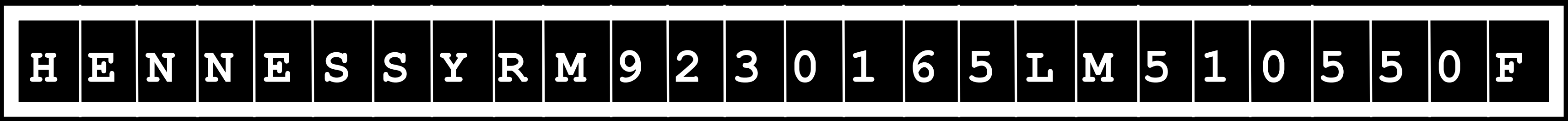
COOBOL Basics
2

Group Items/Records

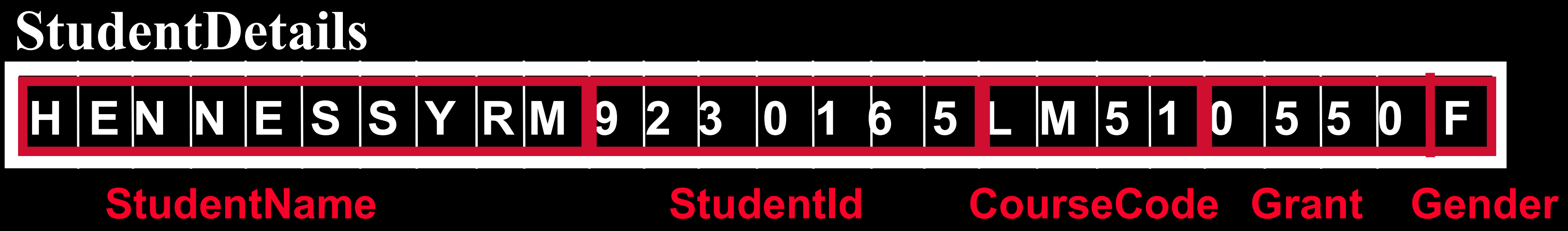
```
WORKING-STORAGE SECTION.
01    StudentDetails             PIC X(26).
```

StudentDetails



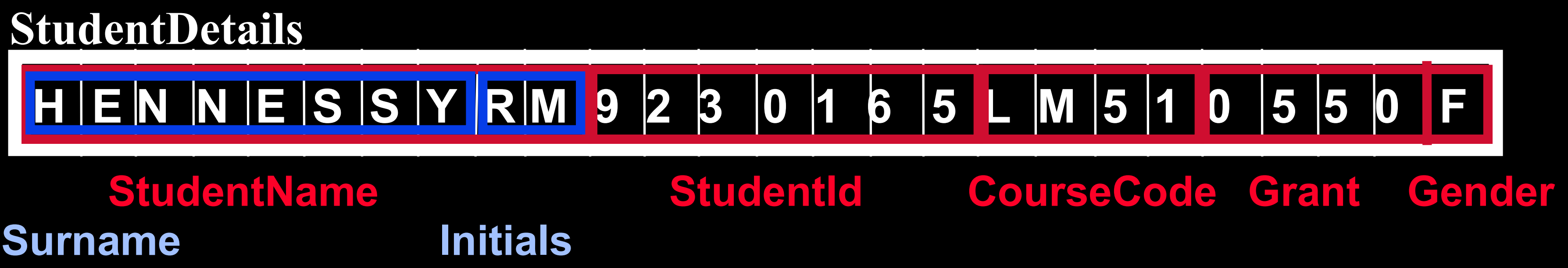
Group Items/Records

```
WORKING-STORAGE SECTION.
01      StudentDetails.
        02      StudentName         PIC X(10).
        02      StudentId           PIC 9(7).
        02      CourseCode          PIC X(4).
        02      Grant               PIC 9(4).
        02      Gender                    PIC X.
```



Group Items/Records

```
WORKING-STORAGE SECTION.
01     StudentDetails.
       02     StudentName.
              03 Surname          PIC X(8).
              03 Initials         PIC XX.
       02     StudentId           PIC 9(7).
       02     CourseCode          PIC X(4).
       02     Grant               PIC 9(4).
       02     Gender              PIC X.
```



LEVEL Numbers express DATA hierarchy

- In COBOL, level numbers are used to decompose a structure into it's constituent parts.
- In this hierarchical structure the higher the level number, the lower the item is in the hierarchy. At the lowest level the data is completely atomic.
- The level numbers 01 through 49 are general level numbers but there are also special level numbers such as 66, 77 and 88.
- In a hierarchical data description what is important is the relationship of the level numbers to one another, not the actual level numbers used.

LEVEL Numbers express DATA hierarchy

- In COBOL, level numbers are used to decompose a structure into it’s constituent parts.
- In this hierarchical structure the higher the level number, the lower the item is in the hierarchy. At the lowest level the data is completely atomic.
- The level numbers 01 through 49 are general level numbers but there are also special level numbers such as 66, 77 and 88.
- In a hierarchical data description what is important is the relationship of the level numbers to one another, not the actual level numbers used

```
01 StudentDetails.
   02 StudentName.
      03 Surname      PIC X(8).
      03 Initials     PIC XX.
   02 StudentId       PIC 9(7).
   02 CourseCode      PIC X(4).
   02 Grant           PIC 9(4).
   02 Gender          PIC X.
```

=

```
01 StudentDetails.
   05 StudentName.
      10 Surname      PIC X(8).
      10 Initials     PIC XX.
   05 StudentId       PIC 9(7).
   05 CourseCode      PIC X(4).
   05 Grant           PIC 9(4).
   05 Gender          PIC X.
```

Group and elementary items.

- In COBOL the term "group item" is used to describe a data item which has been further subdivided.
  - A Group item is declared using a level number and a data name. It cannot have a picture clause.
  - Where a group item is the highest item in a data hierarchy it is referred to as a record and uses the level number 01.
- The term "elementary item" is used to describe data items which are atomic; that is, not further subdivided.
- An elementary item declaration consists of;
  - a level number,
  - a data name
  - picture clause.

  An elementary item must have a picture clause.
- Every group or elementary item declaration must be followed by a full stop.

PICTUREs for Group Items

- Picture clauses are NOT specified for 'group' data items because the size a group item is the sum of the sizes of its subordinate, elementary items and its type is always assumed to be PIC X.
- The type of a group items is always assumed to be PIC X because group items may have several different data items and types subordinate to them.
- An X picture is the only one which could support such collections.

Assignment in COBOL

- In “strongly typed” languages like Modula-2, Pascal or ADA the assignment operation is simple because assignment is only allowed between data items with compatible types.
- The simplicity of assignment in these languages is achieved at the “cost” of having a large number of data types.
- In COBOL there are basically only three data types,
  - Alphabetic (PIC A)
  - Alphanumeric (PIC X)
  - Numeric (PIC 9)
- But this simplicity is achieved only at the cost of having a very complex assignment statement.
- In COBOL assignment is achieved using the **MOVE** verb.

The MOVE Verb

- The MOVE copies data from the source identifier or literal to one or more destination identifiers.
- The source and destination identifiers can be group or elementary data items.
- When the destination item is alphanumeric or alphabetic (PIC X or A) data is copied into the destination area from left to right with space filling or truncation on the right.
- When data is MOVEd into an item the contents of the item are completely replaced. If the source data is too small to fill the destination item entirely the remaining area is zero or space filled.

MOVEing Data

```
MOVE "RYAN" TO Surname.
MOVE "FITZPATRICK" TO Surname.
```

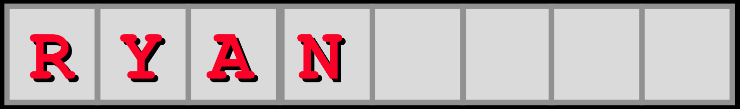
```
01 Surname   PIC X(8).
```

C	O	U	G	H	L	A	N

MOVEing Data

```
MOVE "RYAN" TO Surname.
MOVE "FITZPATRICK" TO Surname.
```

```
01 Surname   PIC X(8).
```



MOVEing Data

```
MOVE “RYAN” TO Surname.
MOVE “FITZPATRICK” TO Surname.
```

```
01 Surname   PIC X(8).
```

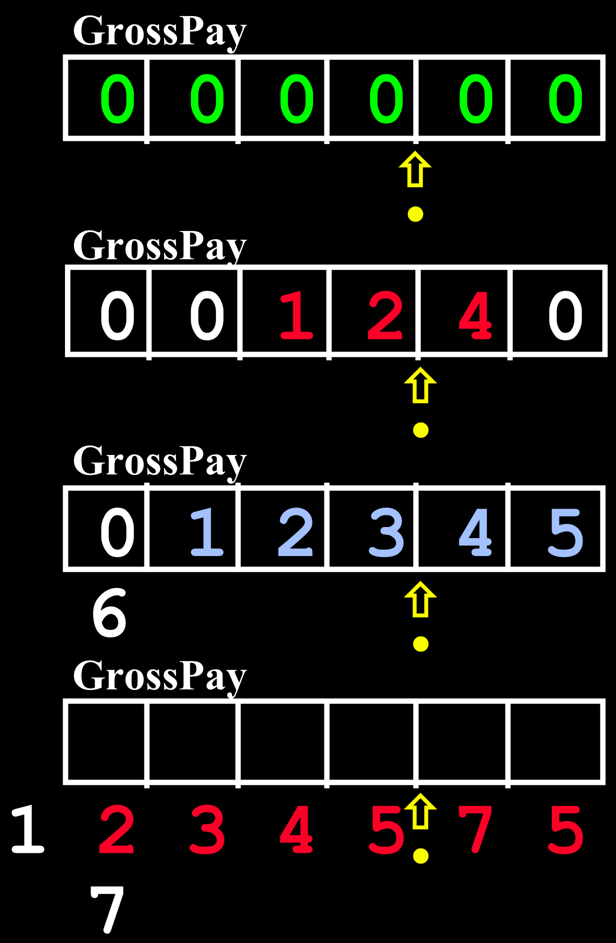
F	I	T	Z	P	A	T	R

I C K

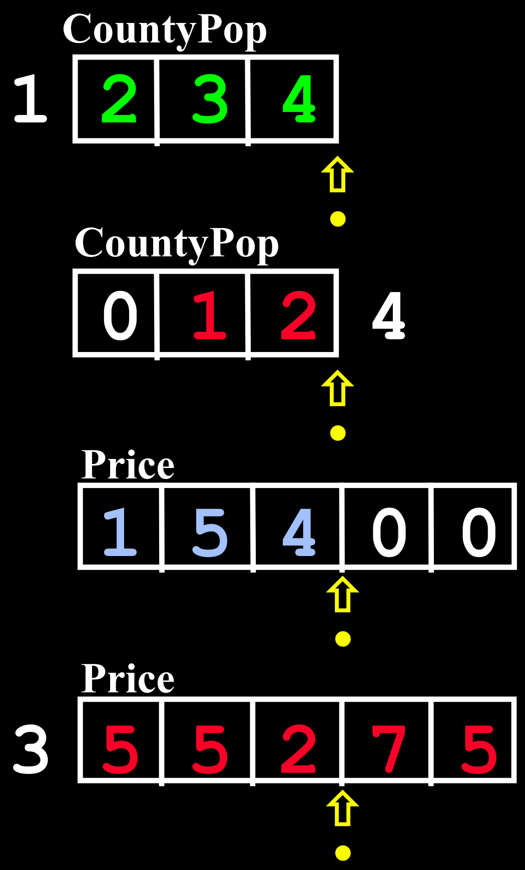
MOVEing to a numeric item.

- When the destination item is numeric, or edited numeric, then data is aligned along the decimal point with zero filling or truncation as necessary.
- When the decimal point is not explicitly specified in either the source or destination items, the item is treated as if it had an assumed decimal point immediately after its rightmost character.

```
01 GrossPay            PIC 9(4)V99.

MOVE ZEROS TO GrossPay.

MOVE 12.4 TO GrossPay.

MOVE 123.456 TO GrossPay.

MOVE 12345.757 TO GrossPay.
```

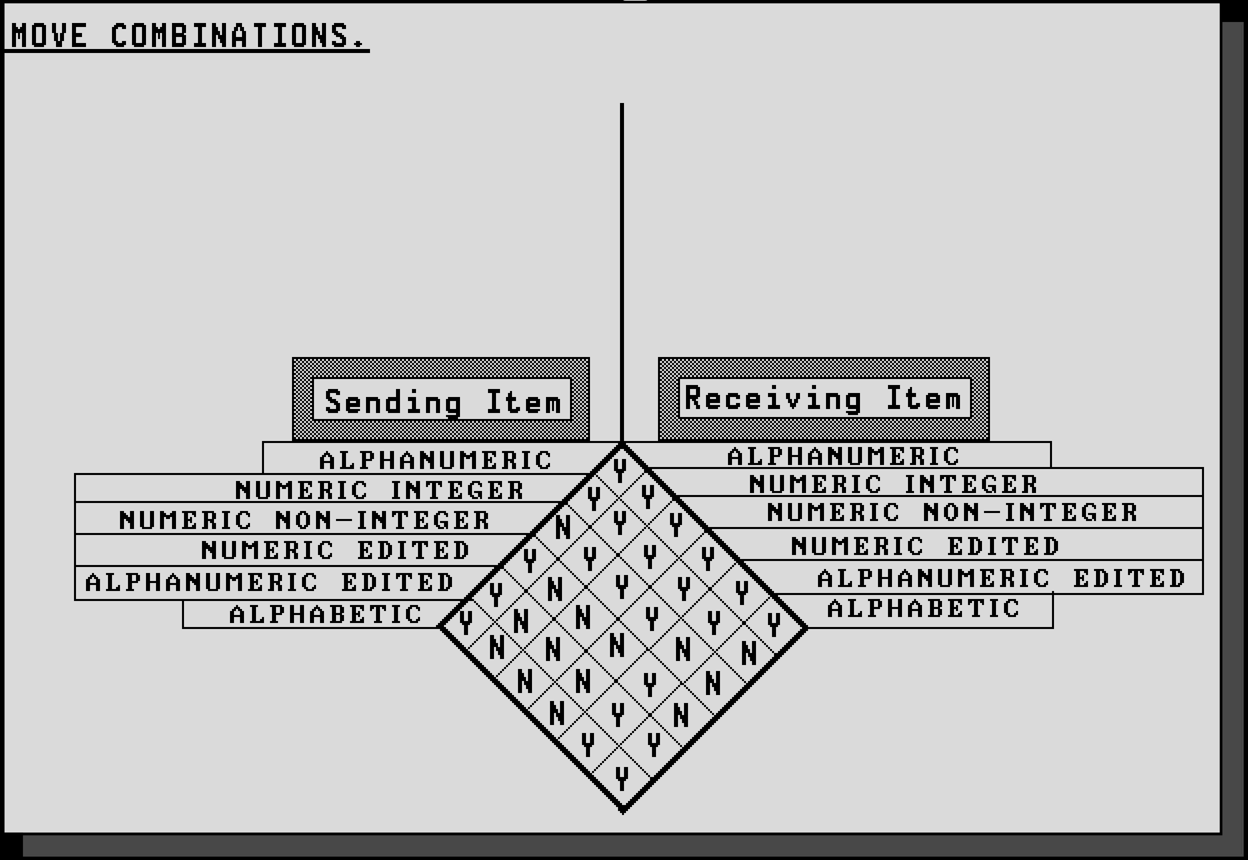


```
01 CountyPop              PIC 999.
01 Price                  PIC 999V99.

MOVE 1234 TO CountyPop.

MOVE 12.4 TO CountyPop.

MOVE 154 TO Price.

MOVE 3552.75 TO Price.
```



Legal MOVEs

Certain combinations of sending and receiving data types are not permitted (even by COBOL).



MOVE COMBINATIONS.

Sending Item \ Receiving Item	ALPHANUMERIC	NUMERIC INTEGER	NUMERIC NON-INTEGER	NUMERIC EDITED	ALPHANUMERIC EDITED	ALPHABETIC
ALPHANUMERIC	Y	Y	Y	Y	Y	Y
NUMERIC INTEGER	Y	Y	Y	Y	Y	N
NUMERIC NON-INTEGER	N	Y	Y	Y	N	N
NUMERIC EDITED	Y	N	N	N	Y	N
ALPHANUMERIC EDITED	Y	N	N	N	Y	Y
ALPHABETIC	Y	N	N	N	Y	Y

The DISPLAY Verb

- From time to time it may be useful to display messages and data values on the screen.
- A simple DISPLAY statement can be used to achieve this.
- A single DISPLAY can be used to display several data items or literals or any combination of these.
- The WITH NO ADVANCING clause suppresses the carriage return/line feed.

The ACCEPT verb

```
01 CurrentDate            PIC 9(6).
* YYMMDD

01 DayOfYear              PIC 9(5).
* YYDDD

01 Day0fWeek              PIC 9.
* D (1=Monday)

01 CurrentTime            PIC 9(8).
* HHMMSSss    s = S/100
```

Run of Accept and Display program

```
Enter student details using template below
NNNNNNNNNSSSSSSSCCCCGGGGS
COUGHLANMS9476532LM511245M
Name is MS COUGHLAN
Date is 24 01 94
Today is day 024 of the year
The time is 22:23
```

```
    $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  Michael Coughlan.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentName.
       03 Surname        PIC X(8).
       03 Initials       PIC XX.
   02  StudentId         PIC 9(7).
   02  CourseCode        PIC X(4).
   02  Grant             PIC 9(4).
   02  Gender            PIC X.

01 CurrentDate.
   02  CurrentYear       PIC 99.
   02  CurrentMonth      PIC 99.
   02  CurrentDay        PIC 99.

01 DayOfYear.
   02  FILLER            PIC 99.
   02  YearDay           PIC 9(3).

01 CurrentTime.
   02  CurrentHour       PIC 99.
   02  CurrentMinute     PIC 99.
   02  FILLER            PIC 9(4).
```

```
PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter student details using template below".
    DISPLAY "NNNNNNNNNSSSSSSSCCCCGGGGS                ".
    ACCEPT  StudentDetails.
    ACCEPT  CurrentDate FROM DATE.
    ACCEPT  DayOfYear FROM DAY.
    ACCEPT  CurrentTime FROM TIME.
    DISPLAY "Name is ", Initials SPACE Surname.
    DISPLAY "Date is " CurrentDay SPACE CurrentMonth SPACE CurrentYear.
    DISPLAY "Today is day " YearDay " of the year".
    DISPLAY "The time is " CurrentHour ":" CurrentMinute.
    STOP RUN.
```