

## COMP205 IMPERATIVE LANGUAGES

### 2. DATA

- 1) Data Attributes
- 2) Declarations, Assignment and Instantiation
- 3) Global and Local Data
- 4) Variables
- 5) Constants
- 6) Example programs

### DATA - "that which is given"

A data item has a number of attributes:

- 1) An *Address*
- 2) A *Value* which in turn has:
  - a) An *Internal Representation* comprising one or more bits and
  - b) An *interpretation* of that representation.
- 3) A *set of Operations* that may be performed on it.
- 4) A *Name* to tie the above together (also referred to as a label or identifier).

The nature of these attributes is defined by the **type** of the data item.

### ADDRESS

- The **address** (reference) of a data item is the physical location in memory (computer store) of that data item.
- The amount of memory required by a particular data item is dictated by its **type**, e.g. integer or character.

### VALUE

The binary number stored at an address associated with a data item is its **value**.

The required interpretation is dictated by the **type** of the integer.

Consider:

0	1	0	1	1	0	1	0
---	---	---	---	---	---	---	---

This might be interpreted as:

- The Decimal integer `90`
- The Hexadecimal integer `5A`
- The Octal integer `132`
- The ASCII character (capital) `Z`

### OPERATIONS

- The type of a data item also dictates the operations that can be performed on it.
- For example *numeric data types* can have the standard arithmetical operations performed on them while *string data types* cannot.

### NAMES

To do anything useful with a data item all the above must be tied together by giving each data item an identifying name (also referred to as a label or identifier):

- Names are usually made up of alphanumeric characters (and should be descriptive).
- White space is usually not allowed (exception Algol 60).
- Some languages (Ada and C included) allow underscores.
- Some languages restrict the number of characters.
- Some allow any length but treat the first N characters as significant (first 31 in ANSI C).
- Case may be significant: in Algol'60, C and Modula 2 case matters, in Pascal and Ada it does not.
- Cannot use key/reserved words as names.
- Naming conventions.

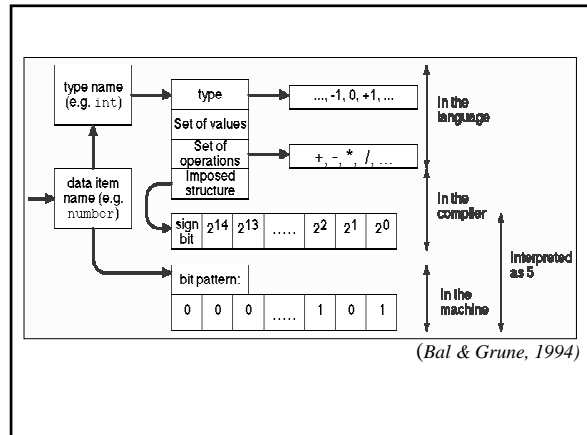
## SUMMARY

- Data Items comprise (a) a name, (b) an address and (c) a value:



(Barron 1977)

- It is important to distinguish between these.
- The type of a data item dictates:
  - The amount of storage required for its internal representation.
  - The interpretation of the internal representation.
  - The operations that can be performed on the item.



## DATA DECLARATIONS, ASSIGNMENT AND INITIALISATION

- A data **declaration** introduces a data item.
- This is typically achieved by associating the item (identified by a name) with a data type.
- Declaration examples (right) in Ada, Pascal and C.
- Assignment** is the process of associating a value with a data item. Usually achieved using an assignment operator.

```
NUMBER : integer;
number : integer;
int number;
```

- Assignment examples in Ada, Pascal and C:

```
NUMBER := 2;
number := 2;
number = 2;
```

- Initialisation** is then the process of assigning an "initial" value to a data item upon declaration (not all imperative languages support this, C and Ada do, Pascal does not).

- Initialisation examples in Ada and C:

```
NUMBER : integer := 2;
int number = 2;
```

## ORDERING OF DECLARATIONS

- In some imperative languages (notably Pascal) declarations must be presented in a certain order, e.g. constants before variables.

## MULTIPLE ASSIGNMENTS

- Many imperative languages (Ada and Pascal, but not C) support the concept of **multiple assignment** where a number of variables can be assigned a value (or values) using a single assignment statement.
- Ada example:

```
NUMBER1, NUMBER2 := 2;
```

- In Pascal we do not have to assign the same value to each variable when using a multiple assignment statement:

```
number1, number2 := 2, 4;
```

- In Pascal we can also write:

```
number1, number2 := number2, number1;
```

Which has the effect of "doing a swap".

## POSITIONING OF DECLARATIONS

- In languages such as C, Ada and Pascal data is declared at the beginning of a function or procedure (other than C global data items).
- Some languages allow declarations to occur anywhere within a procedure or function (for example the object oriented languages C++ and Java).
- Whatever the case remember that:
  - A data item cannot be used until it has been declared.
  - It is good practice to adopt some sensible and systematic procedure for declaring data items, e.g. (if the language supports this) immediately before they are used.

## GLOBAL AND LOCAL DATA

- Data items have associated with them:
  - a) A *life time* - the time-span during the running of a program when they can be said to exist.
  - b) A *visibility* - the parts of the program from where they can be accessed ("seen").
- The nature of the life time and visibility of a data item is dictated by what are referred to as scoping rules.
- In this respect there are two types of data:
  - Global data*
  - Local data*

## VARIABLES

- Given that we can always replace a particular bit pattern with another, the value of a data item can always be changed.
- Imperative languages support such change (destructive assignment).
- Data items that are intended to be used in this way are referred to as variables.
- In most imperative languages (including Ada and C) data items are assumed to be variables by default (in Pascal we must include the key word **var** in the declaration).

## UNINITIALISED VARIABLES

- It is possible to declare a data item that has no value (other than an arbitrary bit pattern).
- Such a data item is referred to as an uninitialised variable.
- Uninitialised variables are dangerous.
- Individual imperative languages treat the phenomena of uninitialised variables differently. Common approaches include:
  - a) Ignore the problem and leave it up to the programmer not to use them.
  - b) Consider their usage to represent an error.
  - c) Allocate an appropriate value to such variables.

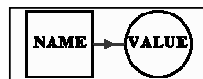


## CONSTANTS

- It is sometimes desirable to define a data item whose value cannot be changed.
- Such data items are referred to as *constants* and are usually indicated by incorporating a predefined key word into the declaration.
- Examples in Ada & C:
 

```
NUMBER: constant := 2;
```

```
const int number = 2;
```
- What we are doing here is telling the Compiler to "flag" the data item as a constant (Software Protection). Theoretically we can still change the bit pattern representing the value.



- In Pascal we declare constants by grouping them together in a `const` section (followed by the `var` section):

```
const
  label = 2;
  pi    = 3.14159;
var
  number integer;
```

- Note: user-defined type declarations (if required) are included in a `type` section located after the `const` section and before the `var` section.

```

#include <stdio.h>

int     GLOBAL_VAR = 0;
const int GLOBAL_CONST = 1;

/* Main function */

int main(void) {
    int     localVar = 2;
    const int localConst = 3;

    printf("%d, %d, %f, %d \n",GLOBAL_VAR,GLOBAL_CONST,
           localVar,localConst);
    GLOBAL_VAR = GLOBAL_CONST + (localVar*localConst);
    printf("%d, %d, %f, %d \n",GLOBAL_VAR,GLOBAL_CONST,
           localVar,localConst);
}

```

```

with TEXT_IO; use TEXT_IO;

procedure TOP_LEVEL is
    package INT_INOUT is new INTEGER_IO(integer);
    use     INT_INOUT;
    GLOBAL_VAR : integer := 0;
    GLOBAL_CONST : constant := 1;
    ----- SECOND LEVEL PROCEDURE -----
    procedure PROC_1 is
        LOCAL_VAR : integer := 2;
        LOCAL_CONST : constant := 3;
    begin
        Output and addition in here
    end PROC_1;
    ----- TOP LEVEL -----
begin
    PROC_1;
end TOP_LEVEL;

```

```

begin
    put(GLOBAL_VAR); put(" ");
    put(GLOBAL_CONST); put(" ");
    put(LOCAL_VAR); put(" ");
    put(LOCAL_CONST); new_line;

    -- Sum

    GLOBAL_VAR := GLOBAL_CONST + (LOCAL_VAR*LOCAL_CONST);

    -- Output

    put(GLOBAL_VAR); put(" ");
    put(GLOBAL_CONST); put(" ");
    put(LOCAL_VAR); put(" ");
    put(LOCAL_CONST);new_line;
end PROC_1;

```

```

program myProg(Output); {Example program}
const
    globalConst = 1;
var
    globalVar : integer;

    procedure procl; {second level procedure}
    const
        localConst = 3;
    var
        localVar : integer;
    begin {Proc_1}
        Output and addition in here
    end; {Proc1}

begin {myProg}
    globalVar := 1;
    procl;
end. {myProg}

```

```

begin {Proc_1}

    write(globalVar);
    write(' ',globalConst);
    write(' ',localVar);
    writeLn(' ',localConst);

    {Sum}
    globalVar := globalConst + (localVar*localConst);

    {Output}
    write(globalVar);
    write(' ',globalConst);
    write(' ',localVar);
    writeLn(' ',localConst);
end; {Proc1}

```

## SUMMARY

- 1) Data Attributes
- 2) Declarations, Assignment and Instantiation
- 3) Global and Local Data
- 4) Variables
- 5) Constants
- 6) Example programs