



## TYPE DECLARATIONS

- There are many situations where it is desirable for programmers to define their own types.
  - 1) To enhance clarity and readability.
  - 2) To obtain a better representation of reality.
  - 3) To facilitate automatic checks on values.i.e. to produce more reliable programs.
- Not all languages support the concept of programmer defined types (Ada does, C does not).
- Where supported programmer-defined types are presented using a *type declaration* statement. This *binds* a "new" type definition to a type name (whereas a data declaration statement binds an existing type definition to a data item name).

## TYPE DECLARATIONS IN ADA

- In Ada type declaration statements have the general form:

```
type <TYPE_NAME> is <TYPE
DEFINITION>;
```
- Example type declarations incorporating specific ranges and/or precisions:

```
type SCORE is range 0 .. 100;
type TEMPERATURE is digits 3;
type PERCENTAGE is digits 4 range 0.0 .. 100.0;
```

- Note that in each case the compiler will deduce that the types SCORE and TEMPERATURE are integer types. Similarly the compiler will deduce that the type PERCENTAGE is a floating point type.

## TYPE DECLARATIONS IN ADA CONTINUED

- Once the above types have been declared they can be utilised in the usual manner:

```
A_VALUE: SCORE;

SUMMER: TEMPERATURE := 20;

X_VALU, Y_VALU: PERCENTAGE := 100.0;
```

- If, in the above declarations, an attempt is made to assign a value outside the specified range an error will result.

## BACK TO ACCESS/REFERENCE VALUES

- To declare an access value we use a type declaration and include the reserved word **access** in the definition.

```
with CS_IO; use CS_IO;

procedure EXAMPLE is
  type INTEGERPTR is
    access integer;
  INT_PTR: INTEGERPTR :=
    new integer'(2);
begin
  put(INT_PTR.ALL);
  new_line;
end EXAMPLE;
```

## DISTINCTION BETWEEN POINTERS AND ACCESS VALUES

- Pointers are more versatile than reference values:
  - 1) We can inspect the value (address) of a pointer.
  - 2) We can perform arithmetic on pointer values.
  - 3) We can look at "the value pointed at" (dereferencing).
- Access values only allow "access" to the data item referred to.

## MORE ON TYPE DECLARATIONS

- We have noted that the use of user defined types offers advantages of.
  - 1) To enhance clarity and readability.
  - 2) To obtain a better representation of reality.
  - 3) To facilitate automatic checks on values.i.e. to produce more reliable programs.
- These advantages can be improved upon through a number of other "typing" techniques:
  - 1) Aliasing/renaming of predefined type names.
  - 2) Creating sub-types of existing types
  - 3) Using macro substitutions

## ALIASING (RENAMING) TYPE NAMES

- It is sometimes desirable, given a particular application, to use more descriptive (application dependent) type names for particular types.
- This may offer further advantages of
  - 1) Clarity, and
  - 2) Brevity
- In Ada this is achieved (again) using a type declaration statement:

```
Type REAL is float  
NUMBER1: REAL;
```

- In C a type definition construct is used:

```
typedef float REAL, *REALPTR;  
REAL x;  
REALPTR xPtr;
```

- In both examples an alternative name for the type "float" has been declared.
- In addition, in the C example, a pointer to the new type has also been declared.
- Note that by convention, in C, alternative names are typed in uppercase.

## SUB-TYPES

- Given a particular application it is sometimes useful to characterise a sub-set of values of some other type.
- This is supported by languages such as Ada (but not C).
- In Ada this is achieved using a *subtype* declaration statement:

```
subtype DAY_NUMBER is integer range 1..31;  
Y: DAY_NUMBER := 10;
```

## PREDEFINED SUB-TYPES

- Some imperative languages that support the concept of sub-types also provide some predefined ("built-in") sub-types.
- For examples Ada supports two predefined sub-types of the type INTEGER which are defined as follows:

```
subtype NATURAL is INTEGER range  
0..INTEGER'LAST;  
subtype POSITIVE is INTEGER range  
1..INTEGER'LAST;
```

(note the use of the integer attribute last).

## ADVANTAGES OF SUB-TYPES

Use of subtypes offers similar advantages to those associated with the use of programmer defined types:

- 1) Readability.
- 2) Good representation of reality.
- 3) Error checking.

## WHEN TO USE TYPE AND SUBTYPE DECLARATIONS

Both offer similar advantages (see above), however generally speaking, where a data item is required to display many of the features of the standard INTEGER or FLOAT types a subtype declaration should be used. Type declarations are usually used to describe compound types (e.g. arrays), but more on this later.

## C MACRO SUBSTITUTION

- Macro substitution is where some string replaces every occurrence of an identifier in a program.
- This is supported by C (but not Ada and Pascal).
- To define a macro substitution we include a statement at the beginning of our program of the form:

```
#define <macro name> <macro string>
```

- For example:

```
#define TRUE 1  
#define FALSE 0
```

- The effect of this is that wherever the name TRUE appears in our program this will be replaced with the string 1 on compilation, and FALSE by 0.

## SUMMARY

- 1) Pointers
- 2) Type declarations
- 3) Access/reference values
- 4) Type renaming/aliasing
- 5) Subtypes
- 6) Macro substitution