

# COMP205 IMPERATIVE LANGUAGES

## 3. DATA AND DATA TYPES

- 1) Data Review
- 2) Anonymous Data Items
- 3) Renaming
- 4) Overloading
- 5) Introduction to Data Types
- 6) Basic types
- 7) Range and precision
- 8) Ada/Pascal "attributes"

## ANONYMOUS DATA ITEMS

- Often we use data items in a program without giving them a name, such data items are referred to as *anonymous data items*.



• Example: `4 + (5*6)`

• or: `x + (5*6)`

- The  $(5*6)$  is an anonymous data item.
- Anonymous data items cannot be changed or used again in other parts of a program.

## SUMMARY

*Variable data item.*

*Constant data items.*

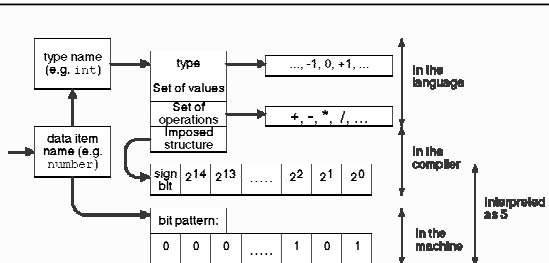
*Uninitialised data items.*

*Anonymous data items.*

## THUS:

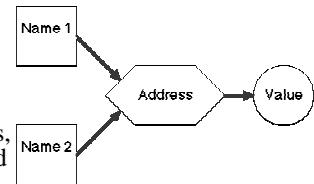
- Data items can be variables or constants or anonymous.
- Variables or constants can be global or local
- Data items are "introduced" using a declaration statement.
- Data items have an "initial" value associated with them through a process known as initialisation (uninitialised data items).
- The value associated with a variable can be changed using an assignment operation.

## RELATIONSHIPS BETWEEN DATA ATTRIBUTES



## RENAMING (ALIASING)

- It is sometimes useful, given a particular application, to rename (alias) particular data items, i.e. provide a second access path to it.



- This is supported by some imperative languages such as Ada (but not C). Example:

```
I: integer = 1;
ONE: integer renames I;
```

It is dangerous to rename variables!

```
I = 1
ONE = 1
I = 10
ONE = 10
I = 100
ONE = 100
```

```
procedure RENAME is
  I: integer := 1;
  ONE: integer renames I;
begin
  put("I = "); put(I);
  put("ONE = "); put(ONE);

  I := I*10;
  put("I = "); put(I);
  put("ONE = "); put(ONE);

  ONE := ONE*10;
  put("I = "); put(I);
  put("ONE = "); put(ONE);
end RENAME;
```

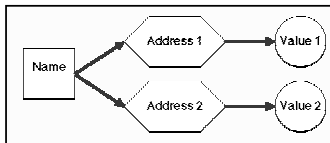
- There are legitimate reasons why we might wish to rename a variable.
- For example:

```
START      : INTEGER := 1 ;
END        : INTEGER := 7 ;
MY_ARRAY   : array (START..END) of
              INTEGER ;
MID_ELEMENT : INTEGER renames
              MY_ARRAY((START+END)/2) ;
```

## OVERLOADING

- Whereas aliasing binds more than one name to one data item, overloading binds two or more data items (possibly of different types) to one name.
- Care must be taken not to cause ambiguity.
- Neither Ada, Pascal or C explicitly support overloading, however:

- 1 In C overloading can be implemented by declaring a compound type known as a *union*
- 2 In Ada the same effect can be contrived using an appropriately defined *record*.



- More generally overloading can best be illustrated by considering the '+' and '-' arithmetic operators.
- Traditionally the operands for these operators are overloaded to allow:

$$5 + 3$$

$$5.6 + 3.2$$

$$5 - 3$$

$$5.6 - 3.2$$

- C also allows mixing of operands (Ada does not) - so called *mixed mode arithmetic*.

$$5 + 3.2$$

$$5.6 + 3$$

$$5.6 - 3$$

$$5 - 3.2$$

## INTRODUCTION TO DATA TYPES

- The type of a data item defines:
  - 1) The nature of the internal representation of the data item.
  - 2) The interpretation placed on the internal representation.
  - 3) As a result of (2) the allowed set of values for the item.
  - 4) The operations that can be performed on it.
- The process of associating a type with a data item is referred to as a data declaration.

## CATEGORISATION OF TYPES

*Pre-defined* and *programmer-defined* types. Pre-defined types are types immediately available to the user. Programmer-defined types are types derived by the programmer.

*Scalar* and *Compound* types. Scalar types define data items that can be expressed as single values. Compound types define data items that comprise several individual values.

*Discrete* and *Non-discrete* types. Discrete types are types for which each value (except its min. and max.) has a predecessor and a successor value. The opposite are non-discrete types.

*Basic* and *Higher Level* types. Basic types are the standard predefined scalar types available for immediate use in any imperative programming language. Higher level types are made up of such basic types and existing higher level types.

## PREDEFINED DATA TYPES

	Ada	C	Pascal	Modula-2
Characters (int. range n..m)	character	char	char	CHAR
Integer (range -n..m)	integer	int	integer	INTEGER
Natural number (range 0..m)				CARDINAL
Real numbers	float	float	real	REAL
Logical type	boolean		boolean	BOOLEAN
Character string	string		text	
Void		void		

## EXAMPLES OF DATA INITIALISATION IN ADA & C

	C	Ada
Same Line	<code>int i = 5, j = 4;</code>	<code>I: integer:= 5; j: integer:= 4;</code>
Multiple declarations	<code>int i = 4, j = 4 int k = i;</code>	<code>I, J: integer:= 4; K: integer:= I;</code>
Using arithmetic expressions	<code>int i = 4*2;</code>	<code>I: integer:= 4*2;</code>
Using existing data items to initialise new data items	<code>int i = 4; int j = i+1;</code>	<code>I: integer:= 4; J: integer:= I+1;</code>
Using functions	<code>int i = timesTwo(2)</code>	

## RANGE AND PRECISION WITH RESPECT TO NUMERIC TYPES

- The term *range* refers to the minimum and maximum value that numeric types can take.
- The term *precision* refers to the number of significant figures with which a number is stored.
- All numeric types have default precisions and/or ranges associated with them.
- Different imperative languages specify ranges and precisions in different ways:
  - 1) Predefined (cannot be influenced by the programmer)
  - 2) Adjectives (e.g. C)
  - 3) Explicit specification (e.g. Ada, Pascal)

## ADJECTIVES

Adjectives are used to specify classes of numeric types.

Some C examples are given below:

Type	Size in bits	Values
Unsigned short int	16	0 to 65535
Signed short int short int short	16	-32768 to 32767
Unsigned long int	32	0 to 4294967296
Signed long int long int long int	32	-2147483648 to 2147483647
float	32	Real number to approx. 6 sig. figs. Real
double	64	Real number to approx. 15 sig. figs.
Long double	128	Real number to approx. 32 sig. figs.

## NOTE ON TWO'S COMPLEMENT NOTATION

- The most common number code for storing integer values.
- Considering the following 8 bit "value box":

-128	64	32	16	8	4	2	1
1	0	0	0	1	1	0	1

- the binary number 10001101 will be equal to:  
 $128+8+4+1 = -115$
- From the above it is evident that any two's complement number with a leading 1 represents a negative number (and vice-versa).

## NOTE ON FIXED & FLOATING POINT STORAGE



- Note: assigning more bits for the exponent will increase the range and decrease the precision.
- Fixed point** = Number of bits available for exponent and mantissa static.
- Floating point** = Number of bits available for exponent and mantissa dynamic.
- Floating point offers the advantage that it can be used to represent both very large and very small numbers without having to store all the zeroes.
- Most modern imperative languages use floating point storage.

## EXPLICIT SPECIFICATION

- Explicit specification of range/precision is the most high level.
- In Ada this is achieved using a general *type declaration* statement:

```
type <type_name> is <type_definition>
```

- Examples:

```
type SCORE is range 0 .. 100;

type TEMPERATURE is digits 4;

type PERCENTAGE is digits 4
  range 0.0 .. 100.0;
```

```
with TEXT_IO; use TEXT_IO;

procedure EXAMPLE is
  type PERCENTAGE is digits 4
    range 0.0 .. 100.0;
  package PERCENTAGE_INOUT is
    new float_io(PERCENTAGE);
  use PERCENTAGE_INOUT;
  N_VAL, M_VAL: PERCENTAGE;

begin
  get(N_VAL);get(M_VAL);
  put(N_VAL);put(M_VAL);
  new_line;
end EXAMPLE;
```

## ADA ATTRIBUTES

- Many imperative languages (Ada, Pascal) allow the programmer to access constants that give information about the default properties of a given type.
- In Ada such constants are referred to as *attributes*.
- Some common (Ada) attributes:

	Attributes
integer	first, last
float	digits, small, large
character	pos, val

```
with CS_IO; use CS_IO;

procedure EXAMPLE is
begin
  put("first = ");
  put(integer'first);
  new_line; put("last = ");
  put(integer'last);
  new_line; put("digits = ");
  put(float'digits);
  new_line; put("small = ");
  put(float'small);
  new_line; put("large = ");
  put(float'large);
  new_line; put("pos(A) = ");
  put(character'pos('A'));
  new_line; put("val(66) = ");
  put(character'val(66));
  new_line;
end EXAMPLE;
```

```
first = -2147483648
last = 2147483647
digits = 15
small = 1.94469227433161E-62
large = 2.57110087081438E+61
pos(A) = 65
val(66) = B
```

## SUMMARY

- 1) Data Review
- 2) Anonymous Data Items
- 3) Renaming
- 4) Overloading
- 5) Introduction to Data Types
- 6) Basic types
- 7) Range and precision
- 8) Ada/Pascal attributes