TEMASEK JUNIOR COLLEGE

Temasek Junior College

JC H2 Computing

Problem Solving & Algorithm Design 7 – Modular Design

1 Modular Design

- A **module** can be defined as a section of an algorithm that is dedicated to performing a single function.
- Use of modules makes an algorithm simpler, more systematic, and more likely to be free of errors.
- Each module represents a single task. The programmer can develop the solution algorithm task by task, or module by module, until the complete solution has been devised.
- A module must be large enough to perform its task and must include only the operations that contribute to the performance of that task.
- A module should have a single entry and a single exit, with a top-to-bottom sequence of instructions.
- The name (identifier) of the module should describe the work to be done as a single specific function.
- A module can be a procedure or function.

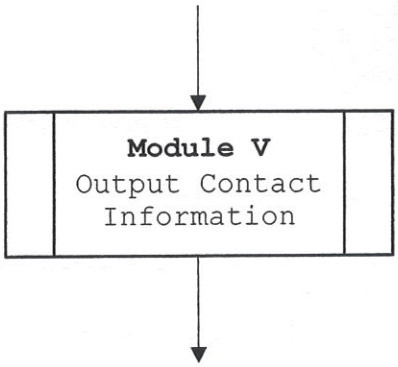
1.1 Procedures and Functions

- A procedure is a block of program code statements designed to carry out a definable task. The procedure has an identifier name.
- A function is a block of program code statements that returns a single value to the program that called it.
- As all programming languages have many "built-in" functions, we should call functions that are designed by the programmer "**user-defined**" functions.
- In general, if a single value is to be calculated, the simplest technique is to code a function.
- If there are no values to returned, then a procedure should be used.
- If more than one value is to be returned, then a procedure must be used.

1.1.1 Different between Procedures and Functions

- A function only accepts input parameters whereas a procedure accepts input or output parameters (passed by reference).
- A procedure is a bundle of code, it does not have return type whereas function has return type. Hence a function will return a value for its input parameters whereas a procedure may not return a value for its input parameters.
- While a procedure does not have a return type based on the inputs parameters, it may still return more than one value using the output parameters (passed by reference, see).
- Functions are normally used for computations whereas procedures are normally used for executing business logic (i.e. executing logical flow of process it is intended for).

1.1.2 Defining and Calling Procedures



Defining a Procedure

A procedure with no parameters is defined as follows:

```
PROCEDURE <identifier>
    <statements>
ENDPROCEDURE
```

A procedure with parameters is defined as follows:

```
PROCEDURE <identifier>(<param1>:<datatype>,<param2>:<datatype>...)
    <statements>
ENDPROCEDURE
```

- The <identifier> is the identifier (name) used to call the procedure.
- When used, param1, param2 etc. are identifiers (names) for the parameters of the procedure. These will be used as variables in the statements of the procedure.

Calling a Procedure

- Procedures defined above should be called/invoked as follows:
  - ✓ Without parameters: **CALL** <identifier>
  - ✓ With parameters: **CALL** <identifier>(Value1,Value2...)
- Unless otherwise stated, it should be assumed that parameters are **passed by value** (see **Section 3**).
- When a procedure is called, control is passed to the procedure.
- If there are any parameters, these are substituted by their values, and the statements in the procedure are executed.
- Control is then returned to the line that follows the procedure call.

Example 1

(a) Defining a procedure without parameters for displaying the address of the college.

```
PROCEDURE Display_College_Address()
    PRINT 'TEMASEK JUNIOR COLLEGE'
    PRINT '22 Bedok South Road, Singapore 469278'
ENDPROCEDURE
```

(b) Defining a procedure with parameters for displaying the address of the college.

Parameters

```
PROCEDURE Display_College_Address(name, block, street, postal: STRING)
    PRINT name
    PRINT block, street, 'Singapore', STRING
ENDPROCEDURE
```

- In this procedure, **name**, **block**, **street** and **postal** are STRINGs, the values of which are passed to the procedure before it can display the address of the school.
- The variables **name**, **block**, **street** and **postal** are called **formal parameters**.
- To use the procedure, the program calls it and specifies the values for **name**, **block**, **street** and **postal**.

Example 2 Program for determining which procedure to execute depending on L1R5.

```
PROCEDURE JC_Institute()
    Stream()
ENDPROCEDURE ②

PROCEDURE Non_JC_Institute(Status : INTEGER)
    CASE OF Status
        0 : ITE()
        1 : Polytechnic()
        2 : Vocational_Institute()
        3 : Others()
        OTHERWISE OUTPUT "ERROR"
    ENDCASE
ENDPROCEDURE ④

MAIN
    IF L1R5 >= 6 AND L1R5 <= 20
        THEN
            CALL JC_Institute() ①
        3E2D // Return address
        ELSE
            CALL Non_JC_Institute(Status) ③
        3FA0 // Return address
    ENDIF
ENDMAIN
```

Called / Invoked procedure
JC_Institute

Caller / Calling Environment
MAIN

Called / Invoked procedure
Non_JC_Institute

Note the use `3E2D // Return address`. Regardless whether `JC_Institute` and `Non_JC_Institute` are procedures or functions, they must return to a memory address referred to as "`return address`" of the caller.

1.1.3 Defining and Calling Functions

Functions operate in a similar way to procedures, except that in addition they return a single value to the point at which they are called. Their definition includes the data type of the value returned.

- User-defined functions can be modified.
- Built-in cannot be modified.
- User defined functions can be designed to meet the user's requirements.
- User-defined functions can only be used in that program / module.

Defining a Function

A function with no parameters is defined as follows:

```
FUNCTION <identifier> RETURNS <data type>
    <statements>
ENDFUNCTION
```

A function with parameters is defined as follows:

```
FUNCTION <identifier>(<param1>:<datatype>,<param2>:<datatype>...) RETURNS <data type>
    <statements>
ENDFUNCTION
```

The keyword **RETURNS** is used as one of the statements within the body of the function to specify the value to be returned. Normally, this will be the last statement in the function definition.

Calling a Function

- Since a function returns a value that is used when the function is called, function calls are not complete program statements.
- The keyword `CALL` should not be used when calling a function. Functions should only be called as part of an expression.
- When the **RETURNS** statement is executed, the value returned replaces the function call in the expression and the expression is then evaluated.

Class discussion 9608/qp22/M/J/s19

1 (i) Procedures and functions are examples of subroutines.
State a reason for using subroutines in the construction of an algorithm. [1]

(ii) Give **three** advantages of using subroutines in a program. [3]

(iii) The following pseudocode uses the subroutine `DoSomething()`. [2]

```
Answer ← 23 + DoSomething("Yellow")
```

State whether the subroutine is a function or a procedure. Justify your answer.
Type of subroutine .....
Justification

Example 3

Program for returning the larger of two given integers.

```
FUNCTION MAX(First: INTEGER, Second: INTEGER) RETURNS INTEGER
    IF First < Second
        THEN
            RETURN Second
        ELSE
            RETURN First
    ENDIF
ENDFUNCTION

MAIN
    PRINT "The larger is ", MAX(6,87)
ENDMAIN
```

Called / Invoked function MAX()

Caller / Calling environment MAIN

Example 4

EmployeeID is used to identify a particular employee and is at most six characters. The first character is an upper case letter and the remaining five are digits, e.g. A23588. Design an algorithm for the validation of the EmployeeID.

```
FUNCTION Validate_EmployeeID(EmployeeID : STRING) RETURNS BOOL
    IF Length of EmployeeID > 6
        THEN
            RETURN False
    ENDIF
    IF first character of EmployeeID is not upper case letter
        THEN
            RETURN False
    ENDIF
    IF any character of remaining five of EmployeeID is not digit
        THEN
            RETURN False
    ENDIF
    RETURN True
ENDFUNCTION
```

2 Passing Parameters

- Each module is a solution to an individual problem.
- Each module has to **interface** with other modules. Modules need to pass values to other modules and be able to accept values from other modules.
- Data can be input to a module, be it a function or a procedure. This is done by means of **parameters**.
- Parameters are temporary variable names (identifiers) defined within modules.
- Within the module, parameters act as placeholders for the argument it is passed.

Example 5

```
{Assign values to address}
SchName ← "Temasek Junior College"
SchBlk ← "22"
SchStreet ← "Bedok South Road"
SchPostal ← "469278"

CALL Display_College_Address(SchName, SchBlk SchStreet, SchPostal)
```

2.1 Actual Parameters

- Within the statements that call the procedure, the data values inside the brackets are called **actual parameters**.
- In **Example 5**, **`SchName`**, **`SchBlk`**, **`SchStreet`** and **`SchPostal`** are **actual parameters**.

2.2 Arguments

- The values that are passed during the call are called the **arguments**.
- An argument represents the value you supply to a procedure parameter (temporary variable) when you call the procedure.
  - In **Example 5**, **`'Temasek Junior College'`**, **`'22'`**, **`'Bedok South Road'`** and **`'469278'`** are **arguments**.
- How the values are passed to the function or procedure depends on the programming language.

Example 6

Program to print the cube of a given number.

```
Function Cube (byVal Vnumber as integer) as integer
    Return Vnumber* Vnumber* Vnumber
Endfunction

MAIN
    Set N as 7
    Print Cube(3)
    Print Cube(N)
ENDMAIN
```

`Vnumber` is a formal parameter

`N` is an actual parameter

`3` and `7` in `N` are arguments

- In `Cube()`, **`Vnumber`** is the parameter for the function. This means that anywhere we see **`Vnumber`** within the function, it will act as a placeholder until a value is passed as an argument.
- To pass an argument to a function is do something such as `Cube(3)`, which will call the `Cube()` function and assign the value `3` (pass the argument `3`) to the parameter **`Vnumber`**. Now, whenever you see the parameter **`Vnumber`** within the function, it will act like a variable with the value of `3`. Hence `Cube(3)` will return 3 cubed which is 27.

3 Passing Parameters by Value or by Reference

Parameters can be passed either by value or by reference.

- **By value (`BYVALUE`)**: the actual value is passed into the procedure.
- **By reference (`BYREF`)**: the address of the variable is passed into the procedure.

The difference between these only matters if, in the statements of the procedure, the value of the parameter is changed, for example if the parameter is the subject (on the left-hand side) of an assignment.

To specify whether a parameter is passed by value or by reference, the keywords BYVALUE and BYREF precede the parameter in the definition of the procedure.

If there are several parameters, they should all be passed by the same method and the BYVALUE or BYREF keyword need not be repeated.

3.1 Passing Parameters by Reference

Example 7
Program to swap the assignment of two integer values.

```
PROCEDURE SWAP(BYREF X : INTEGER, BYREF Y : INTEGER)
    DECLARE Temp: INTEGER
    Temp ← X
    X ← Y
    Y ← Temp
ENDPROCEDURE

PROCEDURE SHOW(BYVALUE X : INTEGER, BYVALUE Y : INTEGER)
    PRINT "X = ", X
    PRINT "Y = ", Y
ENDPROCEDURE

MAIN
    DECLARE A: INTEGER
    DECLARE B: INTEGER
    A ← 8
    B ← 11
    PRINT "Before swapping:"
    SHOW(A, B) //1st call
    SWAP(A, B)
    PRINT "After swapping:"
    SHOW(A, B) //2nd call
ENDMAIN
```

MAIN			
	Type	Class	Address
A	INT	Basic	9EF8
B	INT	Basic	9EF0

SHOW [By value]			
	Type	Class	Address
X	INT	Basic	9DF0
Y	INT	Basic	9DF0

SWAP [By reference]			
	Type	Class	Address
X	INT	**Pointer**	9DF8
Y	INT	**Pointer**	9DF0
Temp	INT	Basic	9DE8

Output

Memory - Stack				
Address	1AE0	1AE8	1AF0	1AF8
Value				
	:	:	:	:
Address	2EE0	2EE8	2EF0	2EF8
	:	:	:	:
Address	9CE0	9CE8	9CF0	9CF8
Value				
Address	9DE0	9DE8	9DF0	9DF8
Value		5	9EF0 / 11	9EF8 / 8
Address	9EE0	9EE8	9EF0	9EF8
Value			11	8 / 11

If the method for passing parameters is not specified, passing by value is assumed.

- If the method for passing parameters is not specified, passing by value is assumed.
- If parameters are passed by reference (as in **Example 7**), an identifier for a variable of the correct data type must be given (rather than any expression which evaluates to a value of the correct type), when the procedure is called.
- A reference (address) to that variable is passed to the procedure when it is called and if the value is changed in the procedure, this change is reflected in the variable which was passed into it, after the procedure has terminated.
- In principle, parameters can also be passed by value or by reference to functions and will operate in a similar way.
- However, it should be considered bad practice to pass parameters by reference to a function and this should be avoided.
- A function should have no other "side effects" on the program other than to return the designated value.
- Passing by reference may lead to unintended "side effects", where the parameter has its value changed in the main program as well as in the procedure.

Example 8 WJEC-w13-1103-01-Computing-CG3
In a computer program data can be passed to a procedure by value.

(a) State the name used for an item of data passed in this way and explain how passing by value works. [2]

Pass by reference

(b) State another method by which data can be passed to a procedure and explain how this method works. [2]

Pass by reference.

(c) Describe one benefit of passing data by value compared with this other method. [1]

[Solution]

a) Parameter.
How passing by value works:
A local copy of the variable is created for procedure (discarded afterwards)

b) Pass by Reference
How passing by reference works:
The address of the required data is passed to the procedure (rather than the actual value of the data

c) Benefits of passing by value:
- Preserves value-at-calling of program
- Avoids unwanted side effects.

4 Scope and Lifetime of Variables

- Variables can be declared so that they are "in scope", i.e. recognised, throughout a program. These are called **global variables** and are declared at the start of a program.
- A variable which is declared inside a procedure or function is said to be a **local variable** with a scope which is local to that procedure or function.
- The scope of variables is a very powerful feature of high-level procedural programming.

```
PROCEDURE SWAP(BYREF X : INTEGER, BYREF Y : INTEGER)
DECLARE Temp: INTEGER
      Temp ← X
      X ← Y
      Y ← Temp
      PRINT "W = ", W, "Z = ", Z
ENDPROCEDURE

PROCEDURE SHOW(BYVALUE X : INTEGER, BYVALUE Y : INTEGER)
      PRINT "X = ", X
      PRINT "Y = ", Y
      PRINT "W = ", W, Z =, Z
ENDSHOW

DECLARE Z: REAL
MAIN
      DECLARE GLOBAL W: INTEGER
      DECLARE B: INTEGER
      A ← 8
      B ← 11
      W ← 109
      Z ← 45.6
      PRINT "Before swapping: "
      SHOW(A, B) //1st call
      SWAP(A, B)
      PRINT "After swapping: "
      SHOW(A, B)// 2nd call
      PRINT "W = ", W, "Z = ", Z
ENDMAIN
```

4.1 Global Variables

- A global variable exists throughout the entire program.
- It is a longstanding variable accessible anywhere throughout the main program and all **subroutines** (i.e. functions/procedures).
- Variables are declared global as part of a module.
- A change to a global variable's value in one place is seen everywhere in the module.
- The lifetime and storage of a global variable exists until the program terminates. (e.g. `GST = 0.07`)

4.2 Local Variables

- A local variable only exists in the subroutine in which it is declared.
- Since a local variable defined within a function/procedure, the scope of this variable is only accessible within the function/procedure in which it is declared in.
- It is not possible for variables declared locally to be modified or accessed by external modules.
- A change to a local variable's value is seen only in the subroutine it is declared in.
- The same local variable name can be used within many different subroutines i.e. different subroutines may use the same identifier (name).
- The lifetime and storage of a local variable exists only when it is needed.
- The memory allocation occurs within the scope of the procedure/function and the memory is deallocated once the procedure/function returns.

4.3 Importance of Local and Global Variables in Production of Software

- Software will be written in modules, therefore it will be necessary to ensure that some values do not overflow into other modules. Hence local variables will be needed.

- Sometimes it will be necessary to use a value from one module in another. Hence global variables will be needed.

Mini Project
Complete all modules for validation checks in [Example 1 CW2011(A) – Data capturing and correctness].

The examinations department decides to store the following data:

- **SubjectCode** is used to uniquely identify a particular subject and is four digits.
- **Name** is the name of the subject and is at most 30 characters.
- **SubType** is the type of subject and can have the values 'A' or 'P'.
- **P1Len** is the duration of the first written paper and can have the values 2.0, 2.5 and 3.0. For a practical subject this field would have the value 0.0.
- **P2Len** is the duration of the second written paper and can have the values 2.0, 2.5 and 3.0. For a practical subject this field would have the value 0.0.
- **PracLen** is the duration of the practical examination and can have values in the range 10.0 to 40.0. For an academic subject this field would contain the value 0.0.
- **CDate** is the final date by which the practical examination should be completed. It is six characters and is in the form 110504 (which represents 4th May 2011). For an academic subject this field would have the value '000000'.

```
DECLARE NoOfRecords : INTEGER
DECLARE SubjectCode : STRING
DECLARE Name : STRING
DECLARE SubType : CHAR
DECLARE P1Len : REAL
DECLARE P2Len : REAL
DECLARE PracLen : REAL
DECLARE CDate : STRING
CONSTANT REC_MIN = 1
CONSTANT REC_MAX = 30

// Validation of CDATE
FUNCTION GetCDate() RETURNS STRING
// Validation of PracLen
FUNCTION GetPracLen() RETURNS REAL
// Validation of P2Len
FUNCTION GetP2Len() RETURNS REAL
// Validation of P1Len
FUNCTION GetP1Len() RETURNS REAL
// Validation of PracLen
FUNCTION GetSubType() RETURNS CHAR
// Validation of Name
FUNCTION GetName() RETURNS STRING

// Validation of SubjectCode
FUNCTION GetSubjectCode() RETURNS STRING
      DECLARE SubjectCode : STRING
      REPEAT
            Valid_Flag ← True
            PROMPT "Subject Code: "
            INPUT SubjectCode
            IF LENGTH(SubjectCode) <> 4 THEN //Presence and length check
                  PRINT "Error! Subject Code must be 4 digits. "
                  Valid_Flag ← False
            ELSEIF NOT every character is a digit THEN     //Format check
                  PRINT "Error! Subject Code contains only digits only. "
                  Valid_Flag ← False
            ENDIF
      UNTIL Valid_Flag = True
      RETURN SubjectCode
ENDFUNCTION
```

```
// Validation of NoOfRecords
FUNCTION GetNoOfRecords() RETURNS INTEGER
   DECLARE NoOfRecords : INTEGER
      REPEAT
            Valid_Flag ← True
            PROMPT "No. of records[1 - 30]: "
            INPUT NoOfRecords
            IF TYPE OF NoOfRecords is not INTEGER THEN     //type check
                  PRINT "Error! No. of records can only be an integer [REC_MIN - REC_MAX]."
                  Valid_Flag ← False
            ELSEIF NoOfRecords < 1 OR NoOfRecords > 30 THEN       //Range check
                  PRINT "Error! No. of records can only be in range [REC_MIN - REC_MAX]."
                  Valid_Flag ← False
            ENDIF
      UNTIL Valid_Flag = True
      RETURN NoOfRecords
ENDFUNCTION

MAIN()
   NoOfRecords ← GetNoOfRecords()

//LOOP NoOfRecords times
DOWHILE NoOfRecords > 0
      SubjectCode ← GetSubjectCode()
      Name ← GetName()
      SubType ← GetSubType()

      //Handling different papers
      IF SubType = 'A' THEN          //Written papers
            PracLen ← 0.0
            P1Len ← GetP1Len()
            P2Len ← GetP2Len()
      ELSEIF SubType = 'P' THEN      //Practical subjects
            P1Len ← 0.0
            P2Len ← 0.0
            PracLen ← GetPracLen()
      ENDIF

      CDate ← GetCDate()
      NoOfRecords ← NoOfRecords - 1
ENDDO
```

5 Concept of Recursion

- A recursive subprogram is one that includes, among the statements making up the subprogram, a call to the **same** subprogram.
- The subprogram will continue calling itself recursively, so it must have a **means of finishing** and continuing the calling program.
- This is done by a **stopping condition (base case)**, which causes the subprogram to exit rather than call itself again.
- **Base Case**, or halting case, of a function is the problem that we know the answer to that can be solved without any more recursive calls. Every recursive function *must* have at least one base case (many functions have more than one).
- Neglecting to write a base case, or testing for it incorrectly, can cause an **infinite loop**.
- The main advantage of recursion is usually simplicity the code.
- The main disadvantage is often that the algorithm may require **large amounts of memory** if the depth of the recursion is very large.
  - Memory overheads of stack use with many recursive procedural calls
  - May result in stack overflow
  - Difficult to program / dry run / debug if there is a fault

- Functional recursion
  - A function may be partly defined in terms of itself.

Example 9 – Summation of Numbers from 1 to n

By Iteration	By Recursion

Let SUM(N)
$= N + (N-1) + (N-2) + \ldots + 2 + 1$
$= \sum_{I=1}^{N} I$
 	
Pseudocode:
 	
SUM(N)
SUM ← 0
FOR I = 1 TO N
SUM ← SUM + I
NEXT I
RETURN SUM
END
	Let SUM(N)
	$= N + (N-1) + (N-2) + \ldots + 2 + 1$
	$= \begin{cases} 1 & \text{IF } (N = 1) \\ N + \text{SUM}(N-1) & \text{IF } (N > 1) \end{cases}$
	
	Pseudocode:
	
	SUM(N)
	IF (N=1) THEN //**Base case, Stopping condition**
	RETURN 1
	ELSE
	RETURN N + **SUM** (N-1)
	ENDIF
	END

5.1 Advantages of Recursion

- When the solution to a problem is essentially recursive (e.g. tree traversal), it enables the programmer to write a programme which mirrors the solution. (We shall learn more about tree traversals when we deal with the topics on Data Structures).
- Recursive solutions are often much shorter than non-recursive ones.

5.2 Disadvantages of Recursion

- If the recursion continues excessively, the stack of return addresses may become full (i.e. no available memory is left). This leads stack overflow and the **program will crash**.
- Recursive routines can be difficult to follow and to debug.
- Recursive routines are sometimes very slow in execution owing to the overheads in memory involved in repeatedly calling the subroutine, and storing and retrieving return addresses and parameters.

3D printed stack illustration

5.3 Recursion vs. Iteration

In programming, iteration allows multiple blocks of data and instructions to be processed repeatedly in sequence using loops. By using the iteration construct, these loops will execute repeatedly until a condition is fulfilled.

Recursion on the other hand is a way of programming that uses the selection construct where a function calls itself one or more times in its body, and then terminates when it reaches a base case.

Example 10
The **factorial** function can be defined recursively by the equation

fact(N) = N * (N – 1) * (N – 2) * ... * 2 * 1

0! = 1 and 1! = 1 (**stopping conditions, base cases**) and,
N! = N (N – 1)! for all N > 0

Neither equation by itself constitutes a complete definition; the first is the base case (**stopping condition**), and the second is the recursive case.

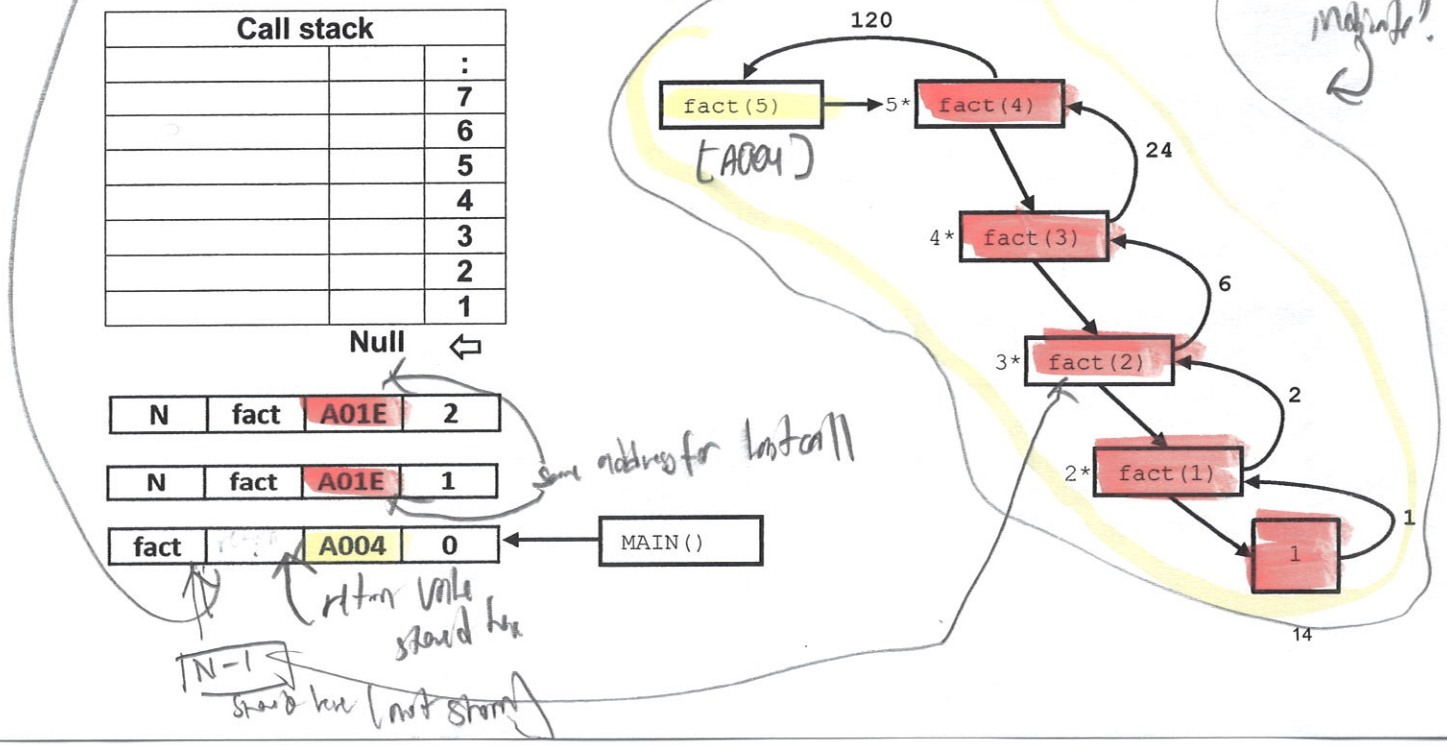
The pseudocode based on recursion is as follows:

```
fact(N)
    IF N = 1 THEN
        RETURN 1
    ELSE
        RETURN N * fact(N-1)
    ENDIF
END

// To find 5!
MAIN
    PRINT fact (5)
ENDMAIN
```

Data used in `fact()`
fact, N, N-1

The diagram below shows a stepwise tracing of the recursive call.



Example 11 – Fibonacci Number Sequence
The Fibonacci number sequence can be defined as follows:

fib(0) = 0 if n = 0
fib(1) = 1 if n = 1
fib(n) = fib(n − 1) + fib(n − 2) if n ≥ 2

For such a definition to be useful, it must lead to values which are non-recursively defined.

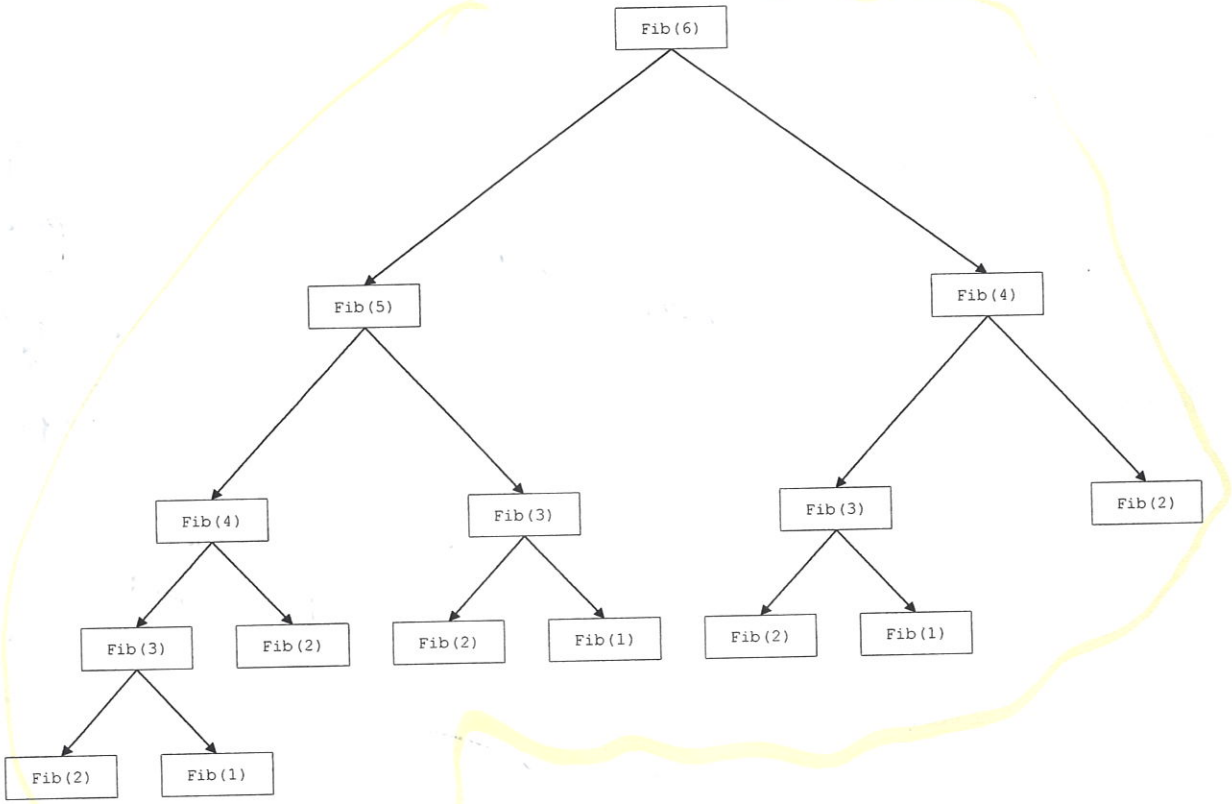
Python language implementation:

```
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Test case data: n ← 1000

In this case,
fib (0) = 0 and fib (1) = 1 (**stopping condition, base case)**.

Tracing Fibonacci Calls



6 Dynamic Programming

- Build up solutions of "simpler" instances from small to large.
- Save results of solutions of "simpler" instances.
- Useful when problem can be solved using solution of two or more instances that are only slightly simpler than original instances, e.g. fibonacci numbers

Example 12 – Fibonacci Number Sequence by Dynamic Programming
The pseudocode below makes use of dynamic programming to generate Fibonacci numbers.

```
FUNCTION fib (N: INTEGER) RETURNS INTEGER
    DECLARE FIBARRAY: ARRAY [1: N] OF INTEGER

    //Instances: Dynamically creating an array of size N
    DECLARE I: INTEGER
    IF (N <= 2)
        THEN
            RETURN 1;
        ELSE
            FIBARRAY[1] ← 1;
            FIBARRAY[2] ← 1;
            FOR I = 3 TO N
                FIBARRAY[I] = FIBARRAY[I-1] + FIBARRAY[I-2]
            NEXT I
    ENDIF
    RETURN FIBARRAY[N];
ENDFUNCTION
```

Example 13 – More Compact Code for Fibonacci Numbers (Using Iteration)

```
FUNCTION fib (N: INTEGER) RETURNS INTEGER
    DECLARE FIBN, FIBN1, FIBN2, I: INTEGER
    FIBN ← 1, FIBN1 ← 1, FIBN2 ← 1;
    IF (N > 2)
        FOR  I = 3 TO N
            FIBN ← FIBN1 + FIBN2
            FIBN1 ← FIBN2
            FIBN2 ← FIBN
        NEXT I
    ENDIF
    RETURN FIBN
ENDFUNCTION
```

Example 14 – Greatest Common Divisor (Self reading)
To write the recursive program for finding the greatest common divisor of any two numbers, we have to express the greatest common divisor (GCD) function in a recursive form:

Note: n % m refers to the remainder of n divided by m

1. if m <= n and n % m = 0 then gcd(n, m) = m(termination step).
2. if n < m then gcd(n, m) = gcd(m, n) (recursive definition of greatest common divisor).
3. if n >= m then gcd(n, m) = gcd(m, n % m) (recursive definition of greatest common divisor).

The pseudocode is as follows:

```
gcd(n, m)

    IF m <= n AND n % m = 0 THEN
        RETURN m
    ENDIF

    IF n < m THEN
        RETURN gcd(m, n)
    ELSE
        RETURN gcd(m, n%m)
    ENDIF

END
```

Tutorial 7.1 Q1, 3

7 Standard Modules

- A standard module is one which carries out a common / standard task / can be used for a standard situation in a (many) program(s)
  Example: print function / input validation

- **Built-in functions**
  - Built-in functions are made available by the programming language / already in the system.
  - Built-in functions are ready made and tested.

    Example: **Random number generation**
    - ✓ Any use of a function to generate random numbers must be clearly explained.
    - ✓ Possible functions for random number generation are:
      - ➢ `randint(min,max)`: generates a random integer between the integers min and max (includes `min` but excludes `max`)
      - ➢ `random()` : generates a random real number between 0 and 1.

    Example: Mathematical functions: `SIN()`, `COS()`, `TAN()` etc.

    Example: String functions: `<str>.upper()` (<str> is any string), etc.

  - **Benefits**
    - ✓ No need to re-code again as the function has already been written. This decreases program development time.
    - ✓ Less likely to have errors because it has already been tested/used ("for real").
    - ✓ Likely to be of high quality/efficient as may have been written by experts in the field.

- **Module Library / Library Programs**
  - Modules can be kept in a module library and reused in other programs.
  - A library program can be defined as a program contained in a program library. There may be programs in a program library but more often they are subroutines that programmers can use in their programs.
  - If a routine exists in a library, it would be very unwise for a programmer to write his or her own routine.
  - Existing library routines will have been extensively tested before release. Even if some residual bugs did exist following testing, the regular use of the routines would almost inevitably lead to their detection.
  - The most obvious examples of library routines are the built-in functions available for use when programming in a particular language.

Tutorial 7.2 Q1, 2, 3

Tutorial 7.1 [Recursion]

Q1, Q3

1* **(a)** Describe the main characteristics of a recursive algorithm. [2]
(b) Describe **two** disadvantages of using a recursive algorithm. [2]

2* The words COW, BEEF and FORTY have all their letters written in alphabetical order. Here is an algorithm for a function which checks whether all the letters in a word are in alphabetical order.

```
01  FUNCTION IsInOrder(Word)
02      IF LENGTH(Word) = 1 THEN
03          RETURN TRUE
04      ELSE
05          FirstChar = First character in Word
06          RestOfWord = All characters in Word except the first
07          IF FirstChar > RestOfWord THEN
08              RETURN FALSE
09          ELSE
10              RETURN IsInOrder(RestOfWord)
11          END IF
12      END IF
13  END FUNCTION
```

Break we can't set Sang Phu)

(a) State what is meant by recursion using this algorithm as an example.

(b) The algorithm is tested with the call IsInOrder("Z").
(i) State the value which will be returned.
(ii) State the lines of the algorithm which will be executed.

(c) Explain what happens if the algorithm is tested with a call IsInOrder(" ") where the value of the argument is the empty string.

(d) Explain what happens when the algorithm is tested with the call IsInOrder("APE"). You should show each call made, the lines of the algorithm executed and the return value of each call. You may use a diagram to illustrate you answer.

a) ~~Definition~~ of function (IsInOrder) is called within the definition of that function at line 10.

Stopping condition when length of word is one letter

The two base cases can be found in Line 2 and Line 7.

b i) True

ii) 1, 2, 3, 4, 12, 13.

c) An infinite loop would occur

d)

3* An algorithm for converting a number n from denary to octal uses the three built-in functions:

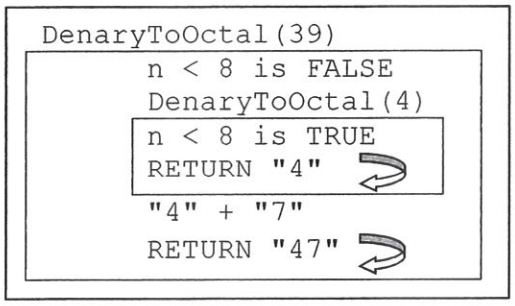
Function	Description
INTMOD(Number, Divisor)	returns the remainder when the first parameter is divided by the second parameter.
	e.g. **INTMOD(7, 3)** returns 1
INTDIV (Number, Divisor)	returns the quotient when the first parameter is divided by the second parameter.
	e.g. **INTDIV (7, 3)** returns 2
SUBSTR (ThisString, Start, Length)	forms a substring from `ThisString`, starting at `Start` (with first index in string zero) and taking `Length` characters
	e.g. **SUBSTR ("abcd", 1, 2)** returns **"bc"**

Study the pseudocode given in the following:

```
01  FUNCTION DenaryToOctal(n : INTEGER) RETURNS STRING
02      OctalDigits ← "01234567"
03      IF n < 8
04          THEN
05              TempString ← SUBSTR(OctalDigits, n, 1)
06          ELSE
07              // '+' is the concatenation operator
08              TempString ← DenaryToOctal (INTDIV(n, 8)) +
                SUBSTR(OctalDigits, INTMOD(n, 8),1))
09      ENDIF
10      RETURN TempString
11  ENDFUNCTION
```

(a) Identify where and why this is a recursive function.

The diagram shows the execution of the call `DenaryToOctal(39)`.



(b) Draw a similar diagram to show the execution of the call DenaryToOctal (67).

Tutorial 7.2 [Modules]

1* **(a)** A software house encourages its programmers to use libraries when developing software.

Explain why it is good programming practice to use such libraries when developing computer programs. [4]

(b) If a program calls a library routine that has not been loaded correctly, an error occurs.

Name this type of error. [1]

2* The museum collects funds from many sources including entrance fees, purchases from the shop and restaurants and donations. It also has to produce the payroll for staff and handle payments to suppliers.

Each of these financial applications is currently controlled by a separate system.

The museum's board of governors has decided to commission a software house to produce an integrated package to handle all financial transactions.

(a) Explain why local and global variables will be important in the production of the software. [4]

(b) The solution contains a number of modules. Three of the modules are

- A module to input relevant data to the system.
- A module to calculate the pay for individual workers.
- A module to output pay slips and electronic transfer of pay to workers' bank accounts.

State the parameters that would be passed between
(i) the input module and the pay calculation module,
(ii) the calculation module and the output module, giving a reason why each parameter is necessary. [6]

3* **(i)** State two differences between a built-in function and a user-defined function. [2]
(ii) State two things that built-in and user-defined functions have in common. [2]