DIMENSIONAL ANALYSIS IN PASCAL

Mukul Babu Agrawal and Vijay Kumar Garg
F 312, Hall One
I.I.T. Kanpur, INDIA

Summary :
The paper attemets to introduce the concept of Dimensions in programming languages as a tool for scientific computations.

key words : pascal dimensions units type

Since the introduction of strong type attributes associated with every variable,the type has come to signify the structure of the variable .This paper explores another attribute which though used so often in daily practice is alas forgotten in programming languages, namely Dimensions.

Providing dimensional attribute has a strong impact towards a new programming style in scientific computations which while being more natural avoids inadvertant errors . A scientist trying to analyze his experimental data prefers to maintain dimensioinal consistency between various data elements and therefore finds data structures such as integers and reals too primitive to provide the facility of carrying units with him.e.g. he would prefer
SPEED := 3.0 METRE / SEC
rather then bare SPEED := 3.0 .An earlier attempt in this direction by ADA using derived types [1] fails to achieve its basic objectives .e.g. It would not allow :

```
procedure DEFICIENT is
   type SPEED is new FLOAT ;
   type TIME is new FLOAT  ;
   type DISTANCE is new FLOAT ;
   S : SPEED ;
   T : TIME ;
   D : DISTANCE ;
begin
   S:=5.0 ; T:= 3.0 ;
   D:=S*T;
   -- The types are incompatible and hence an error would
   -- occur although it was all done with good intentions.
end ;
```

The definition of our design to take care of the dimensional attributes is described next.

The programmer declares all dimensions that he intends to use by a dim declaration (in "declare before use" spirit of PASCAL) .In our implementation the dim declaration is made just after label decl.e.g.

```
label 1,2 ;
dim MASS , LENGTH , TIME ;
```

This declares 3 dimensions namely MASS , LENGTH and TIME which may now be used to define types. e.g.

```
type METRE = real dim
                    LENGTH => 1
             end * 1.0  ;
    NEWTON = real dim
                    MASS   => 1;
                    LENGTH => 1;
                    TIME => -2
             end * 1.0 ;
```

The number and the operator at the end of the defns. denote a conversion factor w.r.t. some basic unit of those dimensions. It may be observed that the units with a conversion factor of 1.0 may be considered as basic units . Dimensional expressions are allowed which may use previously defined types. e.g. JOULE = NEWTON METRE ; is a valid type declaration. The juxtaposition , in accordance with the day to day use of dimensions, means multiplication.It is equivalent to the following definition :

```
      JOULE = real dim
                    MASS => 1 ;
                    LENGTH => 2 ;
                    TIME => -2
             end * 1.0 ;
```

Another noteworthy feature is automatic conversion within units of the same dimension . e.g.

```
ANGSTROM = METRE / 1.0E+10 ;
```

declares ANGSTROM to have same dimensions as that of METRE but with a conversion factor of 1.0E-10 w.r.t. the basic unit METRE. The compiler can automatically generate code to convert all units to basic units , thus relieving the programmer from frequent multiplication with a factor.

With these structures at hand , the compiler can easily do dimensional analysis and point out to the user dimensionally incorrect statements.An assignment would then be between compatible types only . Please note that definition of type compatibility in pascal itself is vague [2] .To avoid going into side tracks let us agree on compatibility as structure compatibility and dimensional compatibility.

In a statement involving ':=' ,'+','-','*','/' the check is made as follows.

':=','+','-' : allowed only if both operands are of same dimensions.Hence an expression of the following type is illegal.

id_of_speed_dim + id_of_time_dim

Similarly , id_of_mass_dim:=id_of_area_dim would be rendered as error.

'*','/' : These operators change the dimensions of the resulting expression by adding or differencing corresponding dimensions. For e.g. an assignment of the kind

id_of_area_dim:=id_of_length_dim * id_of_length_dim ;

is valid because dimensions of R.H.S become length =>1 + 1; On the other hand '/' operator , instead of '*' would not be permissible in the statement because then r.h.s has dim length =>1 - 1 which is different from that of L.H.S. All reals which are declared without DIM are taken as dimensionless quantities.A standard function then needs to be provided.

function MAG(X:real_with_dimensions):real;

This function converts any dimensional quantity to dimensionless real.The mnemonic MAG stands for magnitude.

The programmer may use dimensional expressions in statements . e.g. SPEED := 3.0 METRE / SEC .

```
< BNF lovers > ::=
 <dim_decl>::= dim <id_list> ;
 <id_list>::=id { , id }
 <type_decl>::=types_as_before ! <dim_type>
 <dim_type>::= <sim_dim> ! <comp_dim>
 <sim_dim>::= real dim <dim_assn>
              { ; dim_assn } end < dim_conv_fact >
 <dim_assn>::= dim_id => int_const
 <dim_conv_fact>::= * real_const  ! / real_const
 <comp_dim>::= <dim_exp> <dim_conv_fact>
 <dim_exp>::= dim_fact  [ / dim_fact  ]  ! 1 / dim_fact
 < expression >::= expressions_as_before [ dim_exp ]
 < dim_fact > ::= dim_id ! ( dim_id ** int_const )
```

An example to illustrate various constructs is given below.

```
*****************************************************************
program DEMONSTRATE(INPUT,OUTPUT);
   dim MASS,LENGTH,TIME ;
   type
      KG = real dim
                        MASS => 1
               end * 1.0 ;
      METRE=real dim
                        LENGTH => 1
               end * 1.0 ;
      KM = METRE * 1000.0 ;
      SEC= real dim
                        TIME => 1
               end * 1.0 ;
      NEWTON = KG METRE / ( SEC ** 2) ;
      { Note that the types in this context are generally units }
   const
      G = 6.67E-11 NEWTON (METRE ** 2) /( KG ** 2) ;
      { Note that we are allowing an extra constant
        decl. so as to allow dimensional constants as well }
   var
      CUMULATIVE,BODYMASS,EARTHMASS : KG ;
      DISTANCE :KM ;
      FORCE : NEWTON ;
      COUNT :integer ;
   begin
      CUMULATIVE := 0.0 KG ;
      COUNT := 0 ;
      while not EOF(INPUT) do
      begin
         READLN(BODYMASS,DISTANCE,FORCE);
         { Read BODYMASS in KGs , DISTANCE in KMs , FORCE in NEWTONs }
         CUMULATIVE := CUMULATIVE + FORCE*DISTANCE*DISTANCE/(G*BODYMASS)
         { Note the automatic conversion of KMs to METREs }
         COUNT := COUNT + 1
      end ;
      EARTHMASS := CUMULATIVE / COUNT ;
      WRITELN('Average mass = ',EARTHMASS,' kgs')
   end.
*****************************************************************
```

CONCLUSIONS ::

The text does not claim an exhaustive treatise on the subject but attempts to open a different line of thought in achieving a greater reliability of Software. The advantages of the above constructs are :

(1) Ease in programming :: Automatic conversion .

(2) Easy to catch errors :: It is now possible to catch dimensional inconsistency in formulae.

(3) Documentation :: The reader of the program knows each data element precisely.

The text also points out that structure alone need not specify the type of a variable . Type should encompass other useful attributes as well.

REFERENCES ::

[1] "Reference Manual for the ADA programming language"
U.S. DoD
[2] Ambiguities and insecurities in PASCAL
J.Welsh , W.J. Sneeringer , C.A.R. Hoare
Software practice and experience 1977
[3] PASCAL ,User Manual and report .
K.Jensen and N.Wirth ,Springer Verlag ,1978