

COBOL

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
29-May-2009	5.0	Arjun Singh	Content Creation
08-Oct-2009	5.0	CLS Team	Review and template application
06-July-2010	6.0	Vaishali Kasture	Review and Modification of Labs
22 nd August 2012	6.1	Vaishali Kasture	Revamped after Assignment Review

Table of Contents

Document Revision History.....	2
Table of Contents.....	3
Getting Started.....	5
<input type="checkbox"/> Overview.....	5
<input type="checkbox"/> Setup Checklist for COBOL.....	5
<input type="checkbox"/> Instructions.....	5
<input type="checkbox"/> Learning More (Bibliography if applicable).....	5
<input type="checkbox"/> Prerequisites.....	5
Lab 1: Sequential File Creation.....	6
<input type="checkbox"/> 1.1: Write a program to input the data given below:.....	6
<input type="checkbox"/> 1.2: Write a program to input the data given below:.....	8
Lab 1.3 Analysis and Debugging.....	10
<input type="checkbox"/> 10	10
<input type="checkbox"/> 1.4: Modify the program cobass01.....	10
<input type="checkbox"/> 1.5: Modify program cobass01.....	13
<input type="checkbox"/> 1.6: Print Item Class wise Report.....	14
Lab 1.7 Analysis and Debugging.....	16
<input type="checkbox"/> 16	16
<input type="checkbox"/> TO DO: The participants have to analyze the program, remove the error and successfully execute the program.	16
<input type="checkbox"/> TO DO: The program has to be debugged by the participants till it becomes error-free.....	16
Lab 1.8 Report-ControlBreak.....	17
Lab 1.9: Indexed File from Sequential File.....	18
Lab 1.10: Create an Indexed File from Sequential File.....	19
1.11: Indexed File –Append & Analysis.....	20
Lab 1.12: Updating Indexed File.....	21
<input type="checkbox"/> Update Index file "itemmast.dat" from a sequential file "GRN.DAT".....	21
Lab 1.13: Order booking for a Customer.....	22
<input type="checkbox"/> Order booking by Marketing for a customer.	22
Lab 1.14 Enhancement Assignment.....	25
Lab 1.15 Analysis and Debugging.....	26
Lab 1.16 Bank Transaction Updation and Report Generation.....	27
Lab 1.17 Working with 3 Indexed Files.....	27
Lab 1.18 Online Bus Booking.....	34
Lab Assignment 1.19: Call Statement-Debugging.....	34
Lab Assignment 1.20: Call Statement-Enhancement Assignment.....	39
Lab 1.21: Table Handling Lab Assignment.....	40
Lab Assignment 1.22: Table Handling- Analysis Assignment.....	42
Lab 1.23: String Handling.....	43
Lab 1.24: Mobile Services.....	44
<input type="checkbox"/> Stretched Assignment 1:.....	46
<input type="checkbox"/> Itemwise Sales Order Report.....	46
<input type="checkbox"/> Stretched Assignment 2: Write a program to print the following report using the file created in program COBASS2.....	47
<input type="checkbox"/> Case Study.....	47
Appendices 54	
<input type="checkbox"/> Appendix A: Breakup of Marks.....	54
<input type="checkbox"/> 1. Compiling, Linking, and Executing COBOL programs.....	54

□ 1. Debugging a COBOL program.....	61
□ 3. Invoking Subroutines	65
COBOL STANDARDS.....	56

Getting Started

- **Overview**

This lab book is a guided tour for learning COBOL. It comprises solved examples and 'To Do' assignments. Follow the steps provided in the solved examples and work out the 'To Do' assignments given.

- **Setup Checklist for COBOL**

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- NA

Please ensure that the following is done:

- NA

- **Instructions**

- Create a directory by your empid_name in drive <drive> for storing all COBOL programs.

- **Learning More (Bibliography if applicable)**

NA

- **Prerequisites**

It is expected that participants know the following:

- MVS
- JCL

Day1:Lab 1:Working with Sequential Files and Report Handling
Goals

- Learning COBOL assignments in a real world COBOL application system. This is a basic system with minimum functional complexities.
- To develop a COBOL business application

Introduction:

A company named "ABC Co. Ltd" manufactures computer parts and sells them. The company's stockyard gets orders from customers which are collected by the marketing department.

Everyday the stockyard receives the material through the manufacturing department. These items are received through a document called "Goods Receipt Note" (GRN). The GRN contains the GRN number, date, item number, and the quantity of the item received. Using the GRN, the Item Master is updated to reflect the new inventory stock in hand for a particular item. That is, the **received qty** and the **dispatch qty** fields of the **Item master** are initially set to zero at the beginning of the month. Subsequently, these fields are updated on a regular basis during the month.

Marketing department gets the stock status every day after the updating of the stock based on the Goods Receipt Note. Based on the stock position, order balance, customer priority etc., the Marketing department sends **delivery instructions** to the Stockyard. The **delivery instruction** contains the order, which has to be processed, along with the items and quantities, based on priority of orders. In short, orders are not automatically processed. Depending on the availability of items, a **status flag** is updated to indicate whether the order requirements have been fully/partially supplied. The stockyard prepares the invoice from the Delivery Instruction and this also reduces the stock balance.

At the end of the month, a fresh **stock master file** is created and the last month's file is taken to backup. Final updation of the **Item master** takes place on a monthly basis.

Files are created, updated, and Reports are generated from this data.

- **Day 1:Lab 1.1: Write a program to input the data given below:**

[Duration 1 hour]

Record structure:

IT-ITEMNO	Itemno	pic x(6)	format : char 1-2 "Item Class", char 3 – 6 4 digit serial no.
IT-DESC	Item description	pic x(25)	
IT-MONOPBAL	Item month op. Bal qty	pic 9(6)	
IT-RCPTQTY	Item month receipts qty	pic 9(6)	
IT-DESPQTY	Item month dispatch qty	pic 9(6)	
IT-RATE	Item rate	pic 9(5)v99	

IT-PROCMY	Processing MMYYYY	pic 9(6)	
-----------	-------------------	----------	--

Item's Classes are as follows:

10 – Internal Components, 20 - Input Devices, 30 - Cards Boards, 40 – Storage Devices, 50 – Output Devices, 60 – Cables, 70 – Storage Media

Sample data:

IT-ITEMNO		IT-DESC	IT-MONOPBAL	IT-RCPTQTY	IT-DESPQTY	IT-RATE	IT-PROCMY
IT-Class	IT-SR-No						
10	0001	Celeron 1 GHz	80	25	45	3300	012004
10	0002	P III 833 MHz	40	5	10	3400	012004
10	0003	P IV 2 GHz	36	7	14	4500	082004
10	0004	RAM 128 MB	100	10	40	900	012004
10	0005	RAM 256 MB	70	30	20	1200	112003
10	0006	RAM 512 MB	70	20	30	1800	012004
20	0001	Logitech serial mouse	20	5	11	300	112003
20	0002	Logitech PS/2 mouse	15	3	5	400	122003
40	0001	Sony SDT – 9000	2	3	4	8000	122003
60	0001	40 pin IDE cable	100	40	80	40	012004
60	0002	80 pin IDE cable	90	30	60	50	122003
60	0003	Parallel port cable	115	20	60	80	012004

Solution:

Step 1: Check that the data is of a record structure as shown in the above table. Item class can have values as multiples of 10 from 10 – 70 inclusive only.

Step 2: Input all the items in the data given in the above table. Once the data of a record is complete, the program should display the full group as an item. The program will then ask the user “Do you want to Input More (Y/N)”.

Step 3: Select an option (Y/N). The program should accept the user's response.

- If the response is “Y”, then the program should accept the details on another item and repeat the above steps.

- If the response in “N”, then the program should stop.
 - Perform a Peer Review and record the TestCases in TestCase.xls
- **Day 1:Lab 1.2: Write a program to input the data given below: [Duration 1 hour]**

Record structure:

CU-CUSTNO	Custno	pic x(6)	format : char 1-2 The 1st two letters of customer name, char 3-6 4 digit serial no.
CU-CUSTNAME	Cust Name	pic x(25)	
CU-ADDLIN1	Cust add. Line 1	pic x(25)	
CU-ADDLIN2	Cust add. Line 2	pic x(25)	
CU-REGION	Region	pic x(3)	Coded as 3 Char of city as below only: MUM,DEL,CHN,CAL,BLR,HYD,PUN.
CU-MAXCR	Max credit balance	pic 9(6)	
CU-CURRBAL	Current Balance	pic 9(6)v99	

CU-CUSTNO	CU-CUSTNAME	CU-ADDLIN1	CU-ADDLIN2	CU-REGION	CU-MAXCR	CU-CURRBAL
AB0034	ABC Computer Services	MIDC,	Andheri	MUM	1,50,000	1,00,000
DA7201	Datacomp Services	Anna Salai,	Teynampet	CHN	2,00,000	1,75,000
NE9944	Network Pvt. Ltd	Vasant Vihar,	Thane	MUM	2,00,000	1,00,000
EX1021	Excellent Computers	MIDC,	Bhosari	PUN	1,25,000	90,000

Solution:

Step 1: Check that the data is of a record structure as shown in the above table.

Step 2: Input all items of the data given in the above table. Once the data of a record is complete, the program will display the full group as an item. The program will then ask the user “Do you want to Input More (Y/N)”.

Step 3: Select an option (Y/N). It should accept the user’s response.

If the response is “Y”, then the program should accept the details on another customer and repeat the above steps. If the response is “N”, then the program should stop.

Day 1 : Lab 1.3 Analysis and Debugging [Duration ½ hour]

TO DO: The participants have to analyze the program, remove the error and successfully execute the program.

TO DO: All the participants have to document the error in the excel sheet along with steps taken to resolve the error.

Files for participants: Sequential File Analysis.cbl

-

- **Day 2: Lab 1.4: Modify the program [Duration 3 hours]**

Modify the program cobass01 to have three choices as listed below. The program should function as per the steps of choices listed below.

When the user executes the program the following screen will be displayed:

<p>A. Input Data B. Print Data Q. Quit.</p> <p>Enter your choice : _</p>
--

Ensure that the choice entered is A, B, or Q.

Choice 1: If the user enters 'A', then follow the steps given below.

Input data:

Step 1: Open the file "**itemseq.dat**" in extend mode.

Step 2: Key in the data as in COBASS01. However, instead of displaying on the screen it is to be written to a sequential file named "**itemseq.dat**" with the same format as the group. The program will then ask the user "Do you want to enter more (y/n)?".

Step 3: Select an option (Y/N). It will accept the user's response.

- If the user response is Y, then the program will once again take input and write to file.
- If the user response is N, then the program will close files and go back and display the menu once again.

Choice 2: If the user enters 'B', then follow the steps given below.

Print Data:

Step 1: Open the sequential file “**itemseq.dat**” in **input mode** and the “**itemrep.dat**” in **output mode**.

- It should read the data sequentially until End of file.
- For every valid record that is read, it should print as per the format given below.
- There should be a max of 5 item records per page. The **Run Date** is the system date and the **month/ Year** is the mmyyyy of the data in itemseq.dat’s first record.
- At the end of the report, a grand total of the closing value field will be printed and all files will be closed.

ABC CO. LTD.							Date : dd/mm/yyyy
List of Items in Stock For <month > / Year							Page : 999
Itemno	Item Description	<----- Month Details ----->				Rate	Clo. Value
		Op.Bal	Receipts	Dispatch	Clo-Bal.		
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	999999	999999	999999	999999	99999	9999999
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
PAGE TOTAL :							99999999
List of Items in Stock For <month > / Year							Page : 999
Itemno	Item Description	<----- Month Details ----->				Rate	Clo. Value
		Op.Bal	Receipts	Dispatch	Clo-Bal.		
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	999999	999999	999999	999999	99999	9999999
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
PAGE TOTAL :							99999999
GRAND TOTAL :							99999999

Choice 3: If the user enters ‘Q’, the program should terminate.

Sample output

ABC CO. LTD.

Date : 08/01/2004

List of Items in Stock For 01/2004

Page : 1

```

-----
<----- Month Details ----->
Itemno  Item Description  Op.Bal Receipts Dispatch Clo. Bal. Rate  Clo. Value
-----
100001  Celeron 1 GHz        80    25    45      60  3300.00 198000.00
100002  P III 833 MHz        40     5    10      35  3400.00 119000.00
100003  P IV 2 GHz           36     7    14      29  4500.00 130500.00
100004  RAM 128 MB           100    10    40      70   900.00  63000.00
100005  RAM 256 MB            70    30    20      80  1200.00  96000.00
  
```

PAGE TOTAL : 606,500.00

List of Items in Stock For 01/2004

Page : 2

```

-----
<----- Month Details ----->
Itemno  Item Description  Op.Bal Receipts Dispatch Clo. Bal. Rate  Clo. Value
-----
100006  RAM 512 MB          70    20    30      60  1800.00 108000.00
200001  Serial mouse         20     5    11      14   300.00   4200.00
200002  PS/2 mice            15     3     5      13   400.00   5200.00
400001  Sony SDT-9000         2     3     4       1  8000.00   8000.00
600001  40 pin IDE000        100   40    80      60   40.00   2400.00
  
```

PAGE TOTAL : 127,800.00

List of Items in Stock For 01/2004

Page : 3

```

-----
<----- Month Details ----->
Itemno  Item Description  Op.Bal Receipts Dispatch Clo. Bal. Rate  Clo. Value
-----
600002  80 pin IDE000        90    30    60      60   50.00   3000.00
600003  Parallel port        115    20    60      75   80.00   6000.00
  
```

PAGE TOTAL : 9,000.00

GRAND TOTAL : 743,300.00

• **Day 2: Lab 1.5: Modify program [Duration 1.5 hours]**

Modify program cobass02 to have three choices as listed below. The program should function as per the steps of choices listed below.

When the user executes the program the following screen is displayed:

<p>A. Input Data B. Print Data Q. Quit.</p> <p>Enter your choice: _</p>

Ensure that the choice entered is A, B, or Q.

Choice 1: If the user enters 'A', then follow the steps given below:

Input data:

Step 1: Open the file "**custseq.dat**" in extend mode.

The data is entered as in COBASS01. However, instead of displaying on the screen it is to be written to a sequential file named "**custseq.dat**" with the same format as the group. The program will then ask "Do you want to enter more (y/n)?".

Step 2: Select an option (Y/N). It will accept the user's response.

- If the user response is Y, then the program will once again take input and write to the file.
- If the user response is N, then the program will close files and go back and display the menu once again.

Choice 2: If the user enters 'B', then follow the steps given below.

Print Data:

Step 1: The program will open the sequential file "**custseq.dat**" in **input mode**, and the "**custrep.dat**" in **output mode**.

- It should read the data sequentially until eof.
- For every valid record read, it should print as per the format given below.
- There should be a maximum of 3 customer records per page. The **Run Date** is the system date.
- At the end of the report, a grand total of the Current Balance field should be printed and all files closed.

-----					Date : dd/mm/yyyy	
ABC CO. LTD.					Page : 999	
List of Customers						
Custno	Customer Name	Region	Address Line 1	Address Line 2	Credit-Limit	Current Bal.
xxxxxx	xxxxxxxxxxxxxxxxxx	xxx	xxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxx	999999	999999

- **Day 4: Lab 1.6: Print Item Class wise Report [Duration 1.5 hours]**

Item's Classes are as follows:

For each item class, print the total closing balance and closing value. At the end of the report, print the grand totals of the closing value.

xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	999999	999999	999999	999999	999.99	9999999.99
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:

Item Class Total :					9999999	99999999.99	

Item Class:	xxxxxxxxxxxxxxxxxxxxxx						
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	999999	999999	999999	999.99	9999999.99	
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:

Item Class Total :					9999999	99999999.99	

Grand Total :						99999999.99	

Day 4: Lab 1.7 Analysis and Debugging[Duration 1 hour]

- **TO DO: The participants have to analyze the program, remove the error and successfully execute the program.**
- **TO DO: The program has to be debugged by the participants till it becomes error-free.**

TO DO: The program should be able to successfully calculate vendor total for every vendor.

TO DO: All the participants have to document the error in excel sheet along with steps taken to resolve the error.

Files for participants: - ReportHandlingProgram and Vendor.Dat File

Day4::Lab 1.8 Report-ControlBreak

Goals	<ul style="list-style-type: none">• The program uses the logic of control break processing for generating the report city wise.
Time	1/2 hour

This program generates the electricity bill city wise for the customer file. SRT_ELEC_REC.DAT sorted file.

TO DO:

The participants have to modify the code according to the changes mentioned in the comments.

Files for participants: - ReportHandlingControlBreak.CBL

Day 5:Lab 1.9: Indexed File from Sequential File

Goals	<ul style="list-style-type: none">• Create an Indexed file from an Items sequential file.
Time	1.5 hour

Note: Take the format for “**itemseq.dat**” and create the “**itemmast.dat**” in the same format, except that the field name prefix “IT-” should be replaced with “ITM-”.

Read the sequential file “**itemseq.dat**” and create an indexed file “**itemmast.dat**” with the same format as the input file, and having index keys as:

Itemno as the record Key.

The program should take care of erroneous transactions like, invalid itemno and duplicates or junk transactions (Blank records), and so on.

Note that Month Receipts and Month Despatches are to be initialized to zero before creation.

At the end of program it should display the following:

- “Transactions Read =”,
- “Transactions Created =” and
- “Transactions Rejected =”

Day 5: Lab 1.10: Create an Indexed File from Sequential File

Goals	<ul style="list-style-type: none">• Create an Indexed file from the customer sequential file.
Time	1.5 hour

T

Note: Take the format for “**custseq.dat**” from program **COBASS02** and create the “**custmast.dat**” in the same format, except that the field name prefix “CU-” should be replaced with “CUM-”.

Read the sequential file “**custseq.dat**” and create an indexed file “**custmast.dat**” with the same format as the input file, and having index keys as:

Custno as the record Key.

The program should take care of erroneous transaction like, invalid custno and duplicates or junk transactions (Blank records) etc.

Note that Current balance is to be initialized to zero before creation.

At the end of program it should display the following:

- “Transactions Read =”,
- “Transactions Created =” and
- “Transactions Rejected =”

Day 5:1.11: Indexed File –Append & Analysis

Goals	• Appending Records to existing File.
Time	1.5 hours

The programs append records to the existing file IN_ELEC_REC.DAT (This file must be given to the participant).

The participants have to use existing IN_ELEC_REC.DAT as the input file. The file has 2 records with key values IN-CUST-ID as 2000, 2001.

The participants have to add a record in the input file having IN-CUST-ID Value less than the existing record keys (less the 2000). The record should be written in the file successfully.

The participants have to analyze the program, remove the error and successfully execute the program.

All the participants have to document the error in the excel sheet along with steps taken to resolve the error.

Day 6: Lab 1.12:Updating Indexed File

Goals	<ul style="list-style-type: none"> Update Index file “itemmast.dat” from a sequential file “GRN.DAT”.
Time	2.5 hours

•

A file called “Goods Receipt Note” has data of the days receipts of various items which have been manufactured. The structure of the *Sequential* GRN file is as shown below:

GR-GRNNO	Grn No.	fpic x(5)
GR-DATE	Date	pic 9(8) yyyyymmdd
GR-ITEMNO	Itemno	pic x(6)
GR-QTYMFD	Qty. Mfd	pic 9(5)

Sample data:

GR-GRNNO	GR-DATE	GR-ITEMNO	GR-QTYMFD
GR001	20040402	600002	100
GR002	20040408	200002	30
GR003	20040408	100005	80
GR004	20040506	400001	75
GR005	20040520	200002	29
GR005	20040520	100005	120
GR006	20040520	100009	100
GR006	20040520	600001	70

Logic steps:

Step 1: Take processing dd mm yyyy as input (for example: 20040403 or 20040408).

Step 2: Check if **grn.dat** contains data for the date input. If not, display suitable message and stop run, otherwise continue to step 3.

Step 3: Read a record from the GRN, fetch the corresponding item record from **itemmast.dat** using **itemno** as the key.

- If the record is not found, then display a suitable message as “Item not found in the item master.. Skipping” + itemno.
- If found then add the qty. manufactured to receipts qty.

Step 5: Read the next GRN record, and repeat steps 2 through 5 until end of file of GRN data.

Day 6: Lab 1.13: Order booking for a Customer

Goals	• Order booking by Marketing for a customer.
Time	3.5 hours

Layout of "ORDERS.DAT":

ORD-ORDNO	Order Number	x(5)	Record key
ORD-DATE	Order date	9(8)	yyyymmdd
ORD-CUSTNO	Order Customer no.	x(6)	
ORD-TOTVALUE	Order Total Value	9(7)v99	
ORD-TX1-PER	Tax Percentage – Tax 1	99v99	
ORD-TX2-PER	Tax Percentage – Tax 2	99v99	
ORD-TX3-PER	Tax Percentage – Tax 3	99v99	
ORD-TOTSUPVAL	Order Total Supplied Value	9(7)v99	
ORD-CUSTREFNO	Customer Order Ref. No.	x(10)	
ORD-STATUS-TAG	Order Status Tag	9	0 – New, 1-Partly Supplied 2 – Fully Supplied.

Layout of "ORDITEM.DAT":

OIT-ORDNO	Order No.	x(5)	OIT-KEY
OIT-ITEMNO	Item Number	x(6).	Record Key
OIT-ITEMNO1	Item Number	x(6).	OIT-ALTKEY-1 Alternate Key with duplicates
OIT-DATE	Order date	9(8)	yyyymmdd
OIT-QTY	Item Order Quantity	9(6)	
OIT-UOQ	Item UOQ	X(3)	
OIT-RATE	Item Negotiated Rate	9(5)v99	
OIT-QTYSUPP	Item Qty. Supp	9(6)	
OIT-AMTBILL	Item Value Billed	9(7)v99	
OIT-STATUS-TAG	Order-Item Status Tag	9	0 – New, 1-Partly Supplied 2 – Fully Supplied.

The program should take input from the keyboard of a customer's order and create the order on an index file called "**ORDERS.DAT**" with order header information. The order status tag should be set to '0'.

Multiple items may be ordered against a single order. The same item may not be ordered multiple times against the same order. The details of the items of the order are written in to the order child table called "**ORDITEM.DAT**".

Sample Data:

ORD-ORDNO	ORD-CUSTNO
OR001	DA7201
OR002	AB0034

OIT-ORDNO	OIT-ITEMNO	OIT-QTY	OIT-RATE
OR001	100005	100	800
OR001	200002	2	375
OR002	100005	60	875
OR002	400001	25	7900
OR002	600001	40	40

Steps for accepting a new order are as follows:

Step 1: Accept the **ORD-ORDNO** and **ORD-CUSTNO** from the user. Move the **system date** to **ORD-DATE**.

Step 2: Repeat the following steps for each item of a given order:

- Move **ORD-ORDNO** to **OIT-ORDNO**, and **ORD-DATE** to **OIT-DATE**.
- Accept the **OIT-ITEMNO** from the user, and check if it exists in the **ITEMFILE**.
- If it is a valid item (exists in the **ITEMFILE**), then accept the **OIT-QTY**, **OIT-RATE**, **OIT-UOQ** (IGATE, Doz) from the user.
- Move 0 to **OIT-STATUS-TAG**.
- Write the record.
- If the record is successfully written, then compute the item value ($OIT-QTY * OIT-RATE$).
- Add the item value to **ORD-TOTVALUE**.

Step 3: After all the items of an order are entered, move 0 to **ORD-STATUS-TAG**, and write the record to **ORDERS.DAT**.

The user is prompted to enter the order number.

- If the order number is valid, then the user is asked to enter the item no.
- If the record is found, then the user should be allowed to modify the order quantity if the order status is '0' viz 'New'.
- If the order number is invalid, then an error message should be displayed.

The user is prompted to enter the order number.

- If the order number is valid, then the order details are displayed.
- If the order number is invalid, then an error message should be displayed.

Note: This can also be done using one file with two formats, that is two level 01 record description having some common fields and rest depending on the data – **one record** describing the contents of **orders.dat** and the **other record** describing the contents of **orditem.dat**.

Day 6: Lab 1.14 Enhancement Assignment

Goals	• Adding new Functionality to exsisting Lab
Time	1 hour

Add new functionality in the program i.e. 'Modify an existing order'
It should be the second option in the menu.
Accordingly modify the menu

When the user choose this option:

The user is prompted to enter the order number. If the order number is valid, the user is asked to enter the item no. If the record is found the user should be allowed to modify the order quantity if the order status is '0' viz 'New'. If the order number is invalid, an error message should be displayed.

Files for participants: - Enhancement.CBL, ORDERS.DAT ,ORDITEM.DAT

Day 7:Lab 1.15 Analysis and Debugging

Goals	<ul style="list-style-type: none">This program takes Current-date and Bill-month as input from user and generates Telephone bill for that month for every customer included in an input file .
Time	1.5 hours

TO DO: The participants have to analyze the program, remove the error and successfully execute the program.

TO DO:All participants have to document an error in the excel sheet along with steps taken to resolve the error.

Files for participants: - - CurrentDateBillMonth.CBL and CUSTREC.DAT
Sample Report:- SAMPLE_TELBILL.REP

Day 7:Lab 1.16 Bank Transaction Updation and Report

Goals	A program to read Account Transaction file, Generate a Report & Update the Account Master file. •
Time	8 hours

File Structure
1.1 Transaction File Record Structure

Transaction file is a sequential file of LRECL 80.

Valid Account Types are

- 'CC' – Credit Card
- 'SB' – Savings Acct
- 'CR' – Current Acct

Valid Payment Modes are

- 'CS' – Cash
- 'CQ' – Cheque

```

01 ACCOUNT-TRANS.
05 AT-ACCT-NO          PIC S9(09) COMP-3.
05 AT-ACCT-TYPE        PIC X(02).
05 AT-DUE-DATE.
    10 AT-DUE-YYYY      PIC 9(04).
    10 FILLER           PIC X.
    10 AT-DUE-MM        PIC 9(02).
    10 FILLER           PIC X.
    10 AT-DUE-DD        PIC 9(02).
05 AT-TRANS-DATE.
    10 AT-TRANS-YYYY    PIC 9(04).
    10 FILLER           PIC X.
    10 AT-TRANS-MM      PIC 9(02).
    10 FILLER           PIC X.
    10 AT-TRANS-DD      PIC 9(02).
05 AT-DUE-AMT          PIC S9(09) COMP-3.
05 AT-AMT-CR           PIC S9(09) COMP-3.
  
```

```
05 AT-AMT-DB          PIC S9(09) COMP-3.  
05 AT-PAY-MODE        PIC X(02).  
05 FILLER              PIC X(36).
```

1.2 Master File Record Structure

Master file is an Indexed file of LRECL 100 with AM-ACCT-NO as the Key.

Valid Account Status are

‘ACT’ – Active Account

‘BLK’ – Blocked Account

```
01 ACCOUNT-MAST.  
05 AM-ACCT-NO          PIC S9(09) COMP-3.  
05 AM-ACCT-TYPE        PIC X(02).  
05 AM-ACCT-STATUS      PIC X(03).  
05 AM-NAME.  
    10 AM-FNAME         PIC X(15).  
    10 AM-LNAME         PIC X(15).  
05 AM-DOB              PIC X(10).  
05 AM-AVAIL-BAL        PIC S9(09) COMP-3.  
05 AM-CONTACT-NO       PIC 9(10).  
05 AM-ADDRESS.  
    10 AM-ADDR1          PIC X(10).  
    10 AM-ADDR2          PIC X(10).  
05 FILLER              PIC X(15).
```

1.3 Report REPT of LRECL 160.

2. Program Specification:

1. Main Program: DREPPGM

Read transaction file...

Step 1: For ‘CC’ account type

- 1a. if the due amt \leq 0, read next record.
- 1b. if the due amt $>$ 0 & due date is prior to current date & credited amt $<$ due amt, then:
- Add credited amt to available balance, change acct status to 'BLK' & update master file.
 - Generate a report for the blocked account.
- 1c. if the due amt $>$ 0 & due date is prior to current date & credited amt \geq due amt, then:
- Add credited amt to available balance, change acct status to 'ACT' & update master file.
 - Generate a report.
- 1d. if the due amt $>$ 0 & due date is current date & credited amt $<$ due amt, then:
- Add credited amt to available balance & update master file.
 - Generate a report.
- 1e. if the due amt $>$ 0 & due date is current date & credited amt \geq due amt, then:
- Add credited amt to available balance & update master file.
- 1f. if the due amt $>$ 0 & due date is greater than current date & credited amt $<$ due amt, then:
- If due date is within the next 2 days from current date then:
 - Add credited amt to available balance & update master file.
 - Generate a report.
 - Else
 - Read next record.
- 1g. if the due amt $>$ 0 & due date is greater than current date & credited amt \geq due amt, then:
- Add credited amt to available balance & update master file.

Step2: For 'SB' or 'CR' account type

2a. if credited amt > 0 & debited amt = 0, then:

- Add credited amt to available balance & update master file.
- Generate a report.

2b. if credited amt = 0 & debited amt > 0, then:

- Subtract debited amt from available balance & update master file.
- Generate a report.

2c. if credited amt > 0 & debited amt > 0, then:

- Add credited amt & subtract debited amt from available balance & update master file.
- Generate a report.

2. Sub-program: VALIDATE

Accept Due Date from Driver Pgm (DREPPGM)

Step 1. Date Validation.

Step 2. Leap Year Validation.

Step 3. Validate to check whether due-date falls within the coming 2 days from the current date.

Day 8: Lab 1.17 Working with 3 Indexed Files

Goals	<ul style="list-style-type: none"> Retrieving Data from 3 indexed files
Time	6 hours

iLearn is required to generate a Purchase Report two months before the beginning of each semester to its MS program. This report lists the books that have to be purchased for the coming semester. Trainers/lecturers requirements of books are stored in a Requirements file. This file contains details of the lecturers' book requirements for semesters 1 and 2. There are two other files: Publisher and Book files, which contain publisher and book details respectively.

The participants have to analyze the program and document the Logic of the program.

Wherever require, participants have to add comments to increase the understandability of the program.

Requirements File [Indexed]

There is a record for each book title required by a lecturer.

Note: A book may be required by more than one lecturer.

File Structure

Field	Key Type	Type	Length	Value
PR-Number	Primary	9	4	1-9999
Trainer	Alt with duplicates	X	20	--
Book-id	Alt with duplicates	9	4	1-9999
Copies-required	--	9	3	1-9999
Semester	--	9	1	1/2

Sample Data for the Requirement File

0001	GERARD	0012	025	1
0002	HUTCHISON	0112	150	1
0003	KINSELLS	1111	050	1

Book File [Indexed]
File Structure

Field	Key Type	Type	Length	Value
Book-id	Primary	9	4	1-9999
Publisher-Number	Alt with duplicates	9	4	1-9999
Title	--	X	30	--

Sample Data for the Book File

0012	1111	**NO MATCHING**
0112	2222	FRENCH

1111	2222	MATHS
------	------	-------

Publisher File [Indexed]

File Structure

Field	Key Type	Type	Length	Value
Publisher-Number	Primary	9	4	1-999
Publisher-Name	Alt with duplicates	X	20	--
Publisher-Address	--	X	40	--

Sample Data for the Publisher File

1111	QUE	ENGLAND
2222	PRENTICE	USA
3333	MICROSOFT	USA
4444	MCGRAW-HILL	INDIA

Note :

Accept the semester from the user.

Generate the requirements report that shows book requirements for the semester entered by the user. The report should be sequenced on ascending publisher name and should only list purchase requirements for the semester asked by the user.

Each publisher should appear on a new page.

Assumptions: One book may be required by multiple trainers.

Files for participants: - PurchaseReport.CBL and all input files

Day 8: & Day 9:Lab 1.18 Online Bus Booking

Goals	.Program to read Booking Trans file, Check the availability of the seat in the Bus Detail file, Generate Ticket no., PNR no., Update the Booking Master file, Update Bus Detail file, Print Tickets & Print canceled report in case of cancellation.
Time	8 hours

1. File Structure
1.1 Booking Transaction File Record Structure

Booking Transaction file is a sequential file of LRECL 120.

```

01 BOOK-TRANS-REC.
  05 BT-PNR-NO      PIC X(15) VALUE SPACES.
  05 BT-PASSENGER-NAME.
    10 BT-FNAME     PIC X(15) VALUE SPACES.
    10 BT-LNAME     PIC X(15) VALUE SPACES.
  05 BT-GENDER      PIC X(01) VALUE SPACES.
  05 BT-CONTACT-NO  PIC 9(10) VALUE ZEROES.
  05 BT-AGENT-NO    PIC 9(05) VALUE ZEROES.
  05 BT-BUS-TYPE     PIC X(03) VALUE SPACES.
  05 BT-FROM-LOC     PIC X(10) VALUE SPACES.
  05 BT-TO-LOC      PIC X(10) VALUE SPACES.
  05 BT-SEAT-NO     PIC X(02) VALUE SPACES.
  05 BT-TRAVEL-DATE PIC X(08) VALUE SPACES.
  05 BT-TRAVEL-TIME.
    10 BT-TRAVEL-TIME-HH PIC 9(02) VALUE ZEROES.
    10 BT-TRAVEL-TIME-MM PIC 9(02) VALUE ZEROES.
  05 FILLER         PIC X(22) VALUE SPACES.
  
```

1.2 Master File Record Structure

Booking Master file is an Indexed file of LRECL 180 with BM-PNR-NO as the Key.

```

01 BOOK-MAST-REC.
  05 BM-PNR-NO      PIC X(15) VALUE SPACES.
  05 BM-TICKET-NO   PIC X(12) VALUE SPACES.
  05 BM-STATUS      PIC X(05) VALUE SPACES.
  05 BM-BUS-NO      PIC X(10) VALUE SPACES.
  05 BM-BUS-NAME    PIC X(10) VALUE SPACES.
  05 BM-AGENT-NO    PIC 9(05) VALUE ZEROES.
  05 BM-SEAT-NO     PIC X(02) VALUE SPACES.
  05 BM-FROM-LOC    PIC X(10) VALUE SPACES.
  05 BM-TO-LOC      PIC X(10) VALUE SPACES.
  05 BM-BOOK-DATE   PIC X(08) VALUE SPACES.
  05 BM-CANCL-DATE  PIC X(08) VALUE SPACES.
  05 BM-TRAVEL-DATE.
    10 BM-TRAVEL-DD  PIC X(02) VALUE SPACES.
    10 BM-TRAVEL-MM  PIC X(02) VALUE SPACES.
    10 BM-TRAVEL-YYYY PIC X(04) VALUE SPACES.
  05 BM-TRAVEL-TIME  PIC 9(04) VALUE ZEROES.
  05 BM-FARE         PIC ZZZ9(02).99 VALUE ZEROES.
  05 BM-CANCL-AMT    PIC ZZZ9(02).99 VALUE ZEROES.
  05 BM-PASSENGER-NAME.
    10 BM-FNAME      PIC X(15) VALUE SPACES.
    10 BM-LNAME      PIC X(15) VALUE SPACES.
  05 BM-CONTACT-NO   PIC 9(10) VALUE ZEROES.
  05 BM-BUS-TYPE     PIC X(03) VALUE SPACES.
  05 FILLER          PIC X(14) VALUE SPACES.
  
```

1.3 Bus Detail File Record Structure

Bus Detail file is a sequential file of LRECL 160.

```

01 BUS-DETAIL-REC.
  05 BD-BUS-NO      PIC X(10) VALUE SPACES.
  05 BD-BUS-NAME    PIC X(10) VALUE SPACES.
  05 BD-BUS-TYPE    PIC X(03) VALUE SPACES.
  05 BD-AVAIL-SEATS  PIC X(100) VALUE SPACES.
  05 BD-FROM-LOC    PIC X(10) VALUE SPACES.
  05 BD-TO-LOC      PIC X(10) VALUE SPACES.
  05 BD-FARE        PIC ZZZ9(02).99 VALUE ZEROES.
  05 FILLER         PIC X(09).
  
```

1.4 Report PRNTTKT of LRECL 122.

1.5 Report RFNDTKT of LRECL 122.

2. Program Specification:

1. Program: BUSBOOK

Read booking transaction file...

Step 1: For new bookings

1a. Read bus detail file & Check the availability of Bus & Seat no.

1b. If the preferred bus type & preferred seat no is available for the said date of route.

- Book the ticket (Generate Ticket #, PNR #).
- Update Booking History file.
- Update Bus Detail file
- Print Ticket.

1c. If the preferred bus type & preferred seat no is not available for the said date of route. Book any other available seat no.

- Book the ticket (Generate Ticket #, PNR #).
- Update Booking History file.
- Update Bus Detail file
- Print Ticket.

Step2: For Cancellation

2a. If the cancellation date is less than 24 hrs from the date of journey.

- Print cancellation report without any refund amount.

2b. If the cancellation date is more than 24 hrs from the date of journey. Deduct 15% of the fare

Towards cancellation charges.

- Print cancellation report with the refund amount.

Day 9:Lab Assignment 1.19:Call Statement-Debugging

Goals	• Call Statement
Time	15 minutes

.If program A is called with a call is A using X,Y.Refer to below code
Linkage-Section.

05 M

05 P

Procedure Division using X,Y

----- EXIT.

Debug the program and rectify the error

Day 9:Lab Assignment 1.20:Call Statement-Enhancement Assignment

Goals	• Calling Sub programs
Time	2 hours.

1. Write a separate program which will calculate the tax for the employee and try to call the same program in the EMP file created.
2. Write a COBOL program (using a subroutine to calculate the Gross) for an employee of a Company whose income details (Basic Pay, HRA, PF, DA) are given in the file INCOME.DAT, create the gross pay in OUTPUT.DAT.
 - a. INPUT.DAT
 - b. Emp1 12000 4000 2000 7000
 - c. Emp2 12500 3750 2300 6450
 - d. Emp3 12340 3750 2345 5430
 - e. Emp4 13450 3540 2340 6000
 - f. Emp5 12000 4000 2300 6500
 - g. OUTPUT.DAT should contain the employee name and gross pay.
3. Redo assignment 3 using the CALL statement. The user should be able to select from the menu. The selected menu option should be available as a separate program and called from the main program.

Day 10: Lab 1.21: Table Handling Lab Assignment

Goals	• Working with Arrays (Table Handling)
Time	2 hours

1) Compute the weekly pay for each employee, the rate of pay as found in table TABPAY is multiplied by the number of hours worked (a three-position field with one decimal position). The weekly pay amount is a five-position field with two decimals after rounding.

TABLE

	TABEMP	TABPAY
1001		5.25
1002		6.25
1003		5.40
1004		4.50
1005		5.50
1006		4.75
1007		6.50
1008		4.50
1009		5.00
1010		8.00

Input	Positions	Field
	1 - 4	Employee number
	35 - 37	Hours XX.X

Calculations.

Perform a search of the table using employee number as the search word. After a successful search, perform the necessary calculations.

Output

Leave five positions between each field. Output the employee number, hours worked, rate off pay, and gross pay.

Output is as follows.

EMPLOYEE NUMBER	HOURS WORKED	RATE OF PAY	GROSS PAY
1001	15.5	5.25	81.38
1002	31.5	4.50	146.25
1003	40.0	5.50	220.00
1004	5.0	6.50	31.50
1010	25.0	8.00	200.00

2) A table stores Loan Account Number and Loan Outstanding amount (In lakhs). Use SEARCH to find the number of people who have an outstanding amount of
 Over 10 lakhs
 Between 5 – 10 lakhs (Both inclusive).

Less than 5 lakhs

Day 10: Lab Assignment 1.22:Table Handling- Analysis Assignment

Goals	<ul style="list-style-type: none">The referenced programs have to be analyzed and have to made error free by the participants.
Time	1.5 Hours

Define a 20 items of static Data of Country code and Country Name in the program. Write a program to find out Country Code based on Country Name. The program must use the SEARCH and Arrays statements.

Static Data of the Country Code and Country Name

```
00001USA
00002United Kingdom
00003France
00004China
00005Japan
00006Italy
00007Russia
00008India
00009Nepal
```

Files for participants: - Countrycode.Dat file, Lab22.cbl.Perform Code Review using Code Review excel.

Day 10: Lab 1.23: String Handling

Goals	• Implementing String Handling
Time	1.5 Hours

StreetNo	pic 9(3)
StreetName	pic x(15)
City	pic x(10)
Pincode	pic 9(6)

Print the output in the following format:

StreetNo, StreetName

City – Pincode

1. String four variables A1, B1, C1 and D1 whose values are A1 = BHARAT, B1 = SANCHAR, C1 = NIGAM and D1 = LIMITED into a variable RESULT.
2. UNSTRING the RESULT variable of previous program and store the values in 4 different variables.

Day 11:Lab 1.24: Mobile Services

Goals	<ul style="list-style-type: none"> To develop mobile services which implements all the concepts covered in Cobol.
Time	7 Hours

- Design a Sequential file which will give the details of Customer like CustomerCode, Customer Mobile Number, Customer Name, Address, City, Country, Pincode, Email, Planopted, ServiceType
- Sort the file based on CustomerCode.
- Include Input procedure by selecting data for specific plan (say Plan 99)
- Create an Indexed File for Payment options which includes fields like Payment Date, Mobile No, CustomerCode, Mode of Payment, DueDate, PaymentDate, AmountPaid, BillAmount, Balance amount, paymentoption(direct debit, cash to mobile, vodaphone shop, telebanking, ATM. For e.g for a specific customer if the amountpaid is 200 Bill amount 300 balance has to be 100.
- Include one field of Late Payment charges with following criteria

	Outstanding amount	Late payment Charges
1	Less than Rs. 100/-	Nil
2	>=Rs. 100/- but less than Rs. 300/-	Rs. 25/-
3	>= Rs. 300/- but less than Rs. 500/-	Rs. 50/-
4	>= Rs. 500/-	Rs. 100/-

- Include 5 records which has done late payment for bill and which fulfill the above criteria
- Vodafone has decided instead of country code MUM they will use only M. Implement that change in Sequential file.
- Create an Indexed file of Services which include Customercode and ServiceType from Customer file. This file will give us the information of Services whether opted for prepaid or postpaid. It may include fields like type of service(prepaid, postpaid) rates for STD, rates of ISD, rates of SMS, incoming calls, outgoing calls, roaming charges.
- Create one indexed file called CallsMade which include call details like calls made from, to, duration, cost(cost will be calculated based on the service they have opted and the std rates, isd rates or roaming charges as applicable.
- Generate the following reports
 - To display details of all customers who reside in Mumbai
 - To display details of all customers who have outstanding amount of 1000
 - To display only those customers who have opted for prepaid service
 - To display the number of calls made for a specific month by a specific customer in short all calls made for a specific month.

Analysis	<p>Application Functionality</p> <p>Provide a Summary Comments in the module providing the application overview</p>
Enhancement	<p>Add new features</p> <p>To display all customers who have paid the bills through ATM and who are from India</p>
Problem Solving	<p>Solve current bugs</p> <p>The total bill amount for the customer is invalid based on the published rates.</p> <p>As shown below:</p> <p>Pre-paid Local Call: 1.4 STD rates : 1.3 ISD rates: 3.1</p> <p>Post-paid Local Call: 1.5 STD rates: 1.5 ISD rates: 3.5</p> <p>For a customer the bill should have been X but it is Y Reason: Switching of the Pre Post rates in the files</p>
Debugging	<p>Test Runs</p> <p>Use debugger for the above problem.</p>

- **Stretched Assignment 1:**
- **Itemwise Sales Order Report**

This program prints the Item wise sales orders report for a given period. The report takes data from **orditem.dat** and prints in the format given below.

ABC CO. LTD.				Date : dd/mm/yyyy		
Items wise Sales From dd/mm/yyyy to dd/mm/yyyy				Page : 999.		
Itemno	Item Description	Date	UOQ	Ord-Qty.	Rate	Order-Value
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	dd/mm/yyyy	xxx	999999	999.99	9999999.99
		dd/mm/yyyy	xxx	999999	999.99	9999999.99
		dd/mm/yyyy	xxx	999999	999.99	9999999.99
		:	:	:	:	:
		:	:	:	:	:
Item Totals :			9999999		99,999,999.99	
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	dd/mm/yyyy	xxx	999999	999.99	9999999.99
		dd/mm/yyyy	xxx	999999	999.99	9999999.99
		dd/mm/yyyy	xxx	999999	999.99	9999999.99
		:	:	:	:	:
		:	:	:	:	:
		:	:	:	:	:
Item Totals :			9999999		99,999,999.99	
Grand Total :						99,999,999.99

The program asks for input of start date and end date. It should do a date check of the two dates.

It then starts the **orditem.dat** file to the start date using the alternate keys oit-altkey-1 (Itemno + Date). It should also handle error in case of error in start.

The program should then access the data sequentially. Verify that the data is for the dates in the range, if not then skip the records.

For a valid record, if it is the 1st record of the item, then print the **ItemNo** and the **Item description** (which you get from "**itemmast.dat**"). Otherwise, for other records, print the details without the **Itemno** and **description**.

Take totals of the order-value and quantity. At the change of item, print the Order total as per the report on top.

At the end of the report, print the last order total and the grand total of the value only.

The reading of the file continues until End of File.

- **Stretched Assignment 2: Write a program to print the following report using the file created in program COBASS2.**

ABC CO. LTD.	Date : dd/mm/yyyy
Region wise report	
<hr/>	
Region : XXX	
Customer Name	Address
xxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxxxxxx
:	:
<hr/>	
Total customers in XXX : 999	
<hr/>	
Region : XXX	
Customer Name	Address
xxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxx
	xxxxxxxxxxxxxxxxxx
:	:
<hr/>	
Total customers in XXX : 999	
<hr/>	
Total customers : 9999	
<hr/>	

- **Case Study**

1. Write a program to input the data given below.

Program-Id : STUASS01

ST-ROLLNO	Student rollno	pic 9(4)
ST-NAME	Student name	pic x(15)
ST-CLASS	Student class	pic x(3)
		First 2 digits Std
		3 rd character division.
ST-SUB1-MARKS	Marks of subject 1	pic 9(3).
ST-SUB2-MARKS	Marks of subject 2	pic 9(3).
ST-SUB3-MARKS	Marks of subject 3	pic 9(3).
ST-FEES-PAID	Fees paid	pic 9(4)

Logic :

- The data should be of a record structure as above.
- Std can have values in the range 1-10. Division can have values 'A' or 'B'.
- Marks should be in the range 0-100.
- Input the details for the above record. Once the data of a record is complete, the program should display the full group as an item.

- The program should then ask the user "Do you want to Input More (Y/N)". It should accept the users response.
- If the response is "Y", the program should accept the details on another item and repeat the above steps .
- If the response in "N", the program should stop.

1. Write a program to input the data given below.

Program-Id : STUASS02

MNT-ROLLNO		pic 9(4)
MNT-FLAG	UPDATION FLAG	pic x
		A-Add
		M-Modify (only name can be modified)
		D-Delete
MNT-NAME	STUDENT NAME	PIC X(15)

Logic :

- The data should be of a record structure as above.
- MNT-FLAG can have values the values A, M or D.
- Input the details for the above record. Once the data of a record is complete, the program should display the full group as an item.
- The program should then ask the user "Do you want to Input More (Y/N)". It should accept the users response.
- If the response is "Y", the program should accept the details on another item and repeat the above steps .
- If the response in "N", the program should stop.

3. Modify program STUASS01 to have display the grade of each student.

Program-Id: STUASS03

- Grades are awarded on the following basis :

Percentage	Grade
>= 70	DIST
60-70	CRDT
50-60	PASS
<50	FAIL

If a student scores below 40 in any subject, award a FAIL grade.

At the end display the no. of students passed and failed.

4. Modify program STUASS01 to have three choices as listed below. The program should function as per the steps of choices listed below.

Program-Id : STUASS04

- A. Input Data
- B. Print Data
- C. Quit.

Input data :

- Open the file "studseq.dat" in extend mode.
- The data is entered as in STUASS01 but instead of displaying on the screen it is to be written to a sequential file named "studseq.dat" with the same format as the group.
- When the user enters "Y" to "Do you want to enter more (y/n) ?", the program should once again take input and write to file.
- With user input as "N", the program should close files and go back and display the menu once again.

Print Data

- The program should open the file "studseq.dat" in sequential input mode and the "studseq.dat" in output mode.
- Accept the class from the user & print a report for all students in the class.
- For every valid record read, it should print as per the format given below.
- There should be a max of 5 lines per page.
- The report should be printed in sequence of Rollno.
- At the end of the report, a grand total of the value filed field should be printed and all files closed.

Date : dd/mm/yyyy

Scholars' Academy

Marksheet for Standard : 99 Division :X

Rollno	Student Name	Subject 1	Subject 2	Subject 3	Grade
9999	XXXXXXXXXXXXXXXX	999	999	999	XXXX
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
Total no. of students :					999

Quit :

The program should stop run.

5. Modify program STUASS02 as listed below.
Program-Id : STUASS05

- Open the file "mntseq.dat" in extend mode.

- Print the following standard wise report.**

Scholars' Academy

List of students

RollNo	Name
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50
51	51
52	52
53	53
54	54
55	55
56	56
57	57
58	58
59	59
60	60
61	61
62	62
63	63
64	64
65	65
66	66
67	67
68	68
69	69
70	70
71	71
72	72
73	73
74	74
75	75
76	76
77	77
78	78
79	79
80	80
81	81
82	82
83	83
84	84
85	85
86	86
87	87
88	88
89	89
90	90
91	91
92	92
93	93
94	94
95	95
96	96
97	97
98	98
99	99
100	100

* * * * *
 * * * * *
 * * * * *
 * * * * *

RollNo	Name
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50
51	51
52	52
53	53
54	54
55	55
56	56
57	57
58	58
59	59
60	60
61	61
62	62
63	63
64	64
65	65
66	66
67	67
68	68
69	69
70	70
71	71
72	72
73	73
74	74
75	75
76	76
77	77
78	78
79	79
80	80
81	81
82	82
83	83
84	84
85	85
86	86
87	87
88	88
89	89
90	90
91	91
92	92
93	93
94	94
95	95
96	96
97	97
98	98
99	99
100	100

■	■
■	■
■	■
■	■

99

7. Print the following standard wise, division wise report.

Scholars' Academy
Consolidated Marksheet

STD : 99

Div : A

RollNo	Name	Grade
9999	XXXXXXXXXXXXXXXXXX	XXXX
:	:	:
:	:	:
:	:	:

Total No. of students in Div A : 99

Div : B

RollNo	Name	Grade
9999	XXXXXXXXXXXXXXXXXX	XXXX
:	:	:
:	:	:
:	:	:

Total No. of students in Div B : 999

Total No. of students in Std 99 : 999

:	:	:
:	:	:
:	:	:
:	:	:

Distinction : 99
Credit : 99
Pass : 99
Fail : 99

8. Indexed files

Program-Id : STUASS08

Note : Take the format for “studseq.dat” from program STUASS01 and create the “studmast.dat” in the same format, except that the field name prefix “ST-“ should be replaced with “STM-“.

Read the sequential file “studseq.dat” and create an indexed file “studmast.dat” with the same format as the input file, and having index keys as ;

STM-ROLLNO as the record Key.
STM-CLASS as the alternate record key.

The program should take care of erroneous transaction like, invalid class and duplicates or junk transactions (Blank records) etc. write the erroneous records to the file studerr.dat . This file will have an additional field
Remark Pic X(15)

The values for remark can be any of the following
Duplicated
Out of sequence
The access mode of the file should be sequential.

At the end of program it should display the following

"Student Details Entered =",
"Student Records Created =" and
"Student Records Rejected ="

9. Modify program STUASS6 (single level control break).

Program-Id : STUASS09

Accept a the CLASS from the user.
Print a report for the given CLASS.

10. Using the files created in program STUASS05 (mntseq.dat) and STUASS08 (studmast.dat).

Program-Id : STUASS10

Read a record from the maintenance file (mntseq.dat).
For each record read, add/modify/delete the record in the master file (studmast.dat). Repeat this till the end of the maintenance file.

11. Using the file created in program STUASS04 (studseq.dat). This program will have 3 options, as mentioned below :

Program-Id : STUASS11

- A. Create transaction
- B. Update Master
- C. Exit

Create transaction

Write a program to create the following file (studtran.dat)

Rollno PIC 9(4)
Fees_paid Pic 9999

Note : Enter some invalid rollnos. i.e. which do not exist in the studseq.dat.
There can be multiple transaction records for a student.

Update Master

Write a program to update the studseq.dat file using the studtran.dat.

11. For each Class(irrespective of division) find the highest scorer in each subject.

Program-Id : STUASS11

Print the details in the following format

Class	Subject 1		Subject 2		Subject 3	
	Max	Name	Max	Name	Max	Name
99	99	XXXXX 99	XXXXX 99	XXXXX		
:	:	:	:	:	:	:
:	:	:	:	:	:	:

Assumption: There is only 1 highest scorer in each class.

13. Write a program to accept a company name and display it in the center of the screen.

Note: The user may enter leading/trailing spaces.

14. Modify the second option (Print Data) of the program STUASS04 to receive the class from the main program and print the report.

15. Accept details from the user in the following format :

```

StreetNo          pic 9(3)
StreetName        pic x(15)
City              pic x(10)
Pincode           pic 9(6)
  
```

Print the output in the following format

StreetNo, StreetName

City - Pincode

-----*****-----

Appendices

- **Appendix A: Breakup of Marks**

Each assignments carries 30 marks.

Breakup up of marks is as follows:

- Flowchart/Logic – 5
- Standards - 5
- Coding – 10
- Error Handling - 5
- Output - 5

- **1. Compiling, Linking, and Executing COBOL programs**

Creating Source Code:

Step 1: Run the **CA-Realia workbench** software. Select **file** → **new** and save the file with extension as **.cbl**. On saving the file as **.cbl** it highlights all the COBOL keywords.

Compiling and Linking:

Step 1: Select the menu option **Options** → **Other** from the **Build** menu, and compile the **Source code**.

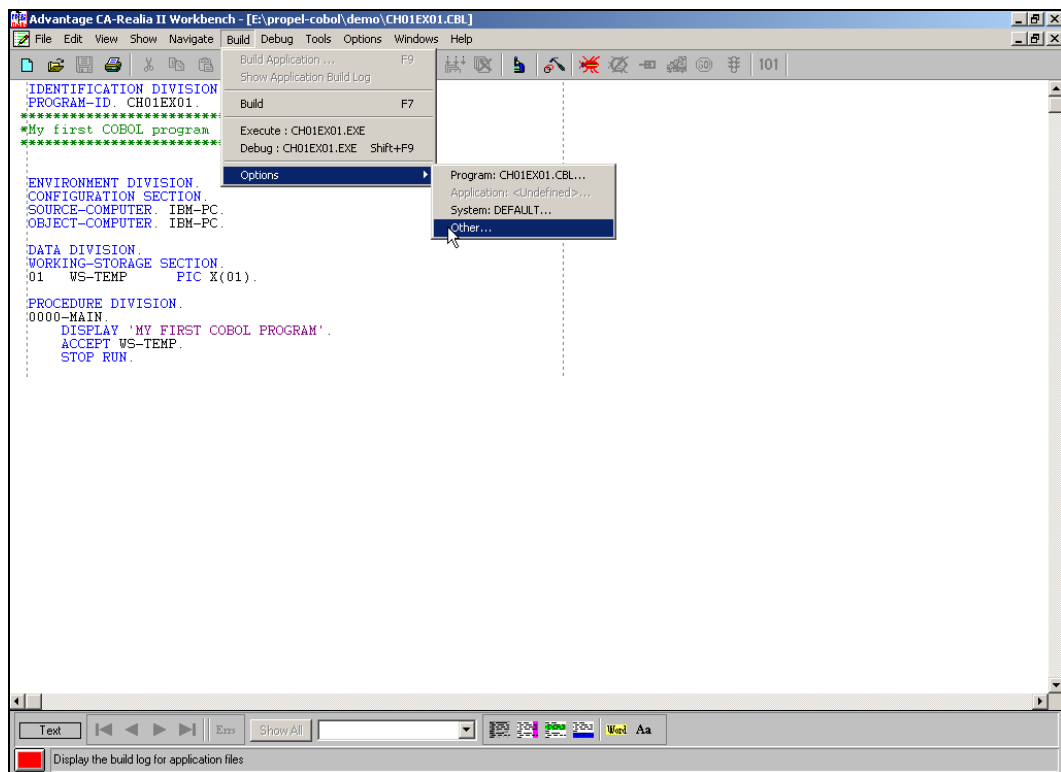


Figure 1: Creating Source Code

Step 2: Select the application type as EXE Application.

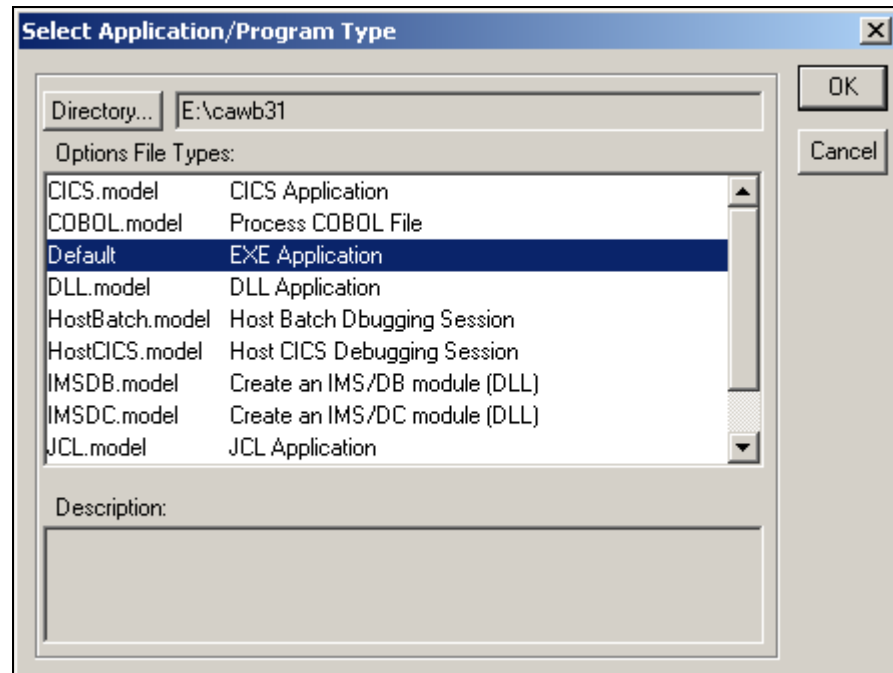


Figure 2: Select Application/Program Type

Step 3: Click **OK**. Select the **Link** tab. Click the **.EXE/LIB** button, and select the path for the file. The exe file will be generated in this directory.

(**Note:** Preferably select the same directory where you have your .cbl file)

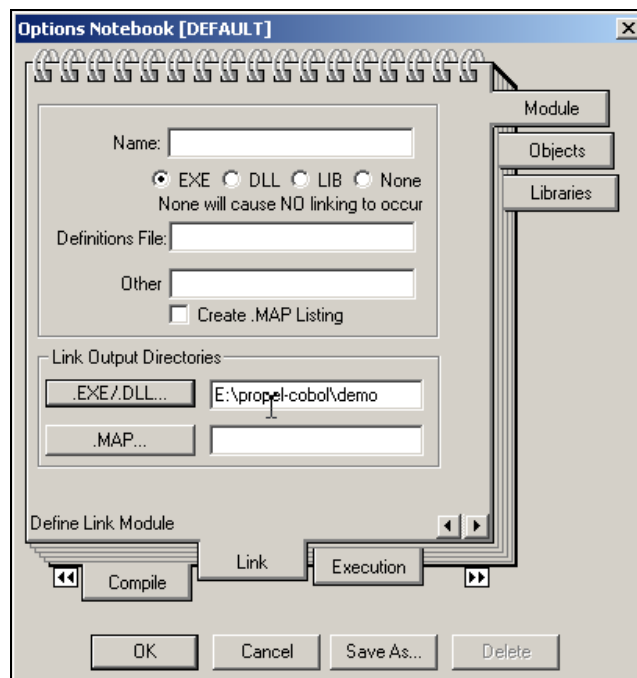


Figure 3: Options Notebook

Step 4: Click **OK**. Select **Build** → **Build**. On successful compilation, the message as shown in the following screenshot is displayed. In this message window, click **No**.



Figure 4: Compile Successful message

Run:

Let us now create the execution profile.

Step 1: Select the following option in **CA-Realia**. Select **Build** → **Options** → **Program** for building the execution profile.

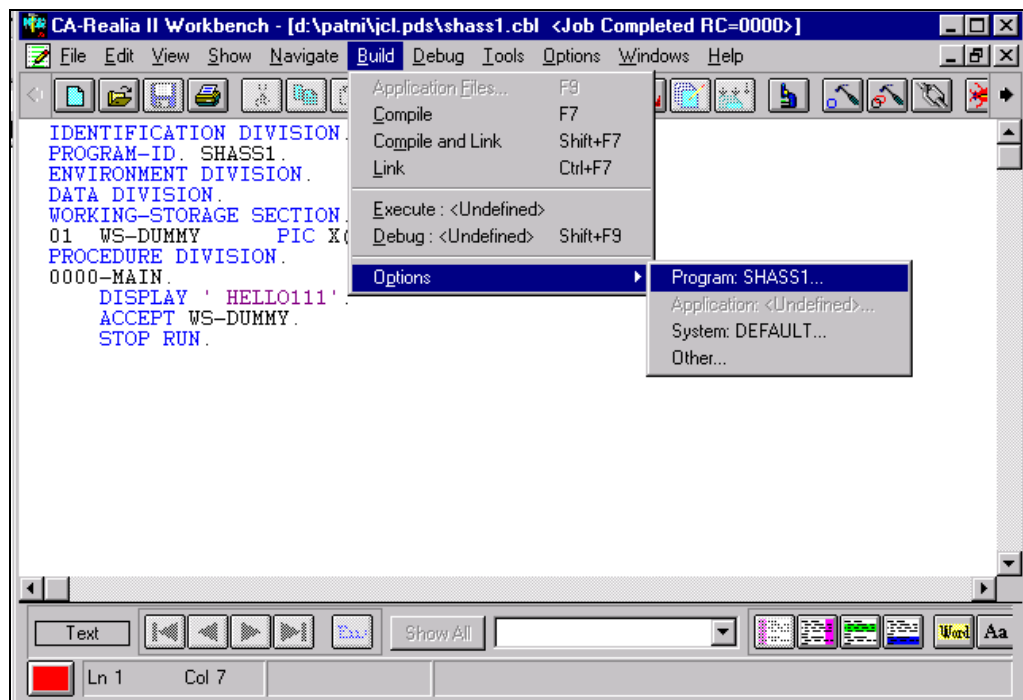


Figure 5: Building execution profile

Step 2: Select the **Execution** tab from **Realia Cobol session** options, and create a new profile.

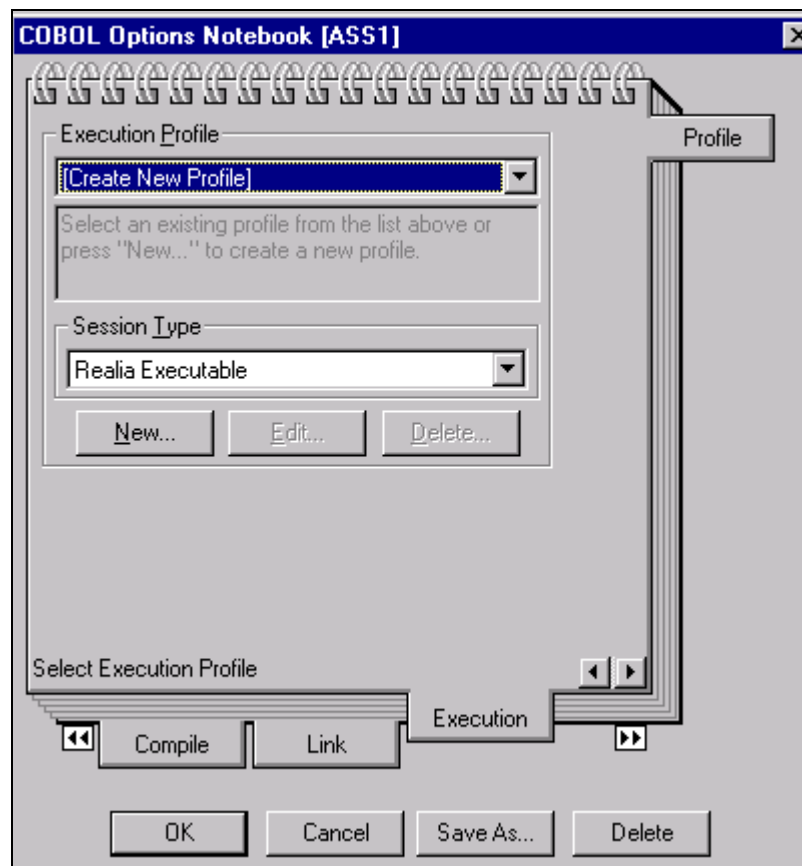


Figure 6: Execution tab

Step 3: Key in the executable filename and path.

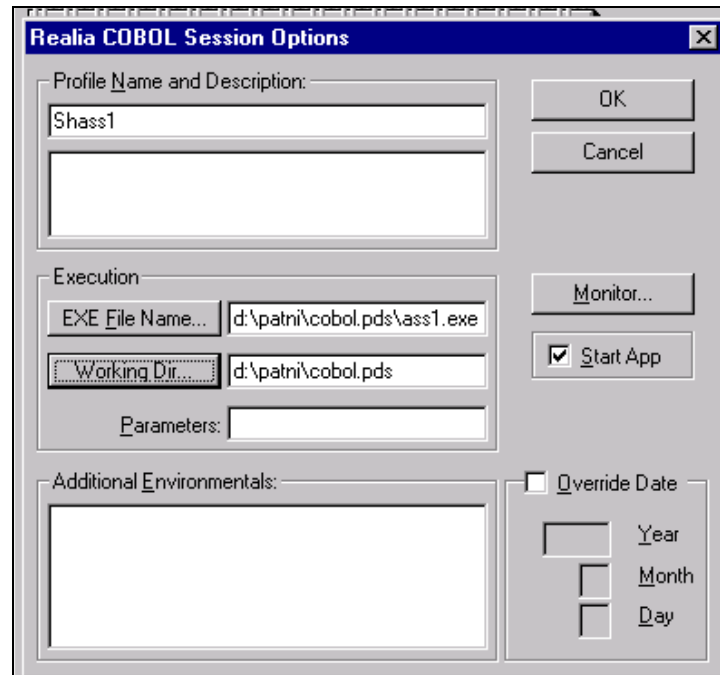


Figure 7: Realia COBOL Session Options

Step 4: After creation of the execution profile, run the .exe by selecting the **Build → Execute** option.

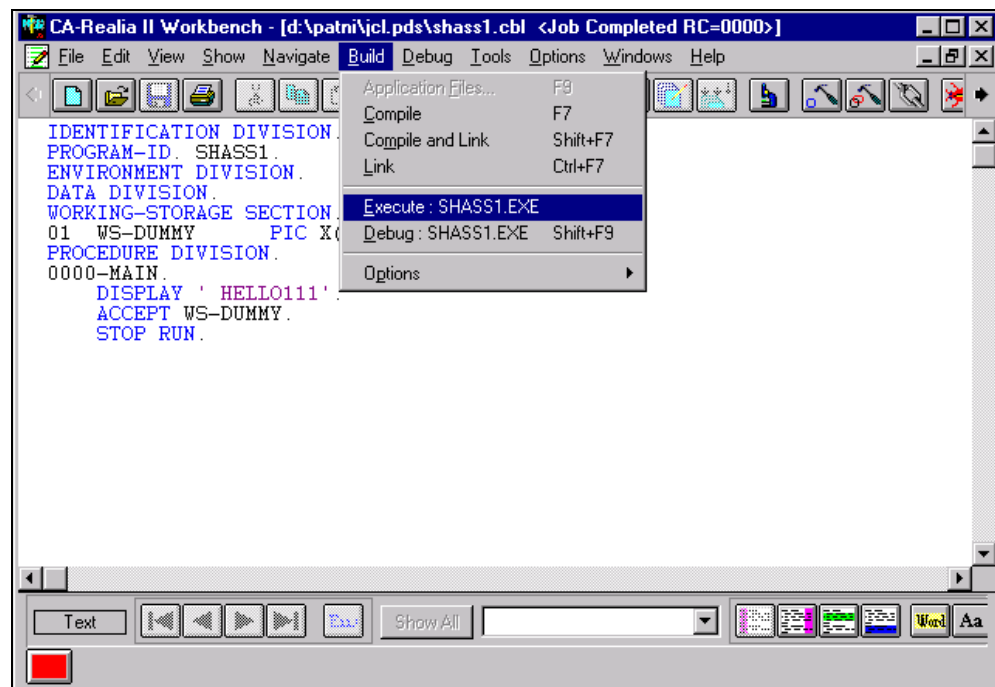


Figure 8: Run .exe

- 1. Debugging a COBOL program

Creating debug profile

After creation of the **Execution Profile**, we can also create a debug profile. This is mainly used for debugging.

Step 1: Select **Debug** option from the **Build** menu.

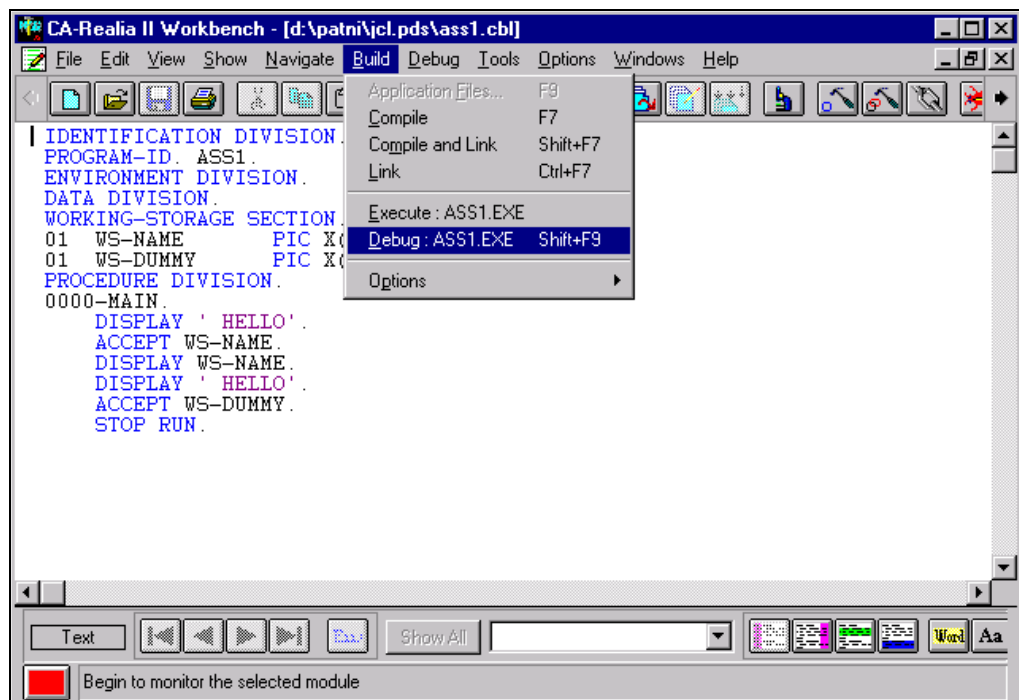


Figure 9: Build menu

Step 2: From the **Execution Options**, select the **Edit** option. From **CA-Realia COBOL session options**, select **Monitor**. Then select the cvx file to be debugged (monitor). The following screen will be displayed.

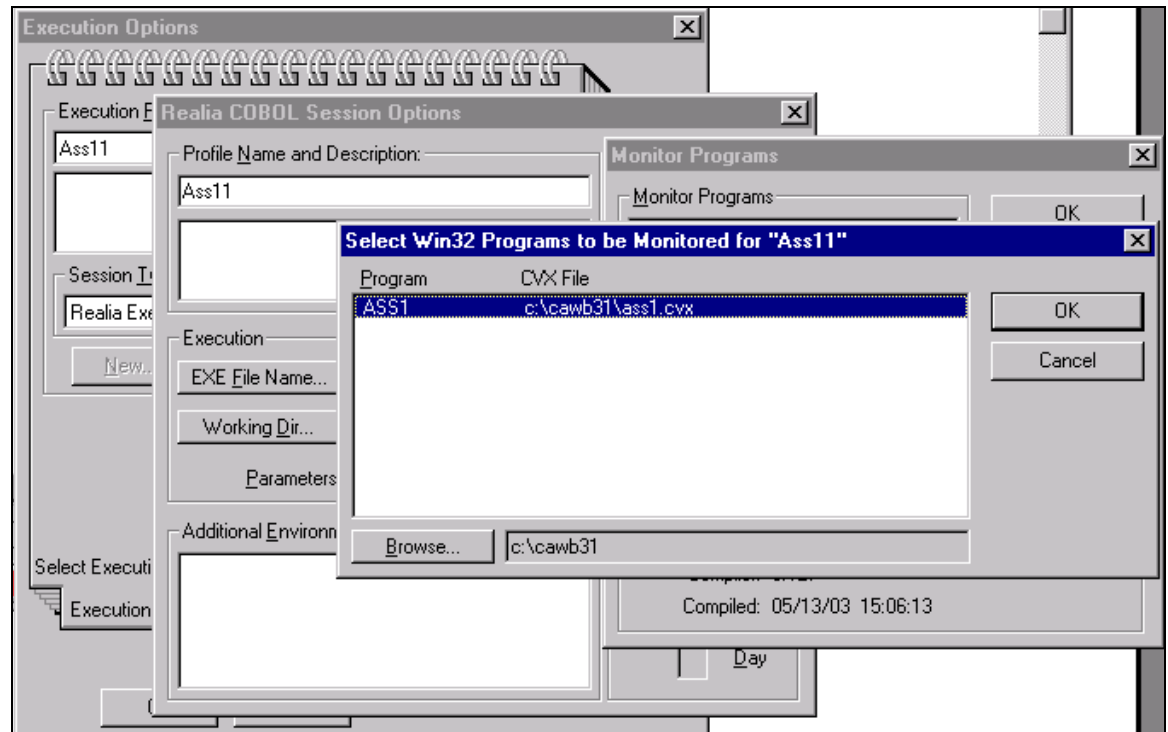


Figure 10: Execution Options

Step 3: Click **OK**. The following screen will be displayed. We can start debugging from the first statement from the procedure division.

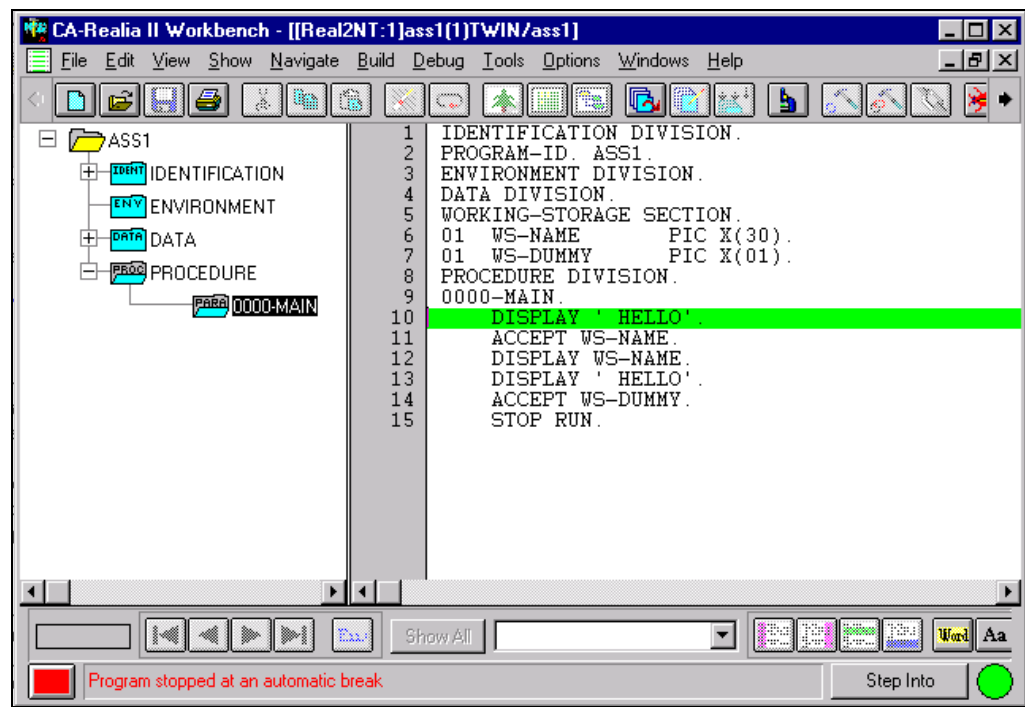


Figure 11: Debugging

Step 4: There are options for tracing in the debug menu. Some important options are as follows:

- Auto-Step:** In this animation style of execution, the program executes statement by statement. The execution path is visible and you can interrupt at any time by using the **Interrupt hot key (Ctrl+Q)**, the **Debug Session Interrupt** command, or the **Interrupt** auto-button.
- Step-into:** This command or auto-button is available when the debugger reaches a CALL statement. When you choose **Step Into**, the debugger steps into the called program and positions the highlighter on the first line.
- Step-out:** This command is available when the current line is within a called program. Choose **Step out** to stop stepping through the called program. The remaining lines of code are executed and the debugger highlights the first executable statement after the CALL statement.
- Step-Over:** This command is available when the debugger reaches a CALL statement. When you choose **Step Over**, the debugger executes the code of the called program. However, it never steps through the code line by line. The only exception is if there is a **breakpoint** in the called program. The debugger honors breakpoints in called programs, and will stop at the breakpoint instead of stepping out of the called program.
- Break points:** They can be set to start debugging from various points in the code. In debugging, breakpoints are predetermined statements in the program where execution of the program is interrupted, either unconditionally or under certain conditions.

- **3. Invoking Subroutines**

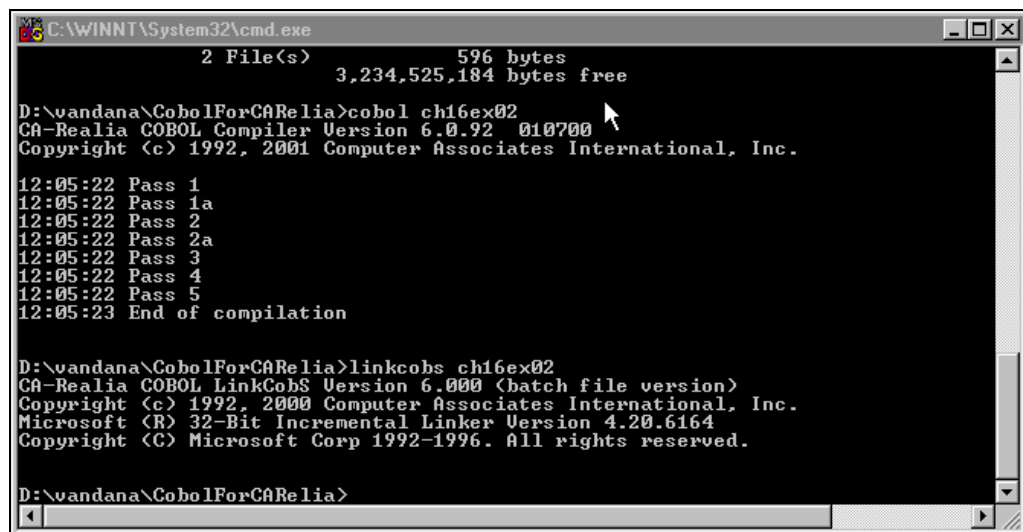
Creation of .DLL for invoking a Sub Program

To invoke a subprogram a .DLL has to be created. To create a .DLL, follow the steps given below:

Step 1: Set the environment settings using the following options:

- Run the **real2env.bat** in the command prompt, and ensure that the path contains **Path=D:\CAWB31**;
- Observe that other environments settings are done automatically and they can be seen by typing set on the command prompt.

Step 2: Compile and link the COBOL program. This creates the DLL.



```
C:\WINNT\System32\cmd.exe
2 File(s)          596 bytes
3,234,525,184 bytes free

D:\vandana\CobolForCARelia>cobol ch16ex02
CA-Realia COBOL Compiler Version 6.0.92 010700
Copyright (c) 1992, 2001 Computer Associates International, Inc.

12:05:22 Pass 1
12:05:22 Pass 1a
12:05:22 Pass 2
12:05:22 Pass 2a
12:05:22 Pass 3
12:05:22 Pass 4
12:05:22 Pass 5
12:05:23 End of compilation

D:\vandana\CobolForCARelia>linkcobs ch16ex02
CA-Realia COBOL LinkCobS Version 6.000 (batch file version)
Copyright (c) 1992, 2000 Computer Associates International, Inc.
Microsoft (R) 32-Bit Incremental Linker Version 4.20.6164
Copyright (C) Microsoft Corp 1992-1996. All rights reserved.

D:\vandana\CobolForCARelia>
```

Figure 12: DLL

Step 3: In the calling program, (for example: in CH16EX01.CBL), select the following options before compiling the program.

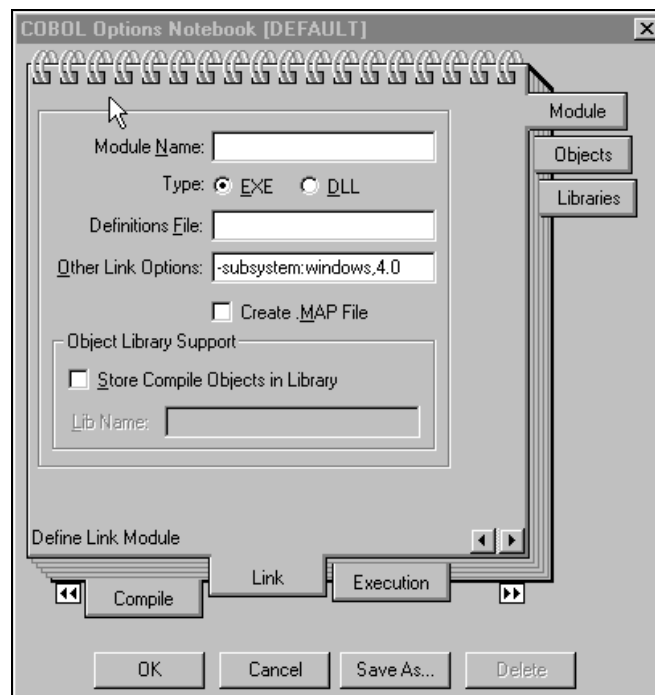


Figure 13: COBOL Options Notebook

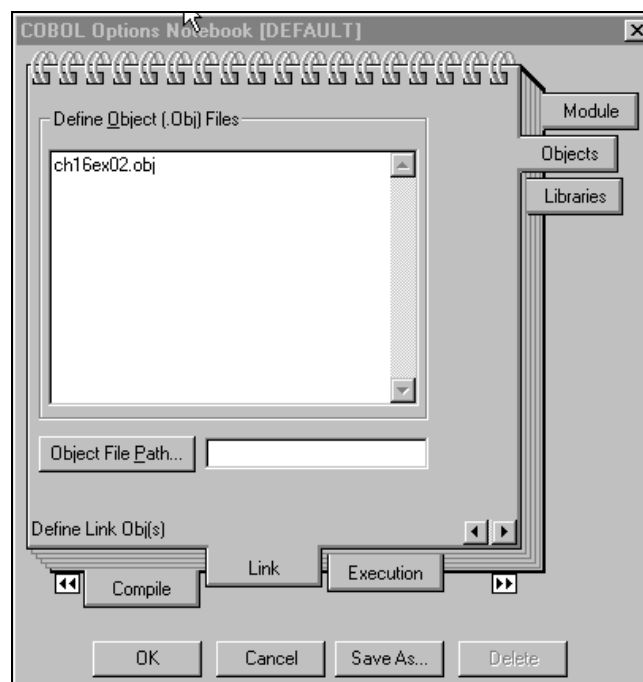


Figure 14: COBOL Options Notebook

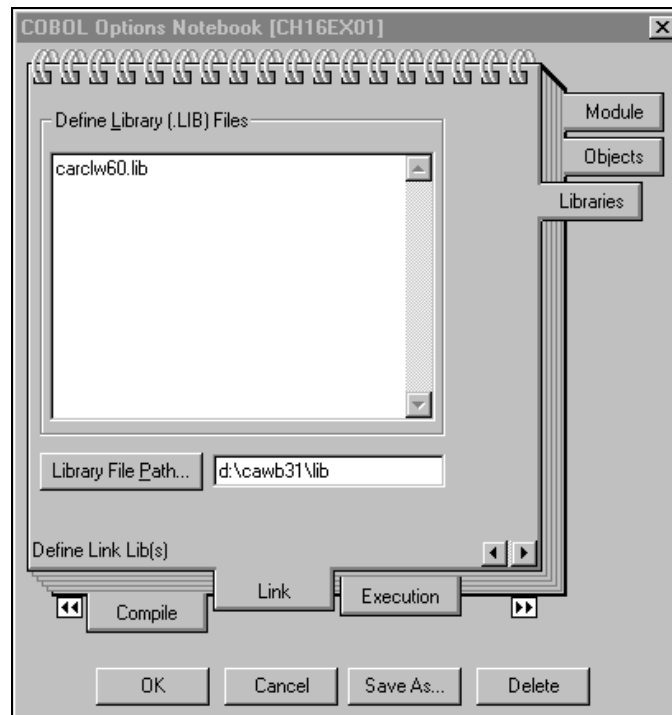


Figure 15: COBOL Options Notebook

Step 4: Compile, link, and execute the calling program.

Note: Creation of DLL is optional. You can compile the called program to create the .Obj file, which can be moved to the library path specified in library path option of the link tab.

COBOL STANDARDS

QMS/1001/STD
Revision No.1.2
(07-Nov-1996)

This document lays down the standards for programming in COBOL. These need to be adhered to in order to achieve uniformity throughout IGATE.

DOCUMENT HISTORY

<u>Date</u>	<u>Revision</u>	<u>Change</u>
04-Sep-1993	0.1D	Created Initial Draft
17-Dec-1993	1.0	Baselined
01-Dec-1994	1.1	Reorganized
04-Nov-1996	1.11D	Reviewed and Revised
07-Nov-1996	1.2	Baselined

DOCUMENT CONTROL INFORMATION**Prepared By**

Saurabhi Kulkarni

Kerman Balsara

Maintained By

All suggestions, comments, change requests and request for copies should be addressed to the CQA Group.

Approved By

Name : Satish M. Joshi

Designation : General Manager

Date :

Signature :

Released By

Name :

Designation : Manager - Quality Assurance

Date :

Signature :

CONTENTS

1.	INTRODUCTION	61
2.	IDENTIFICATION DIVISION	63
3.	ENVIRONMENT DIVISION	65
4.	DATA DIVISION	65
5.	PROCEDURE DIVISION	68
6.	READABILITY FACTORS	61
7.	CODING CONSIDERATIONS	75
8.	DESIGN CONSIDERATIONS	81

1 INTRODUCTION

1.1 The need for a Set of Standards

To ensure that IGATE delivers a quality product, there is a requirement within IGATE for a set of COBOL programming practices and procedures. Code written adhering to these standards will have better readability and maintenance will be easier as well as efficient. Also the time taken for a person to understand the purpose, structure and functionality of the code written by another person will be minimized. Overall the productivity will increase along with reduction in development time as well as maintenance costs.

In addition, the use of standards will improve the consistency of programming code, enhancing the appearance of programs to a great extent. When an applications product using these standards will be delivered to any client it will be a professional, consistent and well finished product.

No set of standards can however, ensure that programmers adhere to them. Thus it is essentially up to the individuals to follow strict discipline in using the standards when writing code. There are a number of guidelines and recommendations which have been included in this document which can be followed in order to design and code good COBOL programs.

1.2 Scope

The standards described in this document are to be followed when writing programs for any COBOL project or product being developed by IGATE.

For any other component of the program, use the respective standards. e.g. CICS standards, DB2 standards, IDMS standards etc.

It should be noted that any client specified standards will override these standards.

1.3 Overview

Taking the COBOL program structure into consideration, the standards have been laid down for coding each division.

Identification Division : Standard comment block which will give general and functional information about the program.

Environment Division : File naming conventions.

Data Division : Naming conventions for data items, logical grouping of data items and their usage.

Procedure Division : Naming conventions for paragraph names.

Readability Factors : Emphasizes the need and provides guidelines for alignment, comments. Use of these guidelines will improve readability of the program substantially.

Coding considerations : Gives standards for writing a structured program. Use of these will enable the programmer to understand and maintain the program in a better fashion.

Design considerations : Gives standards for creating FD's, copyfiles, designing screens, giving control report etc.

2 IDENTIFICATION DIVISION

- 2.1 Program-id must be coded and should be same as the external filename. For IBM users, the program-id should be equal to the TSO member name.
- 2.2 A detailed comment box with the following information should be placed just before the ENVIRONMENT DIVISION.

System Name

Author Name

Brief description of the function of the program

Brief outline of the main logic of the program

Tables and files used by the program along with the mode in which they are opened

All reports generated with their names and descriptions

Invocation method & parameters for the program

Subroutines called along with the parameters passed to them

Change log giving WRI(Work Request Id), modified/created by, modification/creation date, and details of the changes made

External procedures (JCL,etc.) used to run the program.

Screens used, if any like map names etc.

A standard block should be made as :

```
*****
*                                     *
* System name      :                  *
*                                     *
* Author name     :                  *
*                                     *
* Program Title:One line description of the program*
*                                     *
* Purpose        : Explain the purpose of the  *
*                program                    *
*                                     *
* Input         :                  *
*                                     *
* Process       : Brief logic of the program  *
*                                     *
* Output        :                  *
*                                     *
* Subroutines called and parameters passed :  *
*                                     *
* Screens used  :                  *
*                                     *
* JCL to be used:                  *
*                                     *
*****
*                Work Request Log                *
*****
* Date   Programmer WRI   Description   *
*                                     *
* 99/99/99 x.xxxxxxx  xx999  xxxxxxxxxxxxxxxxxxxx*
*                xxxxxxxxxxxxxxxxxxxx*
*                                     *
*****
```

EJECT

3 ENVIRONMENT DIVISION

3.1 Configuration Section

- 3.1.1 If SOURCE-COMPUTER and OBJECT-COMPUTER clauses are to be coded, then use of a copyfile is recommended. Refer to Design considerations.

3.2 Input-output Section

- 3.2.1 Use of standard copyfiles for SELECT clause for the files used in the program is recommended. The copyfile will give the logical and physical file names to be used. Refer to Design considerations.

4 DATA DIVISION

4.1 File Section

- 4.1.1 Copyfile should be used for the FD declaration. Refer to Design considerations.

4.2 Working-storage Section

- 4.2.1 All the copyfiles that are used by this section should be copied at the end of this section.
- 4.2.2 Data definitions should start from 01 level. Further, as much as possible, all levels should be incremented in multiples of 5.
- 4.2.3 All the numeric data fields that are used for internal processing should be defined as COMP-3 with odd number of digits. This improves speed of arithmetic operation and conserves storage.
- 4.2.4 All the datanames (including subscripts) should be meaningful and at least 8 chars long and should not have names like WS-A, WS-B etc. Economy in length of data names should not be done at the cost of understanding and readability.

- 4.2.5 All 88-level entries should be prefixed with '88-'.
- 4.2.6 Data items used for similar purpose should be defined together as a logical group. In some cases the processing logic also needs to be considered. e.g. If monthly, quarterly and yearly totals handled in a program, these can be grouped in two ways :

(1) All totals for a field are grouped together :

```
01      W01-TOTALS.  
      05      W01-FIELD1-MNLY-TOT.  
      05      W01-FIELD1-QTLY-TOT.  
      05      W01-FIELD1-YRLY-TOT.  
  
      05      W01-FIELD2-MNLY-TOT.  
      05      W01-FIELD2-QTLY-TOT.  
      05      W01-FIELD2-YRLY-TOT.
```

(2) The period totals for all fields are grouped together :

```
01      W01-TOTALS.  
      05      W01-FIELD1-MNLY-TOT.  
      05      W01-FIELD2-MNLY-TOT.  
  
      05      W01-FIELD1-QTLY-TOT.  
      05      W01-FIELD2-QTLY-TOT.  
  
      05      W01-FIELD1-YRLY-TOT.  
      05      W01-FIELD2-YRLY-TOT.
```

If the program logic cumulates figures into monthly, quarterly and yearly totals in the same paragraph, then these datanames should be defined as in (1).

If the program logic is such that all monthly totals are done in one paragraph, all quarterly in another and all yearly in a third paragraph then the datanames should be defined as in (2).

- 4.2.7 WORKING-STORAGE groups should be named in ascending order.
eg. W01-, W02- etc.

All datanames in a group should be prefixed with the group prefix and suffixed with 2 or 3 characters indicating the use of the variable. e.g. -SW for Switches, -PRT for print lines, -CNT for counters, -ARR for Arrays, -IND for indexes, -LIT for literals, -SUB for subscripts, -FLAG for flags etc.

eg. 01 W01-COUNTERS.
05 W01-READ-CNT.
05 W01-WRITE-CNT.
05 W01-MODIFIED-CNT.

- 4.2.8 When defining picture, use 2-digit element length. eg. write PIC X(01) instead of PIC X and PIC X(05) instead of PIC X(5).

- 4.2.9 Do not use 77 levels since logical grouping of elements is not possible with 77 level.

- 4.2.10 Report layout should appear after all other variables are defined. The following convention should be used in the report layout :

Header : H01 - H99
Detail : D01 - D99
Total : T01 - T99
Footer : F01 - F99

Later, during modifications, if a new group logically falls between two groups, use alphabets for sequencing. eg. H01A-, H01B- etc.

- 4.2.11 One subscript should NOT be used for more than one table, even if the tables are of similar type. Use different subscripts for each table/array defined. The subscript name should include 3 characters of the table name.

4.2.12 All subscripts should be defined with 'USAGE IS INDEX' and should not be used for any other purpose than subscripting. The advantage of defining them in this manner is that they are automatically assigned PIC S9(04) COMP.

4.2.13 There should be no datanames declared by the programmer which are not used in the PROCEDURE DIVISION.

4.3 Linkage Section

4.3.1 Use prefix 'LN-' for all fields in the linkage section.

5 PROCEDURE DIVISION

5.1 Every paragraph name should be made up of a 4 digit number followed by a '-' and a meaningful name (at least 8 chars) describing the function performed by it. Thus the length of a paragraph name should be at least 13 characters long.

5.2 The 4 digit number in a paragraph name should be incremented by 100 when the program is initially coded so as to facilitate insertion of paragraphs at a later stage.

The exit paragraph name should have the corresponding paragraph number followed by the word '-EXIT'. e.g.

3000-PROCESS.

PERFORM 3100-CALCULATE THRU
3100-EXIT.

PERFORM 3200-CALCULATE-INTEREST THRU
3200-EXIT.

PERFORM 3300-CALCULATE-TAX THRU
3300-EXIT.

3000-EXIT.
EXIT.

- 5.3 New paragraphs inserted during enhancement, should be defined with the 4 digit number which will divide the available range in exactly two halves. In the above example, if a paragraph is to be inserted between the 3100-CALCULATE and 3200-CALCULATE-INTEREST paragraphs, the number should be 3150-.
- 5.4 If all numbers are exhausted then add another set of 4 digits. e.g. paragraph 4155- can have additional paragraph names as 4155-1000-, 4155-2000- etc. If it is initially known that the program will have a huge size, start paragraph numbering with 5 or more digits e.g 10000-.
- 5.5 When coding DECLARATIVES SECTION, code a different paragraph for each file, include file name in paragraph name. The following are the standards for declaratives :

```
*=====
DECLARATIVES.
```

```
9000-DECL-FILE1 SECTION.
```

```
    USE AFTER STANDARD ERROR PROCEDURE ON FILE1.
```

```
    IF W03-ABORT-FLAG = "Y"
      GO TO 9000-EXIT.
```

```
    MOVE W02-FILE1-STAT TO W04-FILE-STATUS.
```

```
    IF 88-FILE-DOES-NOT-EXIST
```

```
      MOVE 'Y'      TO W03-ABORT-FLAG
```

```
      MOVE 099      TO W10-ERROR-CODE
```

```
    PERFORM 9900-DISPLAY-ERR-MSG THRU
      9900-EXIT
```

```
    GO TO 9000-EXIT.
```

```
    ***** HANDLE OTHER ERRORS AS DESIRED
```

```
9000-EXIT.
```

```
EXIT.
```

```
*=====
```

```
9100-DECL-FILE2 SECTION.
```

```
    USE AFTER STANDARD ERROR PROCEDURE ON FILE2.
```

```
    IF W03-ABORT-FLAG = "Y"
      GO TO 9100-EXIT.
```

```
    MOVE W02-FILE2-STAT TO W04-FILE-STATUS.
```

```
    IF 88-FILE-DOES-NOT-EXIST
```

```
      MOVE 'Y'      TO W03-ABORT-FLAG
```



```

MOVE 099      TO W10-ERROR-CODE
PERFORM 9900-DISPLAY-ERR-MSG THRU
              9900-EXIT
GO TO 9100-EXIT.
***** HANDLE OTHER ERRORS AS DESIRED

```

```

9100-EXIT.
EXIT.

```

```

*=====
***** DECLARATIVES FOR OTHER FILES
END DECLARATIVES.
*=====

```

6 READABILITY FACTORS

6.1 Alignment Rules

6.1.1 Align the 'PIC' clause, 'VALUE' clause and 'SPACES'/'ZEROS' for all variables defined. Throughout the program, use 'SPACES' and not 'SPACE'. Similarly, use 'ZEROS' and not 'ZEROES' or 'ZERO'.

6.1.2 The level number 05 should be aligned with the dataname of 01 level entry, level number 10 should be aligned with dataname in 05 level entry, and so on. Two spaces are required between level-number and dataname. Thus the 01 level should start from 8th column, 05 level from 12th column, 10 level from 16th column and so on. e.g.

```

W01-HEADING-LINES.
05      W01-REP-HDG1.
        10 FILLER  PIC X(10) VALUE SPACES.
        10 FILLER  PIC X(03) VALUE "IGATE".

```

6.1.3 Align 'MOVE's and 'TO's as much as possible in a paragraph. But if not possible, they should be aligned at least for a logical block of code.

e.g.

```

MOVE _____ TO _____.
      MOVE _____ TO _____.
      MOVE _____ TO _____.

```

If the 'MOVE TO ' cannot fit on the same line, align the source and destination variables on consecutive lines.

e.g.

```

MOVE _____ TO
_____

```

6.1.4 Every pair of IF and ELSE should be aligned. Condition/Statement after ELSE should be aligned with the condition after IF.

e.g.

```

IF 88-VALID-NAME
    IF 88-VALID-PHONE
        |
    ELSE
        |

ELSE
    IF 88-VALID-ID
        |
    ELSE
        |.
  
```

6.1.5 It is recommended that long lines of code should be split and placed on multiple physical lines and aligned properly.

6.2 All switches should be defined along with the VALUE clause. Comments should be added along with the definition about the various values it takes, their significance, paragraph names where they are set and paragraph names where they are tested.

Information should be provided in the following format :

```

*****
* Wnn-FIRST-SW                      *
* Used for (description in 2/3 lines) *
* Set in paragraphs <para-name>.... *
* Tested in paragraphs <para-name>... *
*                               *
*****
  
```

6.3 The use of nested IF's should be restricted to 3 levels only.

6.4 Every ELSE should be the only statement on the line. Any statement that follows the ELSE should start on the next physical line.

6.5 Do not use NOT with AND and OR.
[As far as possible, positive conditions should be used.]

6.6 The use of complicated conditions like

IF NOT AND (OR NOT)
(AND) OR ... etc.

should be avoided. Such situations should be handled by coding multiple elementary IF's.

6.7 When more than one conditions are involved in an IF-ELSE statement, align them as follows :

e.g IF (condition-1) AND
((condition-2) OR (condition-3)) AND
(condition-4)
.....

ELSE

.....
.....

6.8 Paranthesis should be used when coding multiple conditions in single IF.

6.9 When assigning values to more than one dataname, multiple datanames should not be cluttered on one line. Instead they should be coded on separate lines with all the datanames well indented. Same rule should be followed when opening or closing multiple files in a single command.

6.10 A blank line should be inserted before and after every paragraph definition.

6.11 A blank line should be inserted after every logical group of code.

6.12 All the paragraphs should physically be placed in an ascending order.

6.13 Avoid coding paragraphs of length exceeding one printed page.

- 6.14 For every paragraph, comments must be written. These comments should be enclosed in the standard comment box, which should be placed before the respective paragraph definition. The comments should include the paragraph name, explain the function of the paragraph and the transfer of control from that paragraph.

```
*****
* <para-name>                                     *
* Brief explanation of function of the paragraph. *
* Transfer of control :                           *
* <called para-name> : reason for transfer      *
*      :                                         *
*      :                                         *
*      :                                         *
*****
```

- 6.15 Comments should always add value, not just restate what the code does. i.e. explain 'WHY' and not just the 'WHAT'.

Comments should be modified when the code is changed.

Generally comments should not be in-stream, except where the code needs explanation due to complexity or changes being introduced. Another exception to this is, when a message code is used in the program, the actual message should be coded as a comment just above that line. This improves readability and helps in understanding the program logic.

When maintaining existing code,

- * If a block of code needs to be added, place a comment line at the begin and at the end of the block stating the WRN for which the change is made.
- * If only one or two lines need to be added, add only one comment line before the new code stating the WRN and number of code lines being added.
- * Do NOT delete any existing lines. Only comment them out if they are not required. Use the above method for explaining the action. When commenting original lines, '*D' can be used so that these lines can be easily located if necessary.

- * When modifying code, do not modify the lines directly. Instead comment and add new lines.
- * All lines which have been added or commented for change should have the WRN in the columns 73-80.

7 CODING CONSIDERATIONS

- 7.1 Use TOP-DOWN approach when creating the layout of the program. The standard MAINLINE-PARA should contain only these three paragraphs.

0000-MAINLINE-PARA.

```
PERFORM 1000-INIT THRU --- Open files,  
1000-EXIT. init variables  
PERFORM 2000-PROCESS THRU --- Main process  
2000-EXIT. logic  
PERFORM 9000-WRAPUP THRU --- Close files,  
9000-EXIT. statistics.
```

0000-EXIT.
EXIT.

[Have common para at the end with a four digit prefix of 9999-, 9000 etc.]

- 7.2 There should be only ONE Open & Close statement per file. Project leader's permission should be sought before using more than one OPEN/CLOSE.

All open files should be closed before exiting from the program, whether normally or abnormally.

File status should be checked after every file handling statement.

- 7.3 In any IF condition 88-level datanames should be used instead of using the corresponding data item.

As a good programing practice,

- a> For a group of conditions connected by AND, the most unlikely conditions should be tested first.

b> For a group of conditions connected by OR, the most likely conditions should be tested first.

7.4 The scope of each paragraph should be what the paragraph name indicates.

e.g. The scope of a read paragraph should not extend beyond

- * Setting of key values
- * Reading the file
- * Checking Status
- * Building of control totals

7.5 Branching from a paragraph

Branching from a paragraph (Via a PERFORM) should only be to function related sub-paragraphs or to paragraphs performing common functions. For related paragraphs, a proper sequence of paragraph numbering should be followed and for the common paragraphs a certain range of numbers could be reserved, grouped by the function that these paragraphs perform. e.g. 8000-'s & 9000-'s could be for common routines.

- * The called paragraph number should always be higher than the calling paragraph number.
 - * The calling and called paragraph numbers should lie within the same thousands (ie. they should have the same first digit) if the called paragraph is not a common paragraph. This logic should also be carried down to the hundredths and tens position.
- e.g. 3100- can be called only from 3000-
3110- can be called only from 3100-
3111- can be called only from 3110-.

An exception to this is the 2000-PROCESS paragraph which controls the main logic of the program. Thus this can call 3000-, 4000- etc though they are not common paragraphs.

7.6 Every paragraph should be performed thru its exit.

7.7 'PERFORM..THRU' should be used instead of 'GO TO', in order to keep the programs more structured. However 'PERFORM' should also be used with care, so that there is no overlap in 'PERFORM' ranges and the number of levels in a program are not too many.

7.8 To avoid unstructured coding, the use of 'GO TO' should be restricted only to the exit of the current paragraph. 'GO TO' should NOT be used for any other transfer of control except in exceptional cases given below. Even in this structure 'GO TO' should be used only to transfer control to exit paragraph or to the continue paragraph.

e.g. 2000-process.
.....
.....
If condition
 go to 2000-01-continue.
.....
.....

2000-01-continue.
.....
.....

2000-exit.
 exit.

When there is more than one paragraph in this structure, the paragraphs in between should have a suffix of -nn along with the paragraph no. (example 2000-01-continue).

7.9 While deciding upon the structure of the program, it is necessary that functions which are common i.e. those which can be executed or called a number of times from various places, should be coded in separate paragraphs. Also these paragraphs should be placed towards the end of the program with numbers higher than all the paragraphs that perform these routines.

7.10 COMPUTE statement should be used only when complex expressions are involved. It should not be used when doing simple computations like incrementing value of a single data item, etc. ROUNDED clause must be coded for all COMPUTE statements.

7.11 The body of the procedure division should be as follows:

0000-MAIN.

```
PERFORM 1000-BEGIN      THRU
      1000-EXIT.
PERFORM 2000-PROCESS    THRU
      2000-EXIT.
PERFORM 9000-CLOSE      THRU
      9000-EXIT.
```

0000-EXIT.

EXIT.

1000-BEGIN.

```
PERFORM 1100-INITIALIZE-DATANAMES THRU
      1100-EXIT.
PERFORM 1200-OPEN-FILES    THRU
      1200-EXIT.
PERFORM 1300-ACCEPT-PARAMETERS THRU
      1300-EXIT.
```

|

1000-EXIT.
EXIT.

1100-INITIALIZE-DATANAMES.

|

1100-EXIT.
EXIT.

1200-OPEN-FILES.

|

1200-EXIT.
EXIT.

1300-ACCEPT-PARAMETERS.

|

1300-EXIT.
EXIT.

- 7.12 All the variables should be initialized before use eg. variables defined in tables which are used for computation.
- 7.13 Avoid reuse of variables for different purposes to conserve storage.
- 7.14 ALTER command should not be used.
GOTO depending on should not be used.
- 7.15 MOVE/ADD/SUBTRACT corresponding should not be used since same element names are required in two group items.
- 7.16 The number of switches used in the program should be kept to minimum without affecting the complexity of code.
- 7.17 'EJECT' statement should be appropriately placed in the code.
- 7.18 Avoid using SECTIONS except when using declarative section where it is mandatory.
- 7.19 In report programs, page EJECT should be given at the end of the report and not at the beginning of the report.
- 7.20 When using subroutines, if appropriate, use 'CANCEL' after returning to calling program.
- 7.21 When a program aborts, an information block giving at least the following should be displayed. The corresponding variable names and their usage should be standardised throughout the project by the Project Leader.

Date :
Time :
Program :
Paragraph :
Message :

- 7.22 Program termination statements should be coded.
- 7.23 When making changes to existing code, at times it is better to follow the existing program style even if it means deviation from the standards. But this should be done only after approval from the Project Leader. e.g. Consider a program in which datanames are defined as WS-TOTAL1,..., WS-TOTAL4. If you have to incorporate one more similar total, define it as WS-TOTAL5 instead of the standard Wnn-TOTAL5.

Consider the following piece of code :

```
1000-START-PARA.  
    Read next input-file.  
    ;
```

```
      If condition-1
        GO TO 1000-START-PARA.
      :
      If condition-2
        GO TO 1000-START-PARA.
```

```
1000-EXIT.
      EXIT.
```

There are non-standard GO TO's in this piece of code. But if we are required to add one more similar condition in the above paragraph, it is better to add it in the same style.

7.24 Wherever available :

- * INITIALIZE command should be used to initialize group items.
- * Use of in-stream 'PERFORM' could be done, with maximum of 10 lines in the 'PERFORM'.
- * Scope terminators should be used wherever available. e.g.
Every IF & IF-ELSE should be paired with END-IF

```
IF 88-VALID-NAME
  IF 88-VALID-PHONE
    |
  ELSE
    |
  END-IF
```

```
ELSE
  IF 88-VALID-ID
    |
  ELSE
    |
  END-IF
END-IF.
```

EVALUATE, END-EVALUATE should be used when more than three 'IF's are used at the same level.

8 DESIGN CONSIDERATIONS

8.1 General

- 8.1.1 Data definitions should start from 01 level. Further, all levels should be incremented in multiples of 5.

- 8.1.2 All the numeric data fields that are used for internal processing should be defined as COMP-3 with odd number of digits. This improves speed of arithmetic operation and conserves storage.
- 8.1.3 All the datanames should be meaningful and at least 8 chars long. Economy in length of data names should not be done at the cost of understanding and readability.
- 8.1.4 All 88-level entries should be prefixed with '88-'.
- 8.1.5 When defining picture, use 2-digit element length. eg. write PIC X(01) instead of PIC X and PIC X(05) instead of PIC X(5).
- 8.1.6 Do not use 77 levels since logical grouping of elements is not possible with 77 level.
- 8.1.7 For all error messages, use of either a VSAM file or ISAM file or a DB2 table or a relative file is recommended. A standard error handling routine which will take the error message code from the program and get the actual message associated for that code from the data store should be used.
- 8.1.8 A logging and recovery mechanism should be designed to avoid loss of data consistency due to system / program failures.
- 8.1.9 All files used in the system will have the following Copy files :

SOURCE-COMPUTER, OBJECT-COMPUTER clauses should be put in a file, say, 'CONFIG'.

File Name + suffix of 01 for SELECT Clause.

File Name + suffix of 02 for (FD) Clause and record layout.

File Name + suffix of 03 for File Status variable declarations.

File Name + suffix of 04 for Working-Storage record layout.

When using the Working-Storage copyfile, use a COPY statement as follows :

COPY file-name REPLACING WS by Wnn.
- 8.1.10 At the end of every program, a control report should give the statistics of the Number of Records Read, Processed, Written, Deleted, Modified etc.
- 8.1.11 Internal COBOL SORT should be avoided wherever possible. Instead, use external sorts.

8.2 Configuration Section

- 8.2.1 The SOURCE-COMPUTER and OBJECT-COMPUTER clauses should be kept in a copyfile which will be copied by every program.

8.3 Input-Output Section

- 8.3.1 The SELECT clause for all files to be used in the system should be kept in a copyfile which will be copied by every program using that file. The logical file name should be meaningful and should be at least 8 characters long. The physical filename must be suffixed by a 01 (either as part of the filename or as an extension).

e.g. The logical filename for a Cash Transaction File can be CASH-TRANS-FILE. The physical filename can be CASH-TRANS-FILE.01 (or CSHTRN01 if only 8 character filenames are supported)

- 8.3.2 The 'ASSIGN TO' name for all files should be the same as the file name.

e.g. The Select Clause for a Cash Transaction File can be written as

SELECT CASH-TRANSACTION-FILE
ASSIGN TO CASH-TRANSACTION-FILE etc.

8.4 File Section

- 8.4.1 The file descriptions should be kept in a copy file which will be copied by every program using that file. The physical filename must be suffixed by a 02 (either as part of the filename or as an extension).

e.g. The physical filename for a Cash Transaction File can be CASH-TRANS-FILE.02 (or CSHTRN02 if only 8 character filenames are supported).

- 8.4.2 For all files, the clauses
'BLOCK CONTAINS ' and
'RECORDING MODE IS ' and
'RECORD CONTAINS ' and
'LABEL RECORDS ARE'
should be specified.

- 8.4.3 The record name for every file should be the file name suffixed with '-REC'.

e.g. If the file name is CASH-TRANS-FILE then the record name should be CASH-TRANS-REC.

- 8.4.4 Each field in the file record should be prefixed by a three character abbreviation of the file name.

Field names of the CASH-TRANS-FILE can be prefixed with 'CTF-'.

- 8.4.5 The Copy File must contain comments to indicate the program/s which create the data in this file and the total length of the record.

- 8.4.6 At the end of every record, a FILLER must be defined to allow addition of variables at a later date.

8.5 Working Storage Section

- 8.5.1 The File Status variable (defined in the SELECT clause for a file) should be kept in a copyfile which will be copied by every program using that file. The physical filename must be prefixed by a 03 (either as part of the filename or as an extension).

e.g. The physical filename for storing the File Status variables for a Cash Transaction File can be CASH-TRANS-FILE.03 (or CSHTRN03 if only 8 character filenames are supported)

8.6 Screen Section

For screen interface standards, IBM users should refer to the standards of the corresponding software used e.g. CICS, ADS/O.

The following standards should be followed when using the COBOL screen section :

- 8.6.1 All screens should be designed using a Screen Generation Utility (if available) for the first time. Subsequently they should be inserted in the program. This will facilitate quick generation of code where screens have a large number of items.

- 8.6.2 A screen must have the following attributes:

The company name should be displayed on top appropriately centered with the system date and time on top right hand corner. The program name could be displayed on the top left corner.

The function of the screen should also be appropriately centered on the next line.

Mode to be displayed on top right hand corner below date e.g. add, modify, view etc. for file maintenance programs. The last two lines of the screen should be kept for the function line and the third last line for the messages to be displayed. Screen layout attached herewith.

Screen headings should be in all upper case (capitals).

Literals in screens other than headings to be capitalized.

Screen should not be cluttered up and importance is to be given for alignment and enhanced read ability.

Fields should be bright as against literals.

All numeric fields should be right justified all alpha fields left justified.

8.6.3 Screen group level names should be prefixed with 'SSnn-' where nn is a number between 01 and 99.

8.6.4 All data names in screens should be meaningful. By meaningful it is implied that data name shall indicate the function or attribute for which it is used.

8.6.5 Use of the BLANK SCREEN clause should be substituted by bottom-up line by line blanking. This gives a better touch as far as screen blanking is considered.

While blanking a screen it is to be noted that the standard header occupying the first few lines should never be blanked by the program. Line 24 is for error messages and will be blanked automatically by the next accept statement.

The rest of the screen must be blanked bottom-up. For this as many sub-groups as may be necessary should be defined. e.g.

01 SS01-BLANK-SCREEN.

05 FILLER.

10 SS01-BLANK-23 LINE 23 COL 01 BLANK LINE.
10 SS01-BLANK-22 LINE 22 COL 01 BLANK LINE.
10 SS01-BLANK-21 LINE 21 COL 01 BLANK LINE.
10 SS01-BLANK-20 LINE 20 COL 01 BLANK LINE.
10 SS01-BLANK-19 LINE 19 COL 01 BLANK LINE.
10 SS01-BLANK-18 LINE 18 COL 01 BLANK LINE.
10 SS01-BLANK-17 LINE 17 COL 01 BLANK LINE.
10 SS01-BLANK-16 LINE 16 COL 01 BLANK LINE.

05 SS01-BLANK-SCR-2.

10 SS01-BLANK-15 LINE 15 COL 01 BLANK LINE.
10 SS01-BLANK-14 LINE 14 COL 01 BLANK LINE.
10 SS01-BLANK-13 LINE 13 COL 01 BLANK LINE.
10 SS01-BLANK-12 LINE 12 COL 01 BLANK LINE.
10 SS01-BLANK-11 LINE 11 COL 01 BLANK LINE.

05 FILLER.

10 SS01-BLANK-10 LINE 10 COL 01 BLANK LINE.
10 SS01-BLANK-09 LINE 09 COL 01 BLANK LINE.
10 SS01-BLANK-08 LINE 08 COL 01 BLANK LINE.

05 SS-BLANK-SCR-1.

10 SS01-BLANK-07 LINE 07 COL 01 BLANK LINE.
10 SS01-BLANK-06 LINE 06 COL 01 BLANK LINE.
10 SS01-BLANK-05 LINE 05 COL 01 BLANK LINE.
10 SS01-BLANK-04 LINE 04 COL 01 BLANK LINE.

05 FILLER.

10 SS01-BLANK-03 LINE 03 COL 01 BLANK LINE.

8.6.6 While prompting for confirmation with 'CONFIRM (Y/N)', the default value shall be "N" unless otherwise specified by the user.

8.6.7 Use of escape sequence shall have the following standard route unless otherwise specified by the user.

Escape from first field of the screen shall take control to the previous screen.

Escape from any other field of the same screen shall take control to the first field of the same screen.

8.6.8 While modifying data through screens, the following technique is to be adopted as a standard feature unless otherwise specified by the user :

The serial number of the field to be modified will be accepted and the cursor will be positioned directly on that field instead of having to skip through all the fields.

This applies only to screens having a large number of fields. Screens having a few fields need not adopt this method.

XXXXXXXX	IGATE PVT LTD COMPANY FUNCTION PERFORMED	99/99/99	HH- MM
MESSAGES TO BE DISPLAYED			
F1 :	F2 :	F3 :	F4 :
F5 :	F6 :	F7 :	F8 :