TEMASEK JUNIOR COLLEGE

Temasek Junior College

JC H2 Computing

Problem Solving & Algorithm Design 8 – Modularisation

1 Modularisation

Simple solution algorithm:

1. define the problem
2. write down the control structures required to reach a solution; and
3. devise a **structured-algorithm** solution, which uses a combination of sequence, selection and repetition control structures.

Complex problem:

You must first identify the major tasks to be performed in the problem, then divide the problem into sections that represent those tasks. These sections can be considered subtasks or functions. Once the major tasks in the problem have been identified, the programmer may then need to look at each of the subtasks and identify within them further subtasks, and so on. This process of identifying first the major task, them further subtasks within them, is known as **top-down design** or **functional decomposition.**

By using this top-down design methodology, the programmer is adopting a modular approach to program design. That is, each of the subtasks or functions will eventually become a module within a solution algorithm or program.

2 Hierarchy or structure charts

Hierarchy or **structure charts** use a **top-down approach** to explain how a program is put together. Starting from the program name, the programmer breaks the problem down into a series of steps.

Each step is then broken down into finer steps so that each successive layer of steps shows the subroutines that make up the program in more detail.

The overall hierarchy of a program might look like this:

- programs are made up of modules
- modules are made up of subroutines and functions
- subroutines and functions are made up of algorithms
- algorithms are made up of lines of code
- lines of code are made up of statements (instructions) and data.

In larger programs **a module** is a self-contained part of the program that can be worked on in isolation from all the other modules.

The text in a hierarchy chart at each level consists of only a few words – if you want more detail about what the process involves you need to move further down the diagram. The component parts for each section are organised from left to right to show how the system will work.

A structure diagram [for data or program] shows the overall problem at the top and each level below shows a further refinement, breaking it into smaller problems.

Example 1

A simple calculator is to be created.

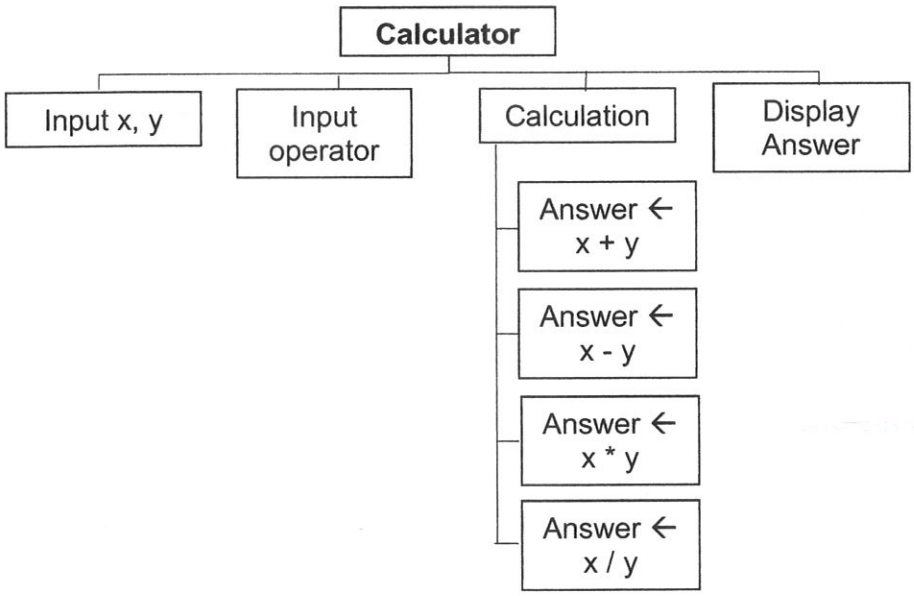
The calculator is to be used as follows:

- User inputs 2 numbers (x and y).
- User inputs an operator (+, –, * or /).
- The calculator computes the answer.
- The calculator displays the answer.

Draw a structure diagram **[program]** for the calculator. The first element is provided. [5]

Calculator

Solution:



Example 2 Construct the structure diagram for Employee Emergency Contact Form for department of data collection

EMPLOYEE EMERGENCY CONTACT FORM

Name
Department

Personal Contact Info:

Home Address
City, State — Postal code
Home Telephone — Hand Phone

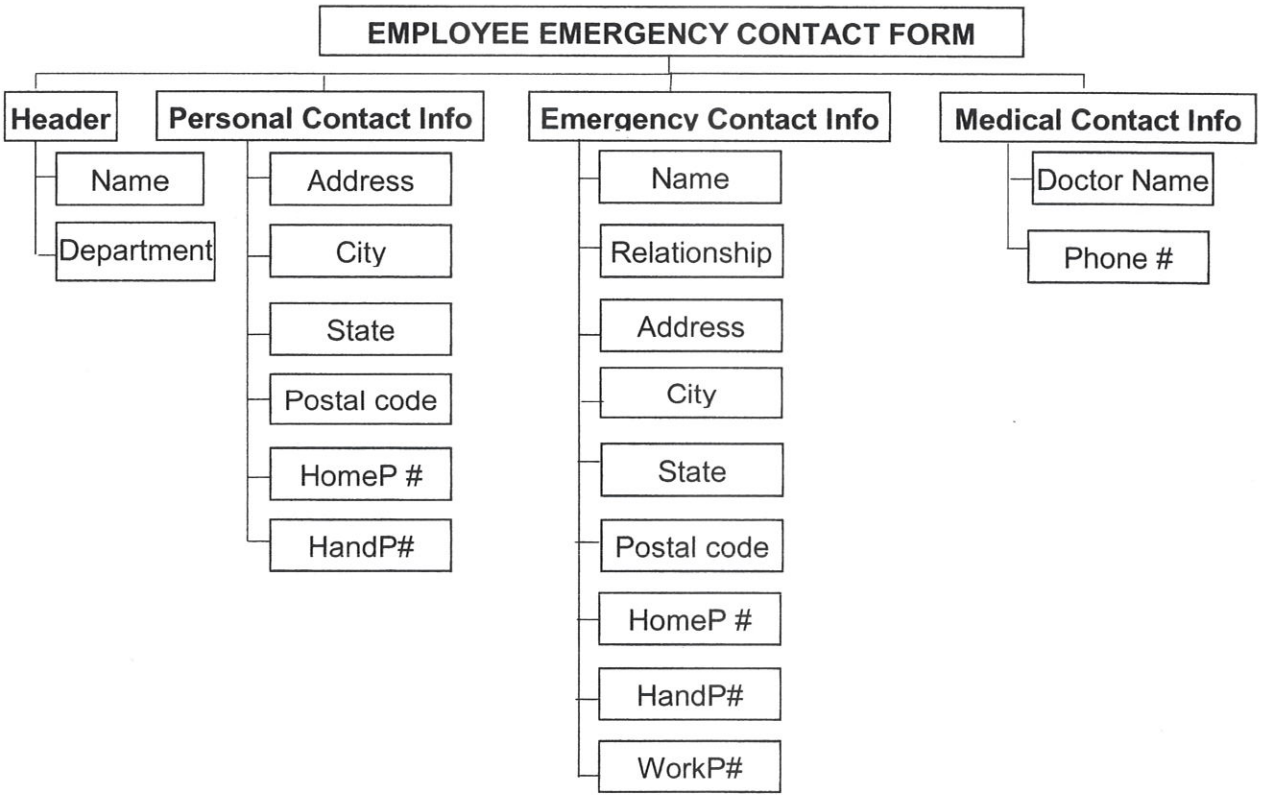
Emergency Contact Info:

Name — Relationship
Address
City, State — Postal code
Home Telephone — Hand Phone
Work Telephone

Medical Contact Info:

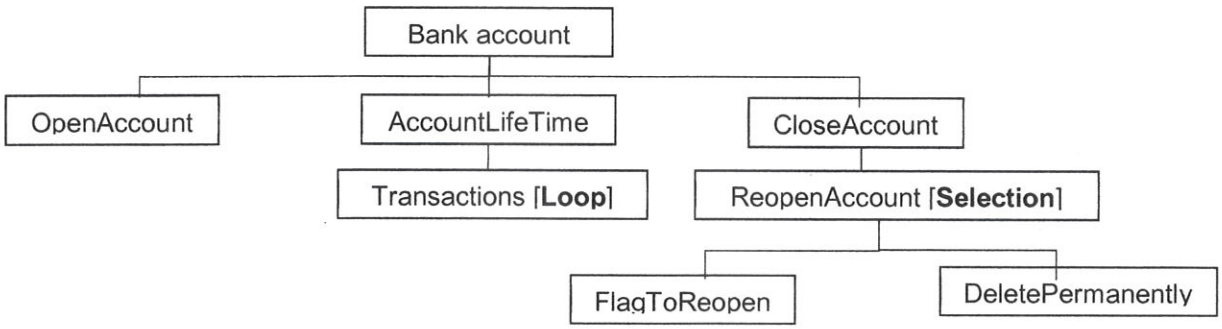
Doctor Name — Phone #

Structure diagram **[data]**



Example 3 9608_w18_qp_42

A bank provides bank accounts to customers.
The program structure diagram is given below:



a) The following pseudocode represents the operation of the bank account.

```
CALL OpenAccount()
CALL AccountLifeTime()
CALL CloseAccount()

PROCEDURE AccountLifeTime()
      REPEAT
            CALL Transactions()
      UNTIL AccountClosed() = TRUE
ENDPROCEDURE

PROCEDURE CloseAccount()
      IF ReopenAccount() = TRUE THEN
            CALL FlagToReopen()
      ELSE
            CALL DeletePermanently()
      ENDIF
ENDPROCEDURE
```

b) A transaction can be a credit (deposit) or a debit (withdrawal).

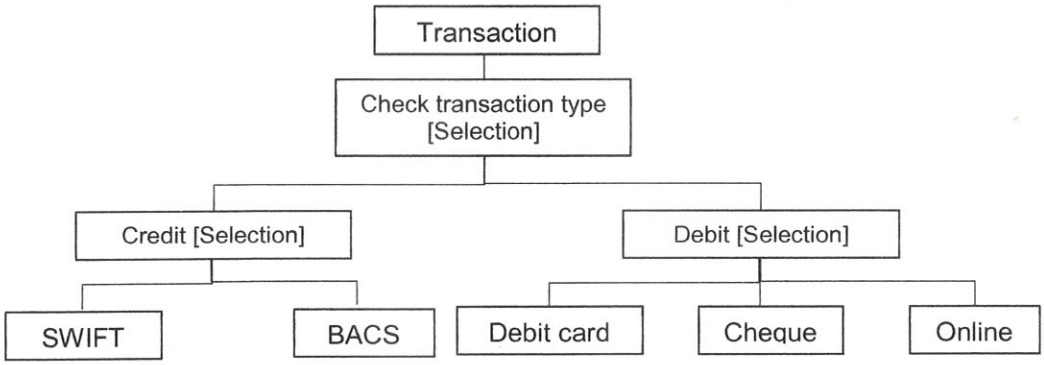
There are two types of transaction that are credits (deposits). These are:

- SWIFT payment
- BACS payment

There are three types of transaction that are debits (withdrawals). These are:

- Debit card payment
- Cheque payment
- Online payment

The **program** structure diagram to represent these additional requirements:

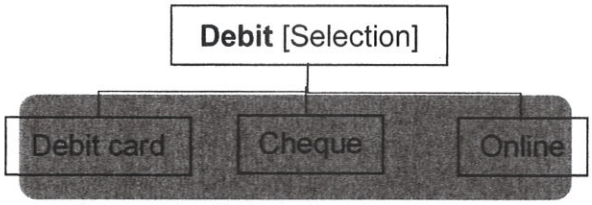


3 Modularisation is the process of dividing a problem into separate tasks, each with a single purpose.

(a) Modularisation provides abstraction

- Abstraction is the process of paying attention to important properties while ignoring nonessential details.
- Modules or subroutines are way to achieve abstraction.

Example 4



Example 5 a payroll program can call a module named CPF_EMPLOYEE. The mathematical details of the function can be written later, or be written by someone else or even purchased from an outside source. When you plan your payroll program, you pay attention only to the fact that somehow a CPF for employee will have to be calculated; you save the details for later.

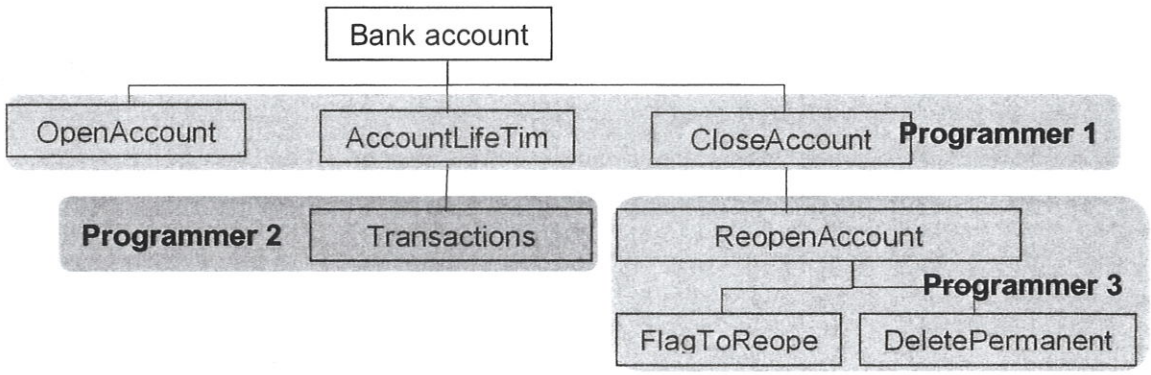
Employee's age (years)	Contribution Rates from 1 Jan 2016 *(for monthly wages ≥ $750)*		
	By Employer (% of wage)	By Employee (% of wage)	Total (% of wage)
55 and below	17	20	37
Above 55 to 60	13	13	26
Above 60 to 65	9	7.5	16.5
Above 65	7.5	5	12.5

(b) Modularisation allows multiple programmers to work on a problem

- When you dissect any large task into modules, you gain the ability to divide the task among various people. Professional software developers are able to come out with new programs in weeks or months instead of years by dividing large programs into modules and assigning each module to an individual coder or programming team.

(c) Modularisation allows you to reuse your work

- If a subroutine or function is useful and well written, you may want to reuse it multiple times within a program or in other programs.



1995

Example 6 A routine VALID_DATE() that checks

- the year (not lower than 1900 or higher than 2999),
- the month (not lower than 1 or higher that 12) and
- the day (between 1 to 30 or 31 for months other than February, between 1 to 28 or 29 for February depending on whether it is a leap year)
- to make sure it is valid is useful in many programs written for a business. A program that uses a personnel file that contains each employee's birth date, hire date, last promotion date, and termination date can use the date-validate module (VALID_ DATE) four times with each employee record. Other programs in an organization can also use the module (VALID_ DATE); these include programs that plan employee's birthday parties and calculate when loan payments should be made.'

• If a group of subroutines can be reused in many other applications, this certainly reduces the time and effort it takes to build that application. It also improves your application's **reliability**.
• Similarly, software that is reusable saves times and money and is more reliable.
• If you create the functional components of your programs as stand-alone modules and test them in your programs, then when you use the modules in future applications, much of the work required will already be done.

(d) Modularisation allows you to identify structure more easily
• When you combine several programming tasks into modules, it may be easier for you to identify structures.

4 Summary:

- In a **structured program:**
  - Only three basic constructs are used; *sequence* (one instruction after another), *selection* (e.g. IF..ELSE..ENDIF statements) and *iteration* (e.g. DOWHLE..ENNDDO, REPEAT-UNTIL and FOR..ENDFOR loops).
  - Each program module is generally not very long and is easily comprehensible.

- **Structured programming** is a methodical approach to the design of a program that emphasises breaking large and complex tasks into smaller subtasks [Tool: structure diagram for data and program]. This is also referred to as *structured planning technique.*

- **Modularity** refers to the concept of making multiple modules first and then linking and combining them to form a complete system. Modularity enables re-usability and minimizes duplication.
  - In addition to re-usability, modularity also makes it easier to fix problems as bugs can be traced to specific system modules, thus limiting the scope of detailed error searching.
  - *Modular programming* is an extensively used concept based on modularity. Modularity is also a feature of object oriented programming.

- **Modular programming** (also known as **top down design** and **stepwise refinement**) is a software design technique that increases the extent to which software is composed of separate, interchangeable components called **modules** by breaking down program functions into modules, each of which accomplishes one function and contains everything necessary to accomplish this.

  The advantages of **modular** programming can be stated as:
  - Individual modules can be separately tested.
  - Modules can be kept in a module library and reused in other programs.
  - Long programs that are split into modules are easier to read, debug and maintain.
  - The modular approach means that several people in a team of programmers can each work on separate modules, thus shortening the total development time for a large project.

- **Top-down programming** or **stepwise refinement** is a particular type of structured programming, where the overall problem is defined in simple terms and then split into a number of smaller subtasks. Each of these subtasks is successively split and refined until they are small enough to be understood and programmed. It is an axiom of this approach that new ideas that were not ill the original global design should not be added. This ensures elements are not missed.

- **Top-down design methodology** allows the programmer to concentrate on the overall design of the algorithm without getting too involved with the details of the lower level modules. Another benefit of top-down design is that separate modules, once identified and written, are easily understood, can be reused, and can be independently modified if necessary.

- **Modular design** is a method of organising a large computer program into self-contained parts, *modules,* which can be developed simultaneously by different programmers or teams of programmers. Modules have clearly defined relationships with the other parts of the system, which enables them to be independently designed, programmed and maintained. Some modules may be independently executable programs, while others may be designed to provide facilities for the suite of programs to which they belong.

  Conceptually, modules represent a separation **of** concerns, and improve maintainability by enforcing logical boundaries between components. Modules are typically incorporated into the program through interfaces. A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface.

Example 7 Student Management System (Python program – see annex)
The administration manager of a school wishes to keep records of the students. Each student has the following data recorded:

- **Name** represents the student's name and is at most 60 characters.
- **Telephone** is the telephone number of the contact in case of a problem and is fixed at eight digits.
- **Email** represents the student's email and is at most 80 characters.

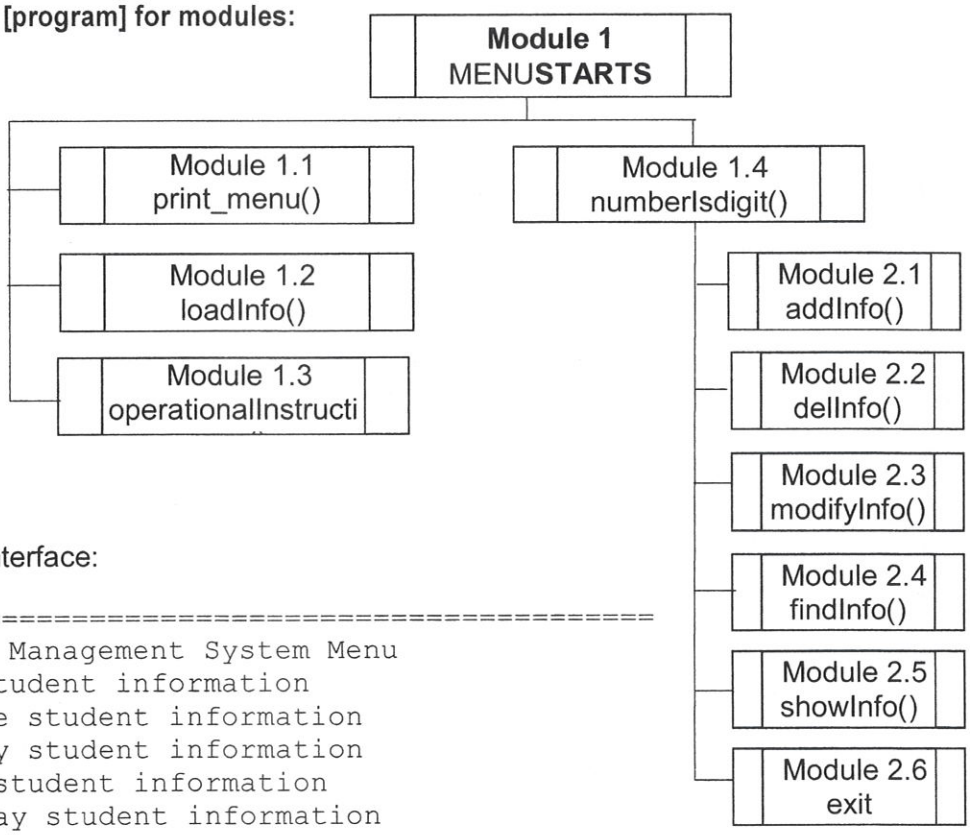
Create a student management system which will achieve adding, deleting, modifying, and checking dictionary data in the list

A file which contains relevant particulars of each student in a class has to be created. There is a maximum of 25 students in each class.

(a) You are to write the algorithm of a function, MENU**STARTS**, which when called will allow the user to enter and store data into a text file named **STUDENTS.DAT**. The file is created fresh, and has the following structure.

<Name><Telephone><Email>
<Name><Telephone><Email>
<Name><Telephone><Email>

Structure diagram [program] for modules:



Command Line Interface:

```
==================================================
      Student Management System Menu
      1:Add student information
      2:Delete student information
      3:Modify student information
      4:Find student information
      5:Display student information
      6:Exit the student management system
==================================================
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1:Added|2:Delete|3:Modified|4:Search|5:Display|6:Exit the system
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Please enter instructions:
```

Example 8 Project A

The manager of a business wishes to keep records of the employees. Each employee has the following data recorded:

- **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.
- **Name** represents the employee's name and is at most 20 characters.
- **TelephoneNumber** is the telephone number of the contact in case of a problem and is fixed at seven digits.
- **EmployeeType** is used to indicate whether the employee is a salaried employee (S) or an hourly paid employee (H).

A file which contains relevant particulars of each employee in a department has to be created. There is a maximum of 20 employees in each department.

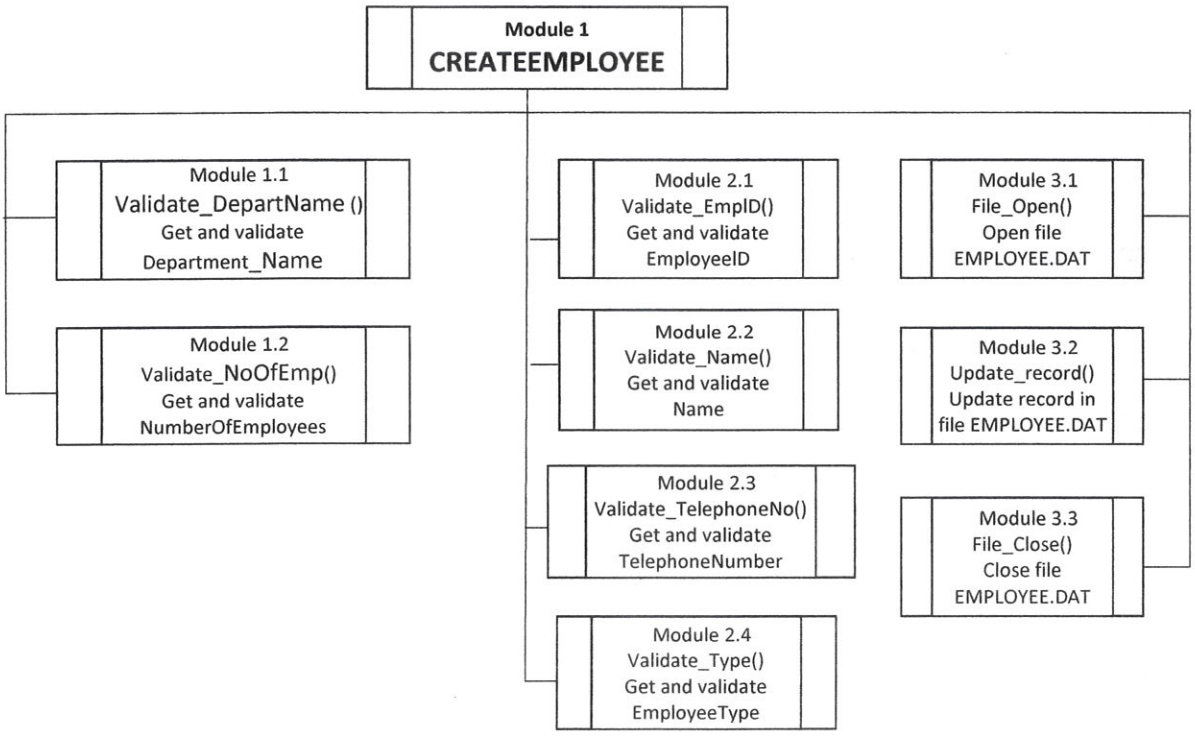
(a) You are to write the algorithm of a function, **CREATEEMPLOYEE**, which when called will allow the user to enter and store data into a text file named **EMPLOYEE.DAT**. The file is created fresh, and has the following structure.

<Department><NumberOfEmployees>
<EmployeeID><Name><TelephoneNumber><EmployeeType>
<EmployeeID><Name><TelephoneNumber><EmployeeType>
<EmployeeID><Name><TelephoneNumber><EmployeeType>

Department is the name of the department and is at most six characters, e.g. MAN4E2. **NumberOfEmployees** is the number of employees in the department, e.g. 18.

(b) Produce a set of test data explaining the purpose of each value

Structure diagram [program] for modules:



```
//Algorithm of a function, CREATEEMPLOYEE, which when called will
allow the user to enter and store data into a text file named
EMPLOYEE.DAT.

FUNCTION CREATEEMPLOYEE() RETURNS BOOL
     DECLARE Department: STRING
     DECLARE NumberOfEmployees: INTEGER
     DECLARE RecordCount: INTEGER
     DECLARE EmployeeID, Name, TelephoneNumber: STRING
     DECLARE EmployeeType: BOOL

     OPENFILE EMPLOYEE.DAT FOR OUTPUT

     REPEAT
          DISPLAY  "Name of Department"
          GET Department
     UNTIL Validate_DepartName(Department) is TRUE

     REPEAT
          DISPLAY  "Number Of Employees (1 to 20)"
          GET NumberOfEmployees
     UNTIL Validate_NoOfEmp(NumberOfEmployees) is TRUE

     WRITE Department, NumberOfEmployees to EMPLOYEE.DAT

     SET RecordCount to 1

     DOWHILE RecordCount <= NumberOfEmployees
        REPEAT
           DISPLAY  "Employee ID:"
          GET EmployeeID
        UNTIL Validate_EmpID(EmployeeID) is TRUE

        REPEAT
           DISPLAY  "Employee Name:"
          GET Name
        UNTIL Validate_Name(Name) is TRUE

        REPEAT
              ....
        UNTIL Validate_TelephoneNo(TelephoneNumber) is TRUE

        REPEAT
              ....
        UNTIL Validate_EmpType(EmployeeType) is TRUE

        WRITEFILE EmployeeID, Name, TelephoneNumber, EmployeeType to EMPLOYEE.DAT

        Increment RecordCount

     ENDDO

     CLOSEFILE EMPLOYEE.DAT

     RETURN True
END FUNCTION
```

```
FUNCTION Validate_EmployeeID(EmployeeID:STRING) RETURNS BOOL

    //Length check
    IF Length of EmployeeID > 6 THEN
        RETURN False
    ENDIF

    IF the first character of EmployeeID is not an upper case letter
      THEN
        RETURN False
    ENDIF

    IF any character of the remaining five of EmployeeID is not digit
      THEN
        RETURN False
    ENDIF

    RETURN True

END FUNCTION

// 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.
```

Test data for Get_and_Validate_ EmployeeID():

EmployeeID	Validate_ EmployeeID		Reason
A23588	True	Valid	Satisfied all condition
A2358	True	Valid	At most six characters
A235889	False	Invalid	More than six characters
	False	Invalid	Empty input
923588	False	Invalid	first character is not an upper case letter
a23588	False	Invalid	first character is not an upper case letter
AB3588	False	Invalid	One of the remaining five is not a digit
A235X8	False	Invalid	One of the remaining five is not a digit

- Appropriate test cases (normal, abnormal, erroneous and boundary data) for testing algorithms and debugging techniques.

With reference to the design of Project A:

Explain the following:

(a) What is structured program?
(b) What is modularity?
(c) What is a module?
(d) What structured programming?
(e) What modular programming?
(f) What is modular design?

Tutorial 8 [Modularisation]

1 Top-down design is a technique used to produce solutions to complex problems.

(a) Explain the term *top-down design.* [3]

(b) Explain three advantages of using top-down design to solve complex problems. [6]

(c) Explain three techniques that can be used to ensure that program code is understandable and can be easily maintained. [6]

2 The following pseudocode represents three separate modules from an algorithm design. The module contents are not shown.

```
FUNCTION Search(AA : INTEGER, BB : STRING) RETURNS INTEGER
        :
ENDFUNCTION

FUNCTION Allocate() RETURNS BOOLEAN
        :
ENDFUNCTION

PROCEDURE Enable(CC : INTEGER, BYREF DD : INTEGER)
        :
ENDPROCEDURE
```

A fourth module, Setup(), refers to the previous three modules as follows:

```
PROCEDURE Setup()
        WHILE Authorised = TRUE
                ThisValue Search(27, "Thursday")
                Authorised Allocate()
                CALL Enable(ThisValue, 4)
        ENDWHILE
ENDPROCEDURE
```

Draw a structure chart to show the four modules and the parameters that these pass between them.

Q3 Decomposition and modularity are important concepts in software development. Explain what do they meant and how do they benefit in software development.

Annex: Python for **Example 7**

```
""" Use python to create a student management system"""
""" Python functional programming realization ideas """
""" Use the dictionary to store the student's name, telephone numbers,
and email """
""" Use list to install student information """
""" Achieve adding, deleting, modifying, and checking dictionary data
in the list """
""" TJC JC1 Computing 2021 """
""" Programmer Fong KK"""
""" Version 2021.NoneOOP v1.1"""

students = []
info  =  "1:Added|2:Delete|3:Modified|4:Search|5:Display|6:Exit  the
system"

def print_menu():
      """Student Management System Menu"""
      print("="*50)
      print("\tStudent Management System Menu")
      print("\t1:Add student information")
      print("\t2:Delete student information")
      print("\t3:Modify student information")
      print("\t4:Find student information")
      print("\t5:Display student information")
      print("\t6:Exit the student management system")
      print("=" * 50)

def operationalInstructions():
      print("\n")
      print("~"*65)
      print(info)
      print("~"*65)

def addInfo():
      Newstudents = {}
      Newstudents['name'] = input("Please enter student name[at most
66 characters]:")
      for stu in students:
            if Newstudents["name"] == stu["name"]:
                  print("Duplicate name, please re-enter!")
                  MENUSTARTS()

      Newstudents['phone'] = input("Please enter the student's mobile
phone number[8 digits only]:")
      if len(Newstudents['phone']) != 8:
            print("Please  enter  the  phone  number  in  the  correct
format!")
            return
      Newstudents['email']  =  input("Please  enter  the  student's
email[at most 66 characters]:")
      students.append(Newstudents)
      saveInfo()
      showInfo()
```

```
def delInfo():
    def_name = input("Please enter the name of the student you want
to delete:")
    print("Are you sure you want to delete the student information?
Deletion is an irreversible operation, please consider carefully!")
    msg = input("To confirm deletion, please press 'y', to return,
please press 'n'")
    if msg == "y":
        for stu in students:
            if def_name == stu["name"]:
                students.remove(stu)
        saveInfo()
        showInfo()
    else:
        showInfo()

def modifyInfo():
    modify_name = input("Please enter the name of the student you
want to modify:")
    flag = 0
    msg = input("To confirm the modification, please press 'y', to
return, please press 'n'")
    if msg == "y":
        for stu in students:
            if modify_name == stu["name"]:
                new_name = input("Please enter a new name:")
                stu["name"] = new_name
                flag = 1
                saveInfo()
                showInfo()
        if flag == 0:
            print("The student %s does not exist" % modify_name)
            showInfo()
    else:
        showInfo()

def findInfo():
    find_name = input("Please enter the name you are looking for:")
    flag = 0
    for stu in students:
        for value in stu.values():
            if find_name == value:
                flag = 1
                print("found it:{}".format(find_name))
                break
    if flag == 0:
        print("could not find it:{}".format(find_name))

def loadInfo():
    global students
    file = open("students.txt","a+")
    file.seek(0,0)
    content = file.read()
    if len(content) > 0:
        students = eval(content)
```

```
def saveInfo():
      file = open("students.txt","w")
      file.write(str(students))
      file.close()

def showInfo():
      print("\n")
      print("\tCurrent student system information")
      print("~"*80)
      print("\tName\t\tPhone number\t\tEmail")

      for st in students:
            msg = "\t"+st["name"]+"\t\t"+st["phone"]+"\t"+st["email"]
            print(msg)
      print("~"*80)

def numberIsdigit():
      number = input("Please enter instructions:")
      if number.isdigit():
            number = int(number)
            if number == 1:
                  addInfo()
            elif number == 2:
                  delInfo()
            elif number == 3:
                  modifyInfo()
            elif number == 4:
                  findInfo()
            elif number == 5:
                  showInfo()
            elif number == 6:
                  msg = input("Are you sure you want to exit the student
management system? , If you are sure to exit, please enter'y', continue
to use please enter'n':")
                  if msg == "y":
                        print("Successfully exit the student management
system, welcome your next use!")
                        return
                  if msg == "n":
                        pass
            MENUSTARTS()
      else:
            print("Please enter the correct command, the command is a
number from 1-6!")

def MENUSTARTS():
      print_menu()
      loadInfo()
      operationalInstructions()
      numberIsdigit()

MENUSTARTS()
```