
         Luas := 0.5 * L * T;
         Writeln('Luas segitiga = ',Luas:9:2);
      END;
   IF Pilihan = 3 THEN
```

```
        BEGIN
            Write('Panjang bujursangkar ? '); Readln(T);
            Write('Lebar bujursangkar ? '); Readln(L);
            Luas := T * L;
            Writeln('Luas bujursangkar =  ',Luas:9:2);
        END;
END.

PROGRAM ContohStatemen2 (Input, Output);
VAR
    Pilihan : byte
    R,L,T,Luas : real;
BEGIN
    Clrscr;
    GotoXY(10,2); Writeln('      <<<PILIHAN>>>');
    GotoXY(10,4); Writeln('1. Menghitung Luas Lingkaran');
    GotoXY(10,6); Writeln('2. Menghitung Luas Segitiga');
    GotoXY(10,8); Writeln('3. Menghitung Luas Bujursangkar');
    GotoXY(10,10); Writeln('0. S e l e s a i');
    Pilihan := 9;
    WHILE (Pilihan < 0) OR (Pilihan > 3) DO
        BEGIN
            GotoXY(10,20); Write('Pilih Nomor (0-3) ? '); Read(Pilihan);
        END;
    Clrscr;
 CASE Pilihan OF
1:  BEGIN
        Write('Jari-jari lingkaran ? '); Readln(R);
        Luas := Pi * R*R;
        Writeln('Luas lingkaran = ',Luas:9:2);
    END;
2:  BEGIN
        Write('Panjang  sisi alas ? '); Readln(L);
        Write('Tinggi segitiga ? '); Readln(T);
        Luas := 0.5 * L * T;
        Writeln('Luas segitiga  = ',Luas:9:2);
    END;
3:  BEGIN
        Write('Panjang bujursangkar ? '); Readln(T);
        Write('Lebar bujursangkar ? '); Readln(L);
        Luas := T  *  L;
        Writeln('Luas bujursangkar = ',Luas:9:2);
    END;
END;
END.
```

```
PROGRAM Rec3; (* Contoh pemakaian WITH *)
TYPE
   RecordTanggal = RECORD
                     Tanggal : 1..31;
                     Bulan : STRING[9];
                     Tahun : 1900..1999;
                   END;
   RecordPegawai = RECORD
                     Nama : STRING[30];
                     TanggalDiterima : RecordTanggal;
                   END;
VAR
   DataPegawai : RecordPegawai;
BEGIN  (* Mengisi field *)
   WITH DataPegawai DO
      WITH TanggalDiterima DO
         BEGIN
            Nama := 'FIKA';
            Tanggal := 1;
            Bulan := 'OKTOBER';
            Tahun := 1989;
         END;
(* Menampilkan isi field *)
WITH DataPegawai, TanggalDiterima DO
   BEGIN
      Writeln('Nama : ',Nama);
      Writeln('Tanggal Diterima: ',Tanggal, ' ',Bulan, ' ',Tahun);
   END;
END.

PROGRAM Rec4
(* Contoh pemakaian VARIANT RECORD untuk *)
(* mengolah gaji pegewai TETAP dan HONORER *)
CONST
   PERJAM = 3000;     (* Upah perjam *)
TYPE
   StatusPegawai = (TETAP, HONORER);
   RecordPegawai = RECORD
                     NRP : STRING[6];
                     Nama : STRING[15];
                     CASE Status : StatusPegawai OF
                        TETAP : (GajiPokok : LONGINT;
                                 JumJamLembur : BYTE);
                        HONORER : (JumJamKerja : BYTE);
```

```
                    END;
VAR
   Karyawan : RecordPegawai;
   Jawaban : CHAR;
   GajiTotal : LONGINT;
BEGIN
   Clrscr;
(* Pemasukan data ke variant record  *);

WITH Karyawan DO
   BEGIN
      Write('Nomor registrasi karyawan :  ');
      Readln(Karyawan.NRP);
      Write('Nama karyawan :  ');
      Readln(Karyawan.Nama);
      Write('Pegawai tetap (Y/T)? ');
      Readln(Jawaban);
      IF UPCASE(Jawaban) = 'Y' THEN
         Status := TETAP
      ELSE
         Status  := HONORER;
      CASE  Status OF
         TETAP : BEGIN
                    Write('Gaji pokok :  ');
                    Readln(GajiPokok);
                    Write('Jumlah jam lembur :  ');
                    Readln(JumJamLembur);
                    GajiTotal := GajiPokok + LONGINT(PERJAM) * JumJamLembur;
                   END;
        HONORER: BEGIN
                    Write('Jumlah jam kerja :  ');
                    Readln(JumJamKerja);
                    GajiTotal := LONGINT(PERJAM) * JumJamKerja;
                   END;
      END;         (* Akhir CASE *)
      Writeln;
      Writeln('Gaji total = Rp.  ',GajiTotal,  ',-');
   END;     (* Akhir WITH *)
END.
```

As we can see from the examples above that a Pascal program begins with the word PROGRAM followed by the name of the program and a list of one or more file names. This

first line of the program is the *heading* . The heading is followed by *declarations*, statements that define the terms used in tthe program. The word BEGIN signals the start of the actual instructitons, or *executable statements*, that the computer will follow. The end of the program is indicated by the word END, followed by a period. Basically the syntax rule of a Pascal program is:

<pascal-program>::= <program-heading> <declaration-part> BEGIN
<statement-part> END"."

That's why this chapter will be divided into three parts, that will discuss : program heading, declaration part, statement part.

III.1. Program heading

DATA

Let's take a look to these program headings:

```
PROGRAM Example (Input, Output);
PROGRAM Payroll (Input, Output, PayFile);
PROGRAM HouseCost(Output);
PROGRAM Game (FileA, FileB, Output);
PROGRAM RataRata (Input, Output);
PROGRAM Contoh (Input, Output);
PROGRAM HapusFile;
PROGRAM GantiNama;
```

ANALYSIS

We can see that the word PROGRAM is always come at the beginning of program heading, followed by the next

words: **Example**, **Payroll**, **HouseCost**, **Contoh**, **RataRata**, **Game**, **GantiNama**, **HapusFile**. These kind of words can be called as ***program-name***: the name of the program. **(Input, Output)**, **(Input, Output, PayFile)**, **(Output)**, **(FileA, FileB, Output)** are called as ***file-name list***, while **input**, **output**, **PayFile**, **FileA**, **FileB** are ***file-name*** (Dale and Weems, 1991:103-104; Kadir, 1991:18;), which is used to communicate between the program and the I/O of computer. File-name list is optional. The last symbol that form program heading is the semi colon (;) sign. Based on the analysis above we can write the syntax rule for program heading:

<program-heading> ::= PROGRAM <identifier> [<file-name-list>];

<file-name-list>::= <file-name> | <file-name-list>,<file-name>

<file-name>::= <identifier>

III.2. Declaration part

This subchapter will be divided into five parts: Constans definition part, Type definition part, Variable definition part, Procedure declaration part, and Function declaration part.

III.2.1. Constants definition part

DATA

```
CONST
   Freeze = 32;
   Boil   = 212;
CONST
   MaxHours = 40.0;
   OverTime = 1.5;
CONST
   Width = 30.0;
   Length =  40.0;
   Stories = 2.5;
   NonLivingSpace = 825.0;
   Price  = 150000.0;
CONST
   JumlahTes = 5;
   MaksSiswa = 20;
```

ANALYSIS

Constants definition part begins with the reserved word CONST. *MaxHours, OverTime, Width, Length, Stories, NonLivingSpace, Price, JumlahTes, MaksSiswa* are identifiers. While *32, 212, 40.0, 15, 30.0, 2.5, 825.0, 1500000.0, 5, 20,* are the value of the constant identifier, that are usually called as *constants*. So the syntax rule for constants definition part are:

<constant-definition-part>::= CONST *<constant-definition>*";" {*<constant-definition>*";"}

<constant-definition>::= *<identifier>* "=" *<constant>*

<constant>::= [*<sign>*] (*<constant-identifier>* | *<number>*) | *<string>*

<constant-identifier>::= *<identifier>*

<number>::= <integer-number> | <real-number>

<string>::= ""<string-character> {<string-character>}""

<integer-number>::= <digit-sequence>

<real-number>::= <digit-sequence> "." [<digit-sequence>] [<scale-factor>]
<digit-sequence> <scale-factor>

<scale-factor>::= ("E" | "e") [<sign>] <digit-sequence>

<digit-sequence>::= [<sign>] <unsigned-digit-sequence>

<unsigned-digit-sequence>::= <digit> {<digit>}

<sign>::= "+" | "-"

<string-character>::= any-character-except-quote | """

III.2.2. Type definition part

```
        DATA

TYPE
   PlayType = (Rock, Paper, Scissors);

TYPE
   Pecahan  = real;
   Logika   = boolean;
   Bulat    = integer;
   Huruf    = string[25];
TYPE
   RekamanSiswa =  Record
                     Nomor : Integer;
                     Nama : String[20]
                     Nilai : Array[1..JumlahTes]  OF Real;
                     Rerata : Real;
                   END;
   ArraySiswa = Array[1..MaksSiswa] OF RekamanSiswa;
TYPE
   RecordTanggal  = RECORD
                     Tanggal : 1..31;
                     Bulan : STRING[9];
                     Tahun : 1900..1999;
                  END;
   RecordPegawai = RECORD
```

```
                Nama : STRING[30];
                TanggalDiterima : RecordTanggal;
              END;
TYPE
  StatusPegawai = (TETAP, HONORER);
  RecordPegawai = RECORD
                NRP : STRING[6];
                Nama : STRING[15];
                CASE Status : StatusPegawai OF
                  TETAP : (GajiPokok : LONGINT;
                           JumJamLembur : BYTE);
                  HONORER : (JumJamKerja : BYTE);
                END;
```

ANALYSIS

If a constant-definition-part begins with the reserved word **CONST**, a type-definiton-part begins with the reserved word **TYPE**. As in constant-definiton-part, words on left side of equal (=) sign are identifiers. While the words on the right side of equal (=) sign are the data type of the identifiers, or it is simply called as *type*. So the syntax rule for type definition part is:

<type-definition-part>::= TYPE <type-definition>; {<type-definition>;}

<type-definition>::= <identifier> "=" <type>

Type itself is consist of *simple-type* and *structured-type*. Simple-type consist of *real, integer, char, boolean, string, subrange-type*.

and *enumerated-type* (Dale and Weems.1991:422). These are the example of simple-type:

```
TYPE
   Pecahan  = real;
   Logika   = boolean;
   Bulat    = integer;
   Huruf    = string[25];
```

enumerated-type:

```
TYPE
   PlayType = (Rock, Paper, Scissors);
    Starch  =  (Corn, Rice, Potato, Bean);
    Grain  =  (Wheat, Rye, Barley, Sorghum);
    Animals = (Rodent, Cat, Dog, Bird, Reptile, Horse,  Bovine, Sheep);
    StatusPegawai  = (TETAP, HONORER);
```

subrange-type:

```
TYPE
  Num = 1..100;
  Letter = 'A'..'Z';
  StudentNum = 1..50;
  Cageable = Rodent..Dog;
  Barnyard  =  Horse..Sheep;
```

Now here is the syntax rule for simple-type.

<type>::= <simple-type> | <structured-type>

<simple-type>::= integer | char | real | string | boolean | <enumerated-type>
| <subrange-type>

<enumerated-type>::= <identifier>, {<enumerated-type>}

<subrange-type>::= lower-value..upper-value

The structured-type consist of *array-type* and *record-type*. These are the examples of structured-type:

array-type:

```
ArraySiswa = Array [1..MaksSiswa] OF RekamanSiswa;
ValueList = Array [IndexRange] OF Integer;
AlphaList = Array [Char] OF Integer;
GradeType = Array [NumGrades] OF Char;
GradeCount = Array [LetterGrades] OF Integer;
AmountType = Array [Drink] OF Real;
```

According to Dale and Weems(1991:465), [1..MaksSiswa],[IndexRange],[Char],[NumGrades],[LetterGrades],[Drink] is called *index-type*, which gives the range of index values. It is used by the compiler to determine how many components are in this *array-type* and how each individual component is accessed. While RekamanSiswa, Integer, Char, and Real are called *component-type*. The component-type describes what is stored in each component of the array. Array components can be any type. Here is the syntax rule for array-type:

<*array-type*>::= ARRAY "["<*index-type*>"]" OF <*component-type*>

<*component-type*>::= <*type*>

Record-type:

```
TYPE
  RekamanSiswa = Record
                   Nomor : Integer;
                   Nama : String[20]
                   Nilai : Array[1..JumlahTes] OF Real;
                   Rerata : Real;
                 END;
TYPE
  RecordTanggal = RECORD
                   Tanggal : 1..31;
```

```
                Bulan : STRING[9];
                Tahun : 1900..1999;
              END;
  RecordPegawai = RECORD
                Nama : STRING[30];
                TanggalDiterima :  RecordTanggal;
              END;
TYPE
  RecordPegawai = RECORD
                NRP : STRING[6];
                Nama : STRING[15];
              END;
```

Nomor, Nama, Nilai, and Rerata are *field-identifier* within the record type RekamanSiswa. And so are Tanggal, Bulan, Tahun are the field identifier within the record type RecordTanggal. Note that each field identifier is given a type.

Nama, Bulan, and NRP are of string type. Nomor is an integer field. Tanggal is an integer field in the subrange 1 to 31. Tahun is an integer field in the subrange 1900 to 1999. TanggalDiterima is of record type RecordTanggal. The record type always begin with the reserved word *Record* and end with *End*. So, the syntax rule for record type is:

<record-type> ::= RECORD <record-section> END ";"

<record-section>::= <field-identifier> "," {<field-identifier>}":" <type> ";"
{<record-section>}

<field-identifier>::= <identifier>

III.2.3. Variable definition part

```
        DATA

VAR
   AvgTemp:          (* Variable to hold the result    *)
      Real;          (* of averaging Freeze and Boil *)
VAR
   PayRate,          (* Employee's pay rate          *)
   Hours,            (* Hours worked                      *)
   Wages,            (* Wages earned                      *)
   Total:            (* Total company payroll             *)
      Real;
   EmpNum:           (* Employee ID number           *)
      Integer;
   PayFile:          (* Company payroll file         *)
      Text;
VAR
   GrossFootage,
   LivingFootage,
   CostPerFoot:
      Real
VAR
   PlayerA,     (* Player A's play  *)
   PlayerB:     (* Player B's play  *)
      PlayType;
   WinsForA,    (* Number of games A wins  *)
   WinsForB,    (* Number of games B wins  *)
   GameNumber:  (* Number of games played  *)
      Integer;
   FileA,            (* Player A's play  *)
   FileB:            (* Player B's play  *)
      Text;
   Legal:            (* True if play is legal *)
      Boolean;
```

ANALYSIS

The variable definition part is almost similar with constant definition part, except it begins with the word

VAR instead of CONST. The word VAR followed by identifiers such as: AvgTemp, PayRate, Hours, Wages, Total, EmpNum,PayFile. All the identifiers have their own type. PayRate, Hours, Wages,and Total are of Real type. EmpNum is integer type, etc. The syntax rule of variable definition part is:

<variable-definition-part>::= var <identifier-list> ":" <type> ";"
<identifier-list>::= <identifier> {"," <identifier>}
<identifier>::= <letter> {<letter> | <digit>} .
<letter>::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "I" | "J" | "K" | "L" |
"M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "X" |
"Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" |
"l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" |
"x" | "y" | "z"
<digit>::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

III.2.4. Procedure declaration part

A procedure is a subprogram, that's way it looks like a program except that PROGRAM heading is replaced by a PROCEDURE heading, and the last END in the procedure has a semicolon following it instead of a period.

A PROGRAM heading names a program and lists the files from which it gets input and to which it sends output. Similarly, a PROCEDURE heading names a procedure

and lists the variables (parameters) that serve as its input and output.

Let's take a look at some of the procedure declaration part:

```
PROCEDURE CalcPay(VAR  PayRate, (* Employee's pay rate *)
                       Hours:        (* Hours worked      *)
                       Real;
                  VAR  Wages:        (* Wages earned      *)
                       Real;
(* Calc Pay  computes wages wages from the employee's  pay rate *)
(* and the hours worked, taking overtime into account          *)
BEGIN  (* CalcPay *)
   IF Hours > MaxHours          (* Check for overtime *)
      THEN                      (* Computes wages if overtime *)
         Wages := (MaxHours * PayRate)  +
                  (Hours - MaxHours) * PayRate * Overtime
      ELSE
         Wages := Hours * PayRate   (* Computes wages if no overtime *)
END;    (* CalcPay *)

PROCEDURE PlayerAWins  (VAR    GameNumber: (* Receives game number *)
                               Integer;
                        VAR  WinsForA:     (* Rec's/returns win count *)
                               Integer);
(* Message  that Player A has won the current game is written,   *)
(* and Player A's total is  updated   *)
BEGIN   (* PlayerAWins  *)
   Writeln('Player A has won game number ',  GameNumber:1, '.');
   WinsForA  := WinsForA + 1
END;    (* PlayerAWins *)
(*************************************************************************)
PROCEDURE PlayerBWins  (VAR    GameNumber    (* Receives game number *)
                               Integer;
                        VAR  WinsForB:       (* Rec's/returns win count *)
                               Integer);
(* Message  that Player B has won the current game is written,   *)
(* and Player B's total is  updated   *)
BEGIN   (* PlayerBWins  *)
   Writeln('Player B has won game number ',  GameNumber:1, '.');
   WinsForB  := WinsForB + 1
END;    (* PlayerBWins *)
(*************************************************************************)
```

```
PROCEDURE ProcessPlays (VAR GameNumber: (* Receives game number *)
                                    Integer;
                                PlayerA,    (* Receives A's play *)
                                PlayerB:    (* Receives B's play *)
                                    PlayType;
                            VAR WinsForA, (* Rec's'returns A's wins *)
                                WinsForB: (* Rec's'returns B's wins *)
                                    Integer;
(* ProcessPlays determines the winning play. If there is a tie, a  *)
(* message is written. Otherwise the number of wins of the winning  *)
(* player is incremented  *)
BEGIN  (* ProcessPlay *)
   IF PlayerA = PlayerB
      THEN
         Writeln('Game number  ', GameNumber:1, ' is a tie.');
      ELSE IF (PlayerA = Paper) AND (PlayerB = Rock)
         THEN
            PlayerAWins(GameNumber, WinsForA)
      ELSE IF (PlayerA = Scissors) AND (PlayerB = Paper)
         THEN
            PlayerAWins(GameNumber, WinsForA)
      ELSE IF (PlayerA = Rock) AND (PlayerB = Scissors)
         THEN
            PlayerAWins(GameNumber, WinsForA)
      ELSE
         PlayerBWins(GameNumber, WinsForB)
END; (* ProcessPlays *)
(*****************************************************************************)

PROCEDURE GetPlays (VAR  PlayerA,  (* Returns A's plays *)
                         PlayerB:  (* Returns B's plays *)
                             PlayType;
                    VAR Legal:     (* Returns True if legal plays *)
                             Boolean);
(* PlayerA's play is read from FileA, PlayerB's play is read from  *)
(* FileB. If both plays are legal, Legal is set to True and both plays  *)
(* are converted to corresponding values of  PlayType. Else Legal  *)
(* is False and PlayerA and PlayerB are undefined. FileA and FileB*)
(* are accessed globally  *)
VAR
   CharForA,  (* PlayerA's input *)
   CharForB:  (* PlayerB's input *)
      Char;
```

```
PROCEDURE Logo;
  BEGIN
    Writeln('Utility menghapus suatu file di disk');
    Writeln('Versi 1.0 (C) Jogiyanto H.M, 1988');
    Writeln;
  END;                    (* Logo *)
```

The above segments are the ***procedure declaration***. Notice that the last END are always followed by a semicolon. Since this are a declaration, they will appear in the declaaration section of the program. Even though a peocedure looks a lot like a program, it is important to remember that only a program ends with a period.

Take a look at the PROCEDURE heading. Just like any other identifier in Pascal, the name of a procedure is not allowed to include blanks. After the naame we'll see some codes that look like a variable declaration between parentheses. This is a ***parameter declaration***. Parameters represent the way the main program and procedures (or procedures and other procedures) communicate. Parameters enable the main program to input (pass) values to a procedure to use in its processing, and the procedure to output (return) results to the program. The parameters in the program's call to a procedure are the ***actual parameters***. The parameters listed in the procedure heading are the ***formal parameters***. These formal parameters are called ***variable*** or ***VAR. parameters*** because the values of the actual parameters

may be changed by the procedure. This is also why they are similar in appearance to a VAR declaration in the main program. A procedure can include variable declarations within the procedure itself. This variables are *local variables* because they are accessible only within the procedure in which they are declared. Based on the examples above, we can describe the rule that governing the procedure declaration part syntax as:

<procedure-declaration-part>::= <procedure-heading> [<local-variables>]
BEGIN <statement-sequence> END ";"

<procedure-heading>::=PROCEDURE <identifier> <formal-parameter-list>

<local-variables>::= <variable-definition-part>

<variable-definition-part>::= var <identifier-list> ":" <type> ";"

<identifier-list>::= <identifier> {"," <identifier>}

<identifier>::= <letter> {<letter> | <digit>}

<letter>::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "I" | "J" | "K" | "L" |
"M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "X" |
"Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" |
"l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" |
"x" | "y" | "z"

<digit>::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<statement-sequence>::= <statement> {";" <statement> }

<formal-parameter-list>::= VAR <identifier> {"," <identifier> } ":" <type>";"
{<formal-parameter-list>}

III.2.5. Function declaration part

Functions are very similar to procedures. The difference is that functions are used when there is only one result value, and that result is to be used directly in an expression. The first thing to note about the function declaration is that it looks like a procedure declaration, except the heading begins with the word FUNCTION instead of PROCEDURE. If we look closely at the heading, we will notice something else: the entire parameter list is followed by a colon (:) and then a data type.

A function returns one value, not through a parameter but through the name of the function. The data type at the end of the heading defines the type of value that the function will return. This type is called the ***function type***, although a more proper term is ***function result type***. These are the examples of function declaration part:

```
FUNCTION Convert (Character:          (* Receives play character   *)
                  Char): PlayType;    (* Returns play literal       *)
(* Converts character into associated value in PlayType *)
BEGIN (* Convert *)
   CASE Character OF
      'R' : Convert := Rock;
      'P' : Convert := Paper;
      'S' : Convert := Scissors
   END; (* Case *)
END; (* Convert *)
```

```
FUNCTION Power (X,                    (* Base number *)
                N:                    (* Power to raise base to *)
                  Integer):
                    Integer           (* Returns X to N power *)
(* This function computes X to the N power  *)
VAR
   Result:                (* Holds intermediate power of X *)
      Integer;
BEGIN          (* Power *)
   Result := 1;
   WHILE N > 0 DO
        BEGIN
           Result := Result * X;
           N := N - 1;
        END;
   Power := Result;
END;           (* Power *)

FUNCTION Factorial (X:          (* The factorial to be computed *)
                    Integer):
                       Integer;
(* This function computes X! *)
VAR
   Result:                (* Holds partial products *)
        Integer;
BEGIN          (* Factorial *)
   Result := 1;
   WHILE X > 0 DO
        BEGIN
           Result := Result * X;
           X := X - 1;
        END;
    Factorial := Result
END;           (* Factorial *)
```

Since that function declaration is almost similar to procedure declaration, then it won't be any difficulties to formulize the syntax rule:

<function-declaration-part>::= <function-heading> [<local-variables>]

BEGIN <statement-sequence> END ";"

<function-heading>::=FUNCTION <identifier> <formal-parameter-list>":"
<function-type>";"
<local-variables>::= <variable-definition-part>
<variable-definition-part>::= VAR <identifier-list> ":" <type> ";"
<identifier-list>::= <identifier> {"," <identifier>}
<identifier>::= <letter> {<letter> | <digit>}
<letter>::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "I" | "J" | "K" | "L" |
"M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "X" |
"Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" |
"l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" |
"x" | "y" | "z"
<digit>::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<statement-sequence>::= <statement>";" { (<statement>";") }
<formal-parameter-list>::= "("VAR <identifier> {"," <identifier> } ":"
<type>";" {<formal-parameter-list>}")"

III.3. Statement part

Statement part is marked by the appearance of the reserved word BEGIN at the initial position, and END at the final position. There are eight kinds of statement: **assignment statement, procedure statement, if statement, case statement, while statement, repeat statement,** and **for statement** . The syntax rule for statement part is:

<statement-part>::= BEGIN <statement-sequence> END"."
<statement-sequence>::= <statement> ";" {(<statement>";")}

<statement>::= <assignment-statement> | <procedure-statement> |
<if-statement> | <case-statement> | <while-statement> |
<repeat-statement> | <for-statement>

III.3.1. Assignment statement

Assignment statement is a statement that gives the value of an expression to a variable. These are the examples of assignment statements:

```
Result := 1;
Result := Result * X;
N := N - 1;
X := X - 1;
Factorial := Result
Power := Result;
```

Result and N (the first four examples) at the left side of ":=" sign are variables, while factorial and power (the last two examples) are function identifier (see FUNCTION power page 28 and FUNCTION factorial page 29). Anything comes up at the right side of ":=" sign is called expression. The syntax rule of assignment statement is:

<assignment-statement>::= (<variable> | <function-identifier>) ":="
<expression>";"

<variable>::= <identifier>

<function-identifier>::= <identifier>

<expression>::= <simple-expression> [<operator> <simple-expression>]

<operator>::= "=" | "<>" | "<" | ">" | "<=" | ">=" | "+" | "-" | "*" | "/" | div | mod

<simple-expression> ::= <variable> | <number>

III.3.2. Procedure statement

Procedure statement has the function to activate or call the procedures have been defined in procedure declaration part or a standard procedure provided by Pascal, such as: read, readln, write, writeln. The basic pattern of procedure statement is a procedure identifier followed by an optional parameter list. The paramater list is delimited by parentheses. It can consists of: a literal value, constants, expressions, variables, functions. These are the examples of procedure statements:

```
Writeln('Water freezes at ', Freeze);
Writeln(' and boils at ', Boil, ' degrees. ');
Rewrite(PayFile);                              (* Open File PayFile *)
Readln(EmpNum);                                (* Read employee ID number *)
PrintResult(Sum(A,B,C)+1);
CetakHasil(SqrX);              (* Memanggil prosedur cetakhasil *)
Write(A+B*C);
```

'Water freezes at' , 'and boils at' are the examples of literal value or string. PayFile, and EmpNum are variables. Sum and Sqr are functions, while A+B*C is an example of expression. Based on the examples above we write the syntax rule for procedure statement:

<procedure-statement>::= <procedure-identifier> [<parameter-list>] ";"

<procedure-identifier>::= <identifier>

<parameter-list>::=<constant> <parameter-list> | <variable>
<parameter-list> | <function-identifier> <parameter-
list> | <expression> <parameter-list>

III.3.3. If statement

The control structure that allows branches in the flow of control is called an IF statement. With it, we can ask a question and choose a course of action: IF a certain condition exists, THEN perform one action, ELSE perform a different action.

The computer actually performs just one of the two actions under any given set of circumstances. But we have to write both actions into the program. Why? Because depending on the circumstances, the computer can choose to execute one of them. The IF statement gives us a way of including both actions in a program and gives the computer a way of deciding which action to take. These are the examples of IF statement.

```
IF Hours > MaxHours                (* Check for overtime *)
    THEN                                (* Computes wages if overtime *)
      Wages := (MaxHours * PayRate) +
               (Hours - MaxHours) * PayRate * Overtime
    ELSE
      Wages := Hours * PayRate   (* Computes wages if no overtime *)

IF WinsForA > WinsForB
    THEN
```

```
        Writeln('Player  A has won the most games.')
    ELSE IF WinsForB > WinsForA
        THEN
            Writeln('Player B has won the most games.')
    ELSE
        Writeln('Player A and B have tied.')

IF IOResult <> 0 THEN
    Writeln('File ini TIDAK DITEMUKAN  di disk !!!',^G)
  ELSE
    Writeln('File ',NamaFileHapus:1,' sudah dihapus');
```

From the above examples we can construct the syntax rule of IF statement.

<if-statement>::= IF <expression> THEN <statement>

[ELSE<statement>]

III.3.4. Case statement

The CASE statement is a selection control structure that allows us to list any number of branches. It is similar to a nested IF statement. The value of the ***case selector***, an expression whose result must match a label attached to a branch, determines which one of the branches is executed. These are the examples.

```
CASE Character OF
    'R' : Convert := Rock;
    'P' : Convert := Paper;
    'S' : Convert := Scissors
  END (* Case *)

CASE Pilihan OF
1: BEGIN
```

```
      Write('Jari-jari lingkaran ? '); Readln(R);
      Luas := Pi * R*R;
      Writeln('Luas lingkaran = ',Luas:9:2);
   END;
2: BEGIN
      Write('Panjang sisi alas ? '); Readln(L);
      Write('Tinggi segitiga ? '); Readln(T);
      Luas := 0.5 * L * T;
      Writeln('Luas segitiga  = ',Luas:9:2);
   END;
3: BEGIN
      Write('Panjang bujursangkar ? '); Readln(T);
      Write('Lebar bujursangkar ? '); Readln(L);
      Luas := T  *  L;
      Writeln('Luas bujursangkar = ',Luas:9:2);
   END;
END;

CASE Status OF
        TETAP : BEGIN
                  Write('Gaji pokok : ');
                  Readln(GajiPokok);
                  Write('Jumlah jam lembur : ');
                  Readln(JumJamLembur);
                  GajiTotal := GajiPokok + LONGINT(PERJAM) * JumJamLembur;
                END;
        HONORER: BEGIN
                    Write('Jumlah jam kerja : ');
                    Readln(JumJamKerja);
                    GajiTotal := LONGINT(PERJAM) * JumJamKerja;
                  END;
     END;       (* Akhir CASE *)
```

Character, Pilihan, and Status are called case selector. 'R', 'P', 'S', 1. 2, 3, TETAP, HONORER, are labels, and the list of labels is the case label lists. Each case label value may appear only once in a given CASE statement. If a value appears more than once, an error will result. The syntax rule of CASE statement is:

<case-statement>::= CASE <case-selector> OF <case-label-list>

(<statement> | (BEGIN <statement-sequence> END) ";") END";"

<case-selector>::= <expression>

<case-label-list>::= <constant> {"," <constant>}

III.3.5. While statement

The WHILE statement, like the IF statement, tests a condition. These are the examples of WHILE statement:

```
WHILE EmpNum <> 0 DO                    (* While employee number <> 0  *)
    BEGIN
       Write('Enter pay rate:  ');                                  (* Prompt *)
       Readln(PayRate);                                  (* Read hourly pay rate *)
       Write('Enter hours worked:  ');                              (* Prompt *)
       Readln(Hours);                                  (* Read Hours worked *)
       CalcPay(Payrate, Hours, Wages);                   (* Computes wages *)
       Total := Total + Wages;                           (* Add wages to total *)
       Writeln(PayFile, EmpNum, PayRate, Hours, Wages); (*Put result in PayFile*)
       Write('Enter employee number:  ');                           (* Prompt *)
       Readln(EmpNum);                                   (* Read ID number *)
    END;

WHILE NOT EOF(FileA) AND NOT EOF(FileB) DO
    BEGIN
       GameNumber := GameNumber + 1;
       GetPlays(PlayerA, PlayerB, Legal);
       IF Legal
          THEN
             ProcessPlay(GameNumber, PlayerA, PlayerB, WinsForA, WinsForB)
          ELSE
             Writeln('GameNumber ', GameNumber:1, 'contained aan illegal play.')
    END;
```

The syntax rule of WHILE statement is almost similar to IF statement.

<while-statement>::= WHILE <expression> DO (<statement> |
<statement-sequence>)";"

III.3.6. Repeat statement

The REPEAT statement is a looping control structure in which the loop condition is tested at the end of the loop. This format guarantees that the loop body is executed at least once. These are the examples of REPEAT statement:

```
REPEAT
    Writeln('Enter a test score. ');
    Readln(Score);
    IF (Score < 0) OR (Score > 100)
       THEN
          Writeln('Invalid score. Score must be ',
                  in the range of 0 through 100. ')
UNTIL (Score >= 0) AND (Score <= 100)

REPEAT
    Writeln(K, 3*K);
UNTIL (K >= 9) AND (K <= 20)
```

The REPEAT statement and WHILE statement in controlling the loop are complements (opposites) of each other. The WHILE continues looping as long as the expression is True, while the REPEAT continues looping as long as the expression is False. Because the WHILE statement tests the condition before executing the body of the loop, it is called a pretest loop. The REPEAT statement does the opposite, and is thus known as a

posttest loop. It is important to remember that WHILE uses the expression to decide when to keep looping, and REPEAT uses it to decide when to stop. To transform a WHILE into a REPEAT or vice versa, the expression must be complemented. The syntax rule is:

<repeat-statement>::= REPEAT <statement-sequence> UNTIL
<expression>

III.3.7. For statement

The FOR statement is designed to simplify the writing of count-controlled loops. The statement

```
FOR Count := 1 TO N DO
    statement;
```

means "Set the loop control variable Count to 1. If N is less than 1, do not enter the loop. Otherwise, execute the statement and increment Count by 1. Stop the loop after it has been executed with Count equal to N." Instead of TO, we can also use DOWNTO. If DOWNTO is used, the loop control variable is decremented by 1 for each iteration, instead of being incremented by 1. These are the examples of FOR statement.

```
FOR EndLetter := 'A' TO 'G' DO
    BEGIN
      FOR PrintedLetter := 'A' TO EndLetter DO
          Write(PrintedLetter);
      Writeln
    END;
```

```
FOR Letter := 'A' TO 'Z' DO
        Write(Letter);
Writeln;

FOR Letter := 'Z' DOWNTO 'A' DO
        Write(Letter);
Writeln;
```

The syntax rule is:

<for-statement>::= FOR <variable-identifier> ":=" <initial-expression> (TO | DOWNTO) <final-expression> DO (<statement> | <statement-sequence>)";"

III.3.8. With statement

In working with record variables, we often need to access one or more fields repeatedly in a small section of code. Field selectors can get rather long. The WITH statement allows us to abbreviate the notation of field selectors by specifying the record name once and then using the field identifiers to select the record components. These are the examples of WITH statements.

```
PROCEDURE WriteMachine (Machine: MachineRecord);
BEGIN (* WriteMachine *)
   WITH Machine DO
       BEGIN
           Writeln(IDNumber);
           Writeln(Description);
           Writeln(History.FailRate);
           Writeln(History.LastServiced.Month:2, '/',
                   History.LastServiced.Day:2, '/',
                   History.LastService.Year:4);
           Writeln(History.DownDays:4);
           Writeln(PurchaseDate.Month:2, '/',
                   PurchaseDate.Day:2, '/',
                   PurchaseDate.Year:4);
           Writeln(Cost:8:2);
```

```
    END;
END;        (* WriteMachine *)

PROCEDURE WriteMachine (Machine: MachineRecord);
BEGIN (* WriteMachine *)
    WITH Machine DO
        BEGIN
            Writeln(IDNumber);
            Writeln(Description);
          WITH History DO
            BEGIN
               Writeln(FailRate);
               WITH LastServiced DO
                  Writeln(Month:2, '/', Day:2, '/', Year:4);
               Writeln(DownDays:4);
            END;
          WITH PurchaseDate DO
            Writeln(Month:2, '/', Day:2, '/', Year:4);
          Writeln(Cost:8:2);
    END;
END;        (* WriteMachine *)
```

From the examples we can construct the syntax rule of WITH statement:

<with-statement>::= WITH <record-variable> {"," <record-variable>} DO
(<statement> | <statement-sequence>)";"

CHAPTER IV

CONCLUSION