COBOL program structure

* **Sections** are the logical subdivision of program logic. A section is a collection of paragraphs.
* **Paragraphs** are the subdivision of a section or division. It is either a user-defined or a predefined name followed by a period, and consists of zero or more sentences/entries.
* **Sentences** are the combination of one or more statements. Sentences appear only in the Procedure division. A sentence must end with a period.
* **Statements** are meaningful COBOL statements that perform some processing.
* **Characters** are the lowest in the hierarchy and cannot be divisible.

*Divisions*

* **Identification Division**

It is the first and only mandatory division of every COBOL program. The programmer and the compiler use this division to identify the program. In this division, PROGRAM-ID is the only mandatory paragraph. PROGRAM-ID specifies the program name that can consist 1 to 30 characters.

* **Environment Division**

Environment division is used to specify input and output files to the program. It consists of two sections −

* **Configuration section** provides information about the system on which the program is written and executed.
* **Input-Output section** provides information about the files to be used in the program. It consists of two paragraphs −
* **File control** − Provides information of external data sets used in the program.
* **I-O control** − Provides information of files used in the program.

Syntax

*ENVIRONMENT DIVISION.*

*CONFIGURATION SECTION.*

*SOURCE-COMPUTER. XXX-ZOS.*

*OBJECT-COMPUTER. XXX-ZOS.*

*INPUT-OUTPUT SECTION.*

*FILE-CONTROL.*

*SELECT FILEN ASSIGN TO DDNAME*

*ORGANIZATION IS SEQUENTIAL.*

* **Data Division**

Data division is used to define the variables used in the program. It consists of four sections

* **File section** is used to define the record structure of the file.
* **Working-Storage** section is used to declare temporary variables and file structures which are used in the program.
* **Local-Storage** section is similar to Working-Storage section. The only difference is that the variables will be allocated and initialized every time a program starts execution.
* **Linkage section** is used to describe the data names that are received from an external program.
* **Procedure Division**

Procedure division is used to include the logic of the program. It consists of executable statements using variables defined in the data division. In this division, paragraph and section names are user-defined.

There must be at least one statement in the procedure division. The last statement to end the execution in this division is either **STOP RUN** which is used in the calling programs or **EXIT PROGRAM** which is used in the called programs.

Data layout

COBOL layout is the description of use of each field and the values present in it.

* Redefines Clause
* Renames Clause
* Usage Clause
* Copybooks
* **Redefines Clause**

Redefines clause is used to define a storage with different data description. If one or more data items are not used simultaneously, then the same storage can be utilized for another data item. So the same storage can be referred with different data items.

*Syntax*

*01 WS-OLD PIC X(10).*

*01 WS-NEW1 REDEFINES WS-OLD PIC 9(8).*

*01 WS-NEW2 REDEFINES WS-OLD PIC A(10).*

* WS-OLD is Redefined Item
* WS-NEW1 and WS-NEW2 are Redefining Item
* Level numbers of redefined item and redefining item must be the same and it cannot be 66 or 88 level number. Do not use VALUE clause with a redefining item. In File Section, do not use a redefines clause with 01 level number. Redefines definition must be the next data description you want to redefine. A redefining item will always have the same value as a redefined item.

Example

*IDENTIFICATION DIVISION.*

*PROGRAM-ID. HELLO.*

*DATA DIVISION.*

*WORKING-STORAGE SECTION.*

*01 WS-DESCRIPTION.*

*05 WS-DATE1 VALUE '20140831'.*

*10 WS-YEAR PIC X(4).*

*10 WS-MONTH PIC X(2).*

*10 WS-DATE PIC X(2).*

*05 WS-DATE2 REDEFINES WS-DATE1 PIC 9(8).*

*PROCEDURE DIVISION.*

*DISPLAY "WS-DATE1 : "WS-DATE1.*

*DISPLAY "WS-DATE2 : "WS-DATE2.*

*STOP RUN.*

*Output*

*WS-DATE1 : 20140831*

*WS-DATE2 : 20140831*

* **Renames Clause**

Renames clause is used to give different names to existing data items. It is used to re-group the data names and give a new name to them. The new data names can rename across groups or elementary items. Level number 66 is reserved for renames.

Syntax

*01 WS-OLD.*

*10 WS-A PIC 9(12).*

*10 WS-B PIC X(20).*

*10 WS-C PIC A(25).*

*10 WS-D PIC X(12).*

*66 WS-NEW RENAMES WS-A THRU WS-C.*

* Renaming is possible at same level only. In the above example, WS-A, WS-B, and WS-C are at the same level. Renames definition must be the next data description you want to rename. Do not use Renames with 01, 77, or 88 level number. The data names used for renames must come in sequence. Data items with occur clause cannot be renamed.

Example

*IDENTIFICATION DIVISION.*

*PROGRAM-ID. HELLO.*

*DATA DIVISION.*

*WORKING-STORAGE SECTION.*

*01 WS-DESCRIPTION.*

*05 WS-NUM.*

*10 WS-NUM1 PIC 9(2) VALUE 20.*

*10 WS-NUM2 PIC 9(2) VALUE 56.*

*05 WS-CHAR.*

*10 WS-CHAR1 PIC X(2) VALUE 'AA'.*

*10 WS-CHAR2 PIC X(2) VALUE 'BB'.*

*66 WS-RENAME RENAMES WS-NUM2 THRU WS-CHAR2.*

*PROCEDURE DIVISION.*

*DISPLAY "WS-RENAME : " WS-RENAME.*

*STOP RUN.*

*Output*

WS-RENAME : 56AABB

* **Usage Clause**

Usage clause specifies the operating system in which the format data is stored. It cannot be used with level numbers 66 or 88. If usage clause is specified on a group, then all the elementary items will have the same usage clause. The different options available with Usage clause are as follows

* **Display**

Data item is stored in ASCII format and each character will take 1 byte. It is default usage.

Example: 01 WS-NUM PIC S9(5)V9(3) USAGE IS DISPLAY.

* It requires 8 bytes as sign and decimal doesn't require any byte.

01 WS-NUM PIC 9(5) USAGE IS DISPLAY.

* It requires 5 bytes as sign.
* **COMPUTATIONAL / COMP**
* Data item is stored in binary format. Here, data items must be integer.

The following example calculates the number of bytes required −

* 01 WS-NUM PIC S9(n) USAGE IS COMP.

*If 'n' = 1 to 4, it takes 2 bytes.*

*If 'n' = 5 to 9, it takes 4 bytes.*

*If 'n' = 10 to 18, it takes 8 bytes.*

* COMP-1

Data item is similar to Real or Float and is represented as a single precision floating point number. Internally, data is stored in hexadecimal format. COMP-1 does not accept PIC clause. Here 1 word is equal to 4 bytes.

* COMP-2

Data item is similar to Long or Double and is represented as double precision floating point number. Internally, data is stored in hexadecimal format. COMP-2 does not specify PIC clause. Here 2 word is equal to 8 bytes.

* COMP-3

Data item is stored in packed decimal format. Each digit occupies half a byte (1 nibble) and the sign is stored at the rightmost nibble.

The following example calculates the number of bytes required −

*01 WS-NUM PIC 9(n) USAGE IS COMP.*

*Number of bytes = n/2 (If n is even)*

*Number of bytes = n/2 + 1(If n is odd, consider only integer part)*

*01 WS-NUM PIC 9(4) USAGE IS COMP-3 VALUE 21.*

*It requires 3 bytes of storage as each digit occupies half a byte.*

*01 WS-NUM PIC 9(5) USAGE IS COMP-3 VALUE 21.*

*It requires 3 bytes of storage as each digit occupies half a byte.*

* **Copybooks**

A COBOL copybook is a selection of code that defines data structures. If a particular data structure is used in many programs, then instead of writing the same data structure again, we can use copybooks. We use the COPY statement to include a copybook in a program. COPY statement is used in the WorkingStorage Section.

example

*DATA DIVISION.*

*WORKING-STORAGE SECTION.*

*COPY ABC.*

String handling

* ***Inspect***

Inspect verb is used to count or replace the characters in a string. String operations can be performed on alphanumeric, numeric, or alphabetic values. Inspect operations are performed from left to right. The options used for the string operations are as follows −

* ***Tallying***

Tallying option is used to count the string characters.

*Syntax*

*INSPECT input-string*

*TALLYING output-count FOR ALL CHARACTERS*

* input-string − The string whose characters are to be counted.
* output-count − Data item to hold the count of characters.

*Example*

*IDENTIFICATION DIVISION.*

*PROGRAM-ID. HELLO*

*DATA DIVISION.*

*WORKING-STORAGE SECTION.*

*01 WS-CNT1 PIC 9(2) VALUE 0.*

*01 WS-CNT2 PIC 9(2) VALUE 0.*

*01 WS-STRING PIC X(15) VALUE 'ABCDACDADEAAAFF'.*

*PROCEDURE DIVISION.*

*INSPECT WS-STRING TALLYING WS-CNT1 FOR CHARACTER.*

*DISPLAY "WS-CNT1 : "WS-CNT1.*

*INSPECT WS-STRING TALLYING WS-CNT2 FOR ALL 'A'.*

*DISPLAY "WS-CNT2 : "WS-CNT2*

*STOP RUN.*

Output

WS-CNT1 : 15

WS-CNT2 : 06

1) Use 1 variable for full name and display it in 3 different variables as first name ,middle name and last name.

IDENTIFICATION DIVISION.

PROGRAM-ID. HELLO-WORLD.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-FULLNAME PIC X(50) VALUE 'ABC DEF XYZ'.

01 WS-FNAME PIC X(15).

01 WS-MNAME PIC X(15).

01 WS-LNAME PIC X(15).

PROCEDURE DIVISION.

UNSTRING WS-FULLNAME DELIMITED BY SPACE

INTO WS-FNAME

WS-MNAME

WS-LNAME

END-UNSTRING

DISPLAY 'FIRST NAME:' WS-FNAME.

DISPLAY 'MIDDLE NAME:' WS-MNAME.

DISPLAY 'LAST NAME:' WS-LNAME.

STOP RUN.

Output:

FIRST NAME : ABC

MIDDLE NAME : DEF

LAST NAME : XYZ

**2) Use 3 different variables (date,month,year) and display in the format of DD/MM/YYYY**

**IDENTIFICATION DIVISION.**

**PROGRAM-ID. HELLO-WORLD.**

**DATA DIVISION.**

**WORKING-STORAGE SECTION.**

**01 WS-DATE PIC X(10) VALUE '10'.**

**01 MONTH PIC X(10) VALUE '02'.**

**01 YEAR PIC X(10) VALUE '2012'.**

**01 WS-FULL-DATE PIC X(25).**

**PROCEDURE DIVISION.**

**STRING WS-DATE DELIMITED BY SIZE**

**'/' DELIMITED BY SIZE**

**MONTH DELIMITED BY SIZE**

**'/' DELIMITED BY SIZE**

**YEAR DELIMITED BY SIZE**

**INTO WS-FULL-DATE**

**END-STRING**

**DISPLAY WS-FULL-DATE.**

**STOP RUN.**

**Output:**

**10/02/2012**

**3) Reverse String**

**IDENTIFICATION DIVISION.**

**PROGRAM-ID. HELLO-WORLD.**

**data division.**

**WORKING-STORAGE SECTION.**

**01 EINA01 PIC X(20) VALUE 'ABCDEFGH0'.**

**01 WORTTXT1 PIC X(20) VALUE SPACES.**

**01 AUSA01 PIC X(20) VALUE SPACES.**

**01 B PIC X(20) VALUE SPACES.**

**01 LOO PIC 9(2) VALUE 0.**

**01 LOP PIC 9(2) VALUE 0.**

**PROCEDURE DIVISION.**

**MOVE 20 TO LOO.**

**MOVE 1 TO LOP.**

**MOVE EINA01 TO WORTTXT1.**

**PERFORM 20 TIMES**

**MOVE WORTTXT1(LOP:1) TO B(LOO:1)**

**SUBTRACT 1 FROM LOO**

**ADD 1 TO LOP**

**END-PERFORM.**

**MOVE B TO AUSA01.**

**DISPLAY AUSA01.**

**STOP RUN.**

**Output:**

**DHGFEDCBA**

**File handling**

**Types of files**

File organization indicates how the records are organized in a file. There are different types of organizations for files so as to increase their efficiency of accessing the records. Following are the types of file organization schemes −

* Sequential file organization
* Indexed sequential file organization
* Relative file organization

***Sequential file organization***

A sequential file consists of records that are stored and accessed in sequential order. Following are the key attributes of sequential file organization −

* Records can be read in sequential order. For reading the 10th record, all the previous 9 records should be read.
* Records are written in sequential order. A new record cannot be inserted in between. A new record is always inserted at the end of the file.

***Indexed sequential file organization***

An indexed sequential file consists of records that can be accessed sequentially. Direct access is also possible. It consists of two parts −

* **Data File** contains records in sequential scheme.
* **Index File** contains the primary key and its address in the data file.

Following are the key attributes of sequential file organization −

* Records can be read in sequential order just like in sequential file organization.
* Records can be accessed randomly if the primary key is known. Index file is used to get the address of a record and then the record is fetched from the data file.

***Relative File Organization***

A relative file consists of records ordered by their **relative address**. Following are the key attributes of relative file organization −

* Records can be read in sequential order just like in sequential and indexed file organization.
* Records can be accessed using relative key. Relative key represents the record’s location relative to the address of the start of the file.

**Access Modes**

**Sequential Access**

When the access mode is sequential, the method of record retrieval changes as per the selected file organization.

* For **sequential files**, records are accessed in the same order in which they were inserted.
* For **indexed files**, the parameter used to fetch the records are the record key values.
* For **relative files**, relative record keys are used to retrieve the records.

**Syntax**

Following is the syntax of sequential access mode −

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT file-name ASSIGN TO dd-name

ORGANIZATION IS **SEQUENTIAL**

ACCESS MODE IS **SEQUENTIAL**

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT file-name ASSIGN TO dd-name

ORGANIZATION IS **INDEXED**

ACCESS MODE IS **SEQUENTIAL**

RECORD KEY IS rec-key1

ALTERNATE RECORD KEY IS rec-key2

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT file-name ASSIGN TO dd-name

ORGANIZATION IS **RELATIVE**

ACCESS MODE IS **SEQUENTIAL**

RELATIVE KEY IS rec-key1

## **Random Access**

When the access mode is RANDOM, the method of record retrieval changes as per the selected file organization.

* For **indexed files**, records are accessed according to the value placed in a key field which can be primary or alternate key. There can be one or more alternate indexes.
* For **relative files** , records are retrieved through relative record keys.

**Syntax**

Following is the syntax of random access mode −

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT file-name ASSIGN TO dd-name

ORGANIZATION IS **INDEXED**

ACCESS MODE IS **RANDOM**

RECORD KEY IS rec-key1

ALTERNATE RECORD KEY IS rec-key2

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT file-name ASSIGN TO dd-name

ORGANIZATION IS **RELATIVE**

ACCESS MODE IS **RANDOM**

RELATIVE KEY IS rec-key1

## **Dynamic Access**

Dynamic access supports both sequential and random access in the same program. With dynamic access, one file definition is used to perform both sequential and random processing like accessing some records in sequential order and other records by their keys.

With relative and indexed files, the dynamic access mode allows you to switch back and forth between sequential access mode and random access mode while reading a file by using the NEXT phrase on the READ statement.

**Syntax**

Following is the syntax of dynamic access mode −

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT file-name ASSIGN TO dd-name

ORGANIZATION IS **INDEXED**

ACCESS MODE IS **DYNAMIC**

RECORD KEY IS rec-key1

ALTERNATE RECORD KEY IS rec-key2

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT file-name ASSIGN TO dd-name

ORGANIZATION IS **RELATIVE**

ACCESS MODE IS **DYNAMIC**

RELATIVE KEY IS rec-key1

**File handling Verbs**

**1) Open Verb**

* Open Verb is the beginning operation of a file. Without opening a file it is not possible to perform any processing operations like read, write or re-write etc.
* The variables in the file structure are available for processing only after opening a file.
* In **open verb,** FILE-STATUS variable is updated after each file operation.
* **1. Input –** This mode is used for existing files. File is used to read only.  
  **2. Output –** It is used for writing or creating a new file, i.e to insert a record in the file.   
  **3. Extend –** To append records in a sequential file extend mode is used, i.e records are inserted at the end. This mode cannot be used when the file access mode is Random or Dynamic.  
  **4. I-O –** Input-Output mode provides read and rewrite the record of a file.

**Syntax:**

OPEN “mode” file-name

**2) Read verb**

* Read Verb is used to read the file records. Only one record is read into the file structure at each read verb.
* The file should be open in INPUT or I-O mode for performing read operation.
* The file pointer is incremented at each read statement, hence the consecutive records are read.

**Syntax:**

READ file-name NEXT RECORD INTO ws-file-structure  
AT END DISPLAY 'End of File'  
NOT AT END DISPLAY 'Record Details:' ws-file-structure  
END-READ.

**3) Write Verb**

* Write verb is used to insert the records in a file. This verb is used when a file is opened in **OUTPUT or EXTEND** mode.
* Write statement is used with the **FROM** option to directly write records from the working storage variables. **FROM** is an optional clause.

**Syntax:**

WRITE record-buffer  [FROM ws-file-structure]  
END-WRITE

**4) Rewrite Verb**

* REWRITE verb is used for updating the records in a file.
* The file must be open in I-O mode for rewrite operations.
* It is used after the successful read operation.

**Syntax:**

REWRITE record-buffer [FROM ws-file-structure]

END-REWRITE.

**5) Delete verb**

* Delete verb is executed when the file is indexed and relative.
* To delete the record file, it should remain open in I-O mode. The record cannot be deleted in sequential file organization.
* In random access mode, first specify the record key, to perform the delete operation.

**Syntax:**

DELETE file-name RECORD

INVALID KEY DISPLAY 'Invalid key'

NOT INVALID KEY DISPLAY 'Record deleted'

END-DELETE.

**6) Start verb**

* Start verb is performed only on the indexed and relative files.
* This verb is used to place the file pointer at a specific record.
* The access mode should be sequential or dynamic in start verb.
* In start verb, a file should be open in I-O or input mode.

**Syntax:**

START file-name KEY IS [=, >, <, NOT, <= or >=] rec-key

INVALID KEY DISPLAY 'Invalid Key'

NOT INVALID KEY DISPLAY 'File Pointer Updated'

END-START.

**7) Close verb**

* Close verb is used to close a file.
* After executing a close operation the variables in the file structure will not be available for processing and link is lost between the program and file.

**Syntax:**

CLOSE file-name

**ADD Verb:**

ADD A B C TO D. 🡪 D=D+A+B+C

ADD A, B GIVING C, D, E. 🡪 C=D=E=A+B

**SUBTRACT Verb:**

SUBTRACT A, B FROM C. 🡪 C=C-A-B

SUBTRACT A, B FROM C GIVING D. 🡪 D= C-A-B

SUBTRACT 15 FROM A B. 🡪 A=A-15 & B=B-15

**MULTIPLY Verb:**

MULTIPLY A BY B. 🡪 B=A\*B

MULTIPLY A BY B GIVING C. 🡪 C=A\*B

MULTIPLY A BY B C D. 🡪 B=A\*B & C=C\*A & D=D\*A

MULTIPLY A BY B C GIVING D E. 🡪 D=A\*B & E=A\*C

**DIVIDE Verb:**

DIVIDE 5 INTO A. 🡪 A=A/5

DIVIDE 5 INTO A GIVING B. 🡪 B=A/5

DIVIDE 5 INTO A GIVING B C. 🡪 B=A/5 & C=A/5

DIVIDE 5 INTO A B GIVING C D. 🡪 C=A/5 & D=B/5

DIVIDE A INTO B GIVING C REMAINDER D. 🡪 C=B/A & D=B%A

DIVIDE A BY 3 GIVING C. 🡪 C=A/3

**COBOL SUBROUTINES**

* Cobol sub routine is a program which can be compiled independently but can not be executed independently.
* It is a set of instructions that performs a specific task for a main routine, requiring direction back to the proper place in the main routine on completion of the task.
* COBOL subroutine has PROCEDURE DIVISION with USING Phrase.

PROCEDURE DIVISION USING data-name-1, data-name-2,…

* Data-name-1, data-name-2 are connected with corresponding data names in the Calling program which are defined in its LINKAGE SECTION.
* Upon completion of execution of subroutine, the control must be transferred to the Calling program using EXIT PROGRAM statement.
* The operands of the USING Phrase in the CALL statement are called Actual Parameters.
* The data names of USING Phrase in the PROCEDURE DIVISION are called Formal Parameters.
* The number of actual and formal parameters must be same and the size, class and usage of the corresponding actual and formal parameters must be identical.
* The process of establishing connection between actual and formal parameters is called parameter passing.
* Call by reference made to the parameter either in calling or called program will make the reference to the same data area in the memory.
* Call by Content or Call by Value will merely affects the copy of the actual parameters and not the original parameter. So in case changes should not affect to the actual parameter, call by Content is used.

**Called program constraints -**

* **Linkage section** must be defined in the called program. It consists of data elements passed in the program. The data items should not have Value clause. PIC clause must be compatible with the variables passed through the calling program.
* **Procedure division using** has a list of variables passed from the calling program and the order must be same as mentioned in the Call verb.
* **Exit program** statement is used in the called program to transfer the control back. It must be the last statement in the called program.

Example:

Main Program-

IDENTIFICATION DIVISION.

PROGRAM-ID. MAIN.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-STUDENT-ID PIC 9(4) VALUE 1000.

01 WS-STUDENT-NAME PIC A(15) VALUE 'Tim'.

PROCEDURE DIVISION.

CALL 'UTIL' USING WS-STUDENT-ID, WS-STUDENT-NAME.

DISPLAY 'Student Id : ' WS-STUDENT-ID

DISPLAY 'Student Name : ' WS-STUDENT-NAME

STOP RUN.

Sub Program-

IDENTIFICATION DIVISION.

PROGRAM-ID. UTIL.

DATA DIVISION.

LINKAGE SECTION.

01 LS-STUDENT-ID PIC 9(4).

01 LS-STUDENT-NAME PIC A(15).

PROCEDURE DIVISION USING LS-STUDENT-ID, LS-STUDENT-NAME.

DISPLAY 'In Called Program'.

MOVE 1111 TO LS-STUDENT-ID.

EXIT PROGRAM.

When you compile and execute the above program, it produces the following result −

In Called Program

Student Id : 1111

Student Name : Tim

**LOOPING STATEMENTS**

The loop statements used in COBOL are −

· Perform Thru

· Perform Until

· Perform Times

· Perform Varying

**Perform Thru**

Perform Thru is used to execute a series of paragraph by giving the first and last paragraph names in the sequence. After executing the last paragraph, the control is returned back.

**Syntax**

PERFORM

DISPLAY 'HELLO WORLD'

END-PERFORM.

**Example**

IDENTIFICATION DIVISION.

PROGRAM-ID. HELLO.

PROCEDURE DIVISION.

A-PARA.

PERFORM DISPLAY 'IN A-PARA'

END-PERFORM.

PERFORM C-PARA THRU E-PARA.

B-PARA.

DISPLAY 'IN B-PARA'.

STOP RUN.

C-PARA.

DISPLAY 'IN C-PARA'.

D-PARA.

DISPLAY 'IN D-PARA'.

E-PARA.

DISPLAY 'IN E-PARA'.

When you compile and execute the above program, it produces the following result −

IN A-PARA

IN C-PARA

IN D-PARA

IN E-PARA

IN B-PARA

**Perform Until**

In ‘perform until’, a paragraph is executed until the given condition becomes true. ‘With test before’ is the default condition and it indicates that the condition is checked before the execution of statements in a paragraph.

**Example**

IDENTIFICATION DIVISION.

PROGRAM-ID. HELLO.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-CNT PIC 9(1) VALUE 0.

PROCEDURE DIVISION.

A-PARA.

PERFORM B-PARA WITH TEST AFTER UNTIL WS-CNT>3.

STOP RUN.

B-PARA.

DISPLAY 'WS-CNT : 'WS-CNT.

ADD 1 TO WS-CNT.

When you compile and execute the above program, it produces the following result −

WS-CNT : 0

WS-CNT : 1

WS-CNT : 2

WS-CNT : 3

**Perform Times**

In ‘perform times’, a paragraph will be executed the number of times specified.

**Syntax**

Following is the syntax of perform times −

PERFORM A-PARA 5 TIMES.

**Example**

IDENTIFICATION DIVISION.

PROGRAM-ID. HELLO.

PROCEDURE DIVISION.

A-PARA.

PERFORM B-PARA 3 TIMES.

STOP RUN.

B-PARA.

DISPLAY 'IN B-PARA'.

When you compile and execute the above program, it produces the following result −

IN B-PARA

IN B-PARA

IN B-PARA

**Perform Varying**

In perform varying, a paragraph will be executed till the condition in Until phrase becomes true.

**Syntax**

Following is the syntax of perform varying −

PERFORM A-PARA VARYING A FROM 1 BY 1 UNTIL A = 5.

**Example**

IDENTIFICATION DIVISION.

PROGRAM-ID. HELLO.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-A PIC 9 VALUE 0.

PROCEDURE DIVISION.

A-PARA.

PERFORM B-PARA VARYING WS-A FROM 1 BY 1 UNTIL WS-A=5

STOP RUN.

B-PARA.

DISPLAY 'IN B-PARA ' WS-A.

When you compile and execute the above program, it produces the following result −

IN B-PARA 1

IN B-PARA 2

IN B-PARA 3

IN B-PARA 4

**GO TO Statement**

GO TO statement is used to change the flow of execution in a program. In GO TO statements, transfer goes only in the forward direction. It is used to exit a paragraph. The different types of GO TO statements used are as follows −

Unconditional GO TO

GO TO para-name.

Conditional GO TO

GO TO para-1 para-2 para-3 DEPENDING ON x.

If 'x' is equal to 1, then the control will be transferred to the first paragraph; and if 'x' is equal to 2, then the control will be transferred to the second paragraph, and so on.

**Example**

IDENTIFICATION DIVISION.

PROGRAM-ID. HELLO.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-A PIC 9 VALUE 2.

PROCEDURE DIVISION.

A-PARA.

DISPLAY 'IN A-PARA'

GO TO B-PARA.

B-PARA.

DISPLAY 'IN B-PARA '.

GO TO C-PARA D-PARA DEPENDING ON WS-A.

C-PARA.

DISPLAY 'IN C-PARA '.

D-PARA.

DISPLAY 'IN D-PARA '.

STOP RUN.

When you compile and execute the above program, it produces the following result:

IN A-PARA

IN B-PARA

IN D-PARA

**TABLE**

A table is a collection of data items that have the same description, such as account totals or monthly averages; it consists of a table name and subordinate items called table elements.

Rather than defining repetitious items as separate, consecutive entries in the DATA DIVISION, you use the OCCURS clause in the DATA DIVISION entry to define a table.

**OCCURS CLAUSE IN COBOL**

**Occurs** clause is used to define a table. Occurs clause indicates the repetition of data name definition. It can be used only with level numbers starting from 02 to 49.

The OCCURS clause cannot be specified for an item whose level number is 01, 66, 77 or 88.

01 table-name.

05 element-name **OCCURS** *n* **TIMES**.

. . . (subordinate items of the table element)

**TWO-DIMENSIONAL TABLE**

To create a two-dimensional table, define a one-dimensional table in each occurrence of another one-dimensional table.

1. TABLE-ONE.

05 TABLE-ROW OCCURS 2 TIMES .

10 TABLE-COLUMN OCCURS 3 TIMES.

15 TABLE-ITEM-1 PIC X(2).

15 TABLE-ITEM-2 PIC X(1).

In TABLE-ONE above, TABLE-ROW is an element of a one-dimensional table that occurs two times. TABLE-COLUMN is an element of a two-dimensional table that occurs three times in each occurrence of TABLE-ROW.

**REFERENCING TABLE**

**INDEXING**

Index can be created by using the INDEXED BY phrase of the OCCURS clause to identify an index-name.

The OCCURS clause which is used to define tables can optionally have an INDEXED phrase

05 TABLE-ITEM PIC X(8)

OCCURS 10 TIMES INDEXED BY INX-A.

To increment the current value of index F1 and Y1 by 2, the following statement may be

used.

SET F1, Y1 UP BY 2.

On the other hand, the statement

SET D1 DOWN BY A indicates that the current value of the index D1 will be

decremented by the current value of the data name A. If before the execution of the

above statement, A and D1 contain 3 and 7 respectively, then after the execution of this

statement, D1 will contain 4.

**SUBSCRIPTING**

Table individual elements can be accessed by using subscript. Subscript values can range from 1 to the number of times the table occurs. A subscript can be any positive number. It does not require any declaration in data division. It is automatically created with occurs clause.

01 ANY-TABLE.

05 TABLE-ELEMENT PIC X(10)

OCCURS 3 TIMES VALUE "ABCDEFGHIJ".

. . .

MOVE "??" TO TABLE-ELEMENT (1) (3 : 2).

**SEARCH**

Search is a linear search method, which is used to find elements inside the table.

It can be performed on sorted as well as unsorted table.

It is used only for tables declared by Index phrase.

It starts with the initial value of index. If the searched element is not found, then the index is automatically incremented by 1 and it continues till the end of table.

SET I TO 1.

SEARCH WS-A

AT END DISPLAY 'M NOT FOUND IN TABLE'

WHEN WS-A(I) = WS-SRCH

DISPLAY 'LETTER M FOUND IN TABLE'

END-SEARCH.

**SEARCH ALL**

Search All is a binary search method, which is used to find elements inside the table. Table must be in sorted order for Search All option. The index does not require initialization. In binary search, the table is divided into two halves and it determines in which half the searched element is present. This process repeats till the element is found or the end is reached.

SEARCH ALL {table-name} [VARYING {index-name}]

[AT END processing-code-1]

{WHEN condition {processing-code-2}

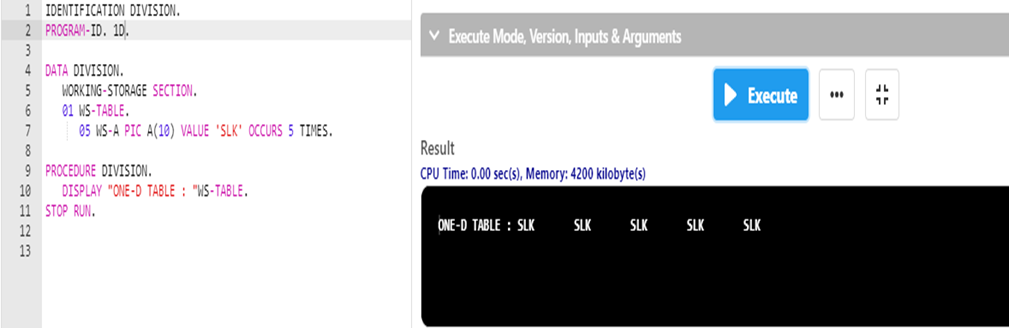
{NEXT SENTENCE    }

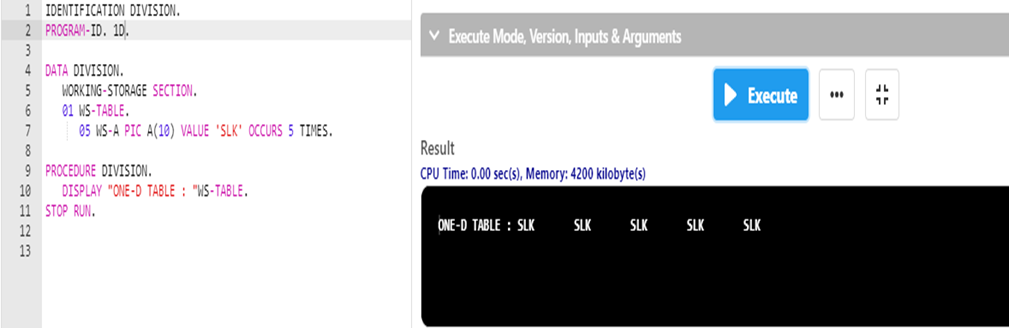
[END-SEARCH]

Assignments

Assignment 01 (Table Processing)

1. One Dimensional creation



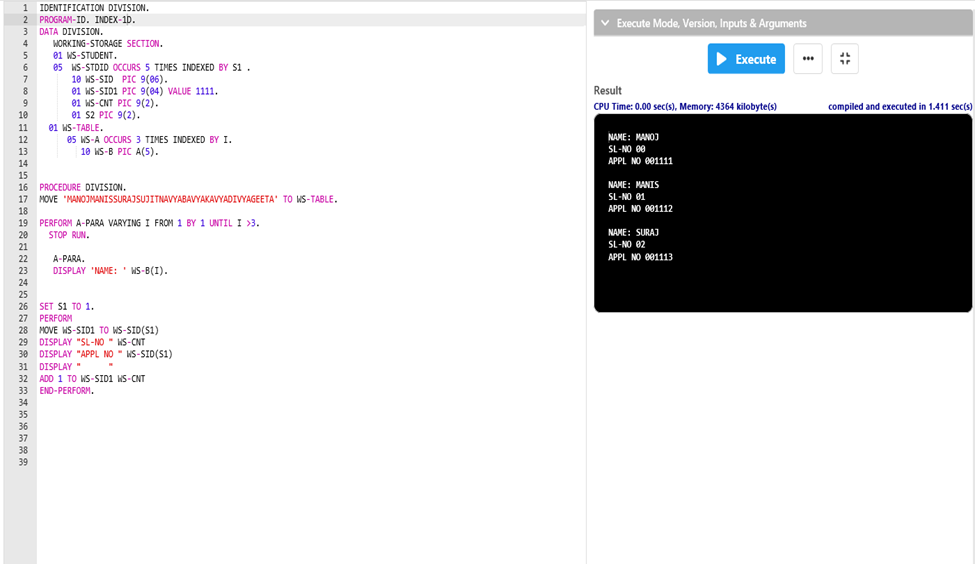


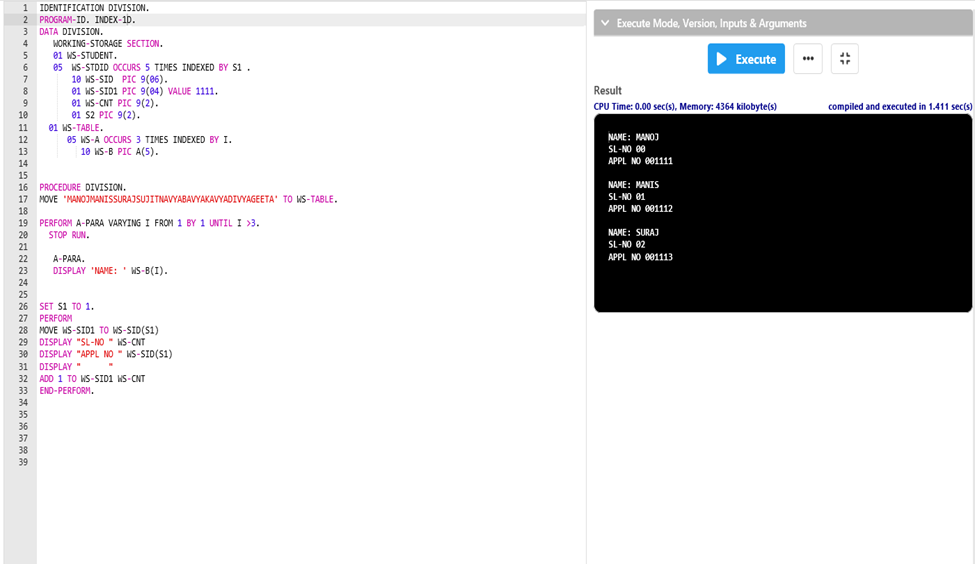
1. Two Dimensional Creation



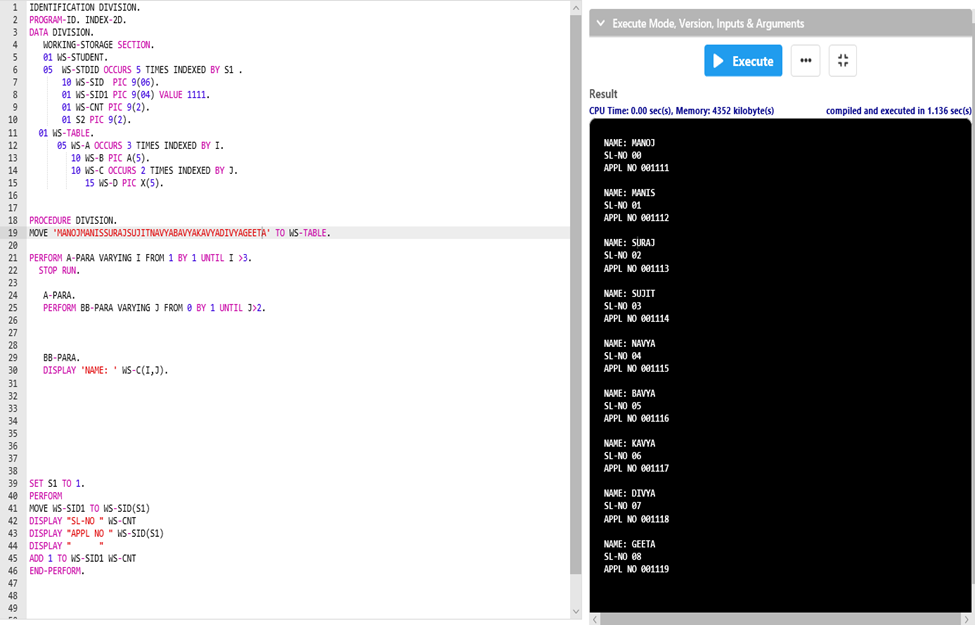


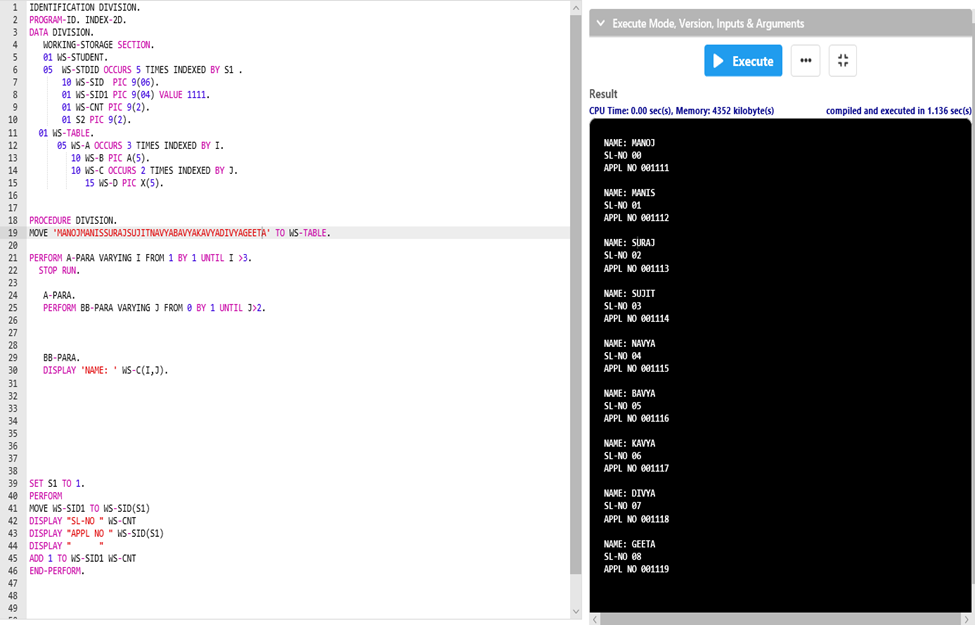
1. Index 1D



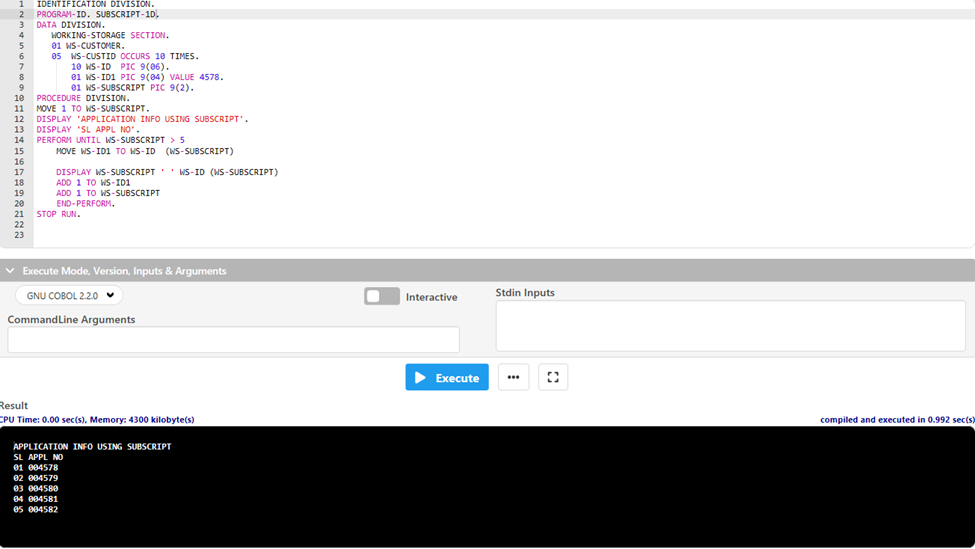
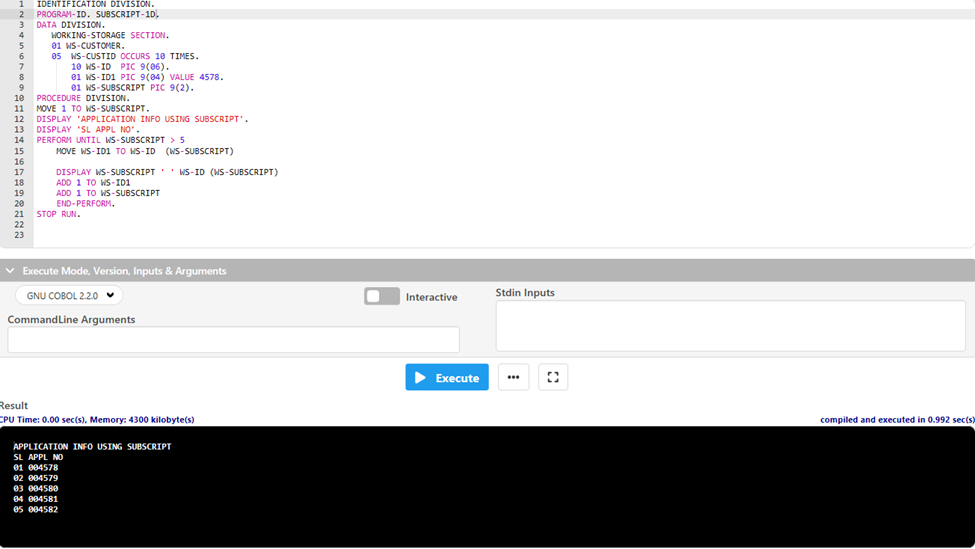


1. Index 2D

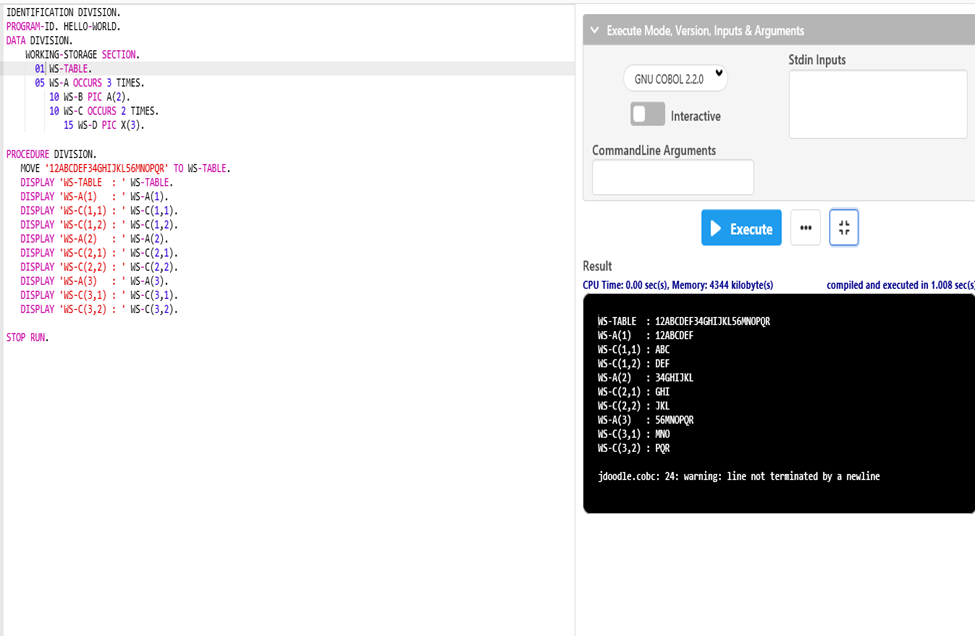
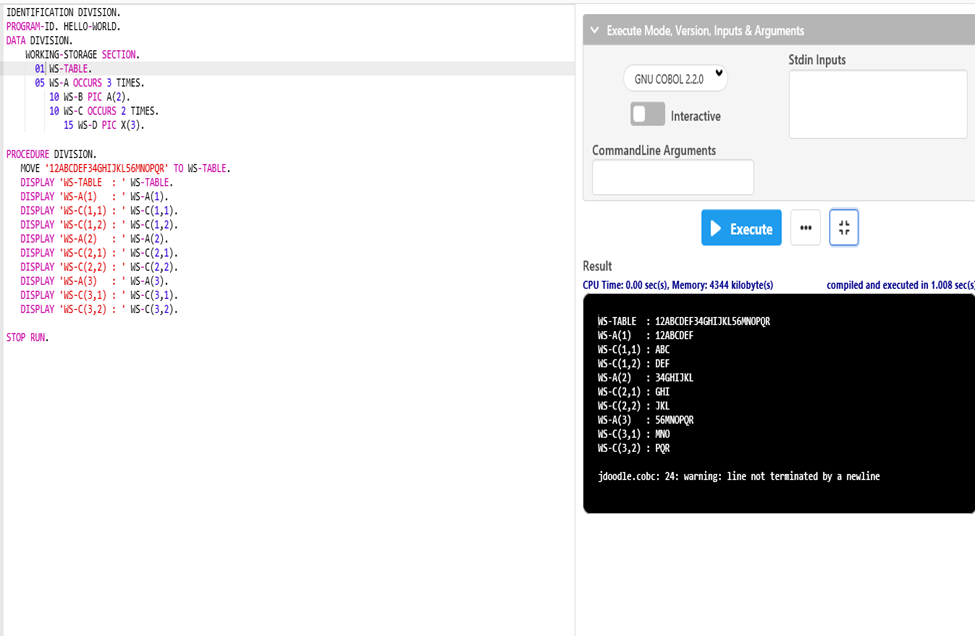




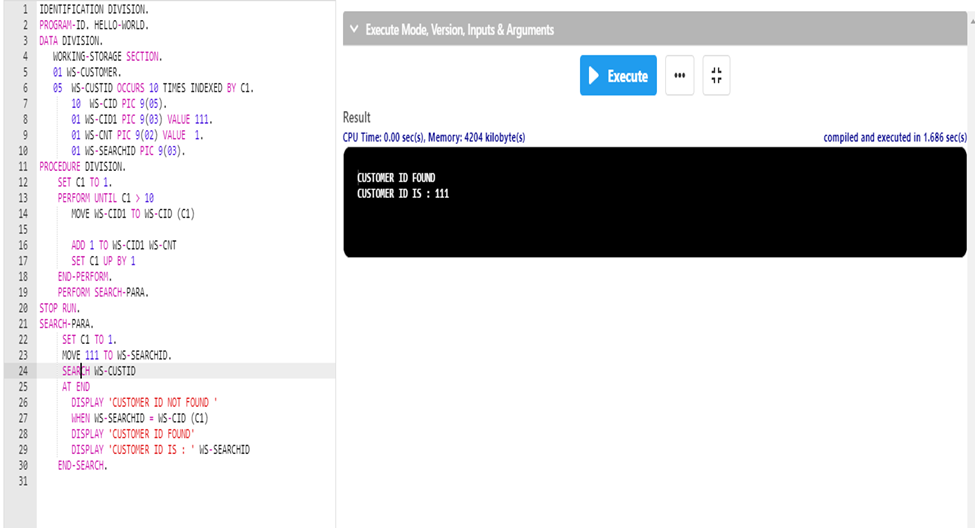
1. Subscript 1D

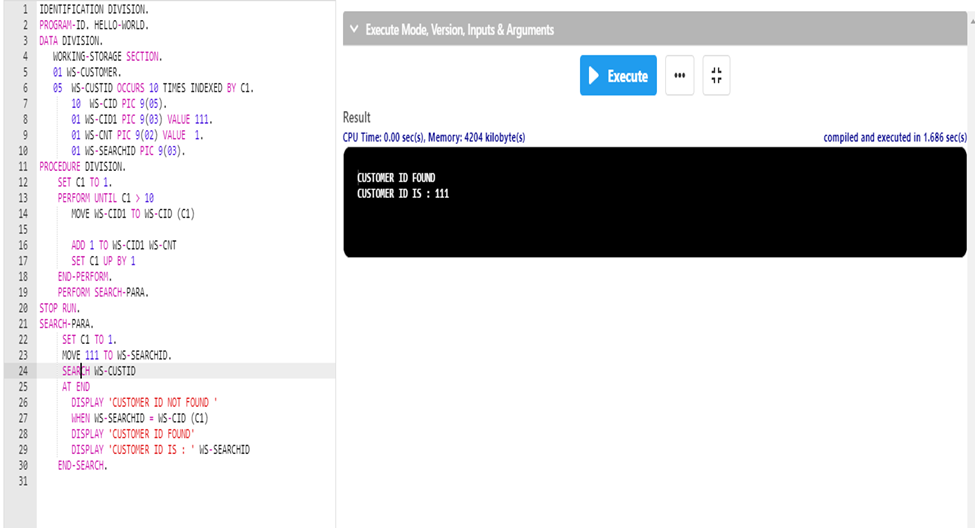
 

1. Subscript 2D

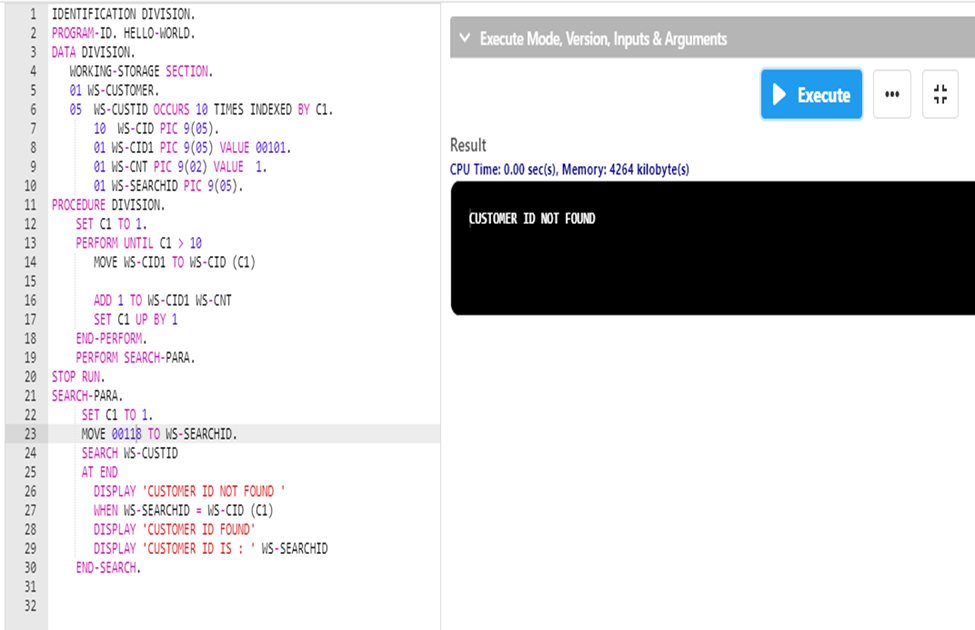


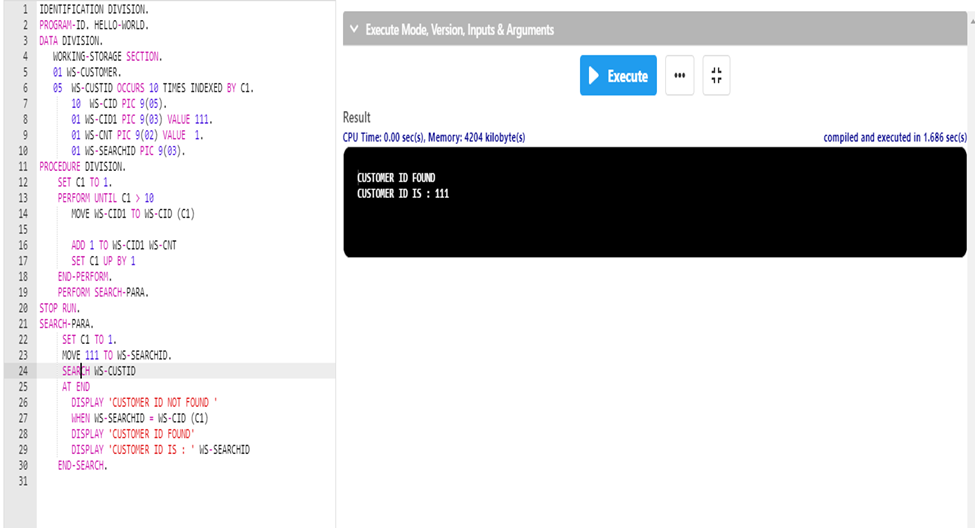
1. Search 1D(Found)



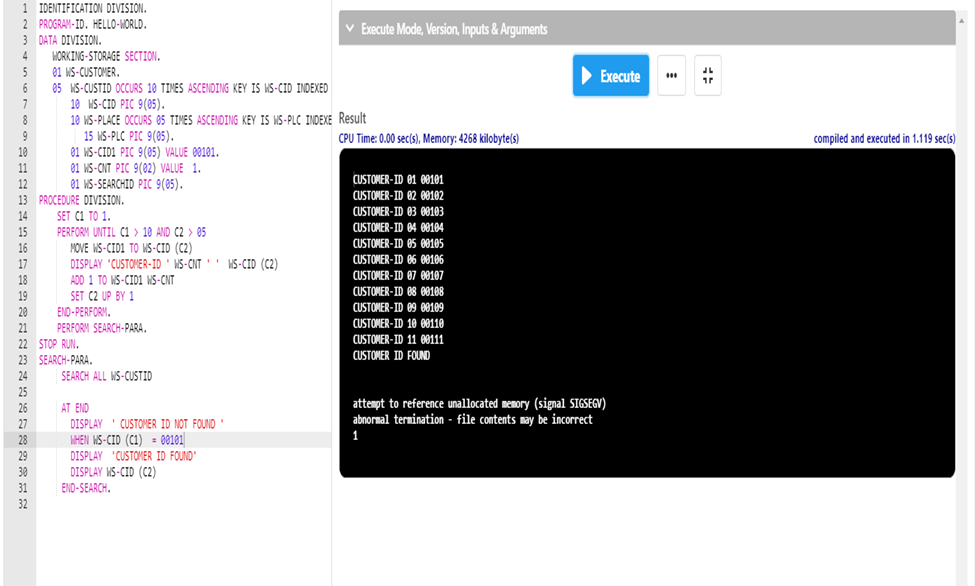
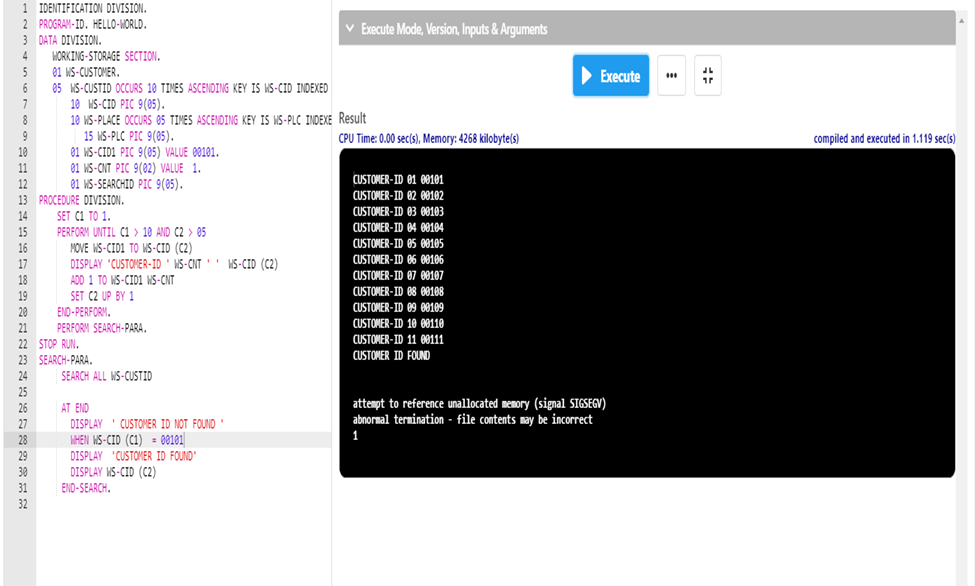


1. Search 1D(Not Found)

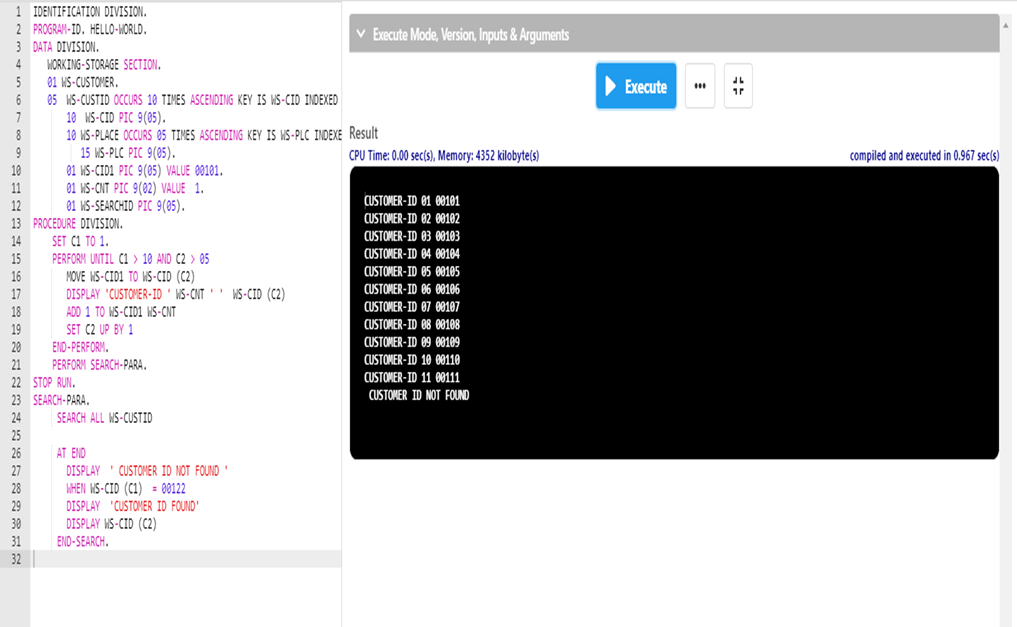


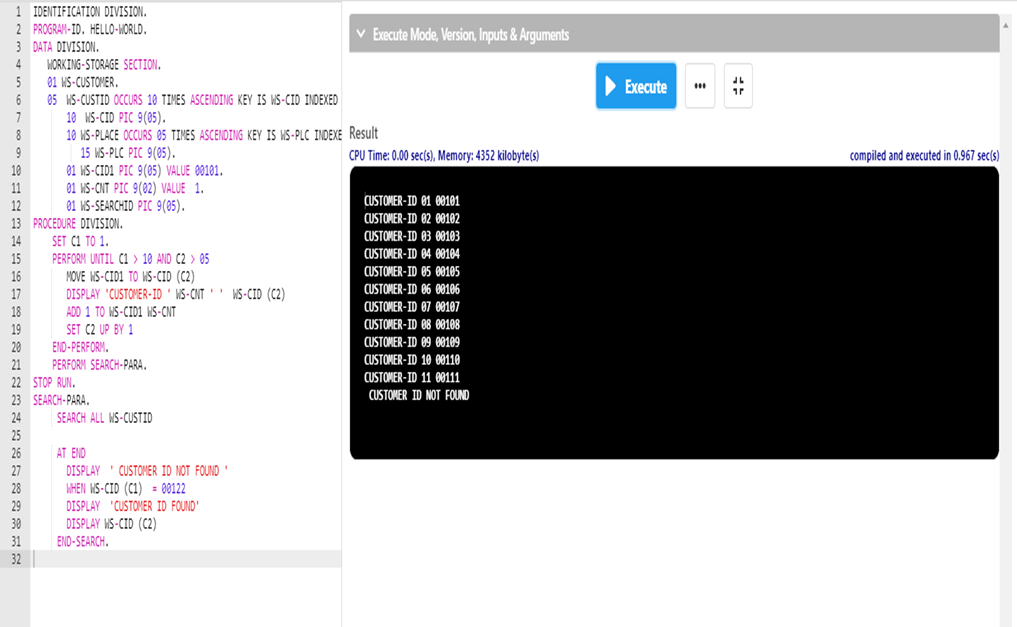


1. Search 2D(Found)

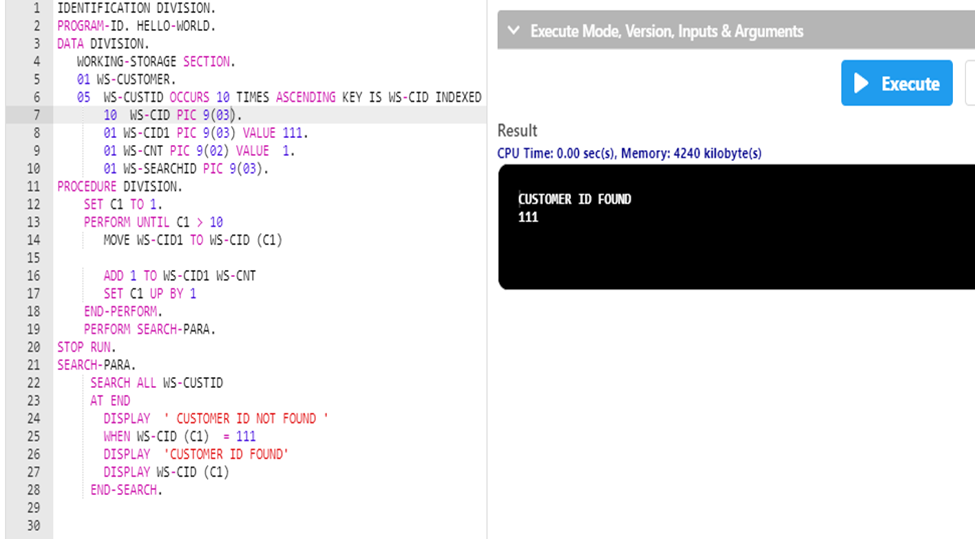


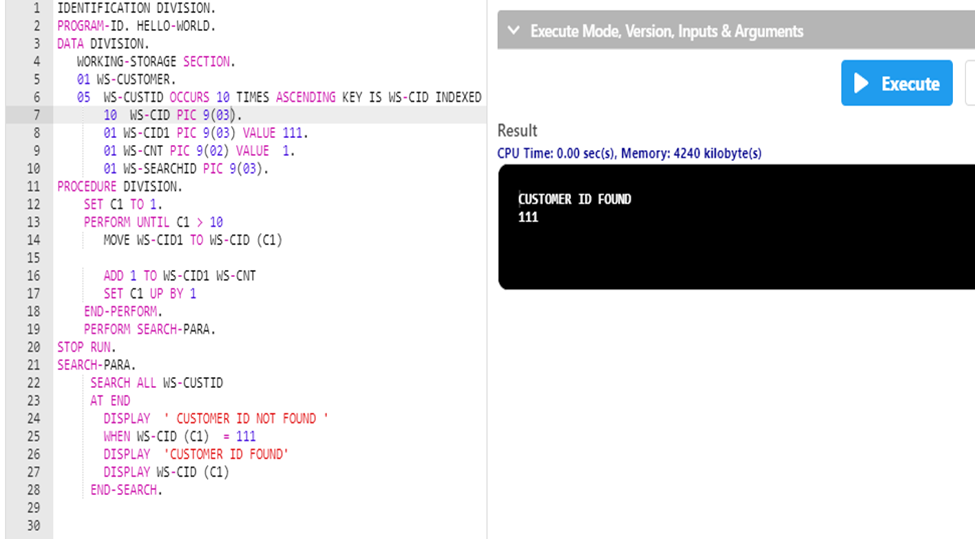
1. Search 2D (Not Found)



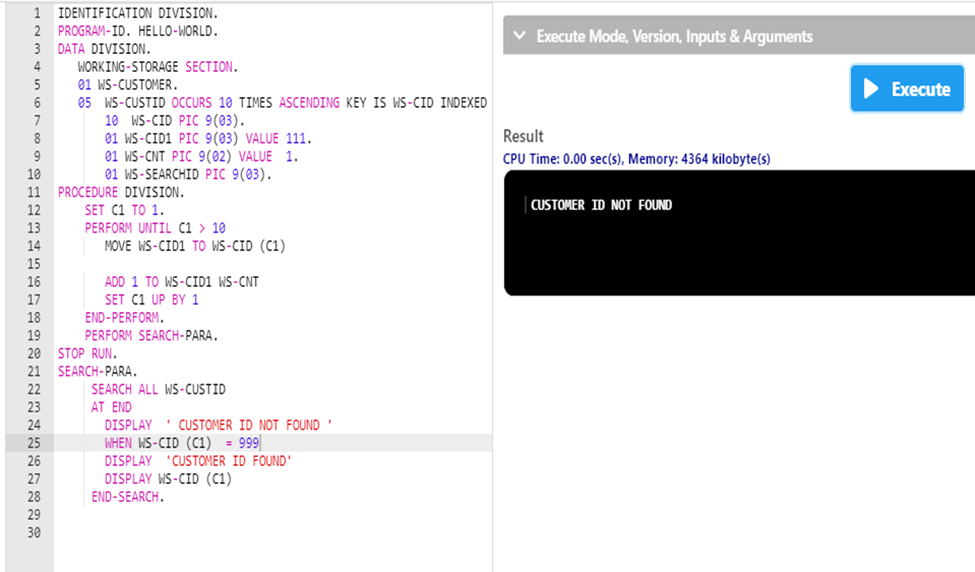


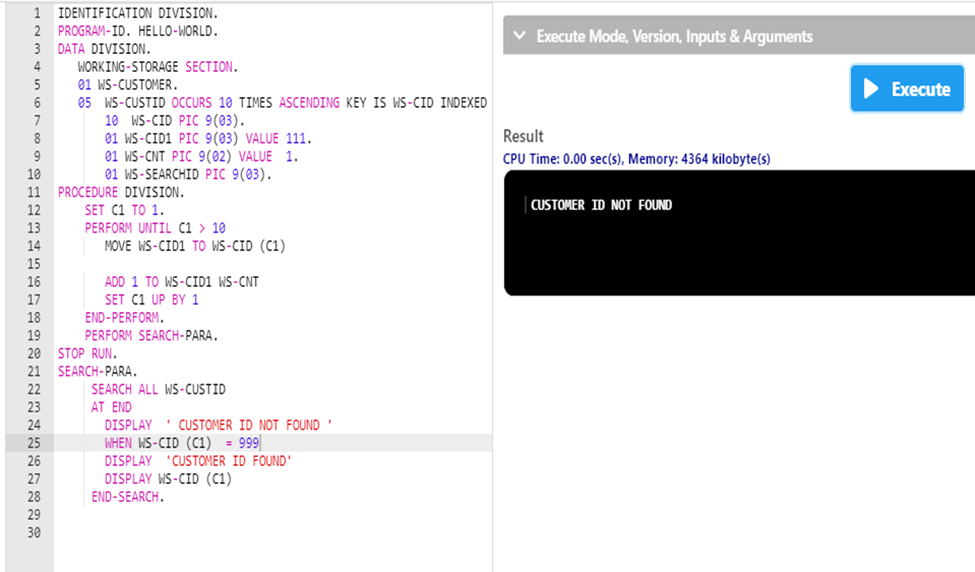
1. Search All 1D (Found)



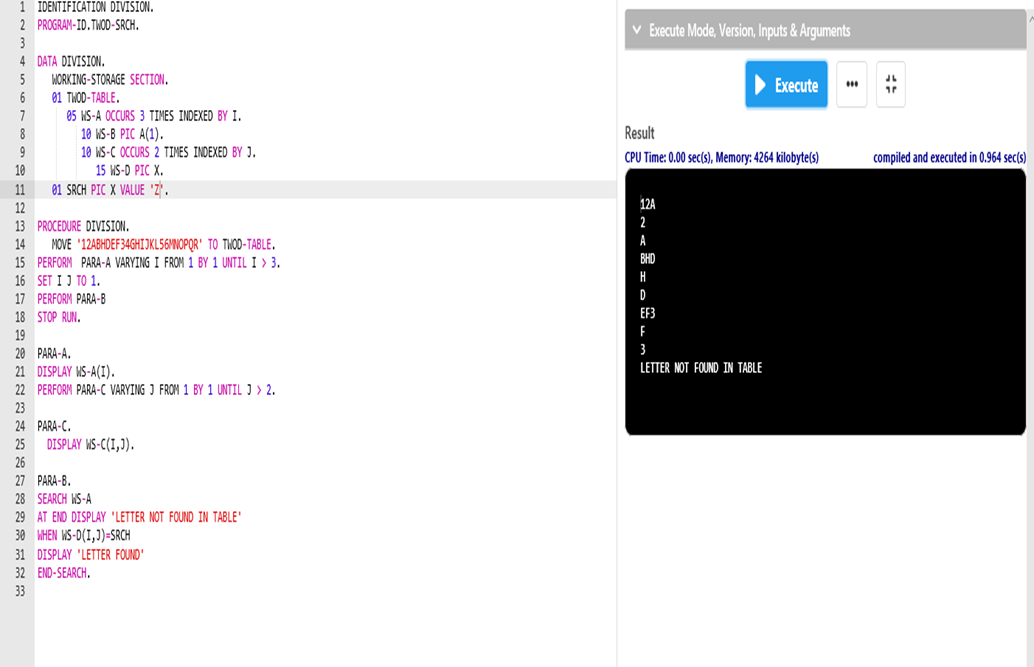
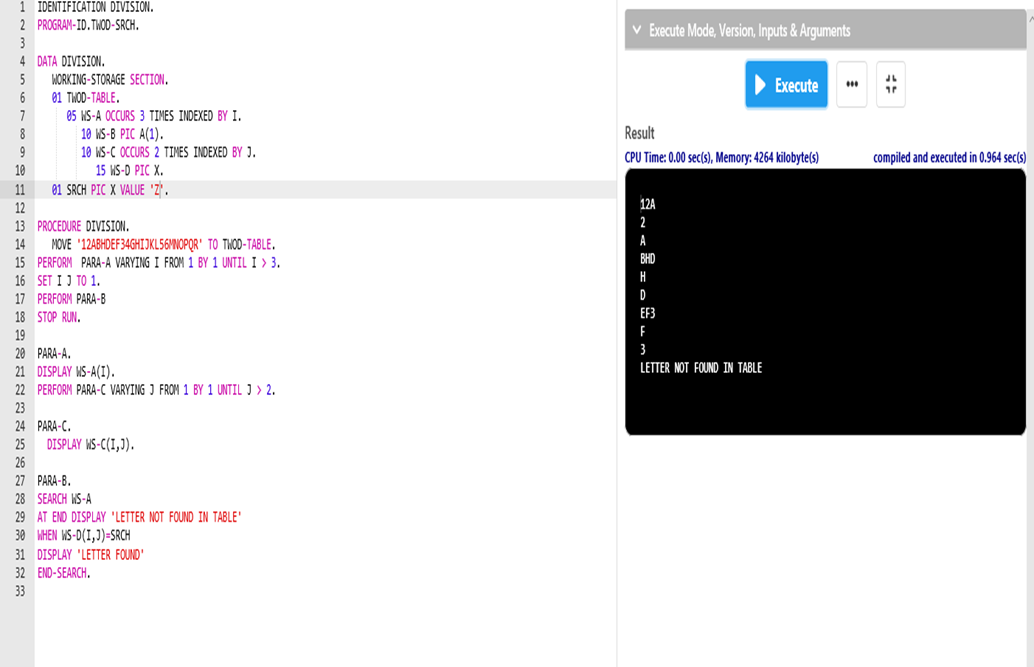


1. Search All 1D(Not Found)



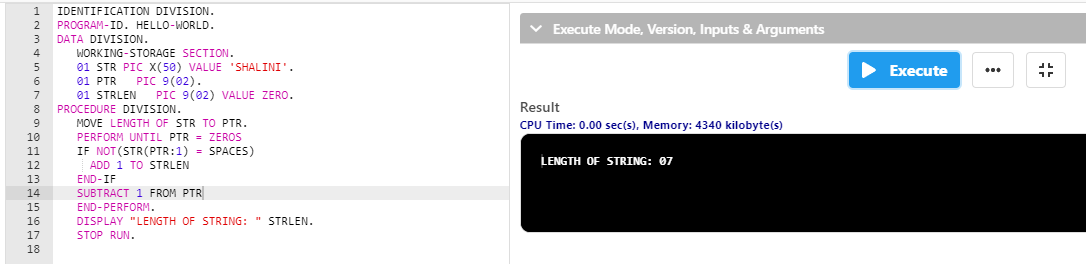


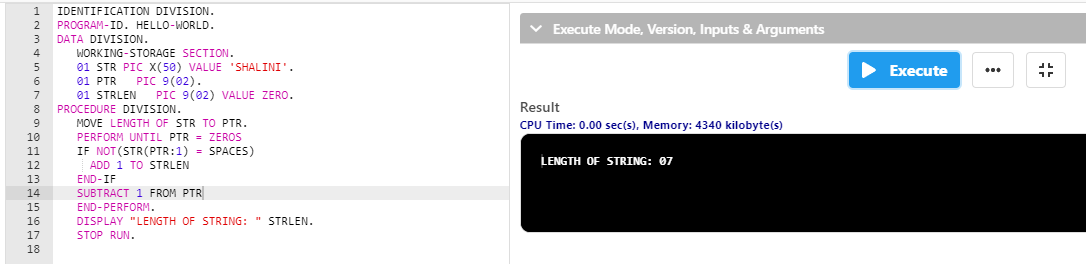
1. Search All 2D(Not Found)

Assignment 02 (String)

1. Length of a String Using Perform





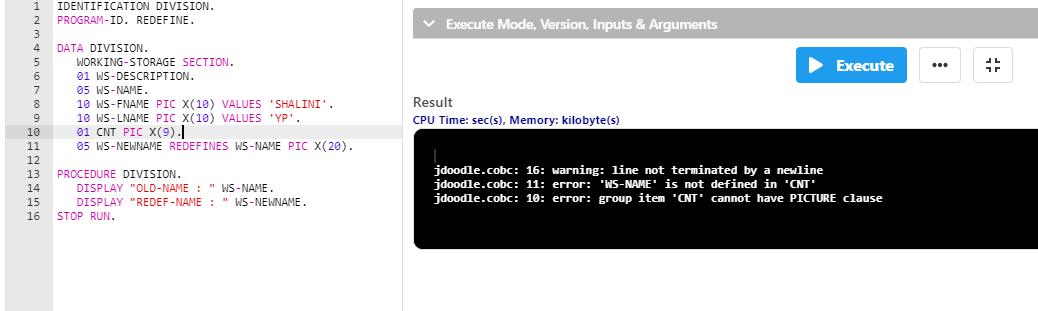
1. Redefine

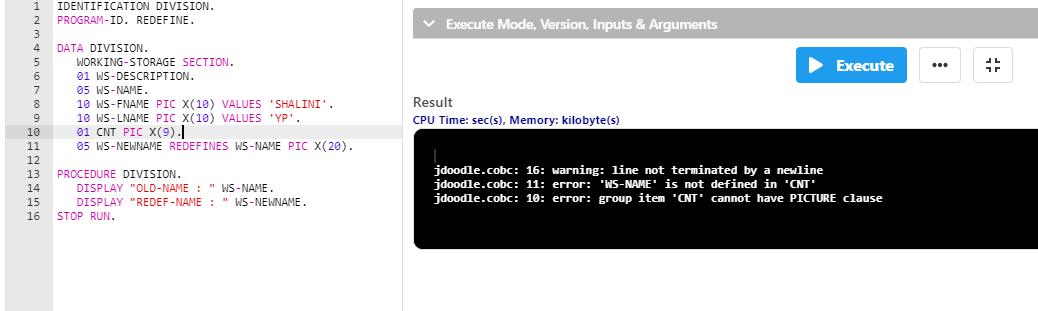




* Redefine with errors

1. Picture clause



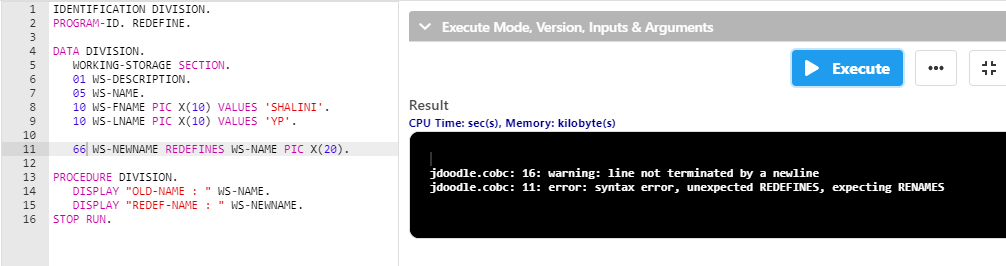


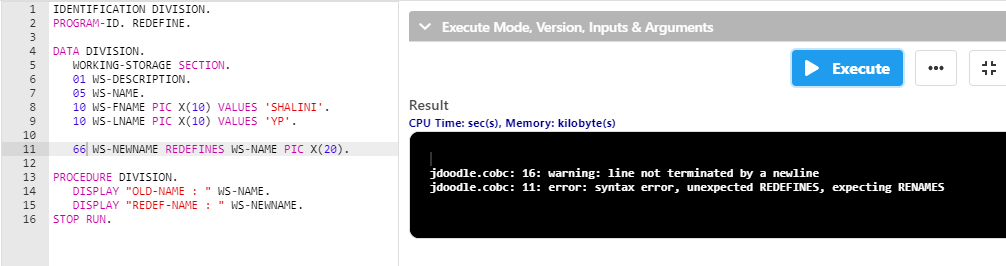
1. Not having same level number



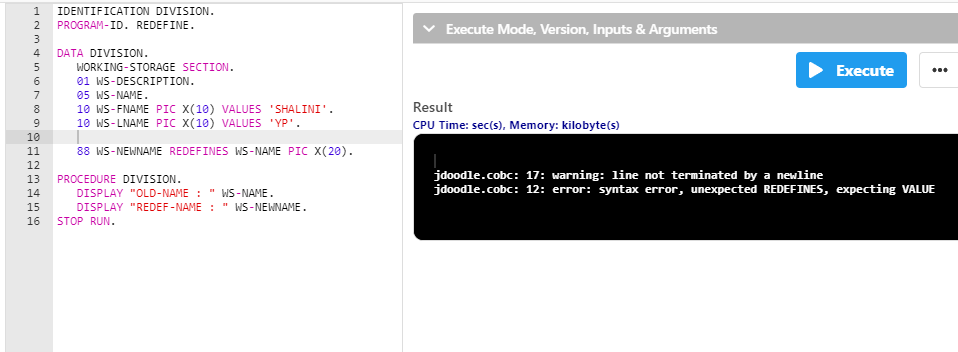


1. Level number 66



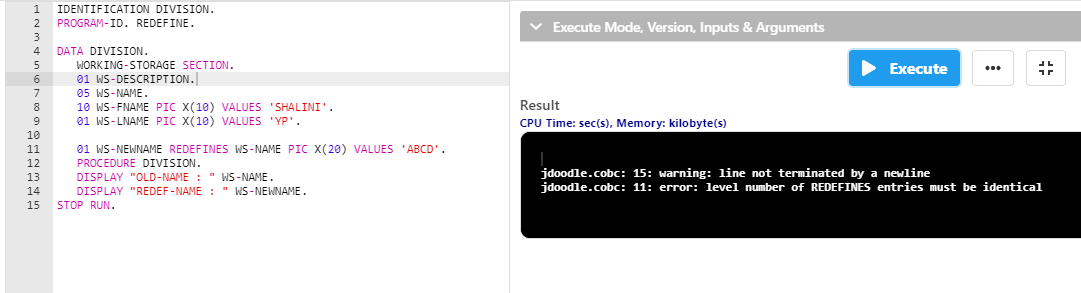


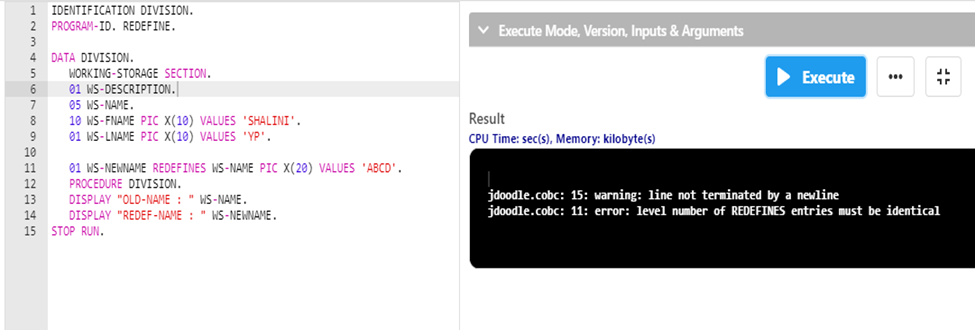
1. Level number 88





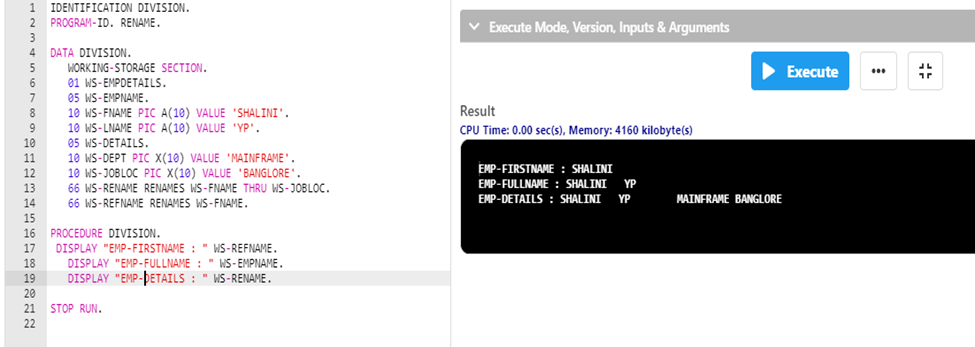
1. Level number 01





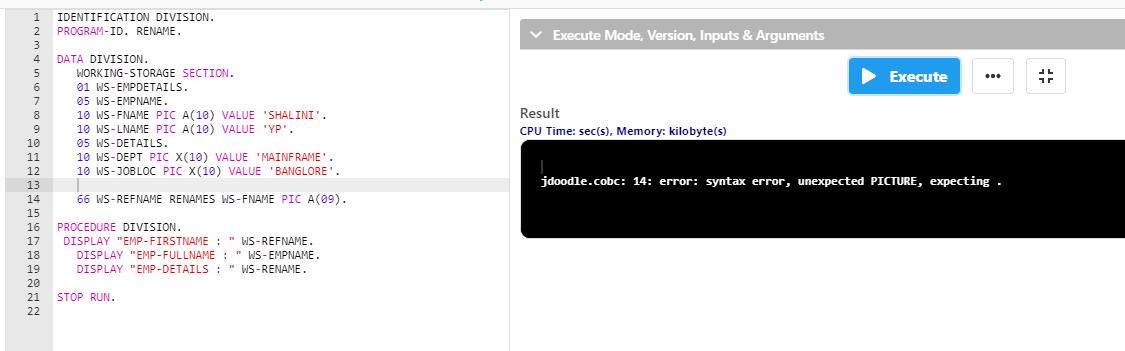
1. Rename

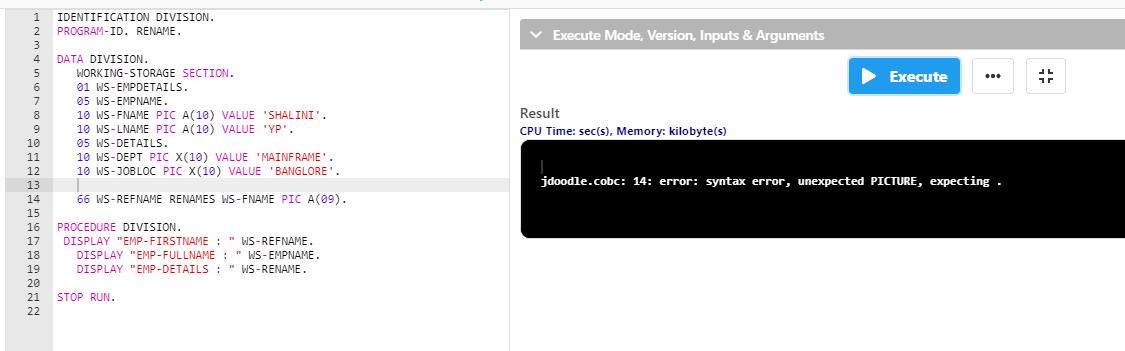




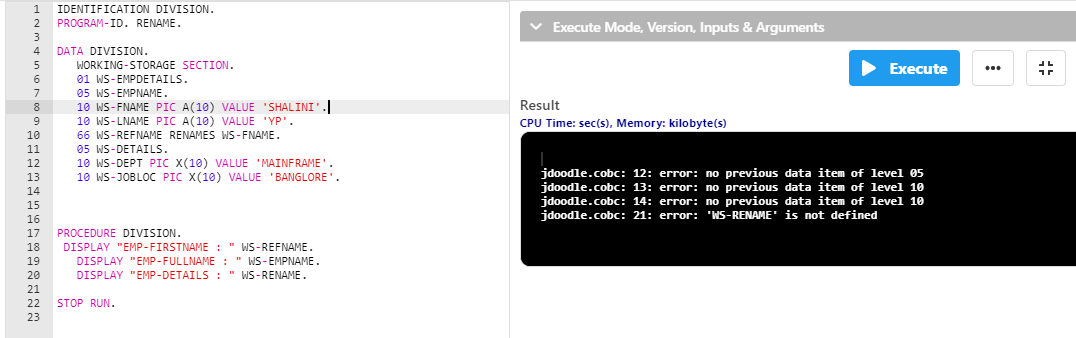
* Rename with Errors

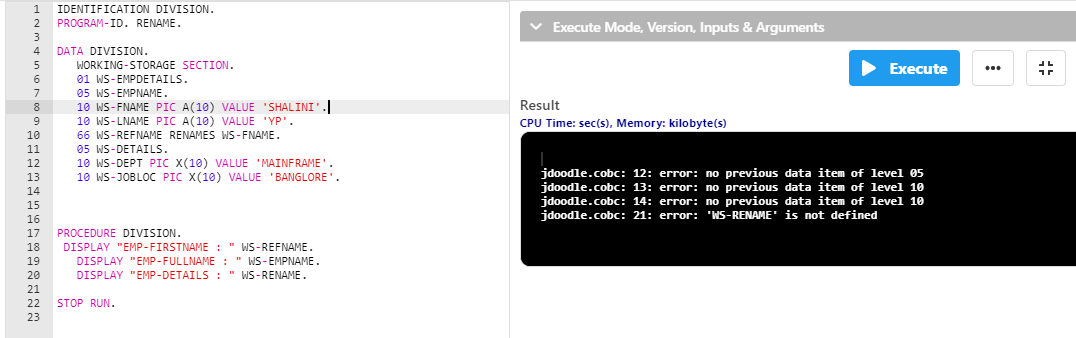
1. Including Picture clause in Rename



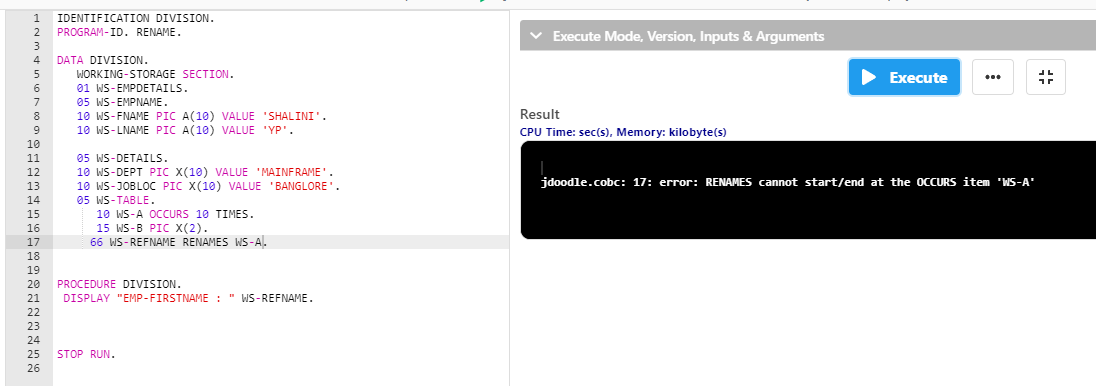


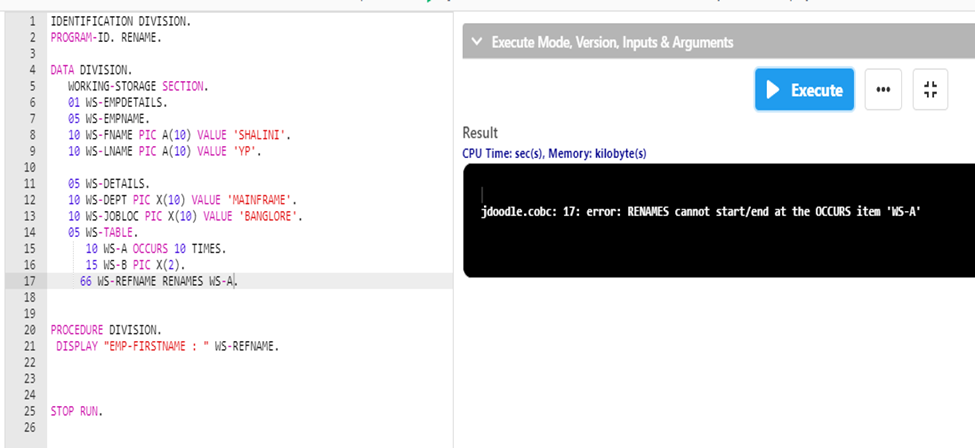
1. Rename clause coded in the middle of the group



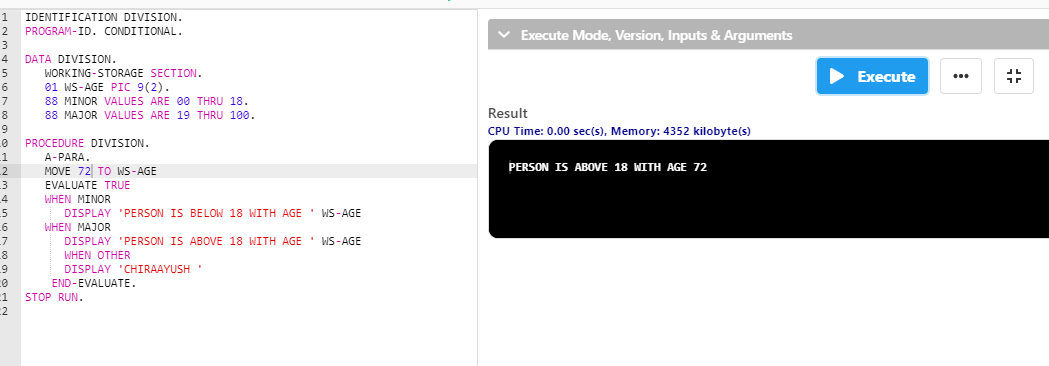


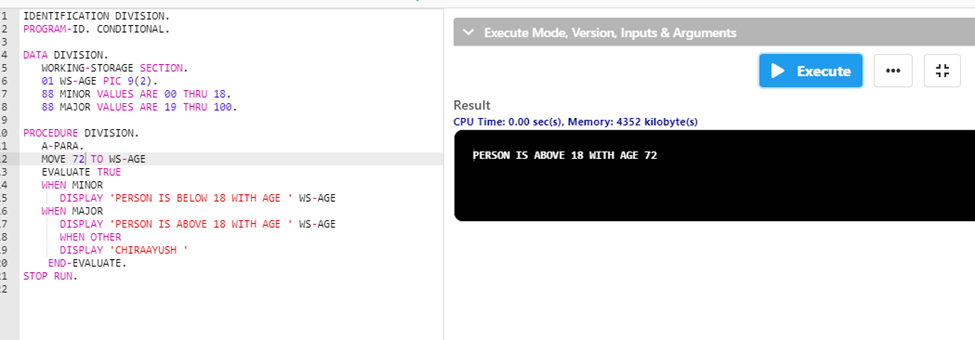
1. Rename clause for occurs





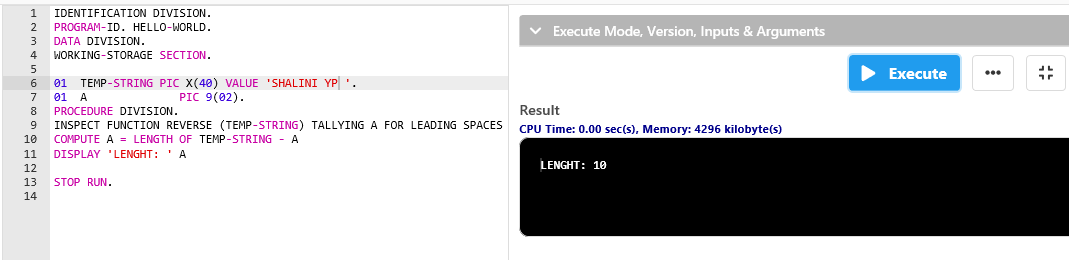
1. 88 LEVEL NUMBER

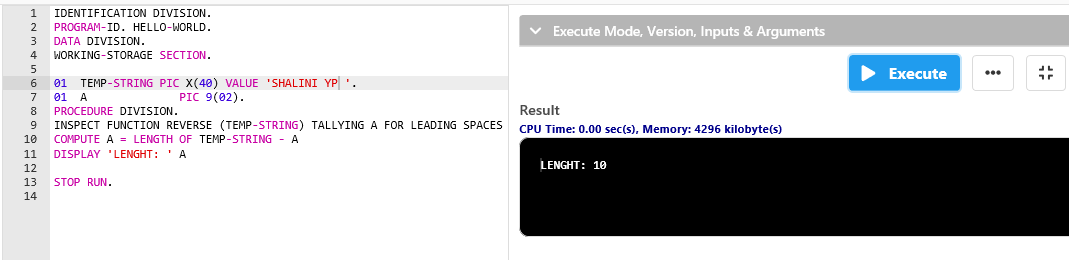




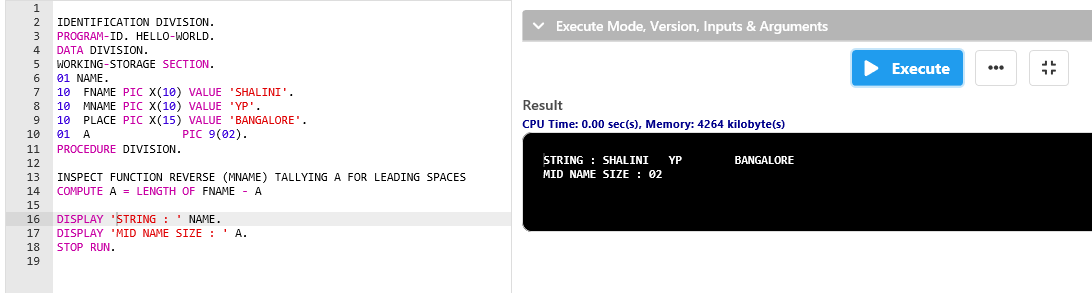
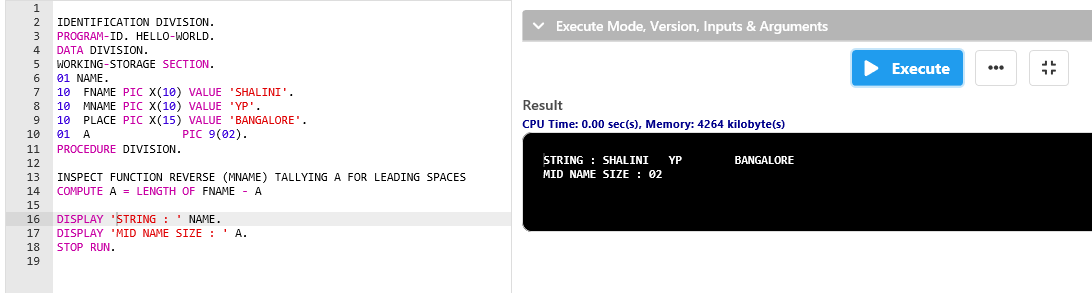
Assignment 03 (Inspect)

1. String length using INSPECT

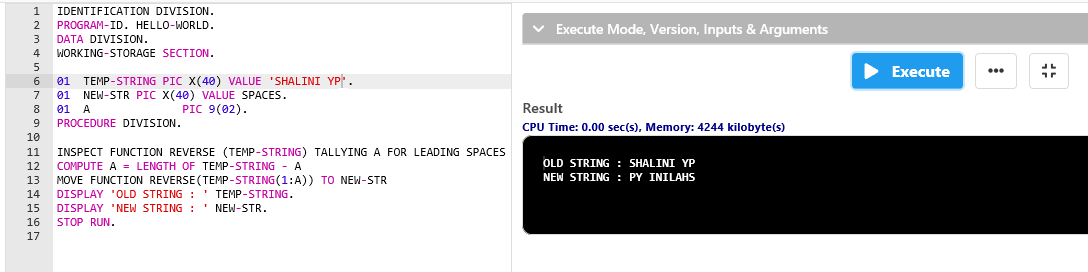


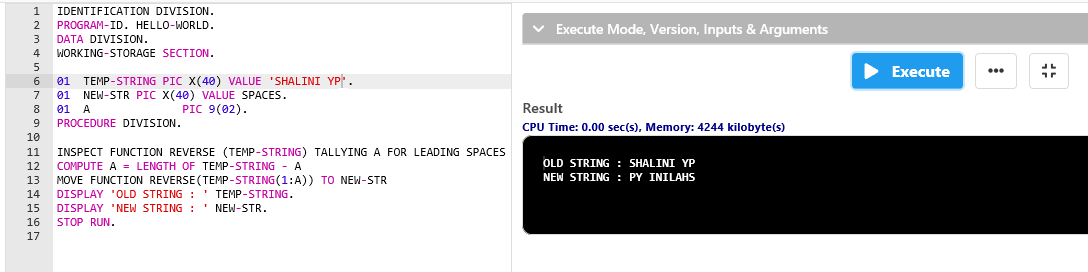


1. String length of Mid-Name using INSPECT

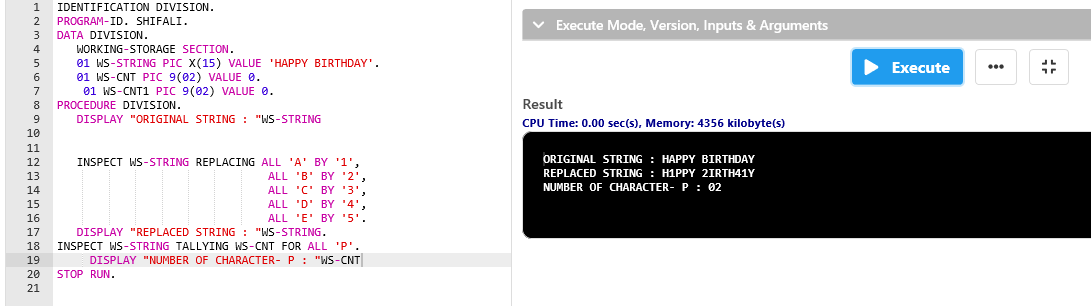


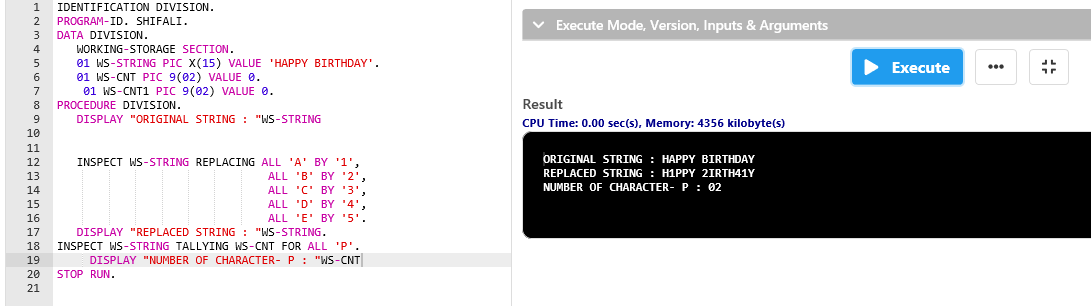
1. String reverse using INSPECT



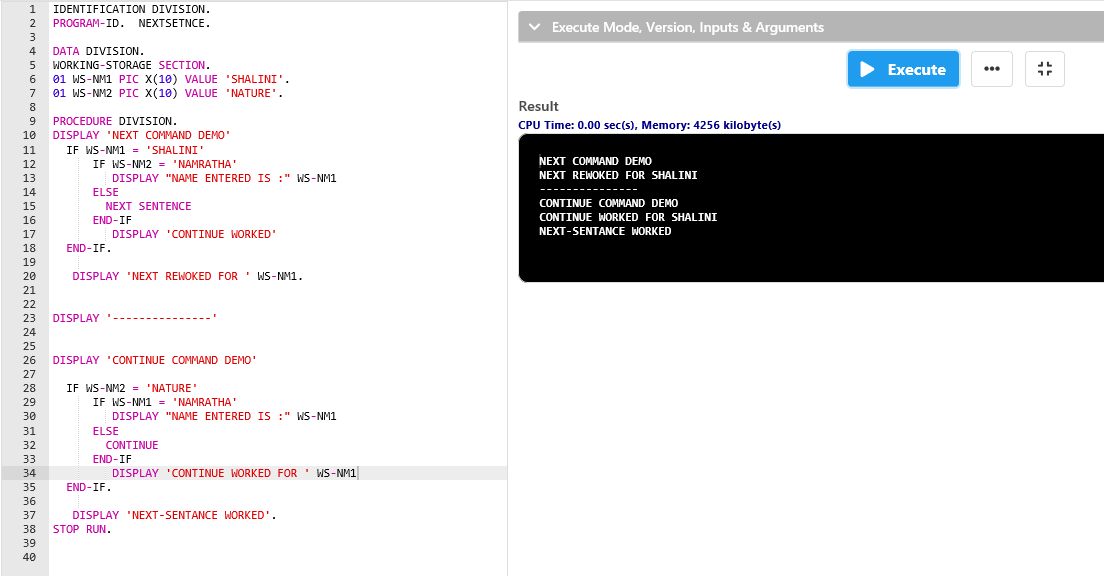


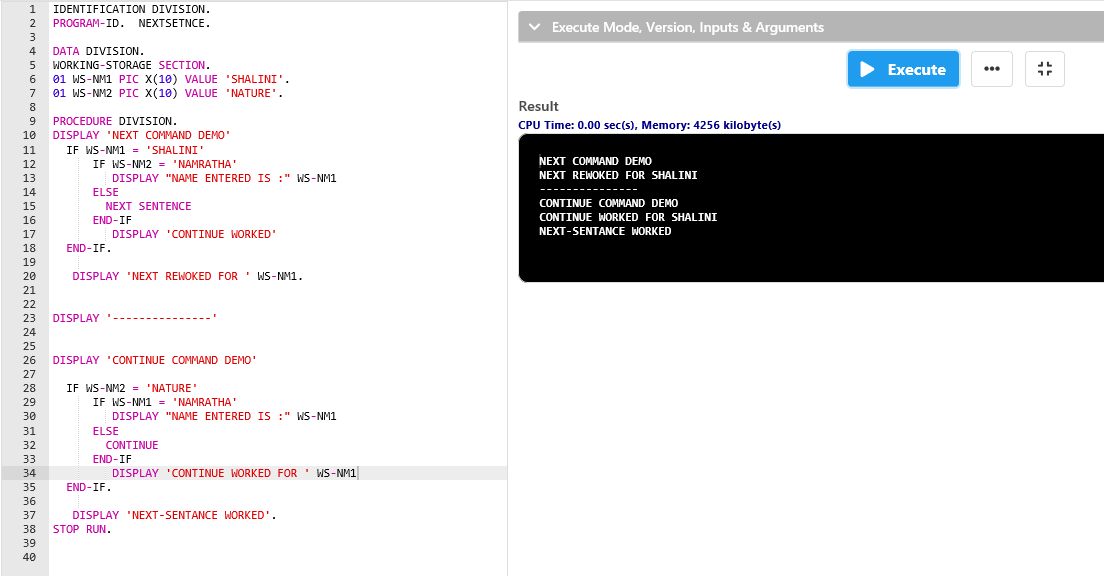
1. String replace using INSPECT





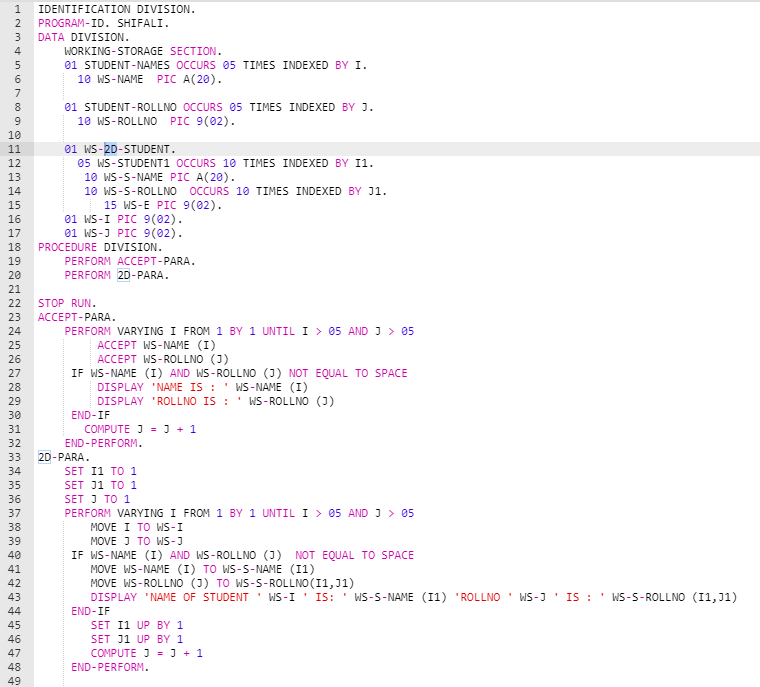
1. CONTINUE AND NEXT





Assignment 04 (Tables)

* Accepting the input into 1D table, and merging 2(1D) table into 1(2D) table



Output:

