

Programming Foundation

Lesson 3: File Handling & Refactoring

Lesson Objectives

- To understand the following concepts:
 - Records
 - File Handling
 - Re-factoring
 - Avoidance of common programming mistakes



3.1 Introduction to Records

What is a record ?

- Record is one of the composite data type, consisting of two or more values or variables stored in consecutive memory positions of different data types.
- Use them to simplify operations on blocks of data
- Use them to simplify module parameter lists
- Example:
 - Call Hardway (Name, Address, Phone, SSN, Sex, Salary)
 - Call Easyway (EmployeeRec)
- Use them to reduce maintenance of related data as changes to a record is easier to implement.
- Used to create user defined data types



Copyright © Capgemini 2015. All Rights Reserved 3

Record: Record is one of the composite data type, consisting of two or more values or variables stored in consecutive memory positions .

Example for record:

```
RECORD Employee
    DECLARE ecode AS INTEGER
    DECLARE ename AS STRING
    DECLARE esal AS INTEGER
    DECLARE edept AS STRING
END RECORD
```

The above record is used to hold details about an employee such as employee code, employee name, employee salary and department in which employee is working.

3.1 Introduction to Records

User Defined Data Type - Example

```
RECORD Student
    DECLARE RollNo AS INTEGER
    DECLARE Sname AS STRING
    DECLARE Course AS STRING
END RECORD

//Pseudo Code to read Student data and print it

BEGIN
    ACCEPT Id, Name, Crs

    Student.RollNo=Id
    Student.Sname=Name
    Student.Course=Crs

    PRINT "Student INFO:"
    PRINT "Student Roll Number      : ", Student.RollNo
    PRINT "Student Name              : ", Student.Sname
    PRINT "Student Course             : ", Student.Course
END
```



Copyright © Capgemini 2015. All Rights Reserved 4

The code given in slide is used to maintain information about a student like rollno, sname and course details.

Functionalities implemented in the above code are

- Accepting student details like id, name and course
- Printing the same.

3.1 File Handling basics

What is a file?

- A related collection of records, stored In a permanent storage device like disk, tape etc
- Files can be classified in terms of:
 - A related collection of records, stored In a permanent storage device like disk, tape etc
 - Content : Text or binary
 - Form of storage: Free or Fixed form
 - Access: Sequential or Random
 - Mode: Input , Output , both



Copyright © Capgemini 2015. All Rights Reserved 5

Files:

So far the input data was read from standard input and the output was displayed on the standard output. These programs are adequate if the volume of data involved is not large. However many business-related applications require that a large amount of data be read, processed, and saved for later use. In such a case, the data is stored on storage device, usually a disk in the form of file.

There are two types of files:

Binary file:

It is efficient for large amount of numeric data.

Text file:

It stores text and numbers as one character per byte.

It consumes large amount of storage for numbers.

It is useful for printing reports and text documents.

Access: Data from the file is accessible either sequentially or randomly.

Mode: Data from the file can be readable(Output) or data can be writable to the file(Input).

3.1 File Handling basics

Basic File Handling Logic

■ Reading from a File

- Open the file in read mode
- Read record into memory variable
- Do the processing
- Close the file

■ Writing to a File

- Open the file
- Write the record from the memory variable written
- Close the file



Copyright © Capgemini 2015. All Rights Reserved

6

File I/O requires four functions:

- **open:** It allows access to file.
- **close:** It ends access to a file.
- **read:** It gets data from file.
- **write:** It adds data into file.

Demo : Reading data from file

- A product file contains the following fields
 - Product Id
 - Name
 - Price
- Refer the pseudo code in the notes page which will list all the products available in the file whose price is above 5,000



Copyright © Capgemini 2015. All Rights Reserved 7

```
RECORD ProductRec
    DECLARE productId AS INTEGER
    DECLARE name AS STRING
    DECLARE price AS INTEGER
END RECORD
BEGIN
    DECLARE file AS FILE
    DECLARE product AS ProductRec
    PROMPT "Enter the filename" AND STORE IN file
    IF(fileExists(file)) THEN
        OPEN file
        READ data from the file AND STORE IN product
        IF(product.price>5000) THEN
            DISPLAY "Product Id" + product.productId
            DISPLAY "Product Name" + product.name
            DISPLAY "Product Price" + product.price
        END IF
    END IF
END
```

3.2 Refactoring

Definition of Refactoring

- Code refactoring is the process of rearranging the source code without modifying its functional behavior in order to improve some of the non-functional attributes of the software
- Refactoring is more essential as it addresses the problems like
 - Maintainability
 - Extensibility
- Some of the refactoring task/techniques which can be used to refactor the code are:
 - Removal of Dead and duplicated code
 - Method/Field/Component name Refactoring
 - Architecture Driven Refactoring
 - Method Slicing/Extraction



Copyright © Capgemini 2015. All Rights Reserved 8

Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behavior. Refactoring requires care because it can introduce bugs to the code

Refactoring is more essential as it addresses the problems like

- Maintainability – Code is not maintainable
- Extensibility – Very difficult to add new feature or upgrade an existing feature

Refactoring task/techniques are:

- Removal of Dead and duplicated code - Remove the unused variables/code, unreachable statements and duplicated code.
- Method/Field/Component name Refactoring – Name of a method/field/component might be confusing, provide a meaningful name for the same and ensure the changes are reflected in other references wherever the variable/field/component is used.
- Architecture Driven Refactoring - Few functionalities in an application can be potentially reused in other applications. In order to reuse such a functionality, separate the code in a separate component by adhering to the architecture design.
- Method Slicing/Extraction - Longer method needs to be broken up into smaller ones to enhance readability and maintainability. Also need to achieve cohesion by implementing only one logic in a method with more suitable name.

3.2 Refactoring

Benefits of Refactoring

- Improves Readability and modularity
- More Maintainable
- Improves extensibility
- Improves internal structure of an application
- Makes code more flexible and reusable
- Improves design of a software
- Retains with the same behavior even though internal structure changed
- If the code works, then use refactoring as valid activity



Copyright © Capgemini 2015. All Rights Reserved 9

Benefits of Refactoring:

- Improves code readability and modularity by improving code structure and design
- Reduced complexity to improve the maintainability of the source code
- Improves extensibility: Easier to add new features
- Refactoring involves improving internal structure of a software without altering its external behavior
- Goal of refactoring is to make the software more flexible and reusable
- It involves changing the design of the software after it has been coded to improve the design of a software. Refactoring is mainly used to improve badly designed software.
- Ensure software's external behavior remains the same after refactoring
- Refactoring is a valid activity only when the code is already in working state

3.2 Refactoring

Match each refactoring task to the code issue it addresses

Refactoring Task	Code Issues
Removal of Dead and duplicated code	Lack of Reusable components
Method/Field/Component name Refactoring	Poor Coding Style
Architecture Driven Refactoring	Lack of Modularity
Method Slicing/Extraction	Code Redundancy
	Lack of Layered Architecture
	Readability
	Lack of Pluggable Components
	Maintainability



Copyright © Capgemini 2015. All Rights Reserved 10

Some of the Code Issues are:

- Lack of Reusable components : Potentially component may not be reused.
- Poor Coding Style : Coding standards/naming conventions has not been followed
- Lack of Modularity : Application components can't be reused as it is tightly coupled.
- Code Redundancy : Duplicate code and dead code exists.
- Lack of Layered Architecture : Any change in one layer causing changes in all other layers
- Readability : Code is not readable due to lack of meaningful name and comments.
- Lack of Pluggable Components : Existing components are not easily replaceable due to the tight coupling code.
- Maintainability: Hard to maintain the code.

3.2 Refactoring

Match each refactoring task to the code issue it addresses - Solution

Refactoring Task	Issue Addressed
Removal of Dead and duplicated code	<ul style="list-style-type: none">• Lack of Reusable components• Code Redundancy
Method/Field/Component name Refactoring	<ul style="list-style-type: none">• Poor Coding Style
Architecture Driven Refactoring	<ul style="list-style-type: none">• Lack of Layered Architecture• Lack of Modularity• Lack of Pluggable Components
Method Slicing/Extraction	<ul style="list-style-type: none">• Readability• Maintainability

Each refactoring task is addressed to the code issue it addresses.

3.2 Refactoring

Example

Example 1 : Can we refactor the following code?

```
IF (State == TEXAS) THEN
    Rate = TX_RATE;
    Amt = Base * TX_RATE;
    Calc = 2 * Basis (Amt) + Extra (Amt) * 1.05;
ELSE IF ((State == OHIO) OR (State == MAINE)) THEN
    Rate = (STATE == OHIO) ? OH_RATE : ME-RATE;
    Amt = Base * Rate;
    Calc = 2 * Basis (Amt) + Extra (Amt) * 1.05;
    IF (State == OHIO) THEN
        Points = 2;
    END IF
ELSE
    Rate = 1;
    Amt = Base;
    Calc = 2 * Basis (Amt) + Extra (Amt) * 1.05;
END IF
```



Copyright © Capgemini 2015. All Rights Reserved 12

The above code is used to calculate tax based on the state. Tax rate varies for each state but similarity exists in the tax calculation.

Rate, Amt and Calc variables are assigned values in each if-else condition.

Use Refactoring for performing the below task:

- Avoid Duplicate code in each if else condition
- Common code should be kept outside if condition

3.2 Refactoring

Example

■ Revised code

```
TEXAS      1
OHIO       2

IF (State == TEXAS) THEN
Rate = TX_RATE;
ELSE IF (State == OHIO) THEN
    Rate = OH_RATE;
Points = 2;
ELSE
    Rate = ME_RATE;
END IF
/* common lines are kept outside the if */

Amt = Base * Rate;
Calc = 2 * Basis (Amt) + Extra (Amt) * 1.05;
```



Copyright © Capgemini 2015. All Rights Reserved 13

Note: The code is more efficient and readable as compared to the previous code.

Are any further improvements possible?

If the program is used for three states, it is likely to be used for other states, as well, in the future.

Instead of using a nested IF ELSE statement, using a table or an array for tax rates of all states will make it easy to add more states later on.

3.3 Common coding mistakes – How to avoid them?

Details

■ Variable Declarations and Initializations

- Ensure the type declaration is correct
 - Do not assume “int” and “long” are the same
 - Not that “char” is not the same as “string”, even if string size is 1
- Avoid global variables to the extent possible
- Use different special naming convention for GLOBAL variables to highlight them
- Instead of using GLOBAL variables directly, use access modules like GetStatus() and SetStatus().



Copyright © Capgemini 2015. All Rights Reserved 14

Variable Declarations and Initializations:

Code Snippet - Example:

```
Boolean more-records = TRUE;
...
while ( more-records ) do ...
```

Better version of the above code snippet:

```
Boolean more-records;
...
more-records = TRUE;
while ( more-records ) do ...
```

Avoid GLOBAL variables to the extent possible. : Note that it is difficult to debug, because a GLOBAL variable can be changed from any module. It may create problems with a recursive module.

Use different special naming convention for GLOBAL variables to highlight them. For example: MAX_USERS_G for global

Instead of using GLOBAL variables directly, use access modules like GetStatus() and SetStatus(). Note that this ensures that the variable is used only at once place inside those modules. This becomes easier to change or debug.

3.3 Common coding mistakes – How to avoid them?

Details

- Use one variable for one specific purpose.
 - Avoid variables with hidden meanings.
 - Do not use one range of values for one purpose and another range of values for another purpose.
 - For example: Employee code above 90000 for temporary employees, employee code between 80000 and 90000 for contract workers



Copyright © Capgemini 2015. All Rights Reserved 15

Use one variable for only one purpose.

For an example, if a variable is used for multiple different what can we provide? Consider a variable that is used to count the number of students, and count their grades. How would you name this variable: count, studentCount/gradeCount?

So create one variable for one purpose, otherwise data management related to the requirement will be complex.

- **Don't use variable with hidden meanings** because it might not be understandable by others.

3.3 Common coding mistakes – How to avoid them?

IF conditions and Case statements

- In case, there are multiple or nested IF conditions, implement the most common scenarios at the beginning
- Use proper indentation and alignment with nested IF conditions
- Use a chain of IF-THEN-ELSE rather than nested IF conditions (recommended)
- Use default case at end in switch case statement to check for unexpected conditions
- Do not have long sequences of code in each case;
 - Call modules, if required



Copyright © Capgemini 2015. All Rights Reserved 16

How to avoid Common Coding Mistakes while using IF and case statements:

- If there are multiple nested IF conditions, then find out the condition which has high priority and specify the same in first if condition.
- Adopt a standard indentation style for your code, and stick with it throughout the program so that program will be easily readable by other programmers.

- **For an Example:**

```
BEGIN
  DECLARE file AS FILE
  PROMPT "Enter the filename" AND STORE IN file
  IF (theFileExists) THEN
    determine the length of the file
    IF(FileLength>0) THEN
      readFile(file); // Invoke readFile module
    ELSE
      PRINT "File doesn't contain data"
    END IF
  ELSE
    PRINT "File doesn't exist"
  END IF
END
```

In the above code, indentation is followed, if-else is used and fileexists condition is checked first before finding length of a file for avoiding common mistakes.

3.3 Common coding mistakes – How to avoid them?

Loops

- Steps for avoiding coding mistakes on the usage of FOR loops; WHILE-DO loops; and DO-WHILE loops are:
 - Check all the loops for the number of times they are executed like
 - Not at all
 - Exactly once
 - More than once
 - For index = start-value to end-value;
 - Be careful if “start-value” and “end-value” are expressions
 - Ensure that end-value \geq start-value
 - Use break carefully, if it is required to break the loop
 - Do not change the loop index inside the loop



Copyright © Capgemini 2015. All Rights Reserved 17

Steps for avoiding Common Coding Mistakes while using Loops are:

- Check for the number of times the loop executed: For an example, execute the below loop thrice, by considering num values as 5, 4 and 2.

```
WHILE (num<5)
DO
....
END WHILE
```

The above loop will be executed once for the value of 4, and the loop will not be executed at all for the num value of 5 and the same loop will be executed more than once if num value is 2.

- For loop:
 - Ensure that endvalue \geq startvalue.

<u>Better version</u>	<u>Not recommended to be used</u>
FOR index= 0 to 10	FOR index=10 to 0
..	...
END LOOP	END LOOP
 - Do not change the loop index within the loop as shown in the below example. In the below example, loop execution will be stopped after the re-initialization of loop index inside the loop.


```
FOR index = 0 to 10
  index=13
END LOOP
```

3.3 Common coding mistakes – How to avoid them?

Loops

- Rigorously check for the end conditions being true.
- Avoid long loops spread across multiple pages.
- Loops are entered only at the top and not in between (Goto).
- Loop index should not be used outside the loop.



Copyright © Capgemini 2015. All Rights Reserved 18

Steps for avoiding Common Coding Mistakes while using Loops are:

Rigorously check for the end conditions being true : Rigorously avoid coding infinite loops.

Avoid long loops spread across multiple pages : The loop should be short enough to view all loop code at once.

Loops are entered only at the top and not in between (Goto) : “goto” statement is always not recommended to be used inside loop as the loop execution always re start from the lower limit.

Loop index should not be used outside the loop. Consider the below code

```
FOR index=1 to 5
END LOOP
index=3;
```

Don't use index outside of the loop. Use another variable, if required as shown below:

```
FOR index=1 to 5
END LOOP
num=3;
```

3.3 Common coding mistakes – How to avoid them?

File IO Operations

- Be aware of the errors returned by the modules, and check for these errors after each call.
 - For example: After opening the file, check for the error.
 - Use appropriate variables to refer errors.
 - For example: FILE-DOES-NOT-EXIST, FILE-ALREADY-EXISTS, FILE-IS-READ-ONLY
 - Display specific, meaningful, and actionable error messages
 - Just “file does not exist” does not make much sense to the user in case the application uses multiple files.



Copyright © Capgemini 2015. All Rights Reserved 19

Points to be considered for avoiding Common Coding Mistakes while performing file IO operations are:

- Be aware of the errors returned by the modules, and check for these errors after each call like
 - Check for the error, if file doesn't exist
 - After opening the file, check for the error.
 - Check for the error, if file size is zero.
 - Check for the error, if file is not readable/writable.
- Use appropriate variables to refer errors. For example: FILE-DOES-NOT-EXIST, FILE-ALREADY-EXISTS, FILE-IS-READ-ONLY
- Display specific, meaningful, and actionable error messages. For an Example, Just “file does not exist” does not make much sense to the user in case the application uses multiple files. Instead use the error message as “Employee.txt file does not exist”, considering “Employee.txt” is a filename.

3.3 Common coding mistakes – How to avoid them?

Calls to modules

- Ensure that the number of parameters, and the sequence is correct for the module call.
- Ensure that there is no “Type mismatch” for any parameter.



Copyright © Capgemini 2015. All Rights Reserved 20

Consider the below signature of a module to calculate total price.

calculateTotal(Integer price, Integer quantity)

Refer the valid and invalid statements to invoke a module

calculateTotal(3,5); //Valid

calculateTotal(4,3,4); //Invalid

calculateTotal('Test',3); //Invalid

Summary

- In this lesson, you have learnt about:

- Record is a composite data type used to store data of different datatypes
- Record is a composite data Usage of files
- Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behavior.
- Use proper variable types, and initialize wherever necessary.
- Avoid GLOBAL and STATIC variables to the best extent possible.
- In case of multiple or nested IF conditions, implement the most common scenarios at the beginning
- In case of loops, rigorously check whether the end conditions are true.
- In case of files, be aware of the errors returned by the functions, and check for these errors after each call.



Review Question

Question 1: Mixing different types of calculation is a good practice.

- A. True
- B. False

Question 2: Which of these are characteristic benefits of refactoring

- A. It enables you to add functionality to your code
- B. Its main function is to optimize performance
- C. It makes code easier to understand and modify
- D. It makes code more flexible and reusable



Review Question

Question 3: Data item that make up structure can be of the different types.

- A. True
- B. False

Question 4: Which type of file format is used to store one character per byte?

- A. Binary File
- B. Text File

