Computer Science

Chapter 12 – Software Development

12.1 Program Development Life cycle

12.2 Program Design

12.3 Program Testing and maintenance

Learning Objectives

12.1 Program Development Life cycle

- Show understanding of the purpose of a development life cycle
- Show understanding of the need for different development life cycles depending on the program being developed (Including, waterfall, iterative, rapid application development (RAD)
- Describe the principles, benefits and drawbacks of each type of life cycle
- Show understanding of the analysis, design, coding, testing and maintenance stages in the program development life cycle

Learning Objectives

12.2 Program Design

- Use a structure chart to decompose a problem into sub-tasks and express the parameters passed between the various modules / procedures / functions which are part of the algorithm design
- Describe the purpose of a structure chart
- Construct a structure chart for a given problem
- Derive equivalent pseudocode from a structure chart.
- Show understanding of the purpose of state-transition diagrams to document an algorithm

Learning Objectives

12.3 Program Testing and maintenance

- Show understanding of ways of exposing and avoiding faults in programs.
- Locate and identify the different types of errors:

❖syntax errors

❖logic errors

❖run-time errors

- Correct identified errors
- Show understanding of the methods of testing available and select appropriate data for a given method (Including dry run, walkthrough, white-box, black-box, integration, alpha, beta, acceptance, stub)
- Show understanding of the need for a test strategy and test plan and their likely contents

Learning Objectives

12.3 Program Testing and maintenance

- Choose appropriate test data for a test plan (Including normal, abnormal and extreme/boundary)
- Show understanding of the need for continuing maintenance of a system and the differences between each type of maintenance (Including perfective, adaptive, corrective)
- Analyze an existing program and make amendments to enhance functionality

12.1 Program Development Life cycle

12.1.1 - The purpose of a development life cycle

The Systems development lifecycle (SDLC) is the process of developing software or information systems from start to finish also it will enables the production of high-quality, low-cost software, in the shortest possible production time. The goal of the SDLC is to produce superior software that meets and exceeds all customer expectations and demands.

Which model you may choose to use depends on a number of factors, including:

- The time-frame for development
- The size of the project
- The budget (also whether the budget is tightly fixed)
- The type of software being developed (e.g. GUI driven versus database driven)
- Whether the design of the final project can be accurately conceived at the Analysis phase.
- Whether final product is intended to be static or constantly added to.

SDLC has seven main phases: Planning, Analysis, Design, Development, Testing, Implementation, and Maintenance.

12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

Click the link below to watch the video:

https://www.youtube.com/watch?v=i-QyW8D3ei0

12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

SDLC is a process that defines the various stages involved in the development of software for delivering a high-quality product. SDLC stages cover the complete life cycle of a software i.e. from inception to retirement of the product.

Adhering to the SDLC process leads to the development of the software in a systematic and disciplined manner.

Purpose:

Purpose of SDLC is to deliver a high-quality product which is as per the customer's requirement.
SDLC has defined its phases as, Requirement gathering, Designing, Coding, Testing, and Maintenance. It is important to adhere to the phases to provide the Product in a systematic manner.

12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

For Example:

A software has to be developed and a team is divided to work on a feature of the product and is allowed to work as they want. One of the developers decides to design first whereas the other decides to code first and the other on the documentation part.

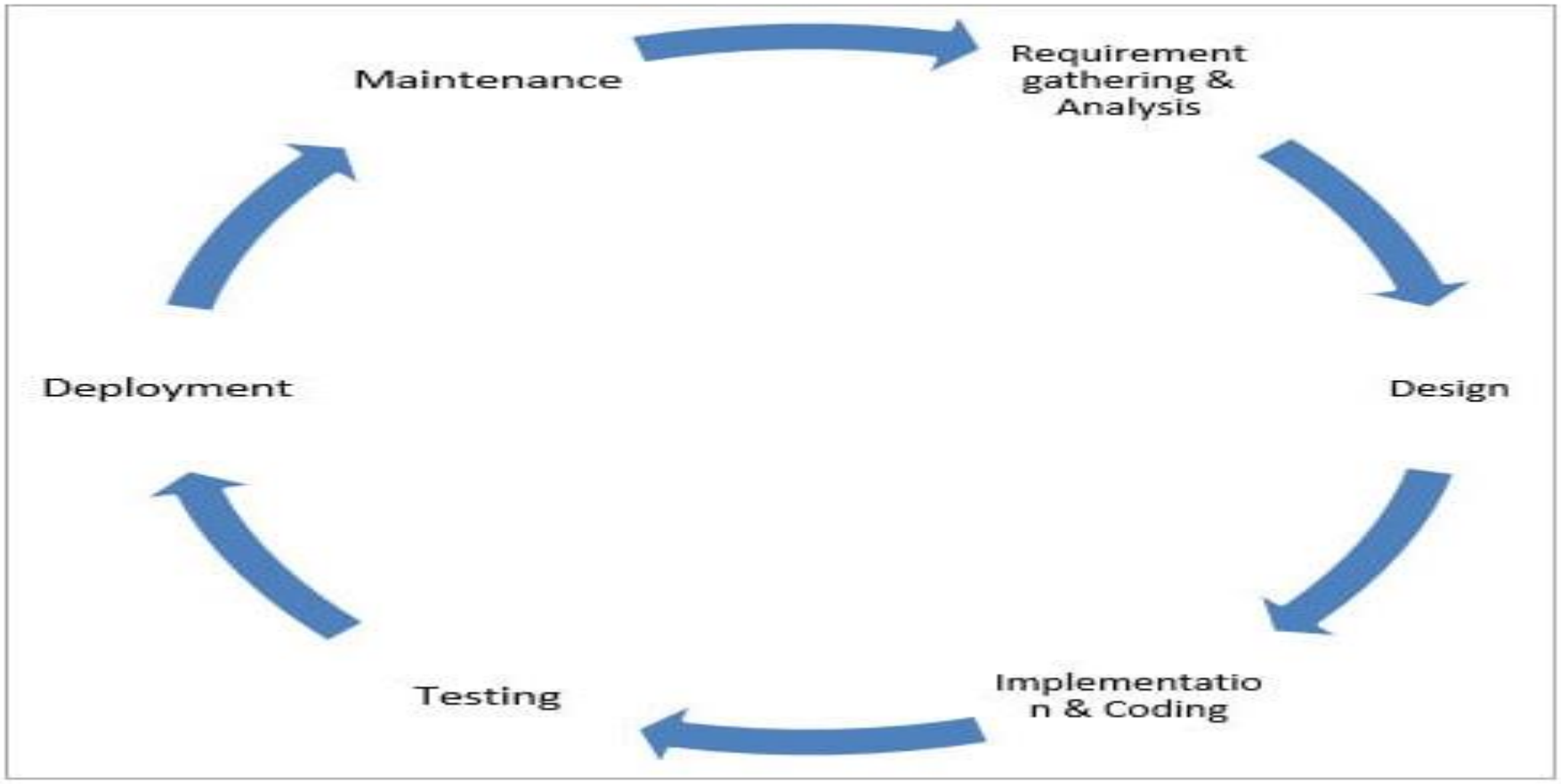
This will lead to project failure because of which it is necessary to have a good knowledge and understanding among the team members to deliver an expected product.

12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

SDLC Cycle:

SDLC Cycle represents the process of developing software: **Below is the diagrammatic representation of the SDLC cycle:**



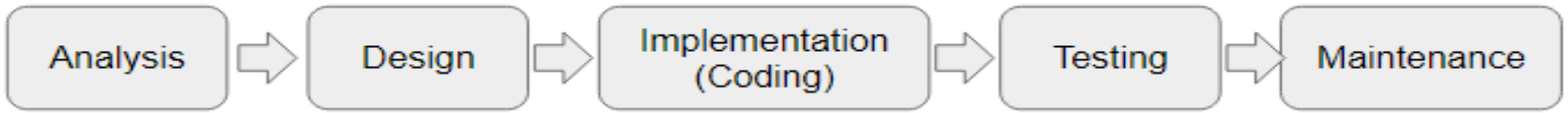
12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

SDLC Phases:

Given below are the various phases:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance



12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

1) Requirement Gathering and Analysis

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

1) Requirement Gathering and Analysis

For Example:

A customer wants to have an application which involves money transactions. In this case, the requirement has to be clear like what kind of transactions will be done, how it will be done, in which currency it will be done, etc.

Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

2) Design

In this phase, the requirement gathered in the document is used as an input and software architecture that is used for implementing system development is derived.

- Frontend - GUI Wireframes, Color schemes, brand requirements

- Processes - Data flow diagrams, Pseudocode, Flowcharts, Gantt charts, high level overview diagrams, abstraction & decomposition - breakdown down into modular parts, Structure Charts, State Transition Diagrams.

- Backend – ERD (entity relationship diagram), Database Schemas

12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

3) Implementation or Coding

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

- Coding of the project.
- Assets pulled together.
- Some testing also takes place - modular test, verification/validation testing.

12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

4) Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

Retesting, regression testing is done until the point at which the software is as per the customer's expectation. Testers refer document to make sure that the software is as per the customer's standard.

- Unit Testing - automated testing using predetermined test
- User Acceptance Testing
- Whole System Testing - server load / concurrent user tests

12.1 Program Development Life cycle

12.1.2 – Stages in the program development life cycle:

5) Deployment

Once the product is tested, it is deployed in the production environment or first UAT(User Acceptance testing) is done depending on the customer expectation.

In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

6) Maintenance

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

- Mostly bug fixes and minor feature addition / improvements.

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

A software life cycle model is a descriptive representation of the software development cycle. SDLC models might have a different approach but the basic phases and activity remain the same for all the models.

1) Waterfall Model:

Click the link below to watch the video on Waterfall model definition and example:

https://www.youtube.com/watch?v=Y_A0E1ToC_I

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

A software life cycle model is a descriptive representation of the software development cycle. SDLC models might have a different approach but the basic phases and activity remain the same for all the models.

1) Waterfall Model:

Waterfall model is the very first model that is used in SDLC. It is also known as the linear sequential model.

In this model, the outcome of one phase is the input for the next phase. Development of the next phase starts only when the previous phase is complete.

- First, Requirement gathering and analysis is done. Once the requirement is freeze then only the System Design can start. Herein, the document created is the output for the Requirement phase and it acts as an input for the System Design.
- In System Design Software architecture and Design, documents which act as an input for the next phase are created i.e. Implementation and coding.

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

1) Waterfall Model:

- In the Implementation phase, coding is done and the software developed is the input for the next phase i.e. testing.
- In the testing phase, the developed code is tested thoroughly to detect the defects in the software. Defects are logged into the defect tracking tool and are retested once fixed. Bug logging, Retest, Regression testing goes on until the time the software is in go-live state.
- In the Deployment phase, the developed code is moved into production after the sign off is given by the customer.
- Any issues in the production environment are resolved by the developers which come under maintenance.

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

1) Waterfall Model:

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

1) Waterfall Model:

Used for: Large projects where the specification/requirements are unlikely to change.

Advantages of the Waterfall Model:

- Waterfall model is the simple model which can be easily understood and is the one in which all the phases are done step by step.
- Deliverables of each phase are well defined, and this leads to no complexity and makes the project easily manageable.

Disadvantages of Waterfall model:

- Waterfall model is time-consuming & cannot be used in the short duration projects as in this model a new phase cannot be started until the ongoing phase is completed.
- Waterfall model cannot be used for the projects which have uncertain requirement or wherein the requirement keeps on changing as this model expects the requirement to be clear in the requirement gathering and analysis phase itself and any change in the later stages would lead to cost higher as the changes would be required in all the phases.

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

2) Iterative Model:

Click to watch the video for SDLC ITERATIVE MODEL:

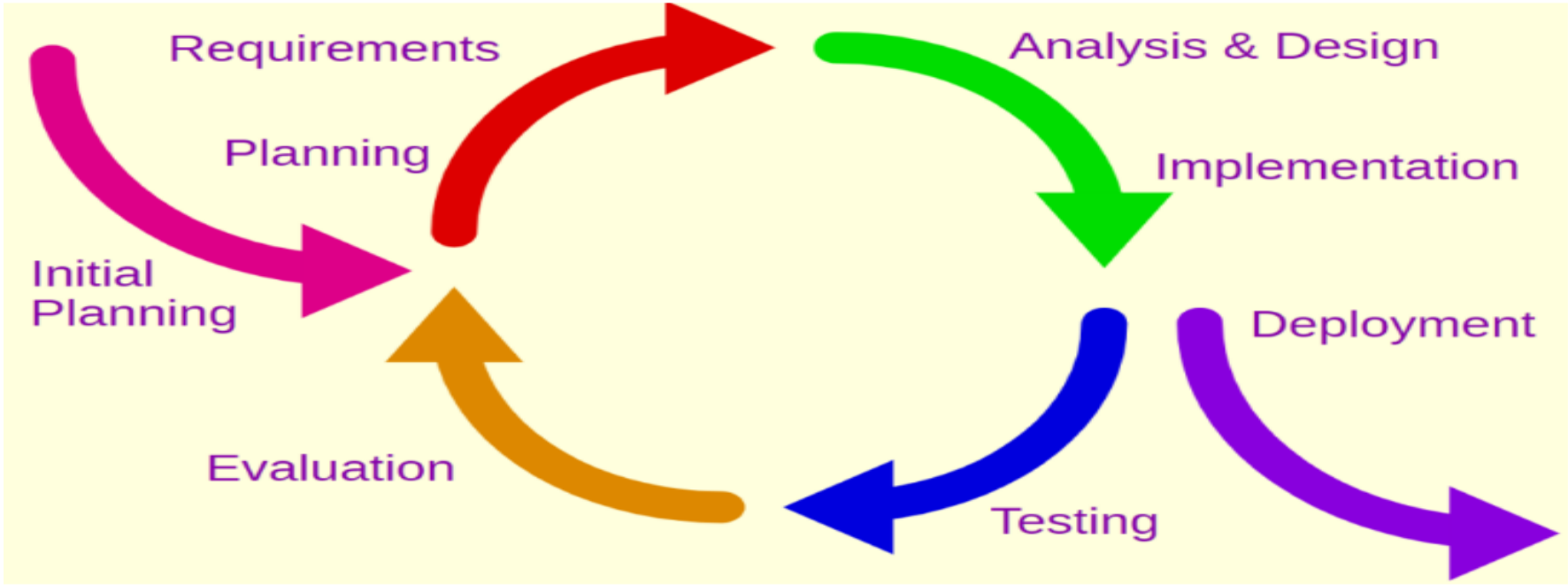
https://www.youtube.com/watch?v=whDPJkSdmCo

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

2) Iterative Model:

The iterative development model is an adaption of the waterfall model where initially only a subset of the full solution is initially created. Once this subset has been released the next iteration begins, each time round adding more components. The process doesn't necessarily end.



Source: Wikipedia

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

2) Iterative Model:

Useful for:

- Rapidly advancing technology, were the end product might not initially be known.

Advantages:

- Constant testing and user involvement means that the product should closely match user requirements
- Extra features can be added depending on the time remaining.

Disadvantages

- Difficult to manage and to know how long before the final product will be ready.

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

2) Iterative Model:

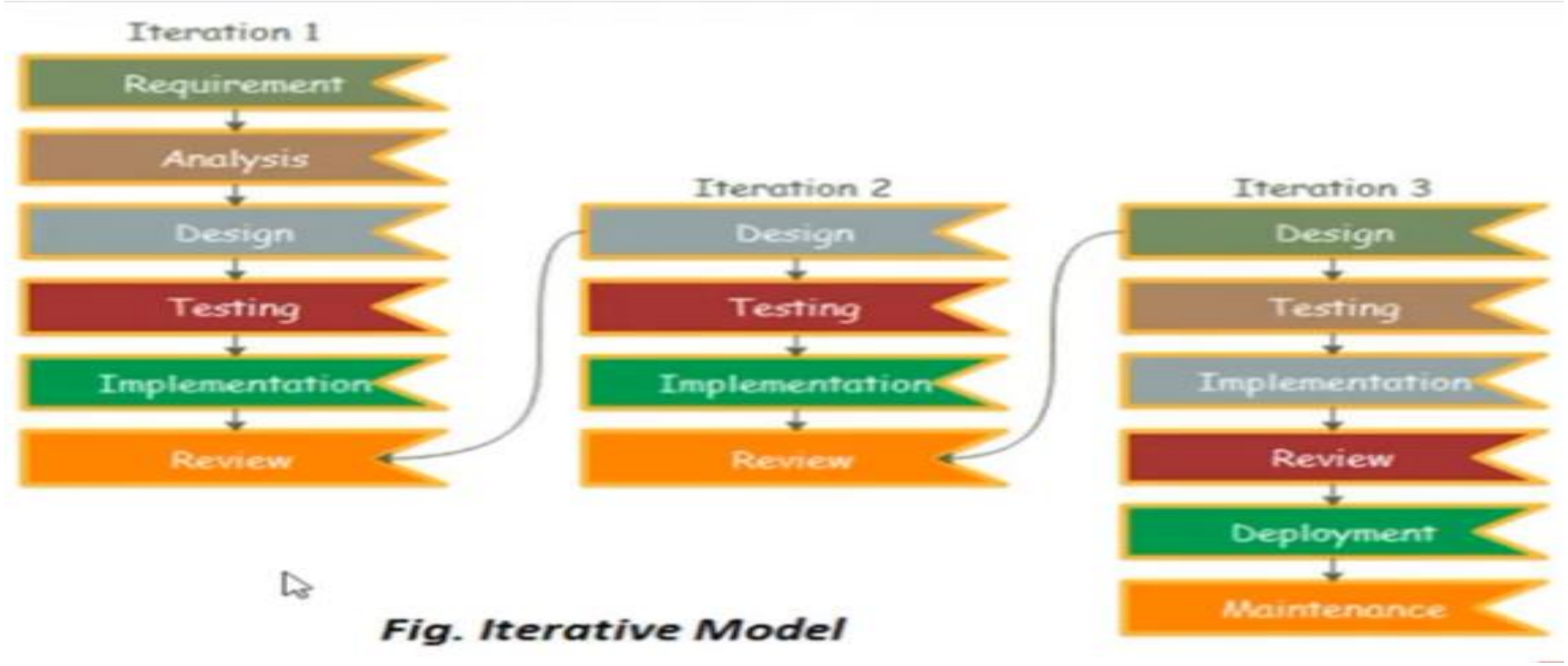


Fig. Iterative Model

12.1 Program Development Life cycle

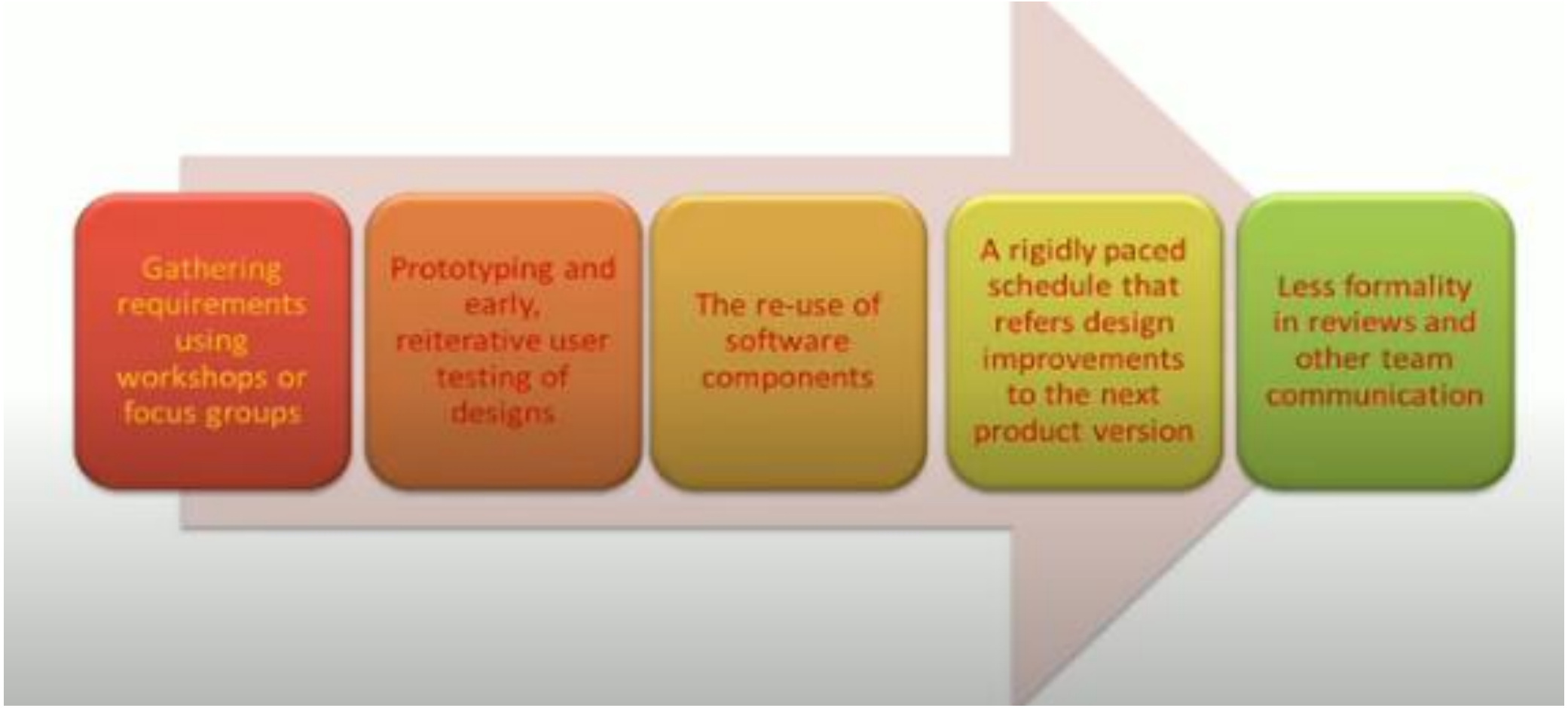
12.1.3 - Different development life cycles or Software Development Life Cycle Models:

3) Rapid Application Development (RAD)

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

3) Rapid Application Development (RAD)



12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

3) Rapid Application Development (RAD)

Rapid application development (RAD) concentrates on user involvement and continuous interaction between users and designers. It combines the planning and analysis phases into one phase and develops a prototype of the system.

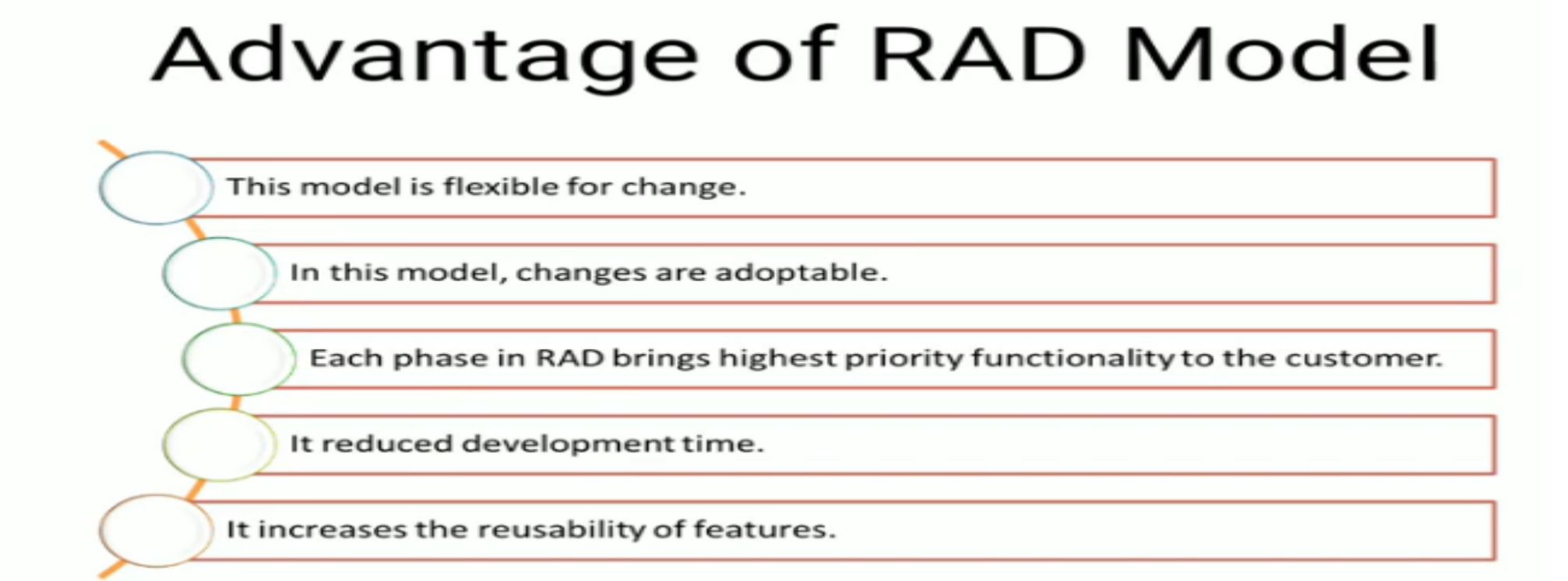
RAD uses an iterative process (also called “incremental development”) that repeats the design, development, and testing steps as needed, based on feedback from users. RAD is also known as low-code or no-code systems development approach. It uses visual interfaces to allow IS personnel to drag various components from the software library, connect them in specific ways, and create an application with little or no coding required.

After the initial prototype, the software library is reviewed, reusable components are selected from the library and integrated with the prototype, and testing is conducted. After these steps, the remaining phases are similar to the SDLC approach. One shortcoming of RAD is a narrow focus, which might limit future development. In addition, because these applications are built quickly, the quality might be lower.

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

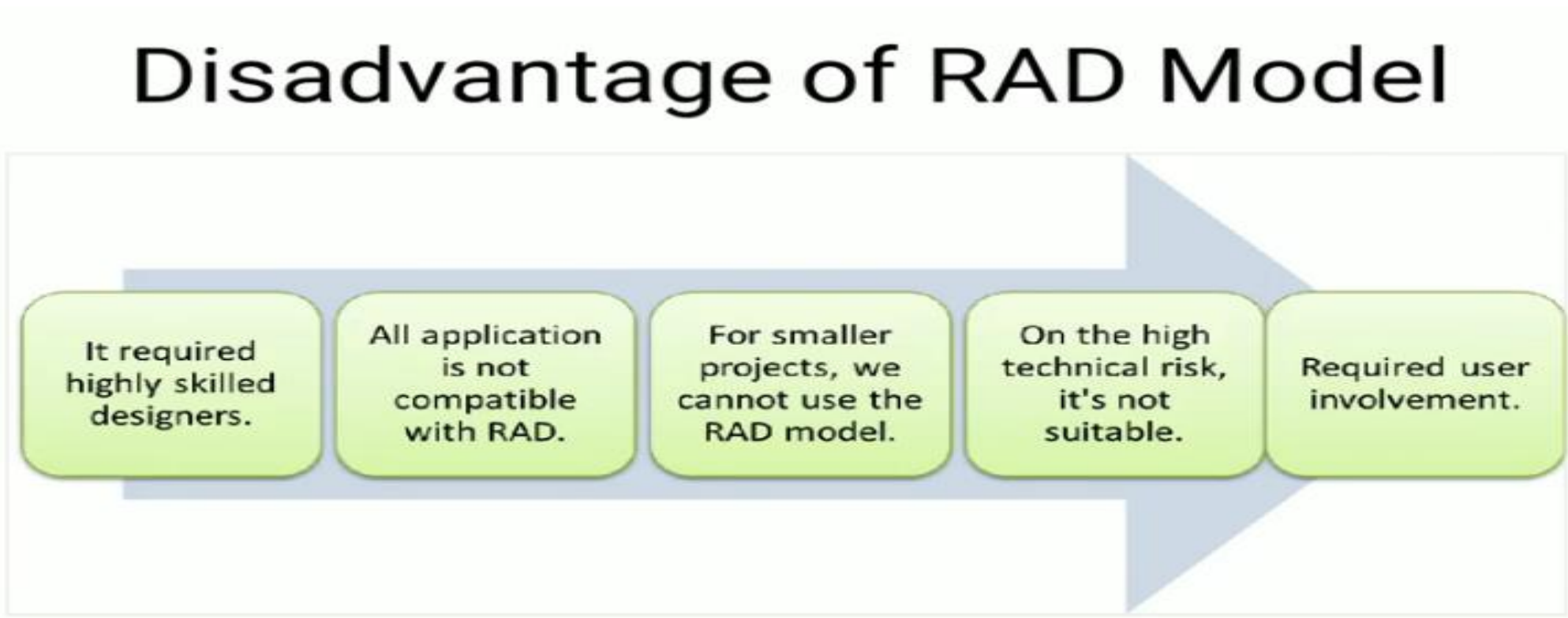
3) Rapid Application Development (RAD)



12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

3) Rapid Application Development (RAD)

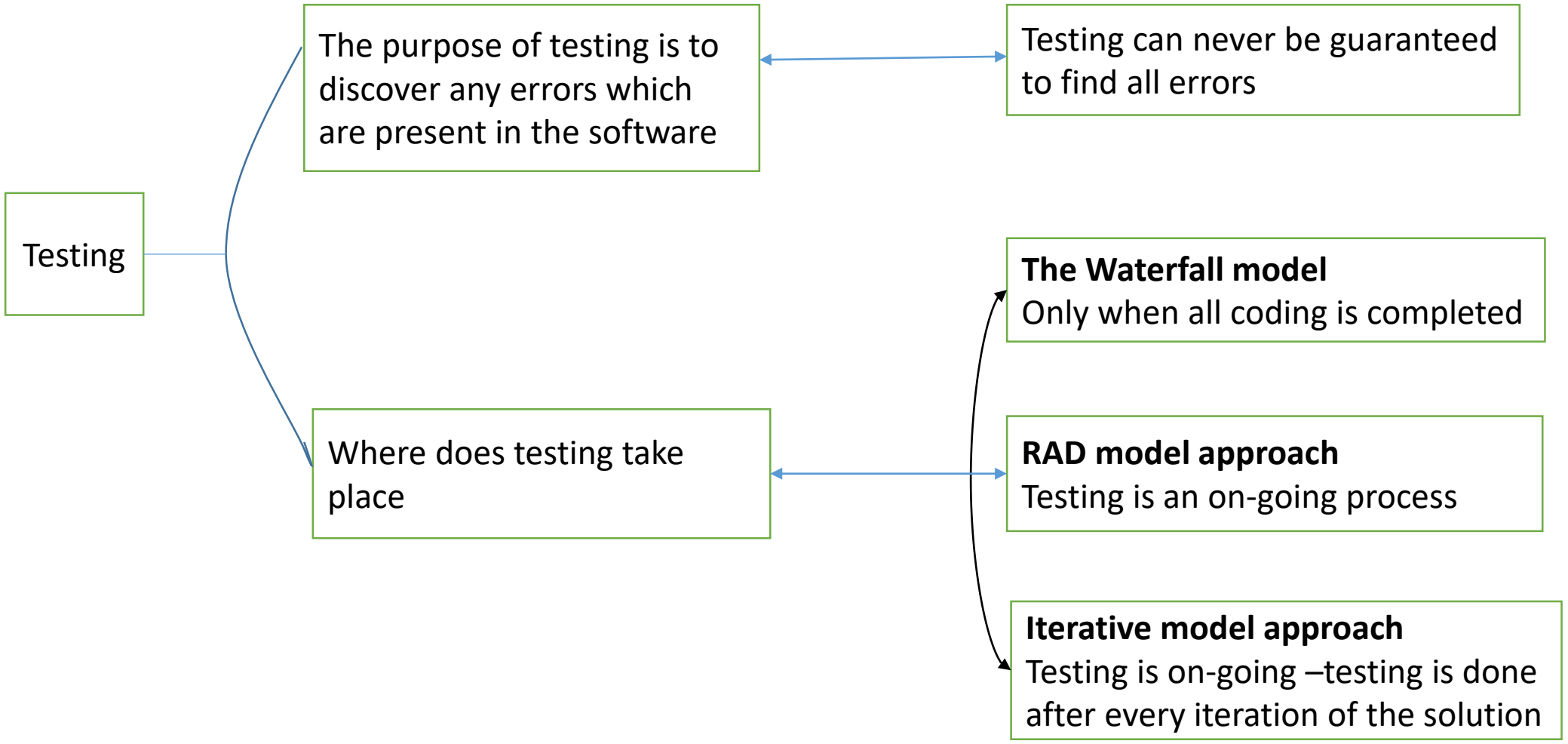


A lack of control means it’s difficult to use with large projects/organisations.

12.1 Program Development Life cycle

12.1.3 - Different development life cycles or Software Development Life Cycle Models:

Maintenance



12.2 Program Design

12.2.1 - The purpose of a structure chart

What Is the Purpose of a Structure Chart?

As a business grows in size, it also grows in complexity -- in terms of both the organization and the types of projects it undertakes. This increasing complexity makes it progressively more difficult to convey the organizational structure of the business and to manage project elements.

At its core, the purpose of a structure chart is to provide a basic, graphical representation of a more complicated organization or process. In the construction industry, for example, a structural chart may outline how the general manager and director have direct or indirect contact with both the engineering and design departments, but the two departments remain effectively isolated from each other. Such graphical representations allow the viewer to grasp the basic relationships between parts of a company or a process without getting bogged down in details.

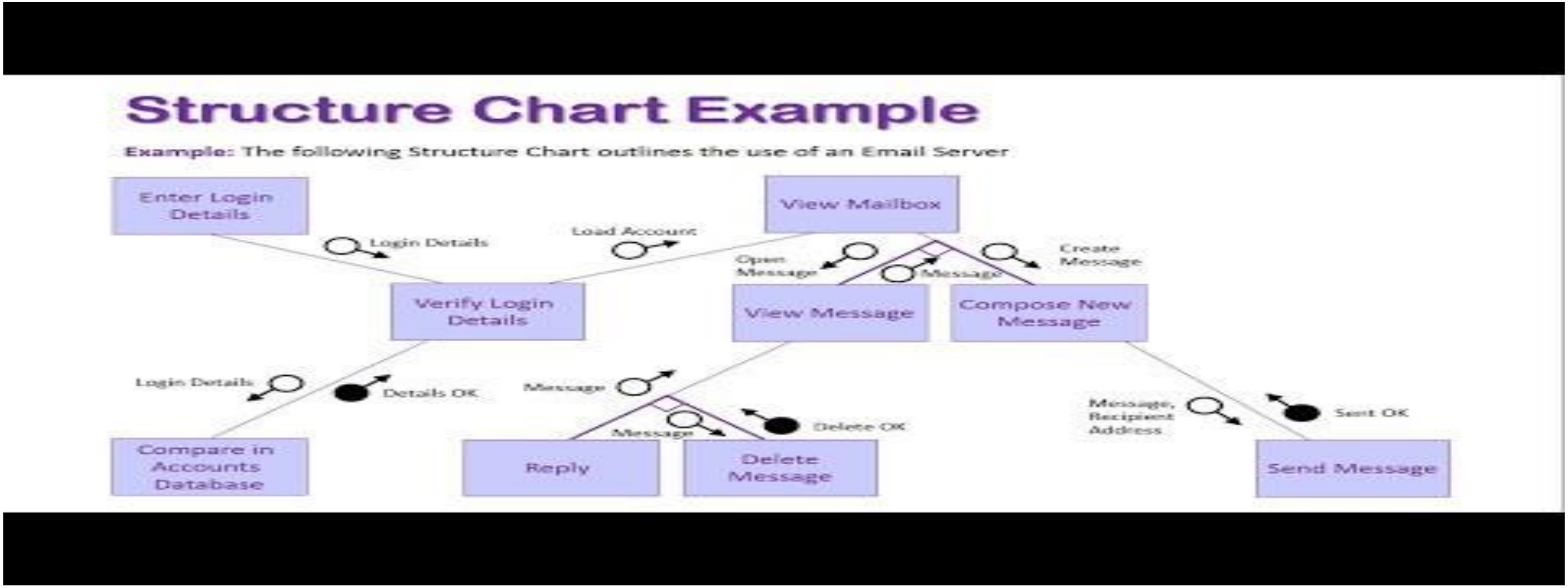
Structure charts provide a simple, visual solution to these kinds of problems.

12.2 Program Design

12.2.1 - The purpose of a structure chart

Click link below to watch the video for the Introduction to Structure Charts: 4 minutes

https://www.youtube.com/watch?v=QN2bjNplGlQ

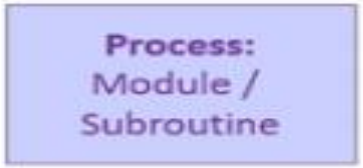


12.2 Program Design

12.2.1 - The purpose of a structure chart

Structure Chart Symbols

Structure charts are used to graphically model the hierarchy of processes within a system. Through the hierarchical format, the sequence of processes along with the movement of data and control parameters can be mapped for interpretation. The control structures of sequence, selection and repetition can all be represented within the chart for a modelled system.

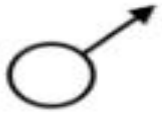


Process: Module / Subroutine
A series of instructions that are to be carried out by the program at a specific point.

Decision
Used to represent SELECTION and split the charts sequence into multiple paths.

Call Line
Indicates the path (SEQUENCE) between modules / subroutines.

Repetition
Used to represent REPETITION and highlight that a process can occur multiple times.

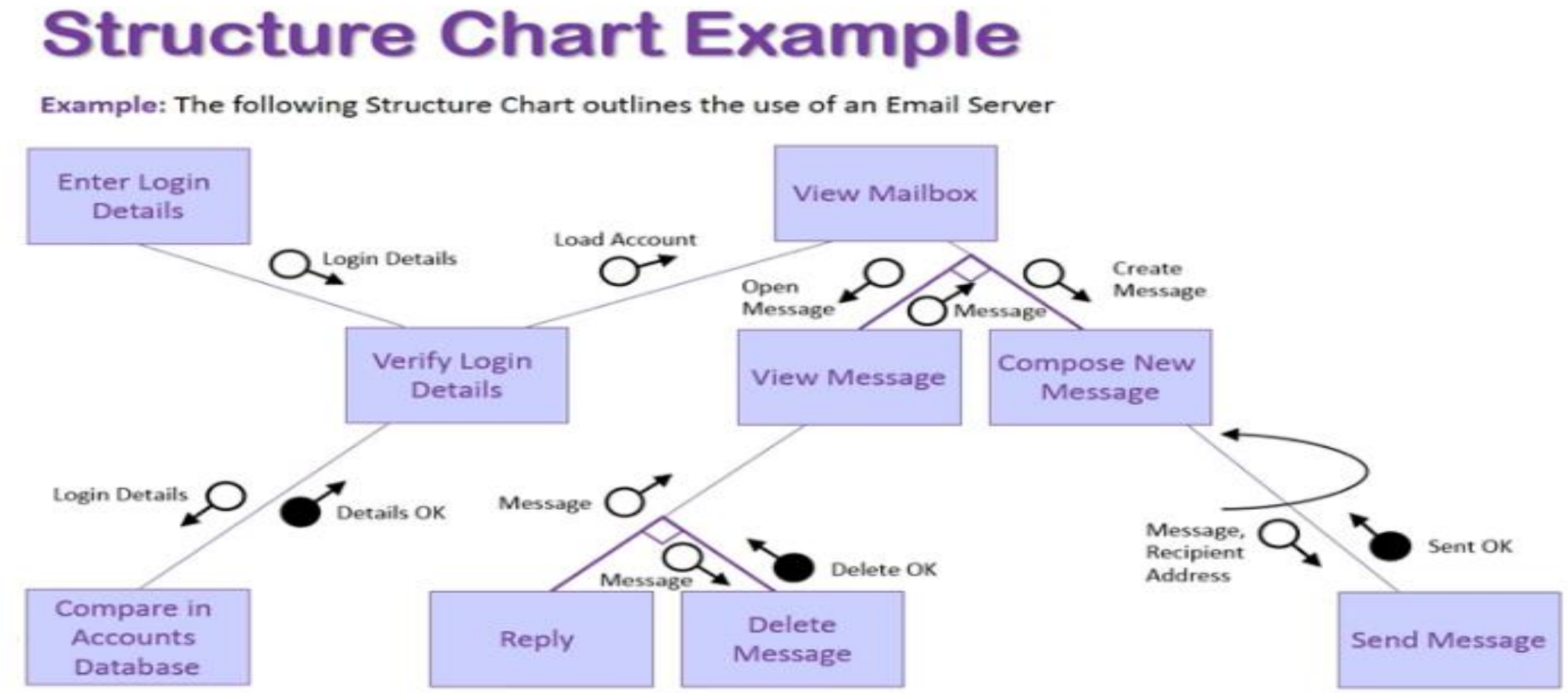


Parameter
Indicates the flow of DATA between processes, which is labelled with the symbol

Control Parameter
Indicate that a criteria has been met, providing confirmation for the system to proceed. E.g. Flags.

12.2 Program Design

12.2.1 - The purpose of a structure chart

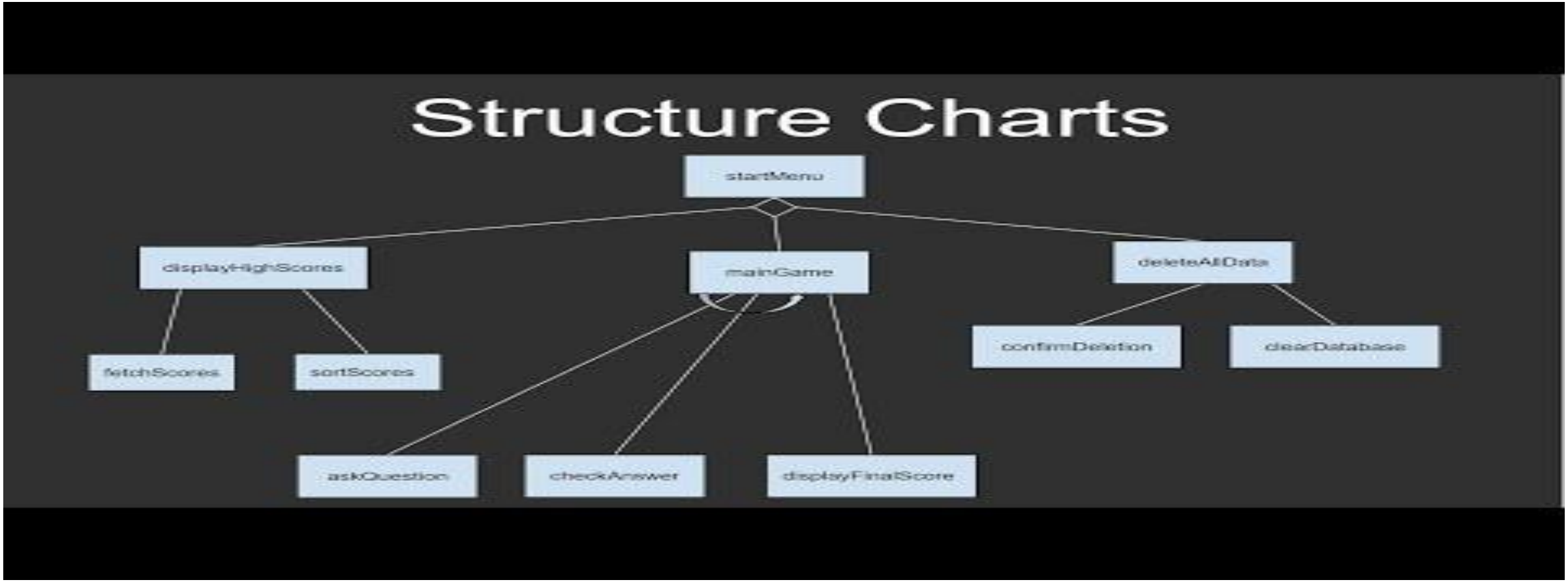


12.2 Program Design

12.2.1 - The purpose of a structure chart

Click the link below to watch the video for Structure Charts: 14 minutes

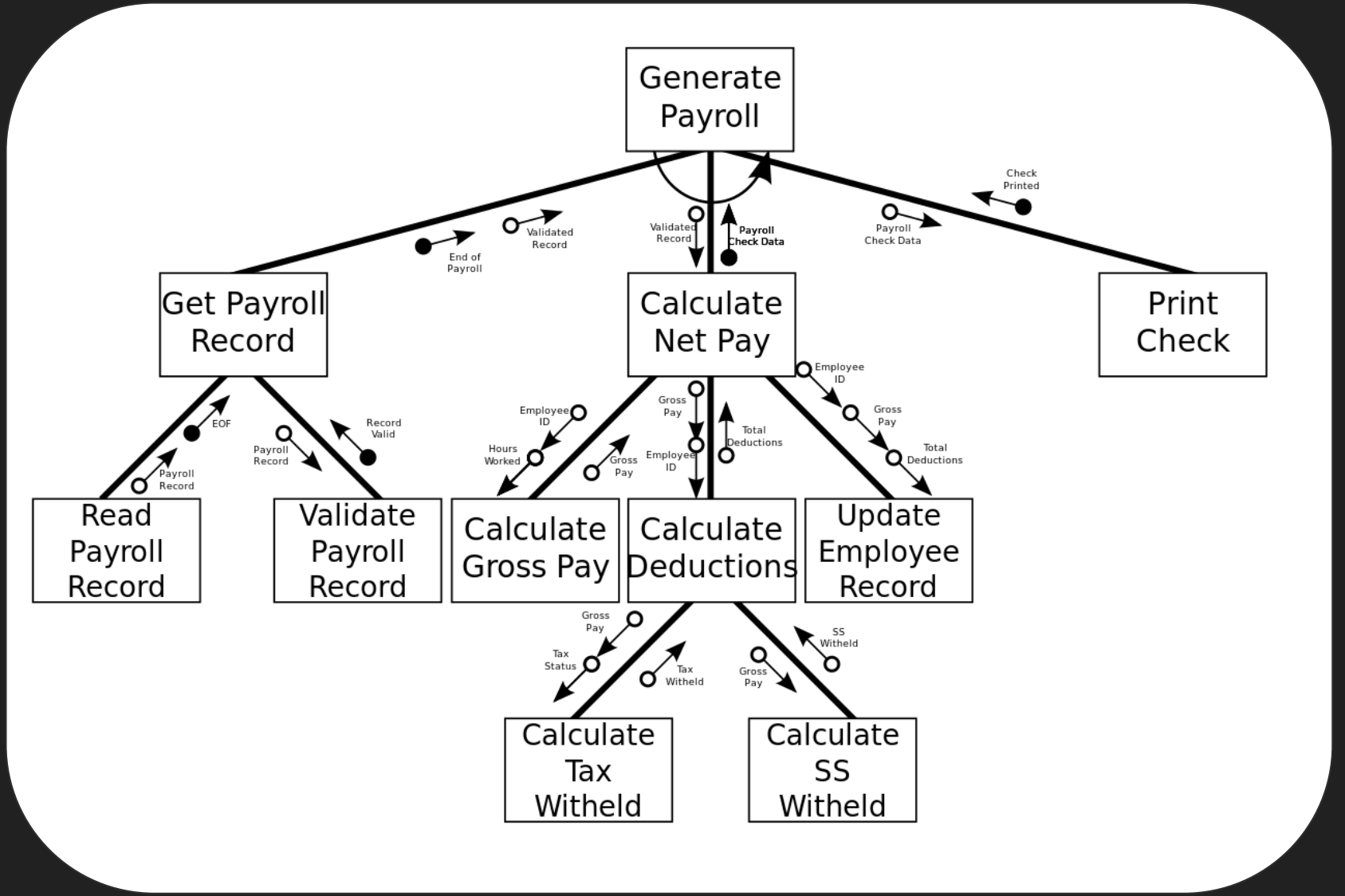
https://www.youtube.com/watch?v=PGLMqdZorDI



Structure Charts

Structure Chart

- Top down inverted tree representation of a system
- Uses functional decomposition to breaks a large program down into programmable components.
- Sequencing, Selection and Repetition can all be mapped to the diagram.
- Data flow and control flow between modules is also included.



https://en.wikipedia.org/wiki/Structure_chart

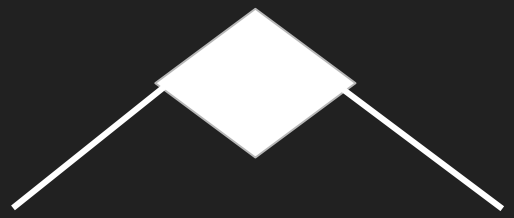
Structure Chart Symbols

Process / Module / Subroutine

Module - Indicates the module/subroutine name

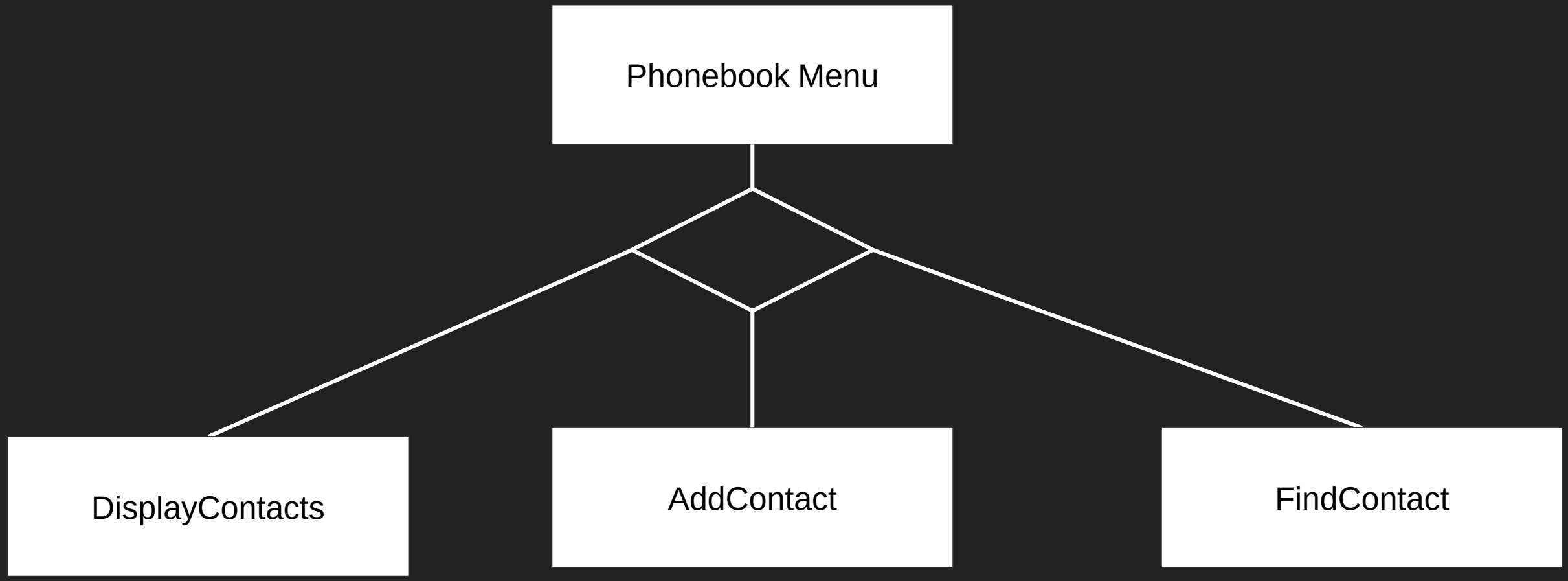
Call line - Shows the program flow path between modules (how they connect)

Repetition - Indicates a process may be repeated

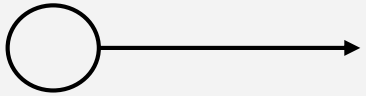


Selection - Shows where program may branch off onto different call lines

Decision Example



Data Flow

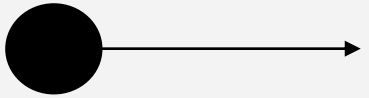


Data Flow

Used to indicate where data is being passed between modules. This can be in the form of :

- Parameters passed to modules/sub-programs
- Return values

Control Flow

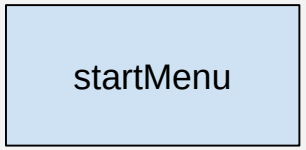


Control Flow

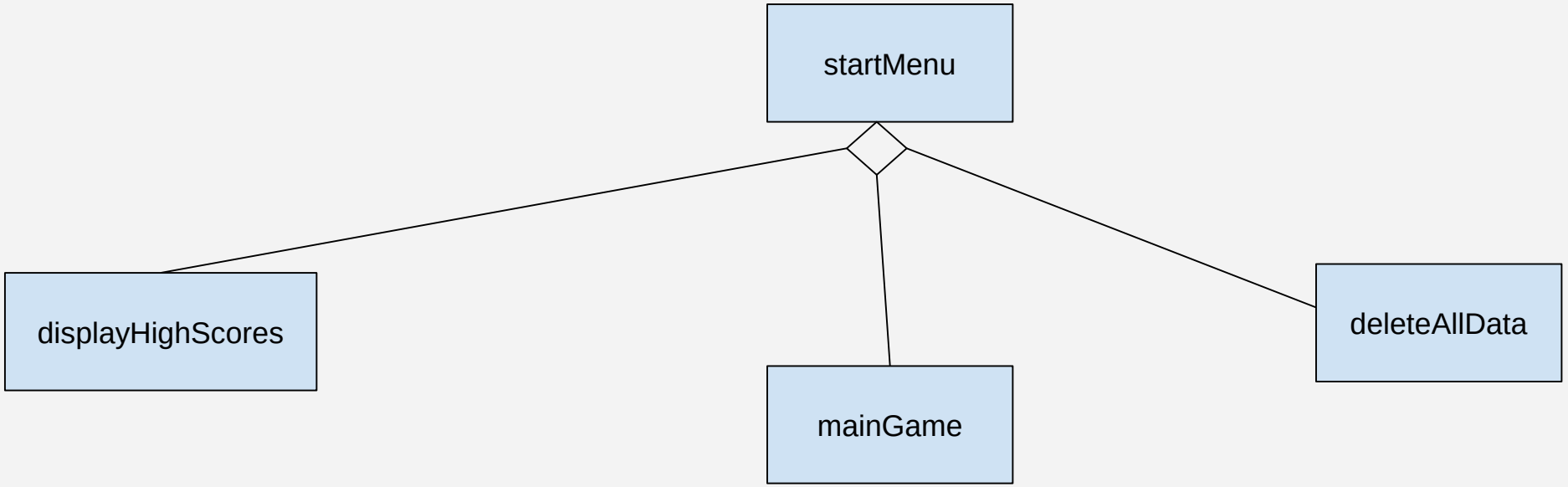
Used to indicate where the data passed is used to control the flow of the program.

- Usually this is some kind of True/False flag passed as a return value

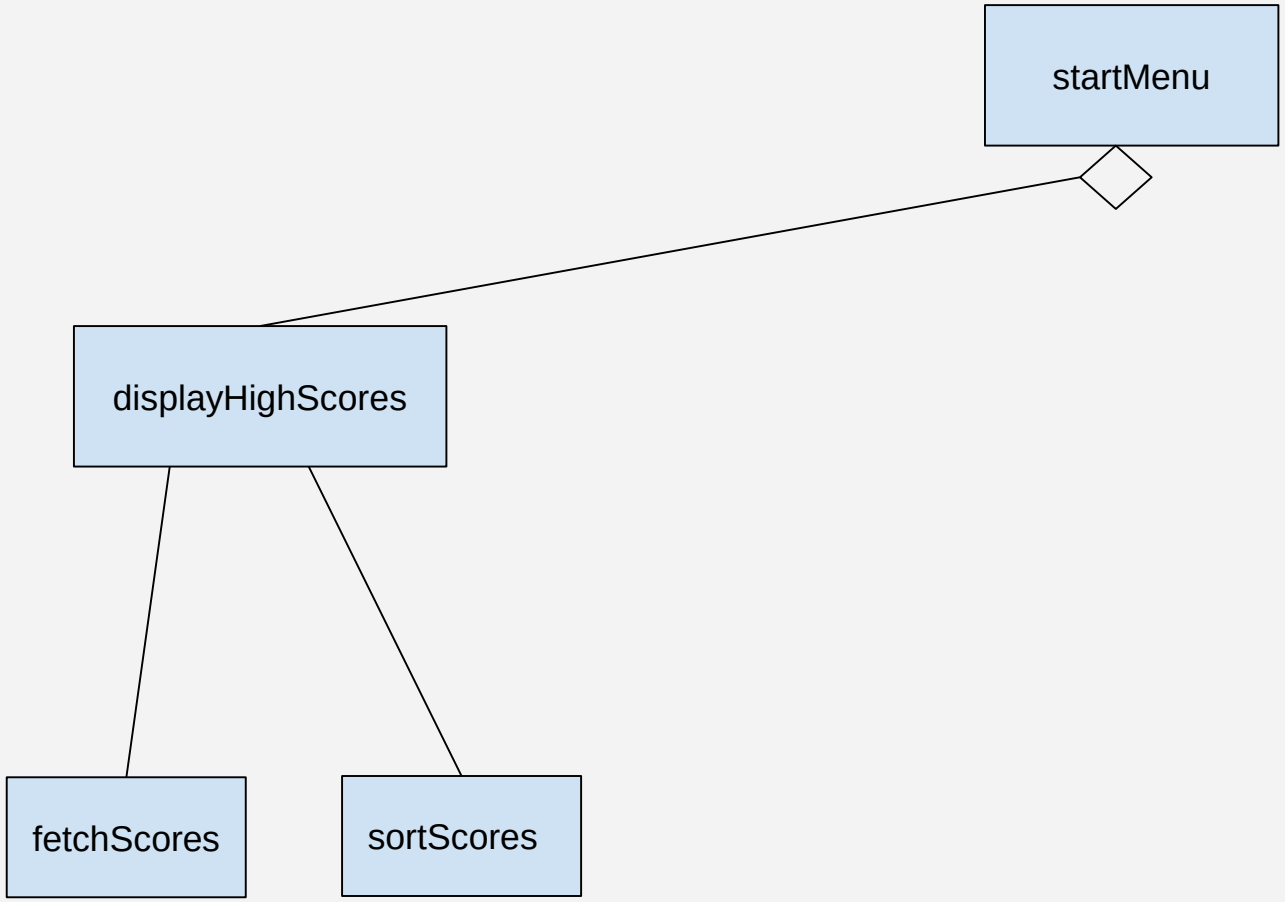
Structure Chart Example - Trivia Quiz



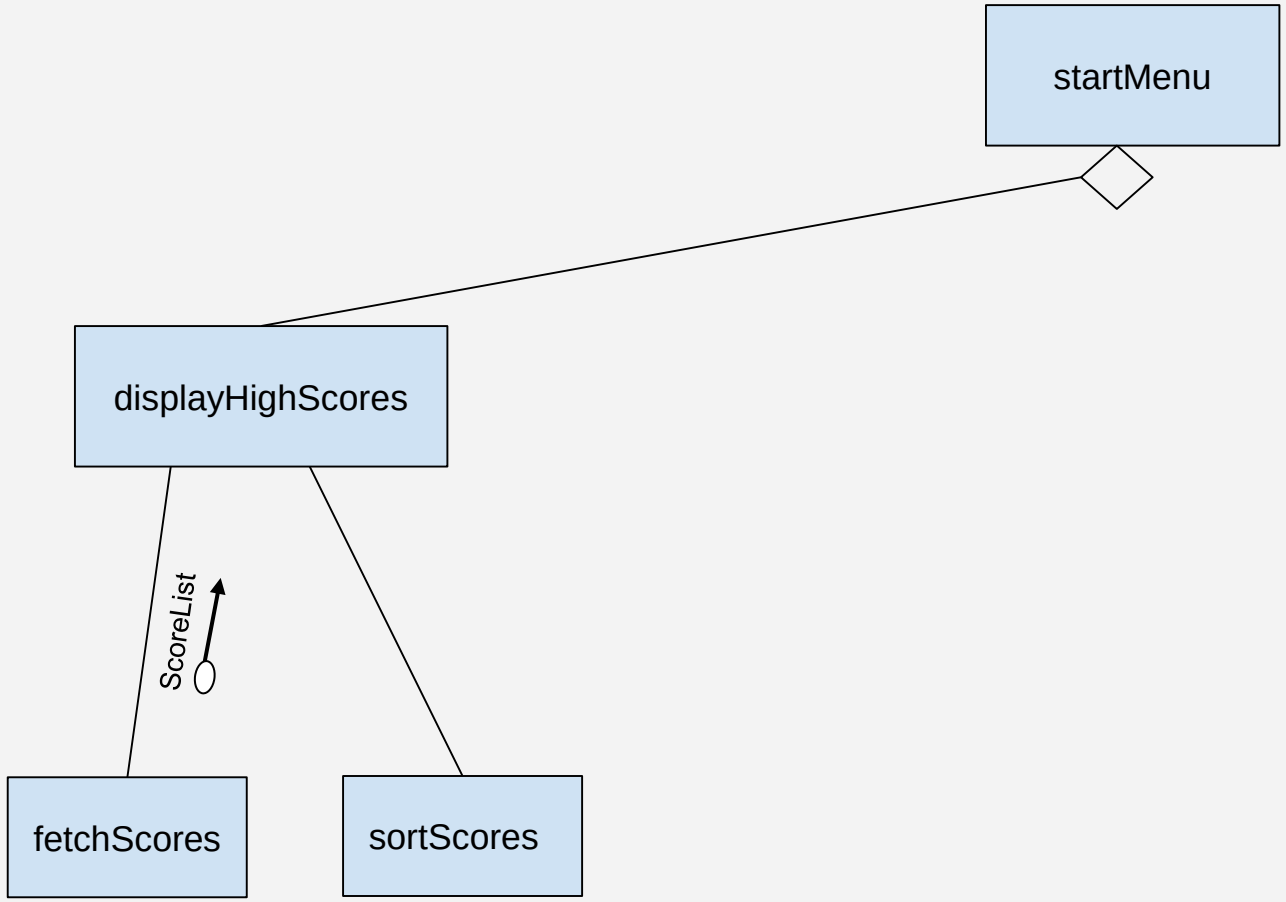
Structure Chart Example – Example : Trivia Quiz



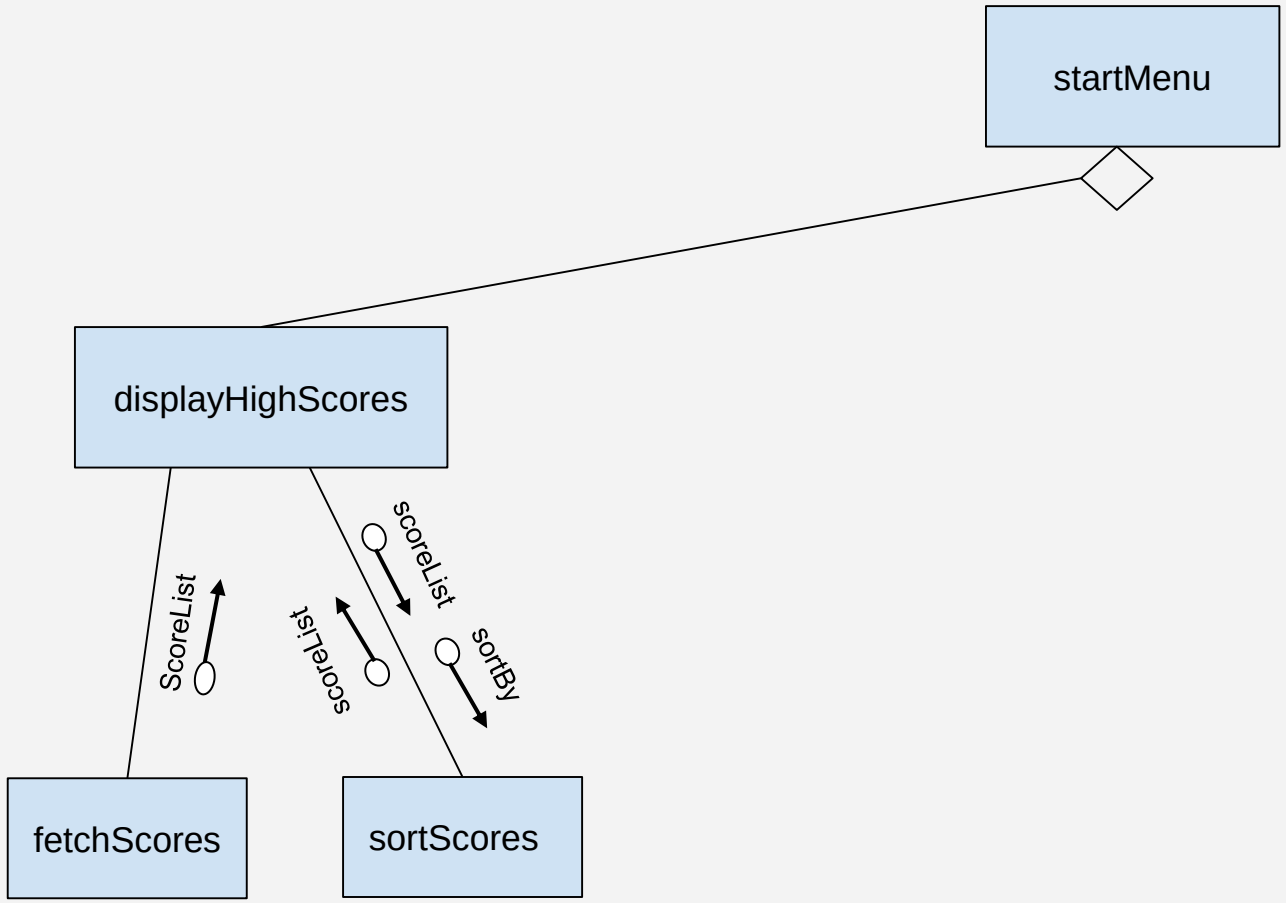
Structure Chart Example - Trivia Quiz



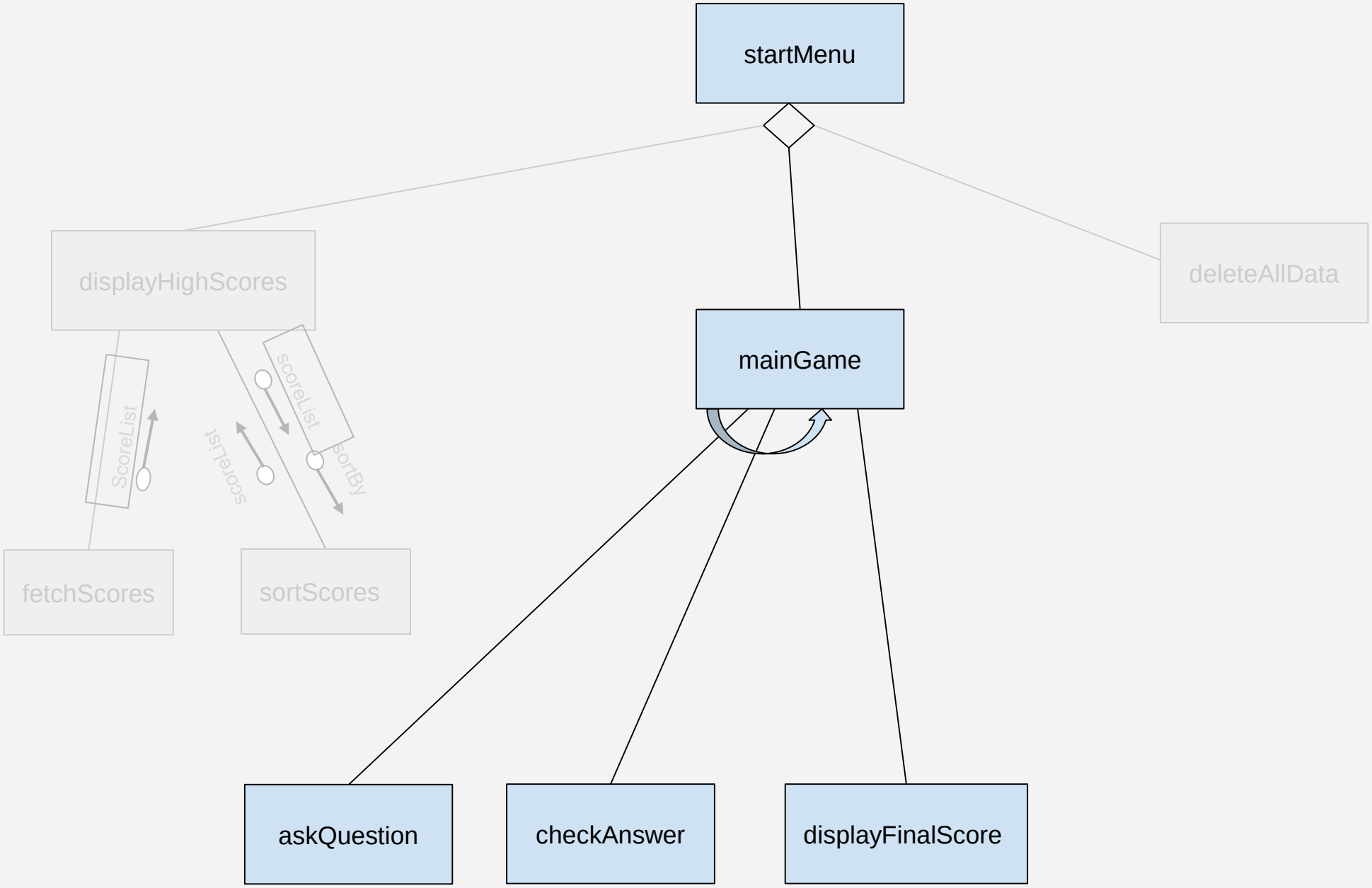
Structure Chart Example - Trivia Quiz



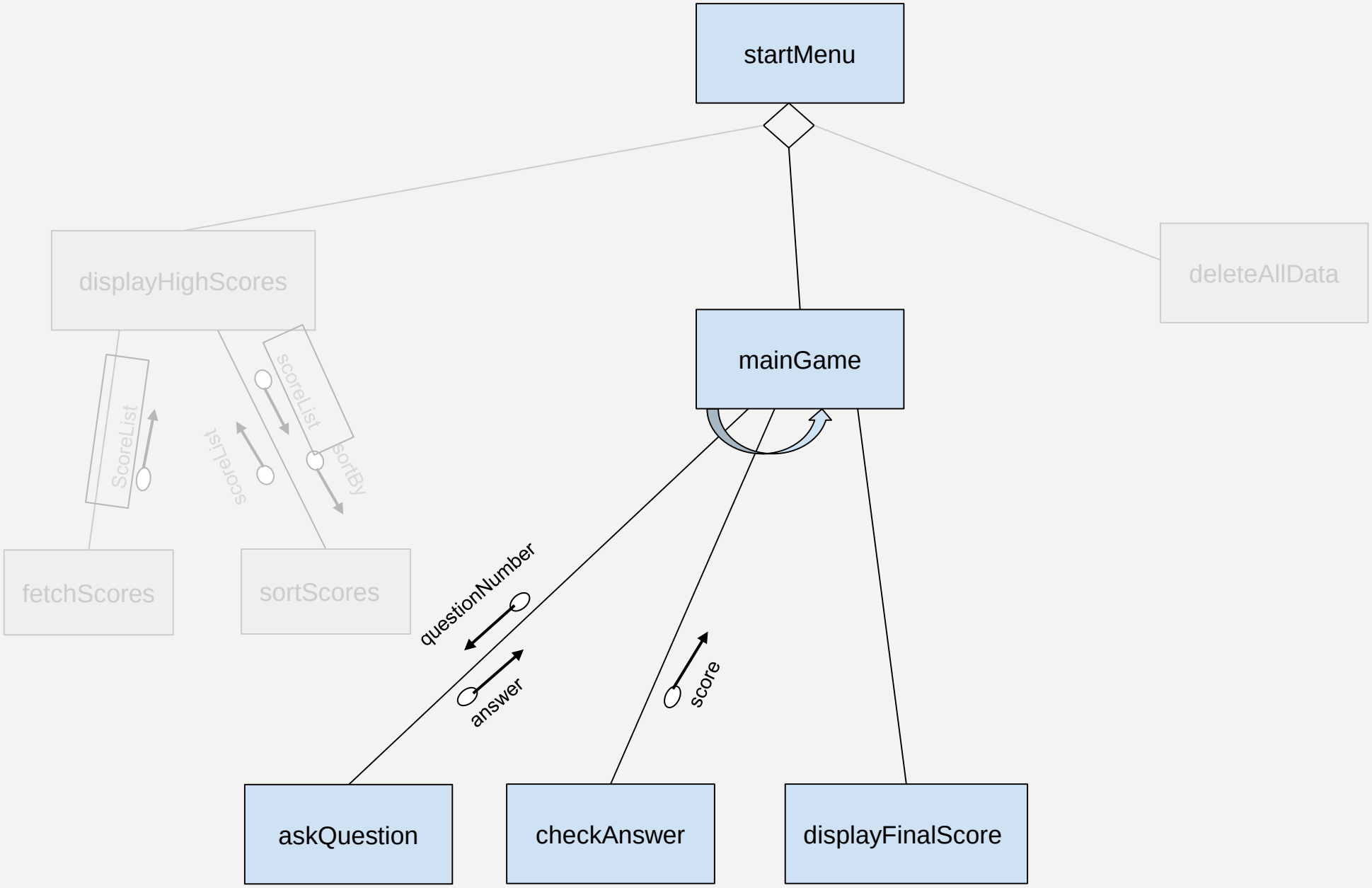
Structure Chart Example - Trivia Quiz



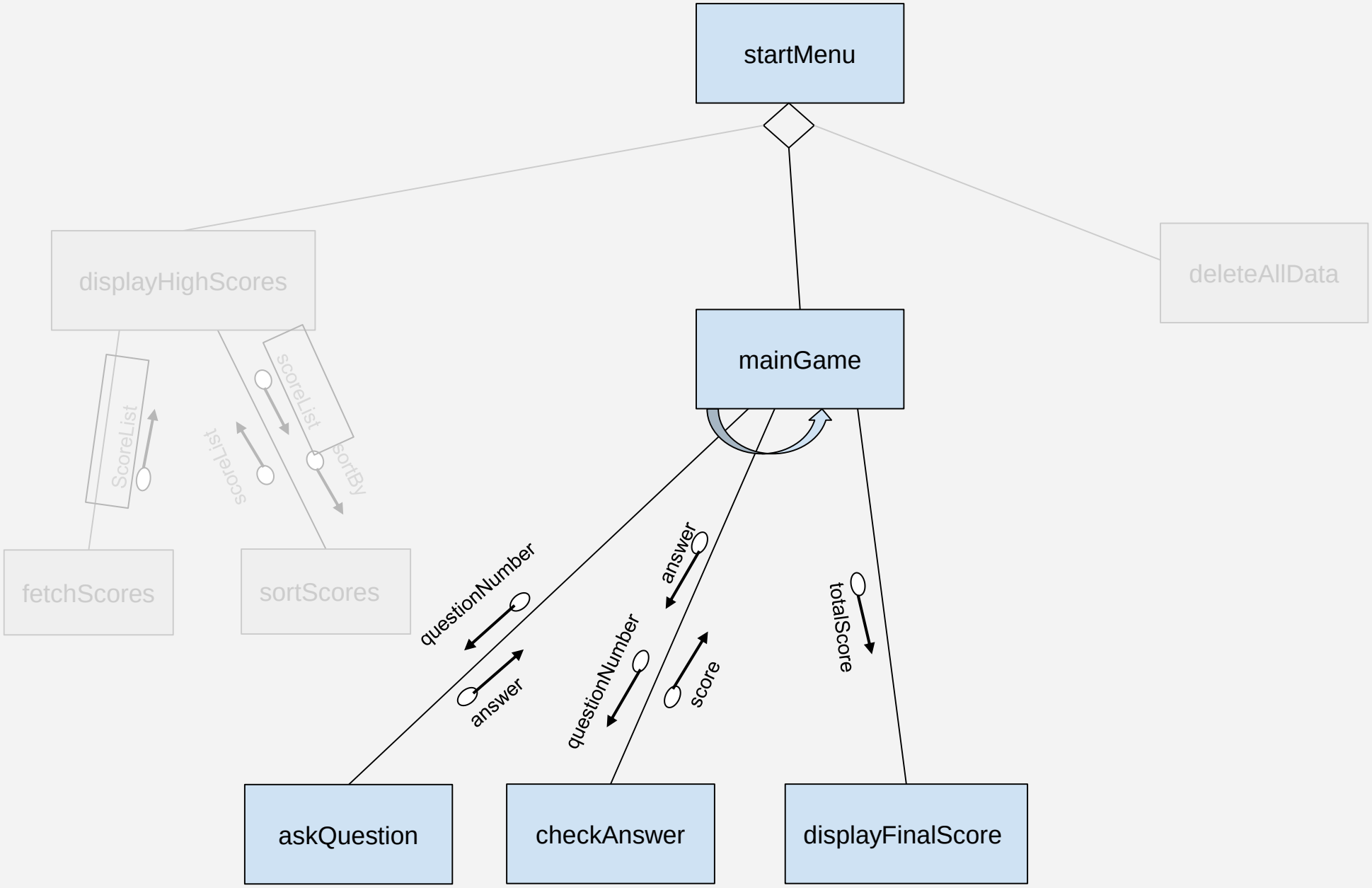
Structure Chart Example - Trivia Quiz



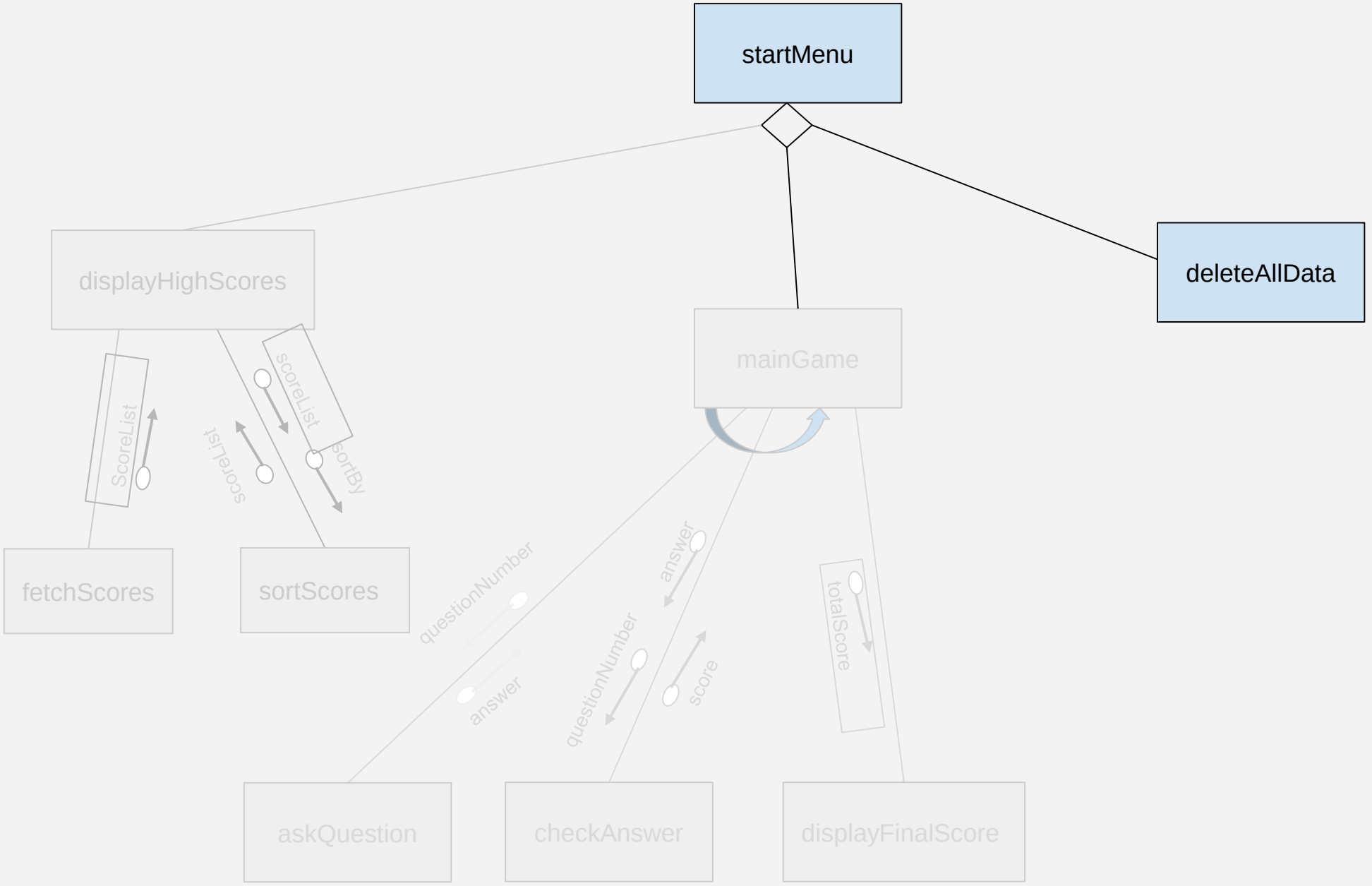
Structure Chart Example - Trivia Quiz



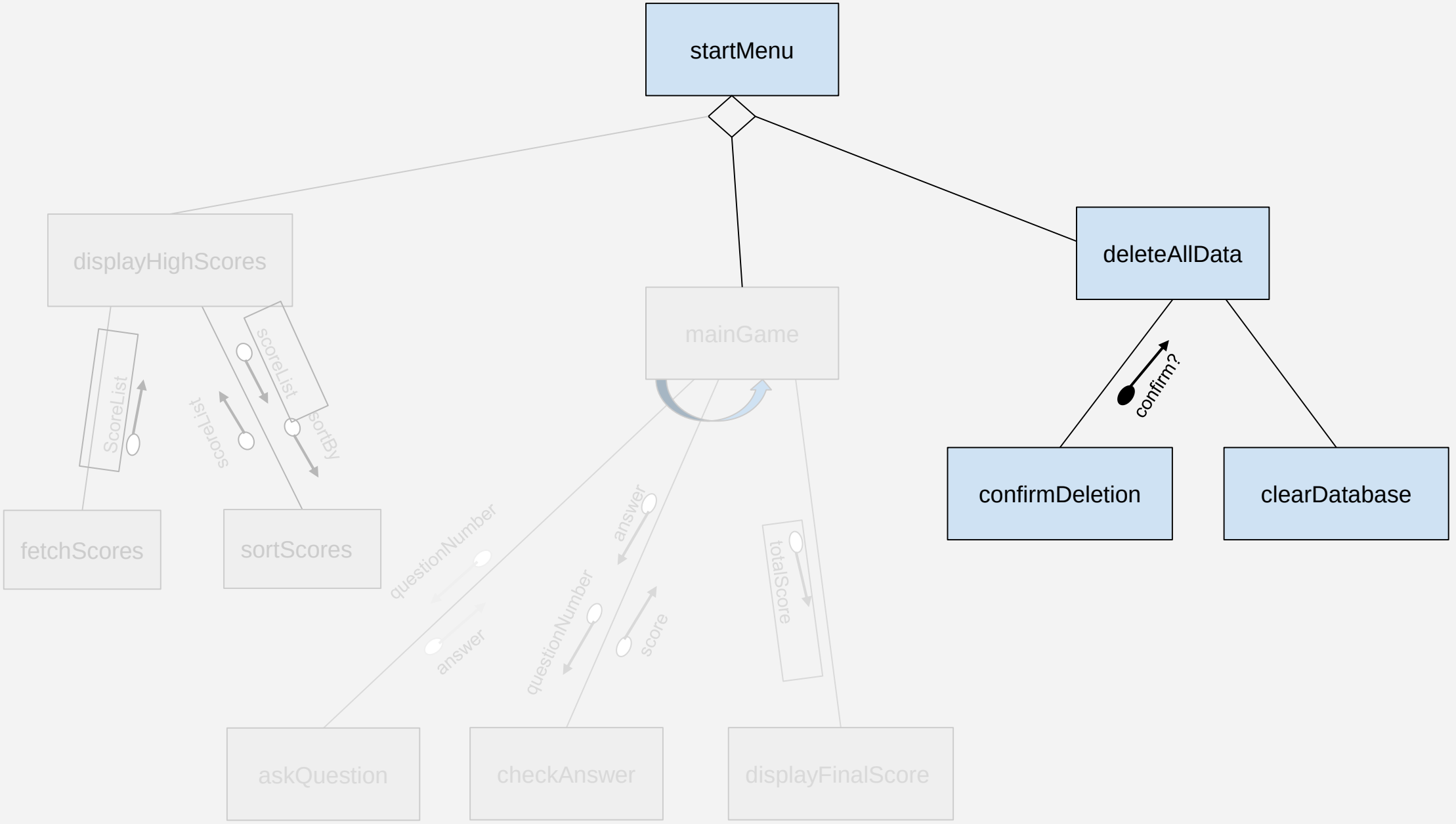
Structure Chart Example - Trivia Quiz



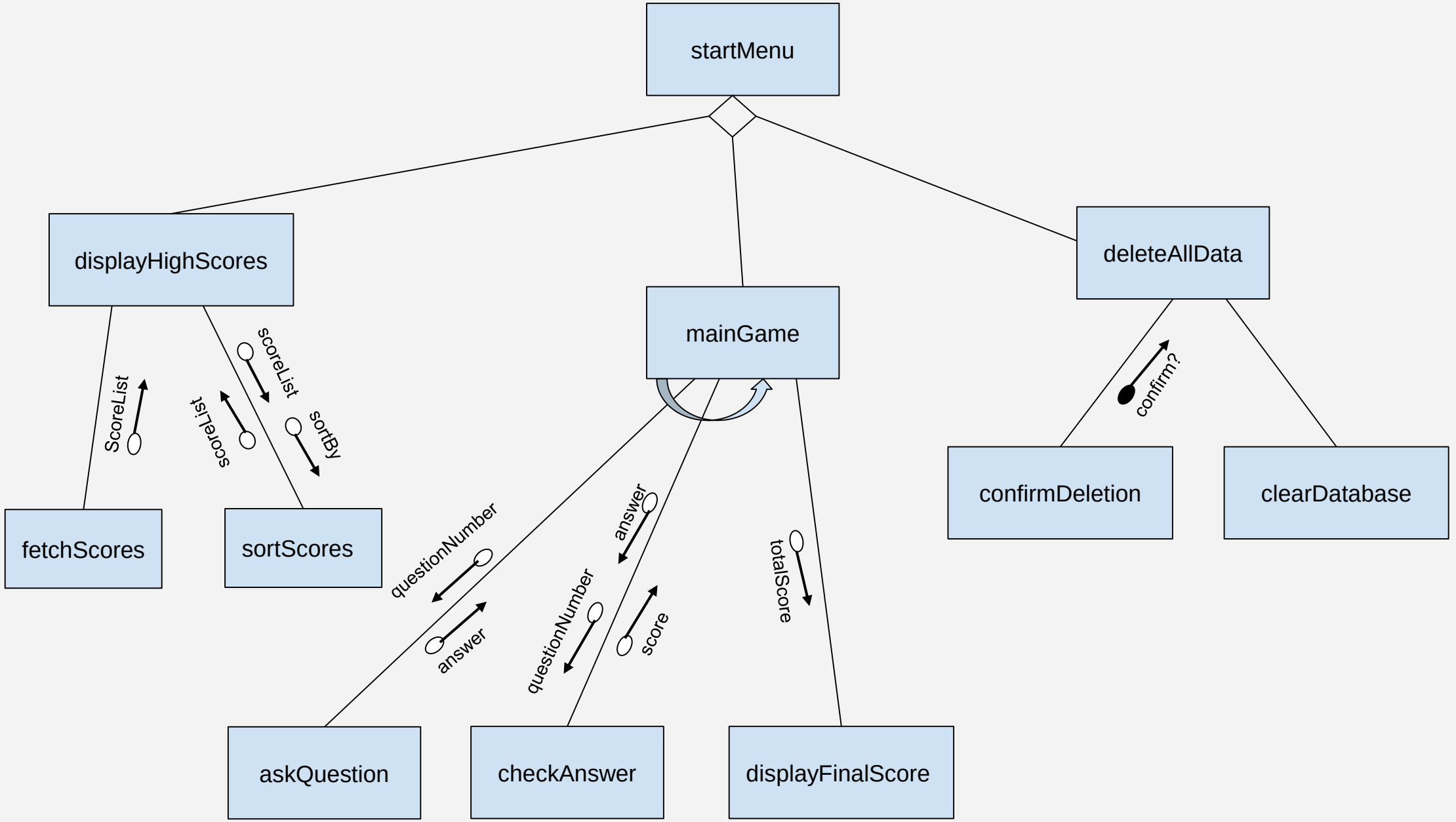
Structure Chart Example - Trivia Quiz



Structure Chart Example - Trivia Quiz



Structure Chart Example - Trivia Quiz



12.2 Program Design

12.2.1 - The purpose of a structure chart

A structure chart is a modelling tool used in program design to decompose a problem into a set of sub-tasks. The structure chart shows the hierarchy or structure of the different modules and how they connect and interact with each other. Each module is represented by a box and the parameters passed to and from the modules are shown by arrows pointing towards the module receiving the parameter. Each level of the structure chart is a refinement of the level above.

Figure 12.2.1 shows a structure chart for converting a temperature from Fahrenheit to Celsius. The top level shows the name for the whole task that is refined into three sub-tasks or modules shown on the next level.

12.2 Program Design

12.2.1 - The purpose of a structure chart

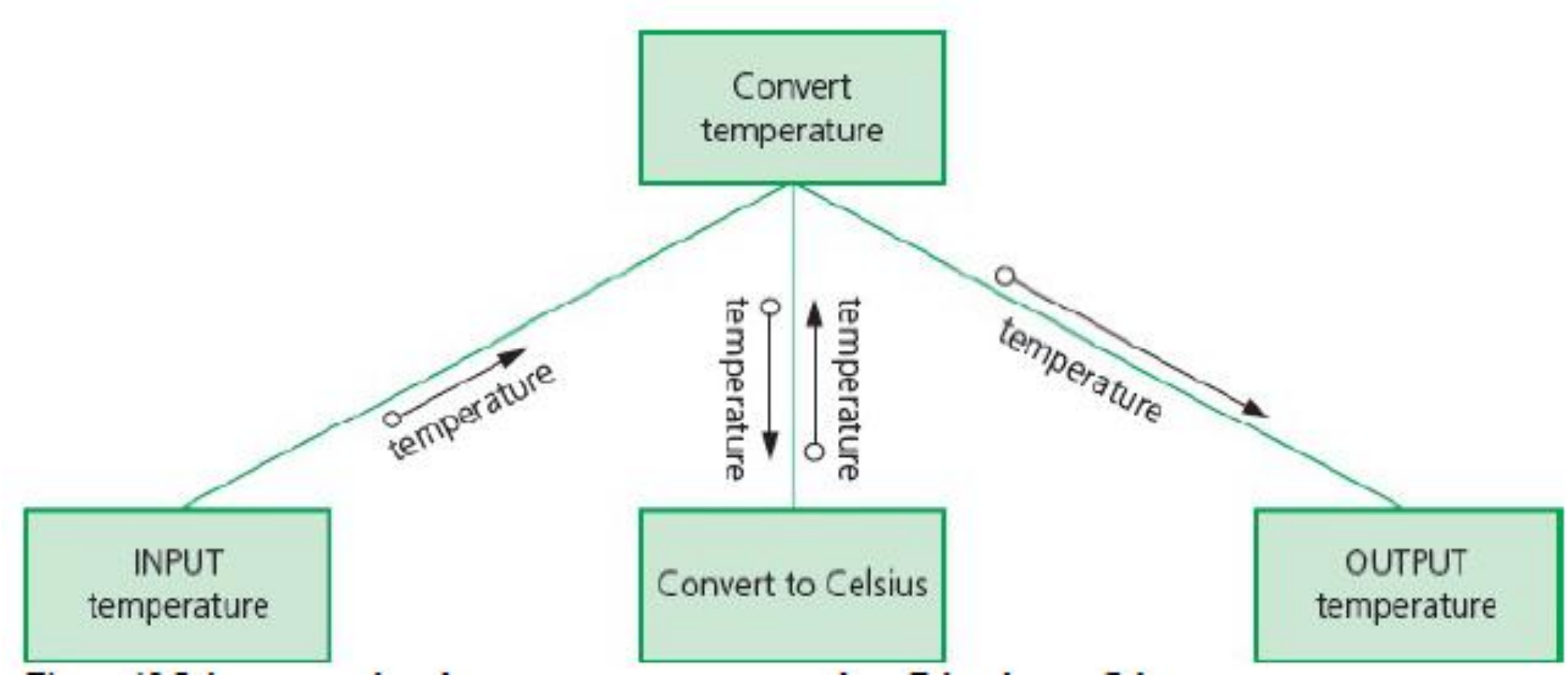


Figure 12.2.1 A structure chart for converting a temperature from Fahrenheit to Celsius

Figure by: Hodder Computer Science Books

12.2 Program Design

12.2.1 - The purpose of a structure chart

Structure charts can also show selection. The temperature conversion task above could be extended to either convert from Fahrenheit to Celsius or Celsius to Fahrenheit using the diamond shaped box to show a condition that could be true or false, as shown in Figure 12.2.2.

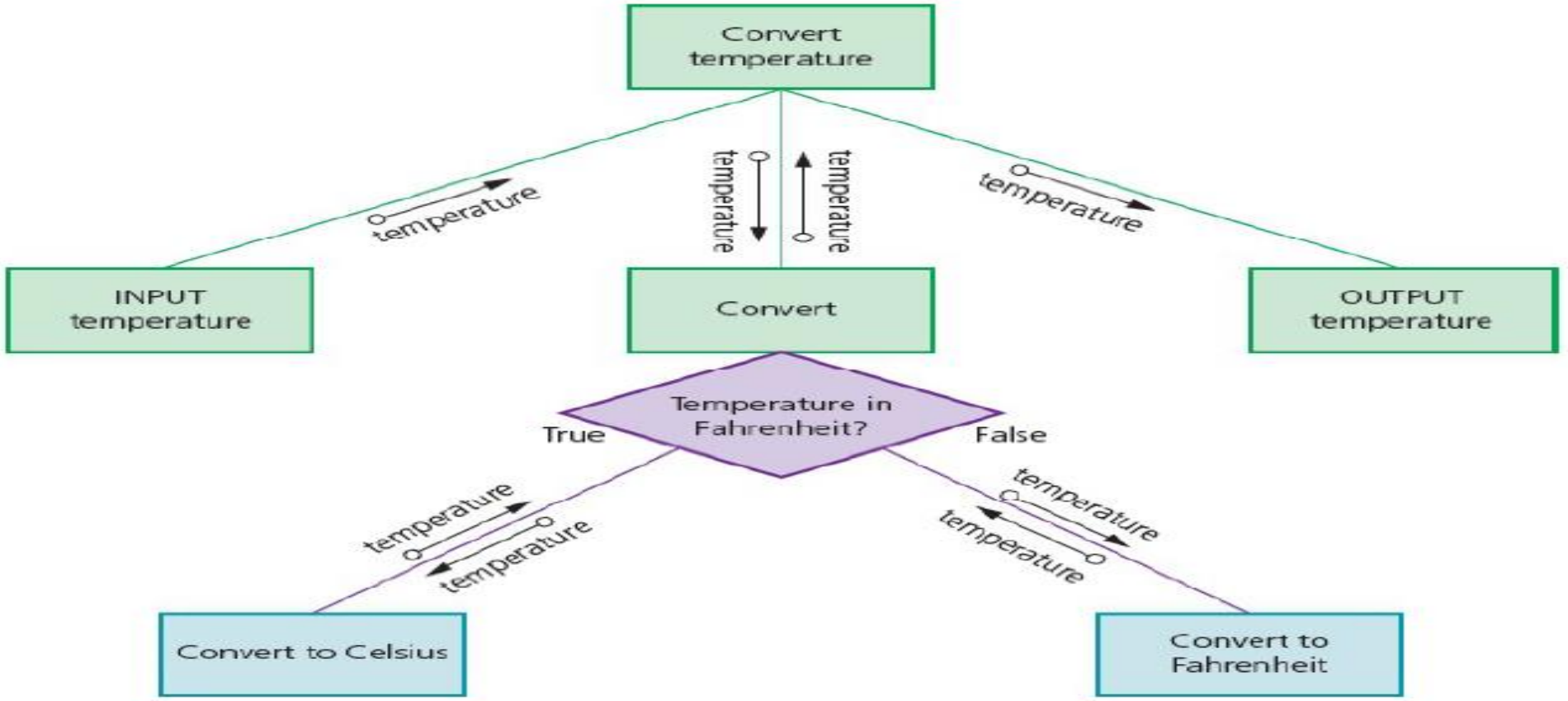


Figure 12.2.2 by: Hodder Computer Science Books

12.2 Program Design

12.2.1 - The purpose of a structure chart

Structure charts can also show repetition. The temperature conversion task above could be extended to repeat the conversion until the number 999 is input. The repetition is shown by adding a labelled semi-circular arrow above the modules to be completed (Figure 12.2.3).

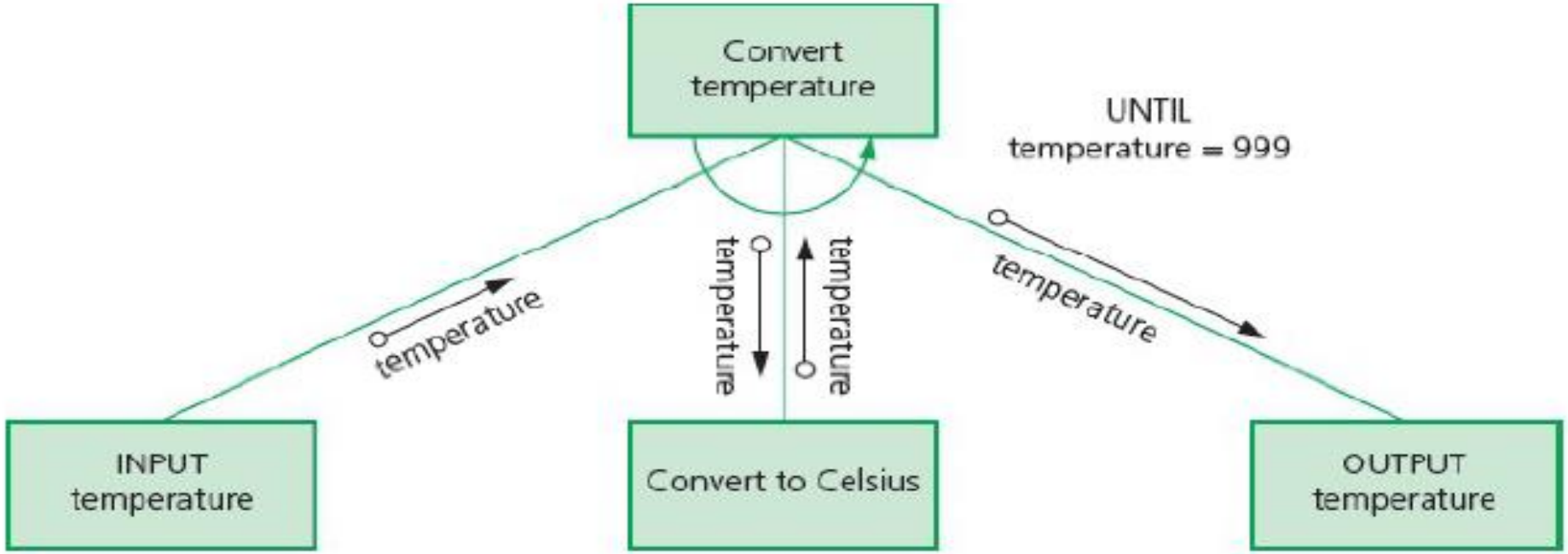


Figure 12.2.3 by: Hodder Computer Science Books

Once a structure chart has been completed, it can be used to derive a pseudocode algorithm.

12.2 Program Design

12.2.1 - The purpose of a structure chart

ACTIVITY 1: Amend your structure chart to input the radius of a sphere, calculate and output either the volume or the surface area. The algorithm should repeat until a radius of zero is entered.

Solution: Figure 12.2.4 shows a possible structure chart for Activity 1.

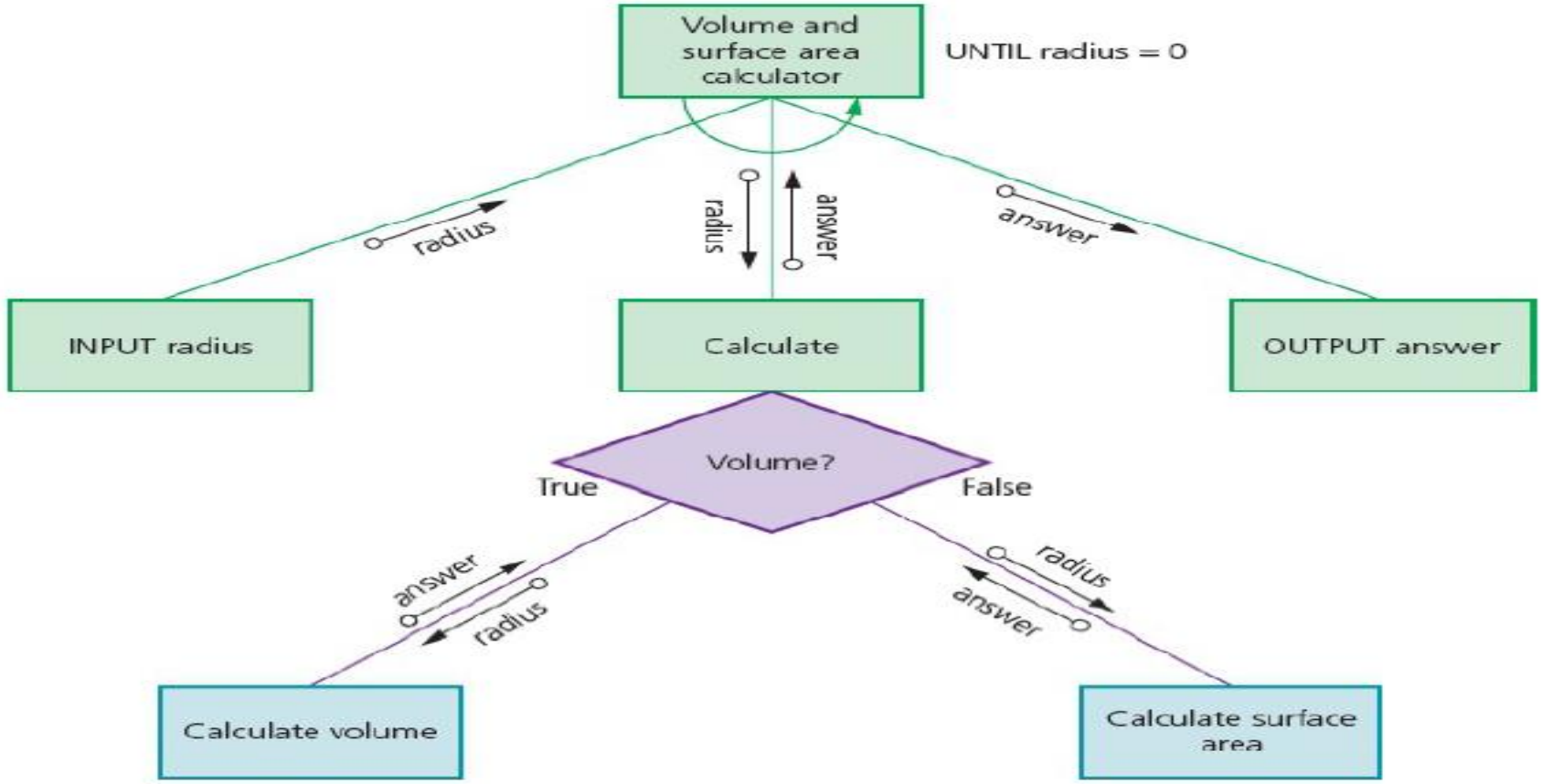


Figure 12.2.4 by: Hodder Computer Science Books

12.2 Program Design

12.2.1 - The purpose of a structure chart

Recap: Difference between Function and Procedure.

Definition:

- A function is used to **calculate result** using given **inputs**.

Example: `FUNCTION calculateVolume (radius:real)` **or** **Shop()**

- A procedure is used to perform certain task in **order**.

Example: `PROCEDURE inputRadius`

Call:

- A function **can be called** by a procedure.
- A procedure **cannot be called** by a function.

12.2 Program Design

12.2.1 - The purpose of a structure chart

To derive the pseudo code first, you will need to create an identifier table.

Identifier name	Description
radius	Stores radius input
answer	Stores result of calculation
pi	Constant set to 3.142

Then declare constants and variables in pseudocode. You can identify two of the variables required from the parameters shown in the structure diagram.

DECLARE radius : REAL

DECLARE answer : REAL

CONSTANT pi ← 3.142

12.2 Program Design

12.2.1 - The purpose of a structure chart

Provide pseudocode for the modules shown in the structure diagram. As Calculate volume and Calculate surface area provide the answer to a calculation, these can be defined as functions.

```
FUNCTION calculateVolume (radius:real) RETURNS real
    RETURN (4 / 3) * pi * radius * radius * radius
ENDFUNCTION
FUNCTION calculateSurfaceArea (radius:real) RETURNS real
    RETURN 4 * pi * radius * radius
ENDFUNCTION
```

12.2 Program Design

12.2.1 - The purpose of a structure chart

The input and output modules could be defined as procedures.

```
PROCEDURE inputRadius
    OUTPUT "Please enter the radius of the sphere "
    INPUT radius
    WHILE radius < 0 DO
        OUTPUT "Please enter a positive number "
        INPUT radius
    ENDWHILE
ENDPROCEDURE
PROCEDURE outputAnswer
    OUTPUT answer
ENDPROCEDURE
```

12.2 Program Design

12.2.1 - The purpose of a structure chart

The pseudocode for the whole algorithm, including the selection and repetition, would be as follows.

```
DECLARE radius : REAL
DECLARE answer : REAL
CONSTANT pi ← 3.142
FUNCTION calculateVolume (radius:real) RETURNS real
    RETURN (4 / 3) * pi * radius * radius * radius
ENDFUNCTION
FUNCTION calculateSurfaceArea (radius:real) RETURNS real
    RETURN 4 * pi * radius * radius
ENDFUNCTION
PROCEDURE inputRadius
    OUTPUT "Please enter the radius of the sphere "
    INPUT radius
    WHILE radius < 0 DO
        OUTPUT "Please enter a positive number "
        INPUT radius
    ENDWHILE
```

12.2 Program Design

12.2.1 - The purpose of a structure chart

```
PROCEDURE outputAnswer
    OUTPUT answer
ENDPROCEDURE
CALL inputRadius
WHILE radius <> 0
    OUTPUT "Do you want to calculate the Volume (V) or Surface Area (S)"
    INPUT reply
    IF reply = "V"
      THEN
        answer ← calculateVolume(radius)
        OUTPUT "Volume "
      ELSE
        answer ← calculateSurfaceArea(radius)
        OUTPUT "Surface Area "
    ENDIF
    CALL outputAnswer
    CALL inputRadius
ENDWHILE
```

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 1-(a)

(a) The following table contains information about five modules in a program. It describes the calls made and the parameters passed.

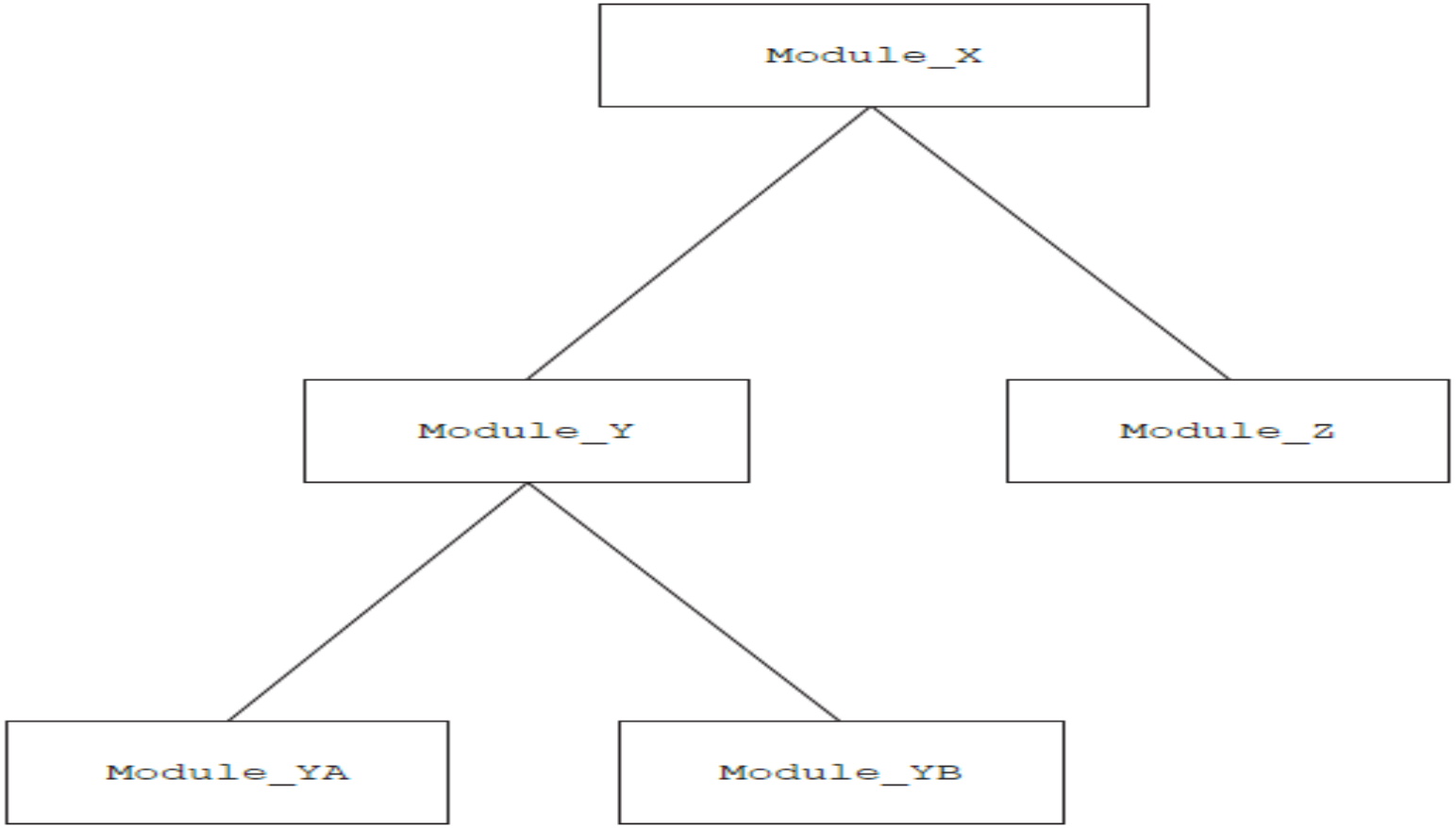
Module	Description
`Module_X`	• repeatedly calls `Module_Y` then `Module_Z` • passes a parameter of type `REAL` to `Module_Y` • passes two parameters of type `INTEGER` to `Module_Z`
`Module_Y`	calls either `Module_YA` or `Module_YB`
`Module_Z`	called with two parameters of type `INTEGER`
`Module_YA`	• called with a parameter of type `REAL` • parameter is passed by reference
`Module_YB`	• called with a parameter of type `INTEGER` • returns a `BOOLEAN` value

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 1-(a)

Complete the structure chart to include the information given about the five modules.

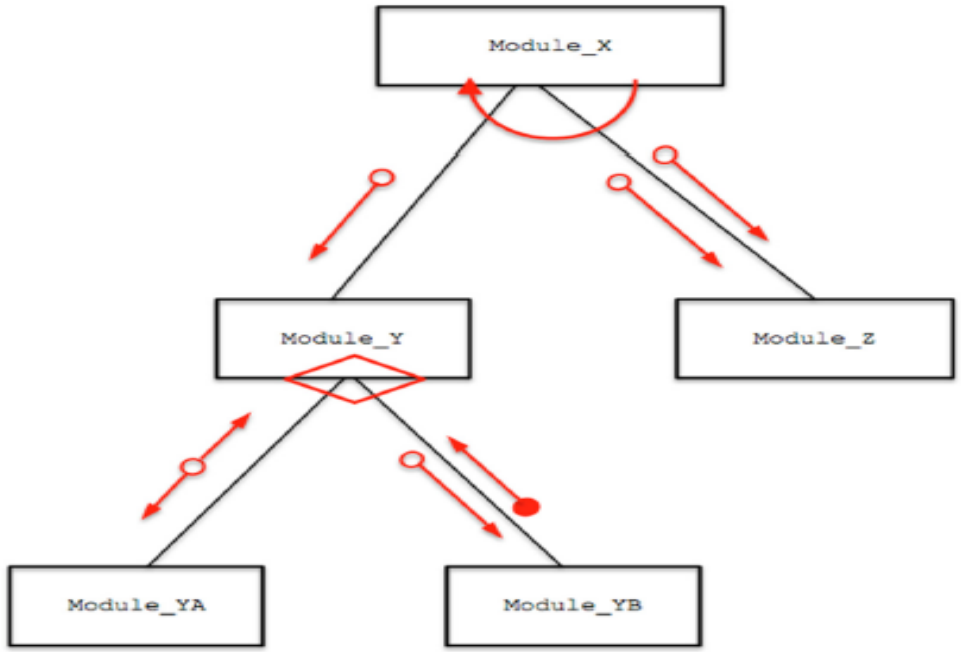


[5]

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 1- (a)

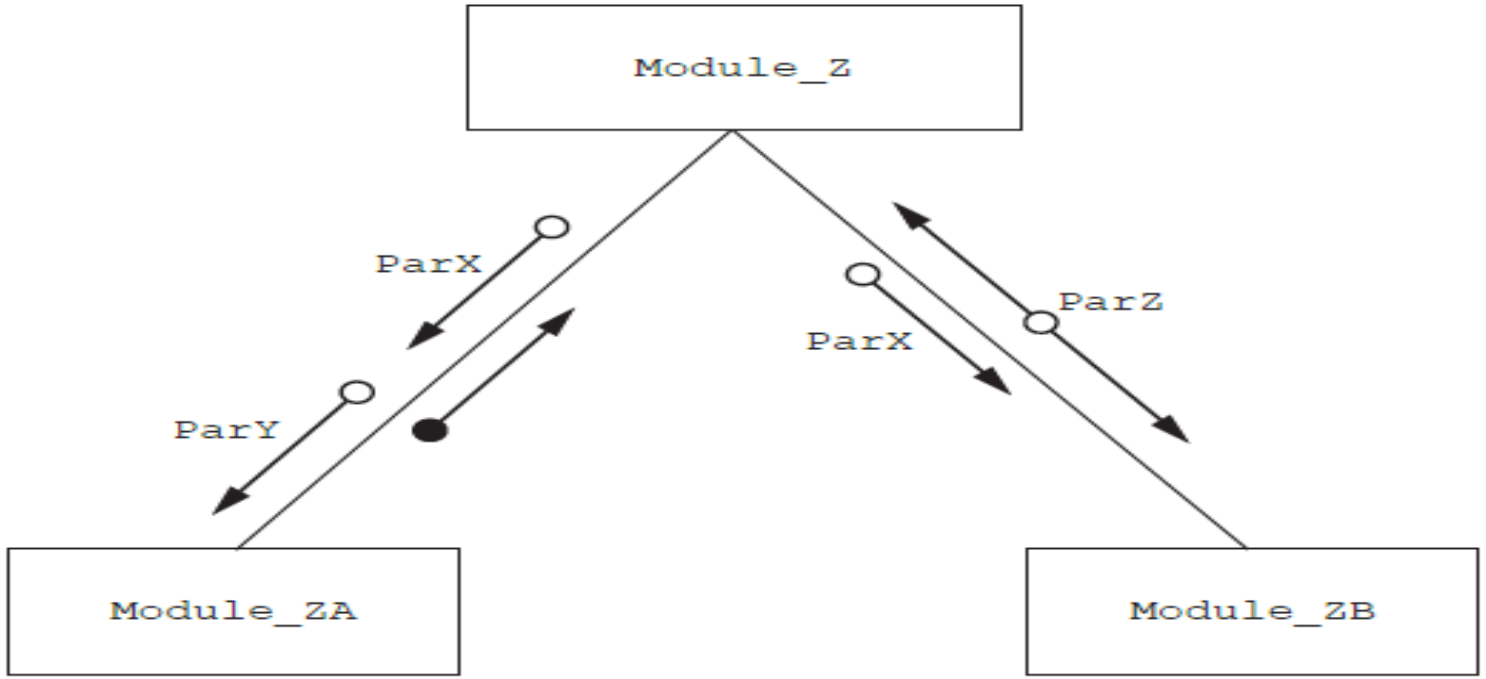
Question	Answer	Marks
3(a)	 1 mark for each of: 1 Iteration arrow 2 Selection diamond 3 Both sets of parameters from Module_X 4 Parameter ByReferene to Module_YA 5 Parameter (ByValue) **and** return Boolean from Module YB	5

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 1-(b)

(b) Two more modules are added to the chart below `Module_Z` as shown:



Parameter data types are:

```
ParX   : REAL
ParY   : INTEGER
ParZ   : STRING
```

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 1-(b)

(i) State whether `Module_ZA()` is a function or a procedure **and** justify your choice.

.....

.....

..... [2]

(ii) Write the **pseudocode** header for `Module_ZB()`.

.....

.....

..... [3]

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 1-(b)

(i) State whether `Module_ZA()` is a function or a procedure **and** justify your choice.

One mark for each statement:

- it is a function
- because it returns a value

[2]

(ii) Write the **pseudocode** header for `Module_ZB()`.

```
PROCEDURE Module_ZB (BYVALUE ParX : REAL, BYREF ParZ : STRING)
```

One mark for:

- Procedure declaration
- `ParX : REAL` and `ParZ : STRING`
- `ByRef` for `ParZ`

[3]

Condone missing `BYVALUE` for `ParX`

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 2-(a)

(a) A structure chart is often produced as part of a modular program design. The chart shows the hierarchy of modules and the sequence of execution.

Give **two other** features the structure chart can show.

Feature 1 .....

.....

Feature 2 .....

.....

[2]

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 2-(a)

One mark per bullet point: • Parameters passed between modules // the interface between modules • Module Iteration • Module selection	2

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 2-(b-i)

(b) Six program modules implement part of an online shopping program. The following table gives the modules and a brief description of each module:

Module	Description
`Shop()`	Allows the user to choose a delivery slot, select items to be added to the basket and finally check out
`ChooseSlot()`	Allows the user to select a delivery time. Returns a delivery slot number
`FillBasket()`	Allows the user to select items and add them to the basket
`Checkout()`	Completes the order by allowing the user to pay for the items. Returns a Boolean value to indicate whether or not payment was successful
`Search()`	Allows the user to search for a specific item. Returns an item reference
`Add()`	Adds an item to the basket. Takes an item reference and a quantity as parameters

(i) The online shopping program has been split into sub-tasks as part of the design process.

Explain the advantages of decomposing the program into modules. Your explanation should refer to the scenario and modules described in **part (b)**.

.....

.....

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 2-(b-i)

Advantages include:

- Easier to solve / implement / program the solution as online shopping is a complex task
- Easier to debug / maintain as each module can be tested separately e.g. test `FillBasket()` first then test `Checkout()`
- Tasks may be shared among a team of programmer. e.g. `Checkout()` and `Search()` modules could be developed in parallel / by teams with different expertise

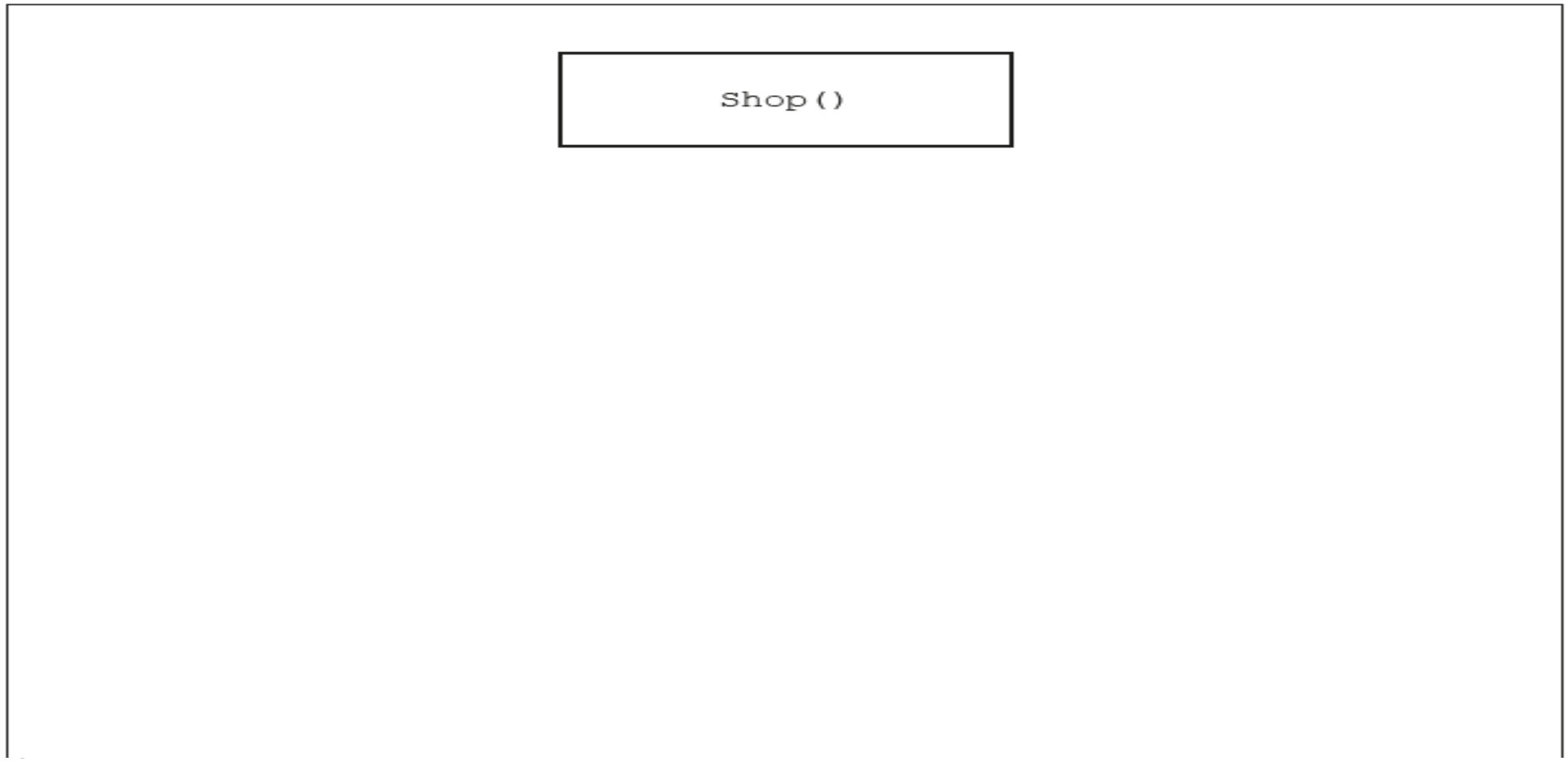
Note:
Must include reference to given scenario to achieve all 3 marks - Max 2 if no reference.

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 2-(b-ii)

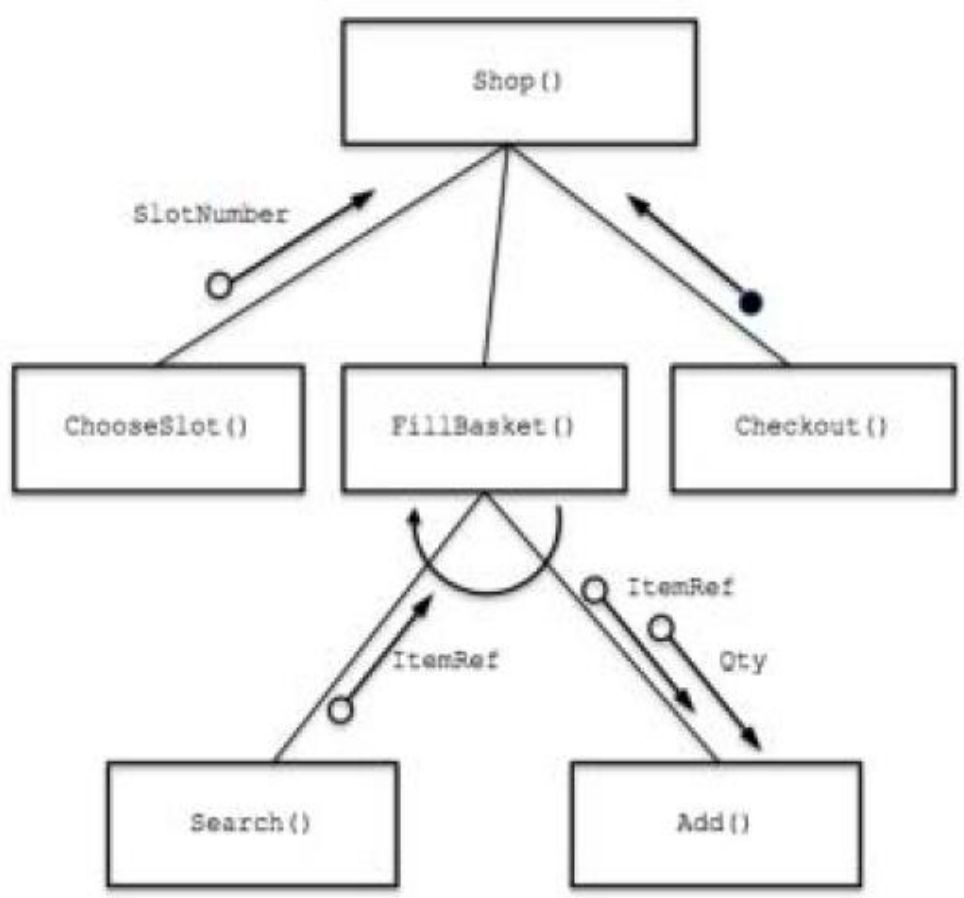
(ii) Complete the structure chart for the six modules described in **part (b)**.



12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 2-(b-ii)



6 One Mark for

1 Three middle row boxes correctly labelled and connected to `Shop()`

2 Two bottom row boxes correctly labelled and connected to `FillBasket()`

3 Iteration arrow on `FillBasket()`

4 Return parameters from `ChooseSlot()` **and** `Checkout()`

5 Return parameters from `Search()`

6 Two input parameters to `Add()`

Notes:

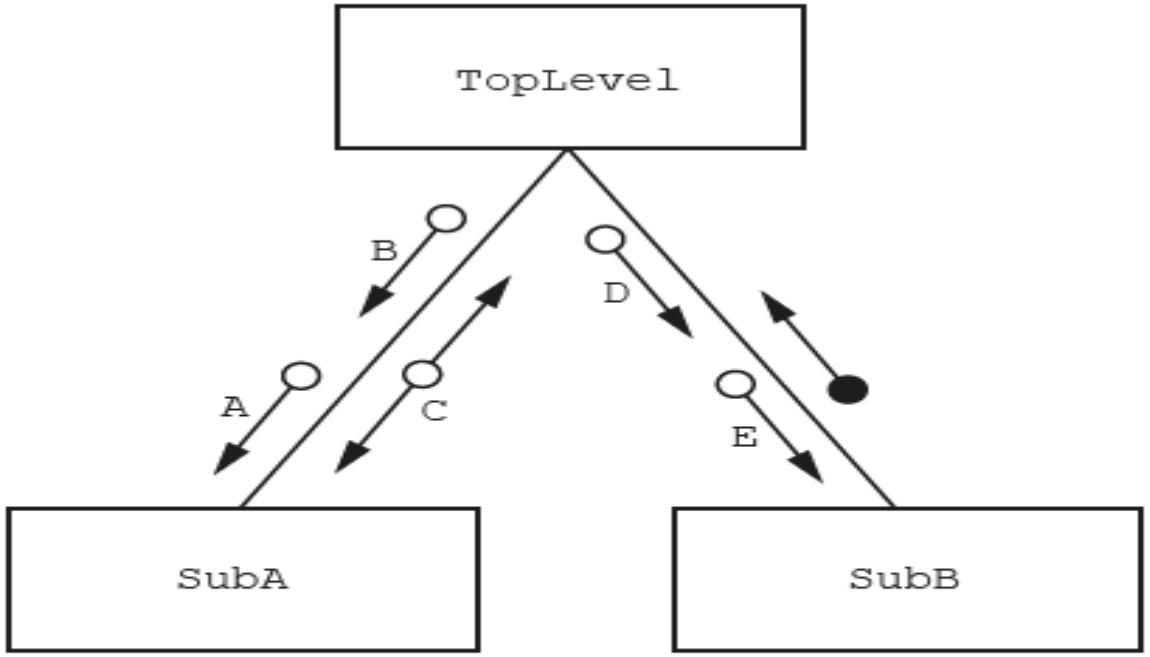
Parameter types must be as shown but ignore parameter names (if given)

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 3

The following structure chart shows the relationship between three modules.



Parameters A to E have the following data types:

```
A, D  : STRING
C     : CHAR
B, E  : INTEGER
```

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 3-(a and b)

(a) **(i)** Write the **pseudocode** header for module `SubA()`.

.....

.....

..... [3]

(ii) Write the **pseudocode** header for module `SubB()`.

.....

.....

..... [3]

(b) Module hierarchy and parameters are two features that may be represented on a structure chart.

State **two other** features than can be represented.

Feature 1 .....

Feature 2 .....

[2]

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 3

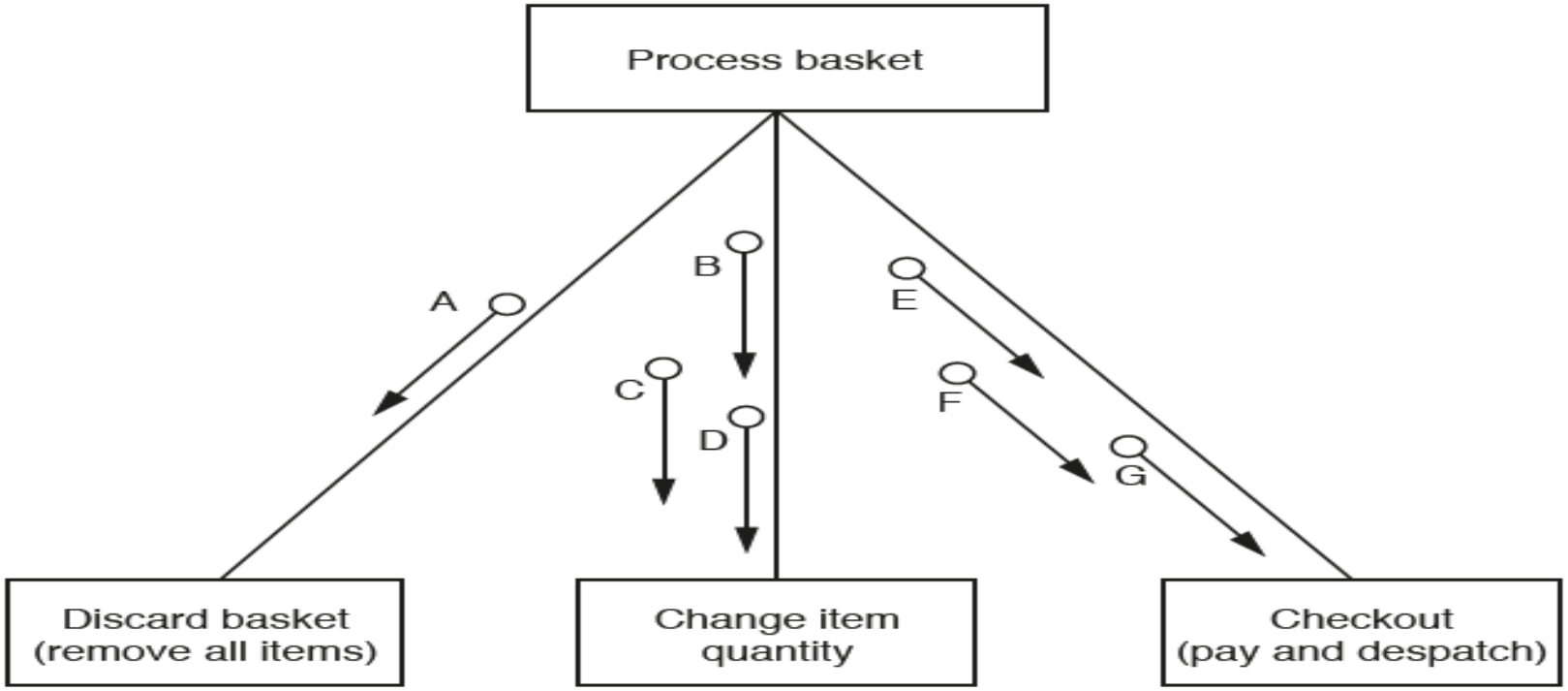
Question	Answer	Marks
3(a)(i)	`PROCEDURE SubA (A : STRING, B : INTEGER, BYREF C : CHAR)` One mark for each underlined part Ignore `BYVAL` for parameter A and/or parameter B Parameter order / names not important but must be correct data types	3
3(a)(ii)	`Function SubB (D : STRING, E : INTEGER) RETURNS BOOLEAN` One mark for each underlined part Ignore `BYVAL` for parameter D and/or parameter E Parameter order / names not important but must be correct data types	3
3(b)	• Selection • Iteration • Sequence One mark per bullet to max. 2	2

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 4

The structure chart shows part of the design of a program for an online shopping system.



(a) (i) Draw on the chart to show the following facts.

- Each of the modules at the lower level returns a Boolean parameter, X.
- Process basket will call only one of the modules shown at the lower level.

[2]

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 4

(ii) The parameters A to G shown on the chart will be used to pass the following information.

```
PaymentDetails
Quantity
BasketID
DeliveryAddress
ItemID
```

Complete the following table to show the parameter and the information it represents.

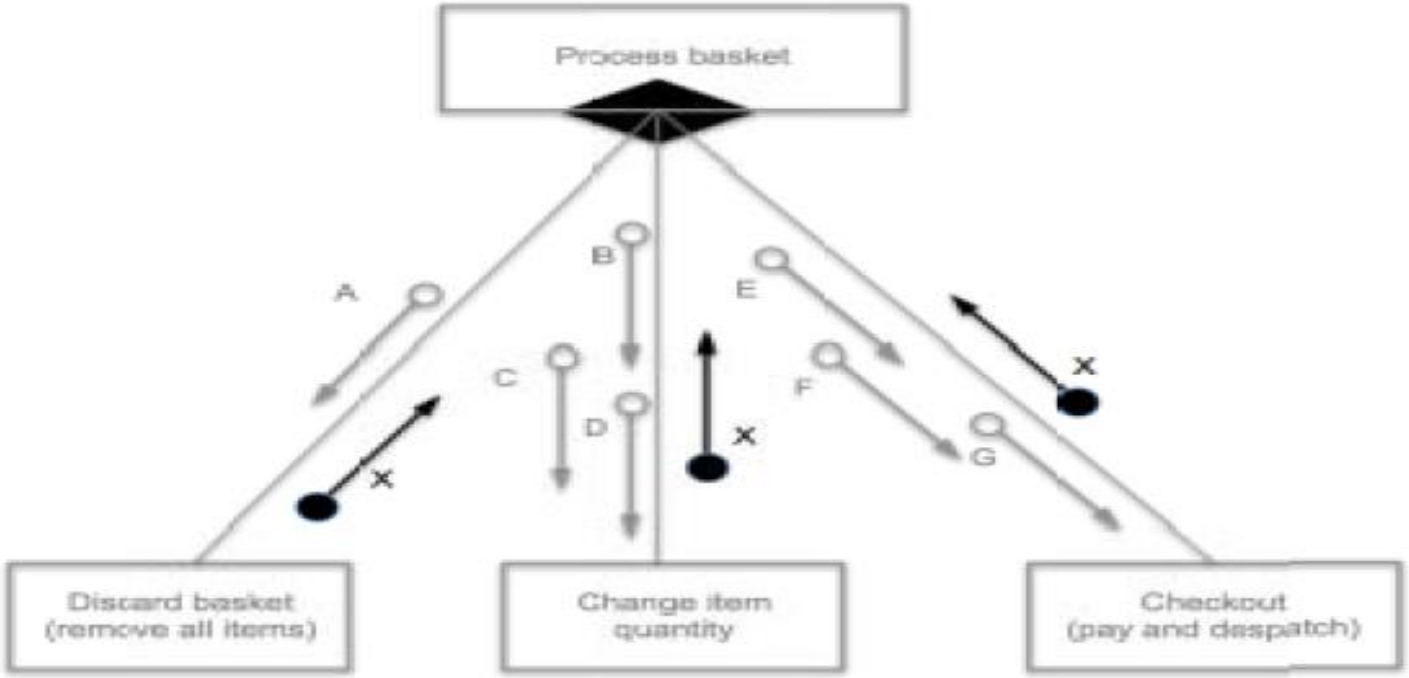
Parameter	Information
A	
B	
C	
D	
E	
F	
G	

[3]

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 4

Question	Answer	Marks
4(a)(i)	 Mark as follows: • One mark for three arrows all labelled X (with filled circles) • One mark for diamond symbol	2

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 4

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(a)(ii)</td><td>
<table>
<tr><th>Parameter</th><th>Information</th></tr>
<tr><td>A</td><td>BasketID</td></tr>
<tr><td>B</td><td rowspan="3">BasketID, ItemID, Quantity
(In any order)</td></tr>
<tr><td>C</td></tr>
<tr><td>D</td></tr>
<tr><td>E</td><td rowspan="3">BasketID, DeliveryAddress, PaymentDetails
(In any order)</td></tr>
<tr><td>F</td></tr>
<tr><td>G</td></tr>
</table>
Mark as follows:

• One mark for parameter A
• One mark for parameters B, C & D
• One mark for parameters E, F & G
</td><td>3</td></tr>
</table>

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 5 –(a)

(a) A structure chart is a tool used in modular program design.

State **three** pieces of information that a structure chart can convey about a program design.

1 .....

.....

2 .....

.....

3 .....

.....

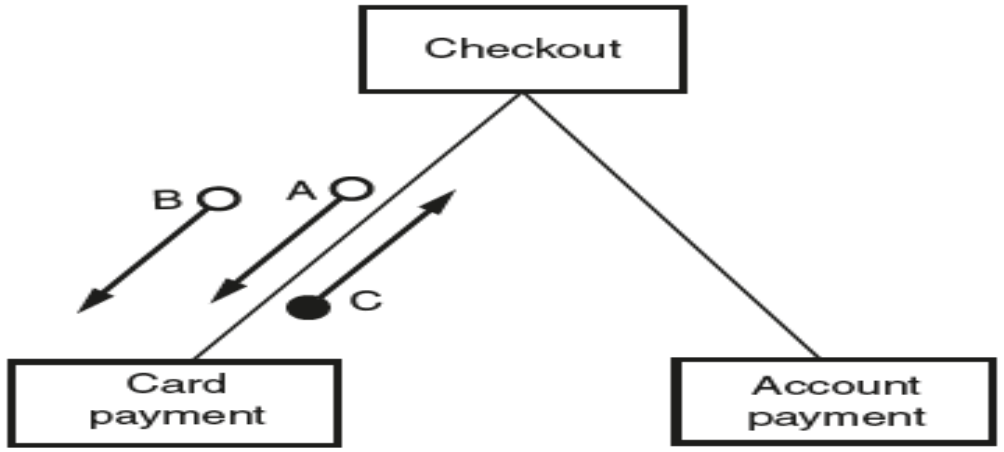
[3]

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 5 –(b)

(b) The following diagram shows part of a structure chart.



Examples of the data items that correspond to the arrows are given in this table:

Arrow	Data item
A	`234.56`
B	`"Mr Robert Zimmerman"`
C	`True`

Use **pseudocode** to write the function header for the **Card payment** module.

.....

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 5

Question	Answer	Marks
4(a)	• The hierarchy of modules • Parameters that are passed between modules // The interface between the modules / • The sequence • Iteration / selection One mark per item	3
4(b)	`FUNCTION CardPayment (ParamA : REAL, ParamB : STRING) RETURNS BOOLEAN` One mark per underlined part Order not significant for `ParamA` and `ParamB` Function name and parameter names not important but must be present	3

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 6-(a)

(a) Name **two** features of your chosen high-level programming language that support the implementation of a modular design.

1 .....

2 .....

[2]

AS/A level past paper 2 exam: Answer 6-(a)

(a)
- Functions / Procedures
- Ability to pass parameters between modules
- Use of local / global variables

[2]

12.2 Program Design

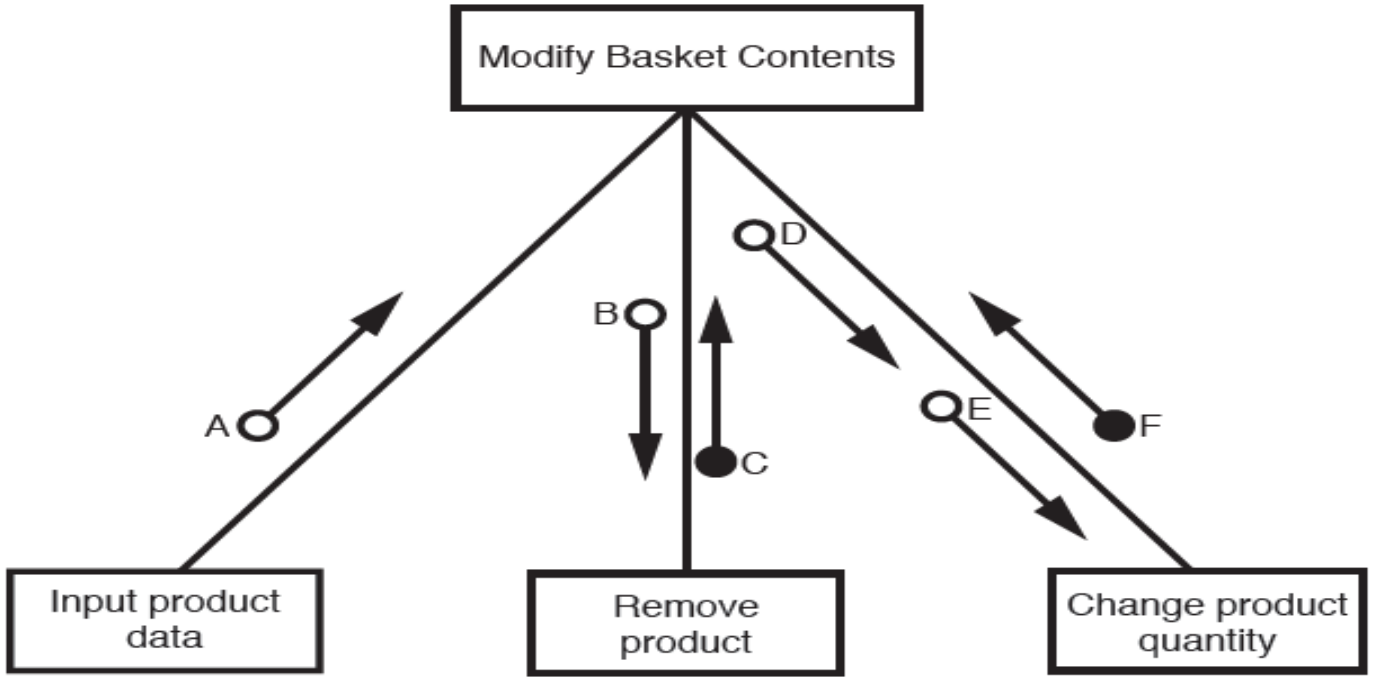
12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 6-(b-i)

(b) (i) The structure chart shows part of the design of a program for an online shopping system.

The user has already added a number of products to their virtual basket.

Draw on the chart, the symbol to show that the process of modifying the basket contents may be iterated (repeated).



[1]

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 6-(b-ii)

(ii) Each arrow in the structure chart above represents a parameter.

The table below shows the three data items that the six parameters pass between modules.

Tick (✓) to match each parameter to the correct data item.

Data item	Parameter					
	A	B	C	D	E	F
Product ID						
Quantity						
Flag Value – indicating operation success or fail						

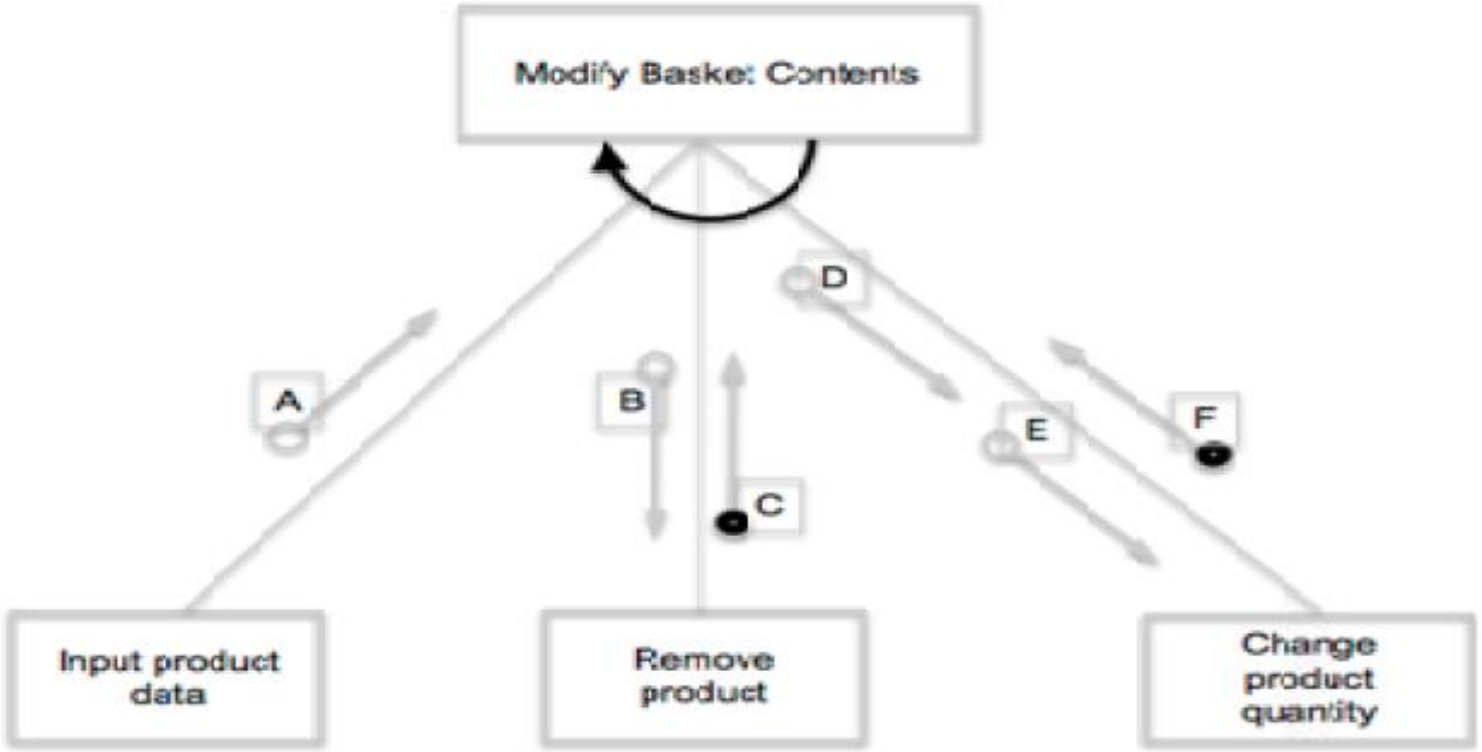
[4]

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 6-(b-i)

(b) (i) [1]



One mark for correct arrow as shown – accept either direction

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 6-(b-ii)

(ii) [4]

Data Item	Parameter					
	A	B	C	D	E	F
Product ID	✓	✓		✓	(✓)	
Quantity				(✓)	✓	
Flag Value – indicating operation success or fail			✓			✓

Mark as follows:
Row 1: One mark for tick in A **AND** B, one mark for D **OR** E
Row 2: One mark for D **OR** E (must be opposite of Row 1)
Row 3: One mark for C **AND** F

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 7

When the guarantee on a computer runs out, the owner can take out insurance to co.
and repairs.

The price of the insurance is calculated from:

- the model of the computer
- the age of the computer
- the current insurance rates

Following an enquiry to the insurance company, the customer receives a quotation letter with the price of the insurance.

A program is to be produced.

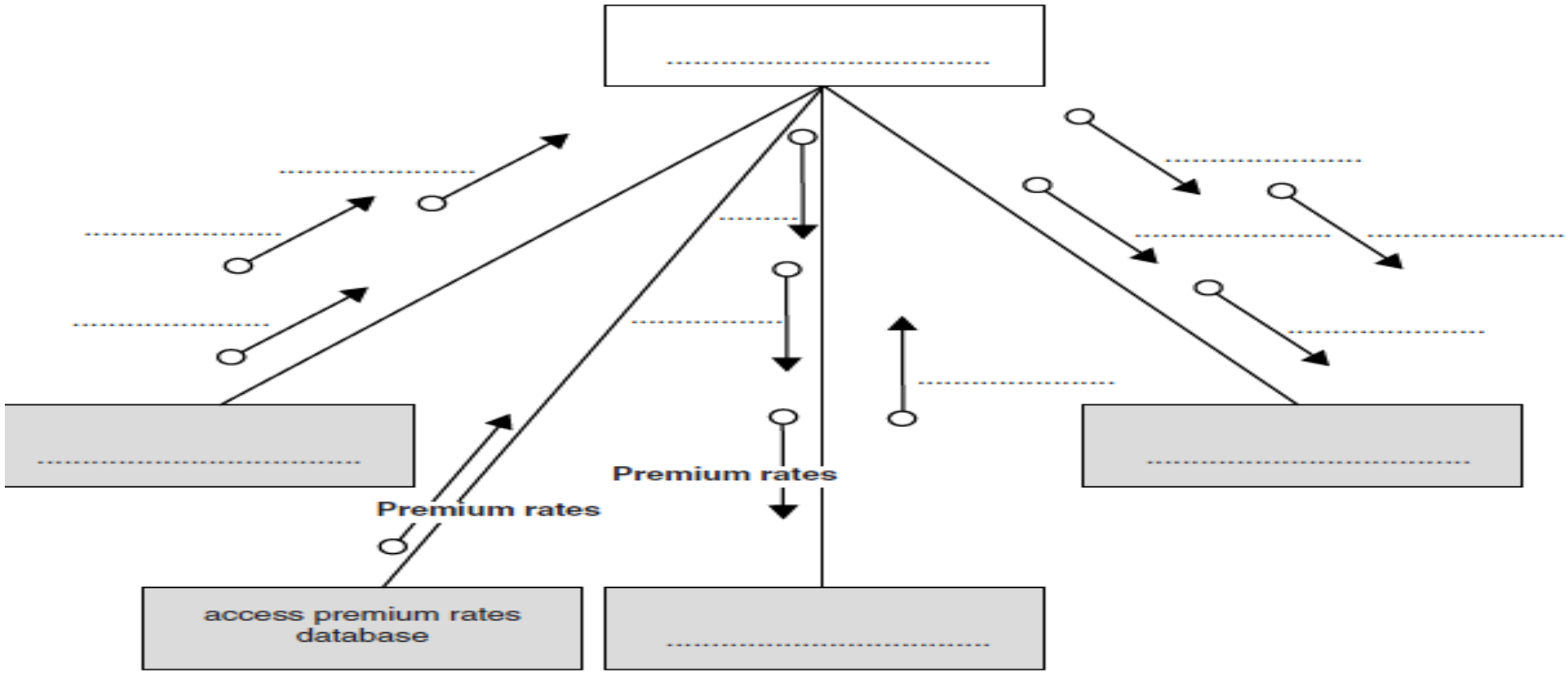
The structure chart below shows the modular design for this process:

12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 7

The structure chart below shows the modular design for this process:



12.2 Program Design

12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Question 7

(a) Using the letters **A** to **D**, add the labelling to the chart boxes on the opposite pag

Modules	
A	Send quotation letter
B	Calculate price
C	Produce insurance quotation
D	Input computer details

[2]

(b) Using the letters **E** to **J**, complete the labelling on the chart opposite.

Some of these letters will be used more than once.

Data items	
E	`CustomerName`
F	`CustomerEmail`
G	`Model`
H	`Age`
I	`PolicyCharge`
J	`PolicyNumber`

[4]

12.2 Program Design

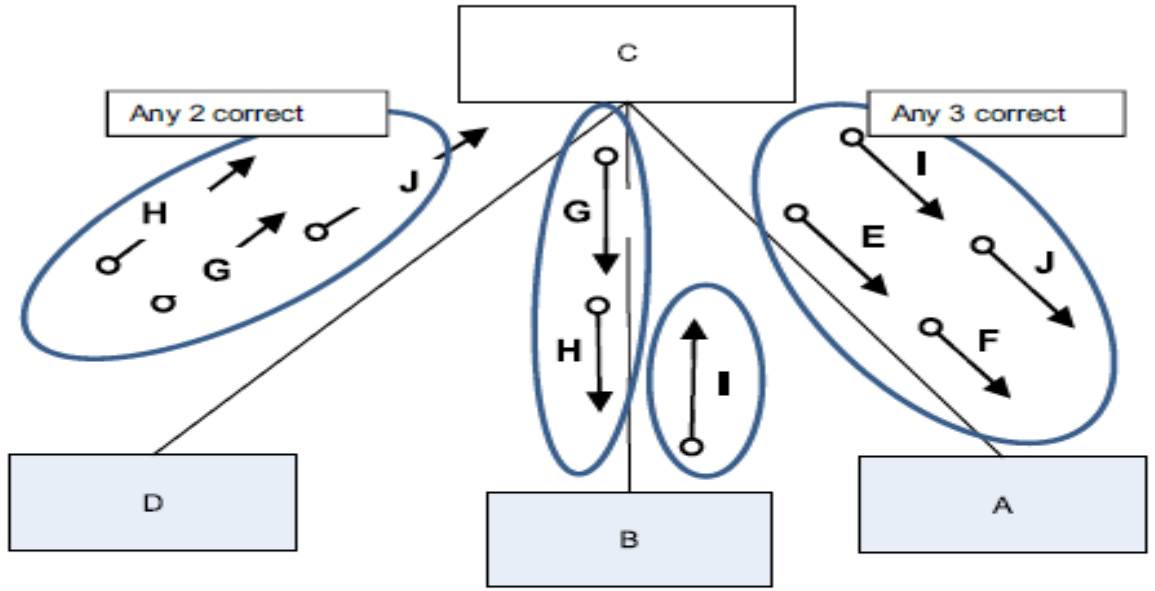
12.2.1 - The purpose of a structure chart

AS/A level past paper 2 exam: Answer 7

(a) Control box – C // Produce insurance quotation

D // Input customer details + A // Send quotation letter is correct positions

(b)



Data items	
E	CustomerName
F	CustomerEmail
G	Model
H	Age
I	PolicyCharge
J	PolicyNumber

12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

Click the link below to watch the video below:

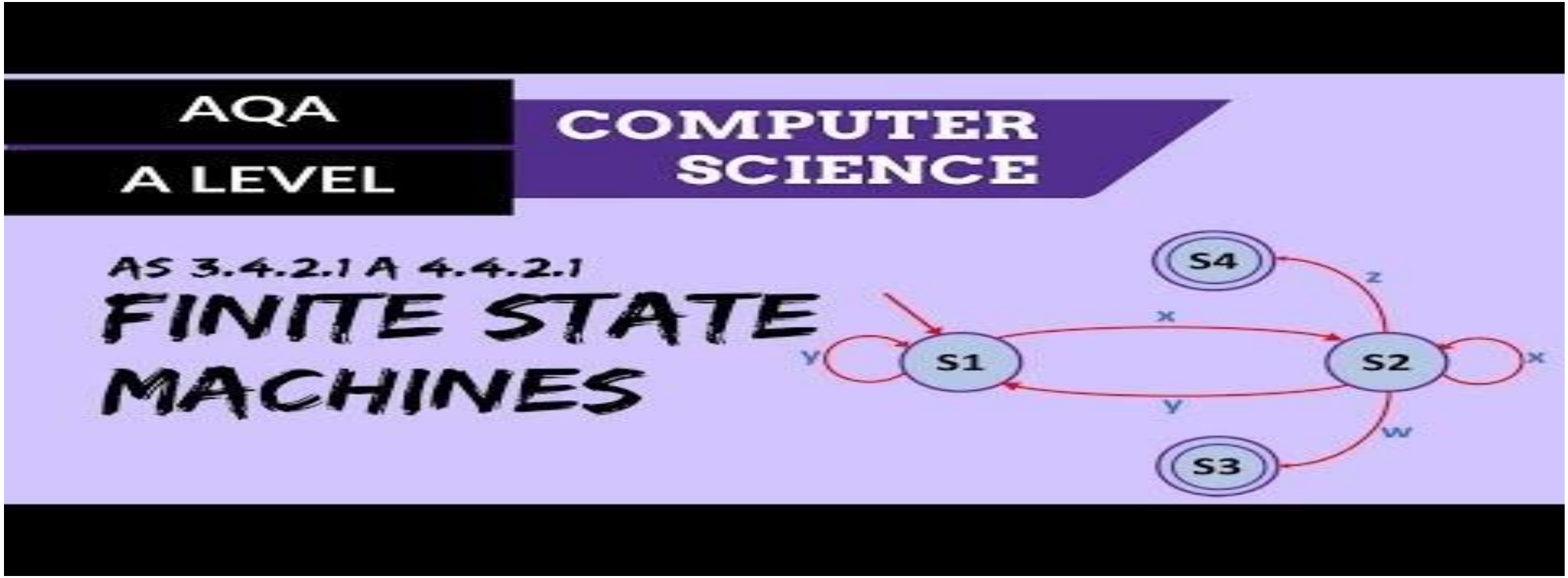
https://www.youtube.com/watch?v=PF9QcYWIsVE

12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

Click the link below to watch the video below:

https://www.youtube.com/watch?v=9YnjgXmv6fU



12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

A **finite state machine (FSM)** is a mathematical model of a machine that can be in one of a fixed set of possible states. One state is changed to another by an external input, this is called a transition. A diagram showing the behavious of an FSM is called a **state-transition diagram**.

State-transition diagrams show the conditions needed for an event or events that will cause a transition to occur, and the outputs or actions carried out as the result of that transition.

State-transition diagrams can be constructed as follows:

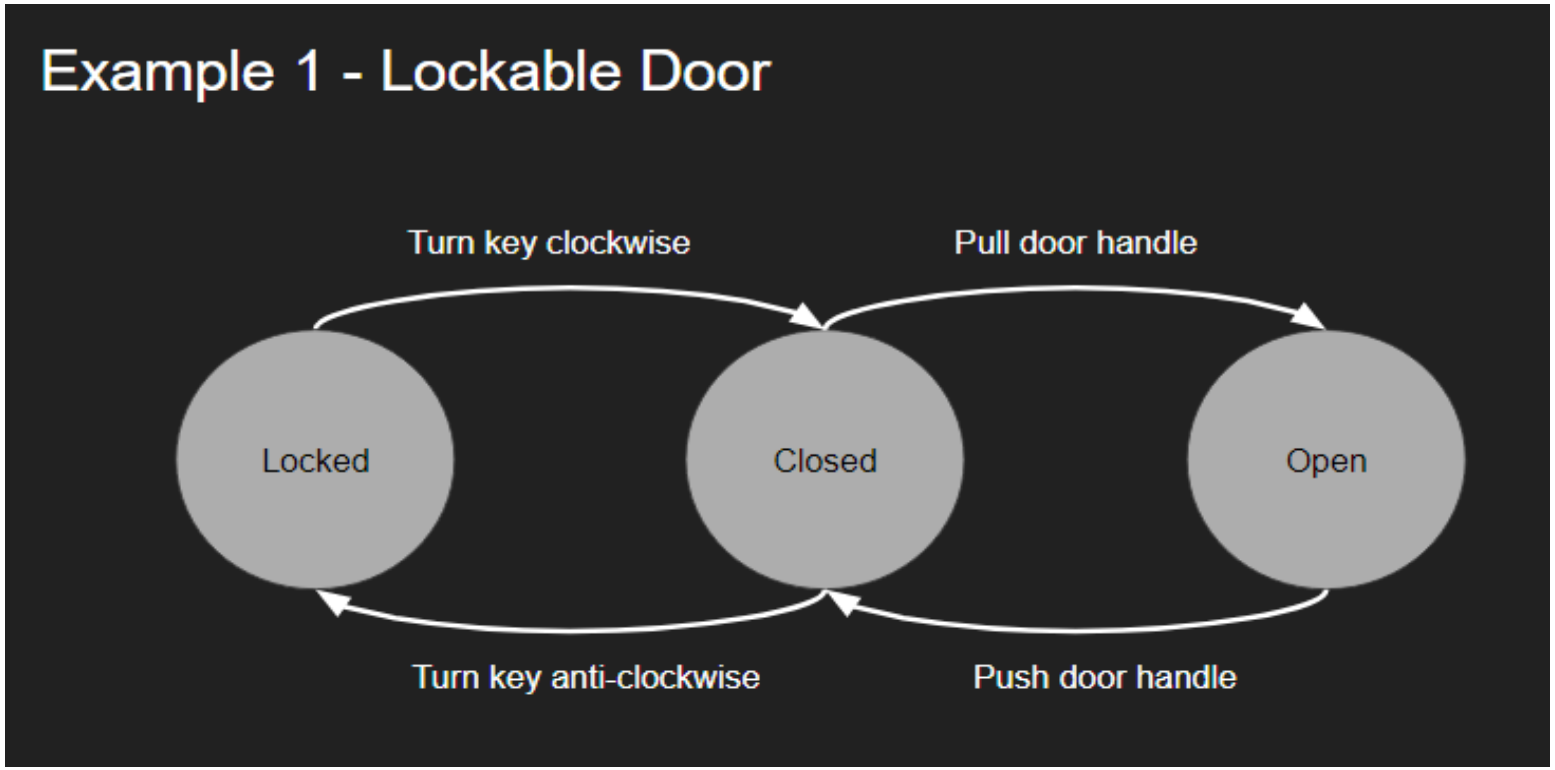
- States are represented as nodes (circles).
- Transitions are represented as interconnecting arrows.
- Events are represented as labels on the arrows.
- Conditions can be specified in square brackets after the event label.
- The initial state is indicated by an arrow with a black dot.
- A stopped state is indicated by a double circle.

12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

They are used to graphically visualize the behavious of Finite State Machines.

- At any moment in time a Finite State Machine can only be in one particular state. When an event happens and/or a guard condition is met then the machine will transition to a different state.
- Movement from state to state is restricted by the direction of the arrows and as such a State Transition Diagram is a form directed graph.



12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

Example 1:

The algorithm for unlocking a door using a three-digit entry code can be represented by a state transition diagram. If the door is unlocked with a three-digit entry code, the lock can be in four states

- locked and waiting for the input of the first digit
- waiting for the input of the second digit
- waiting for the input of the third digit
- unlocked.

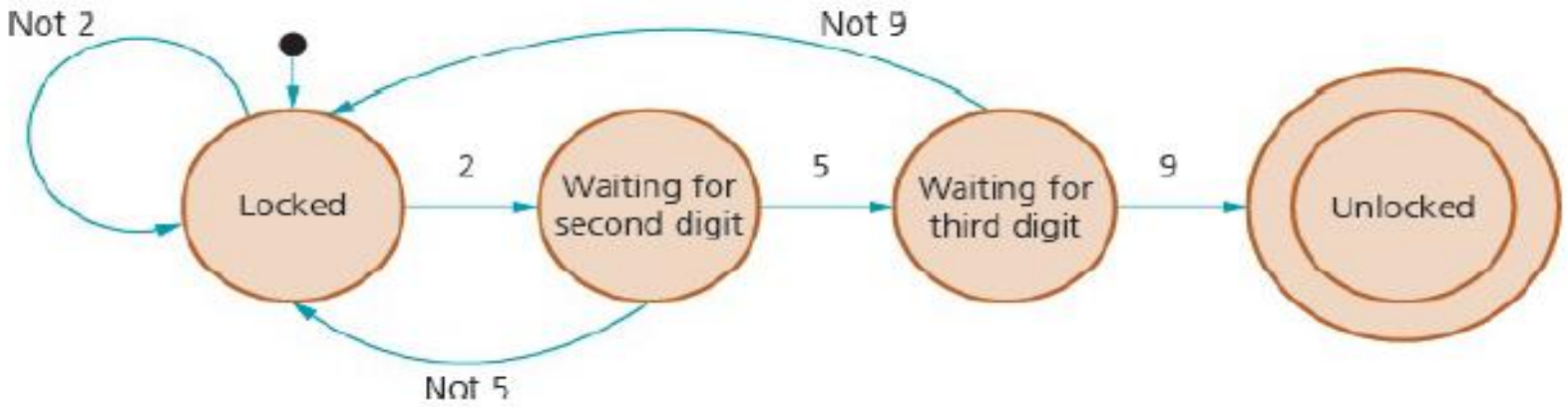
If an incorrect digit is input, then the door returns to the locked state. The algorithm halts when the door is unlocked. A **state-transition table** shows every state, each possible input and the state after the input. The state-transition table for a door with the entry code 259 is shown below.

12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

Current state	Event	Next state
locked	2 input	waiting for input of 2nd digit
locked	not 2 input	locked
waiting for input of 2nd digit	5 input	waiting for input of 3rd digit
waiting for input of 2nd digit	not 5 input	locked
waiting for input of 3rd digit	9 input	unlocked and stopped
waiting for input of 3rd digit	not 9 input	locked

The state-transition diagram for a door with the entry code 259 is shown in Figure below:



12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

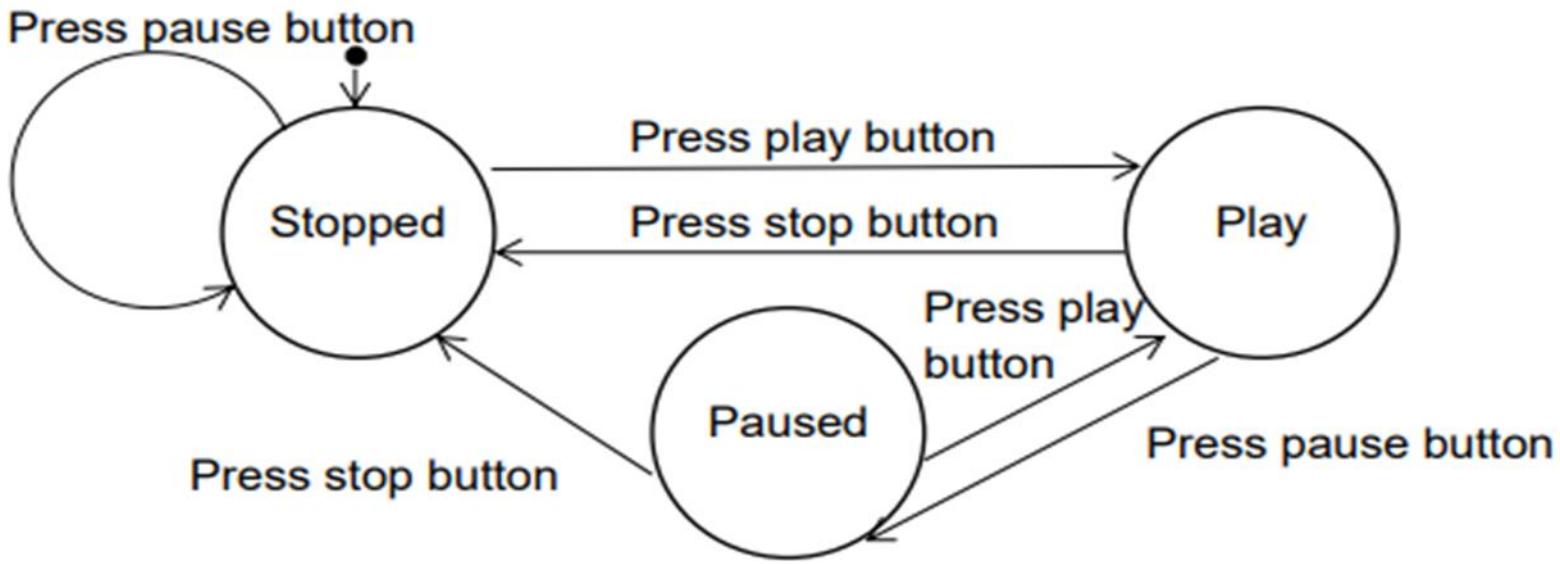
Example 2: Media player: The example below shows a simple state-transition diagram for a media player with three buttons: stop, play and pause. The initial state of the player is stopped. In each state, only the buttons for the other states can be pressed (e.g. in play, only the stop and pause buttons can be pressed). Pressing the pause button when the player is stopped does not result in any change to the player.

The event (press pause when state is Stopped) that does not cause any change in state is indicated by the circular arrow. A finite-state machine can also be represented by a state-transition table, which lists all the states, all possible events, and the resulting state. The following is the state-transition table for the diagram below:

12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

Current State	Event	Next State
Stopped	Press play button	Play
Stopped	Press pause button	Stopped
Play	Press stop button	Stopped
Play	Press pause button	Paused
Paused	Press play button	Play
Paused	Press stop button	Stopped



12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

Example 3– combination lock: State-transition diagrams are also useful for showing the working of algorithms that involve a finite number of states. The following algorithm is for a three-digit combination lock where the correct combination to unlock is '367'. The initial state is Locked, each correct digit changes the state, until the combination unlocks the lock. An incorrect digit returns the lock to the original locked state.

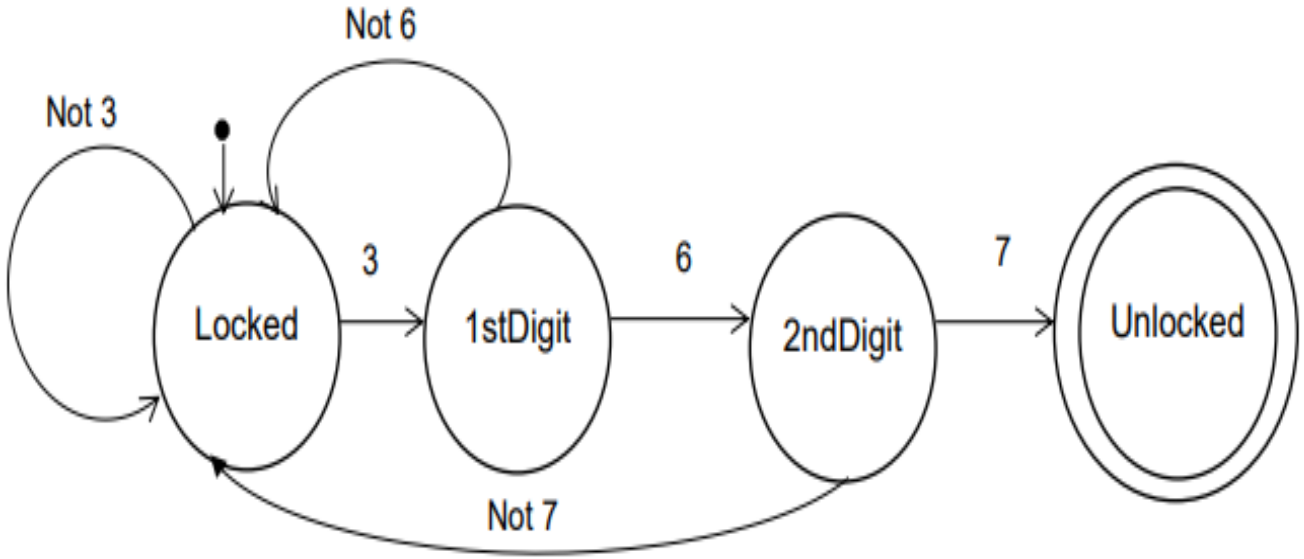
```
DECLARE State : String
DECLARE Number : Integer

State ← Locked
INPUT Number
CASE OF Number
   3 : IF State = Locked
         THEN State ← 1stDigit
       ENDIF
   6 : IF State = 1stDigit
         THEN State ← 2ndDigit
         ELSE State ← Locked
       ENDIF
   7 : IF State = 2ndDigit
         THEN State ← Unlocked
         ELSE State ← Locked
       ENDIF
ENDCASE
```

12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

Example 3– combination lock:

The state-transition diagram for the algorithm is shown below:




```
DECLARE State : String
DECLARE Number : Integer

State ← Locked
INPUT Number
CASE OF Number
   3 : IF State = Locked
         THEN State ← 1stDigit
       ENDIF
   6 : IF State = 1stDigit
         THEN State ← 2ndDigit
         ELSE State ← Locked
       ENDIF
   7 : IF State = 2ndDigit
         THEN State ← Unlocked
         ELSE State ← Locked
       ENDIF
ENDCASE
```

The double line around the Unlocked state indicates that lock halts in this state – this is also known as the ‘accepting state’

12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

1. Key terms

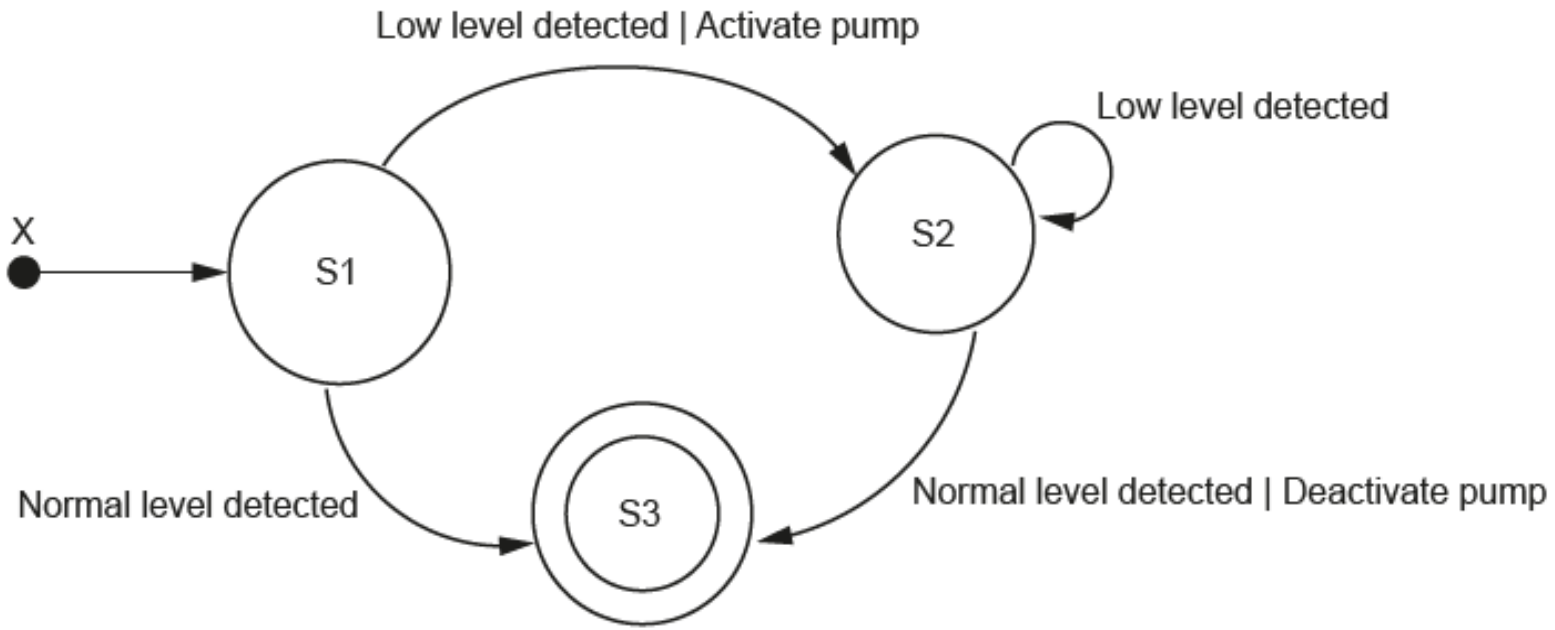
Word/phrase	Meaning
accepting state	A state the system reaches when the input string is valid
event	Something that can happen within a system, such as a timer event, or an input to the system, that may trigger a transition to another state
finite state machine (FSM)	A system that consists of a fixed set of possible states with a set of allowable inputs that may change the state and a set of possible outputs
guard condition	A condition which must be met for a transition to occur from one state to another
state	The value or the position in which a system is at a given point
state transition diagram	A graphical representation of a finite state machine
state transition table	A table that shows all the states of an FSM, all possible inputs and the state resulting from each input
transition	The change from one state to another state

12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

AS/A level past paper 2 exam: Question 1

(a) Examine the following state-transition diagram.



12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

AS/A level past paper 2 exam: Question 1

(i) Complete the table with reference to the diagram.

	Answer
The number of transitions that result in a different state	
The number of transitions with associated outputs	
The label that should replace 'X'	
The final or halting state	

[4]

(ii) The current state is S1. The following inputs occur.

1. Low level detected
2. Low level detected
3. Low level detected
4. Low level detected

Give the number of outputs and the current state.

Number of outputs .....

Current state .....

[2]

12.2 Program Design

12.2.2 - The purpose of state-transition diagrams to document an algorithm

AS/A level past paper 2 exam: Answer 1

Question	Answer	Marks
2(a)(i)	The number of transitions that result in a different state: **3** The number of transitions with associated outputs: **2** The label that should replace 'X': **Start** The final or halting state: **S3** One mark per row	4

Question	Answer	Marks
2(a)(ii)	Number of outputs: 1 Current state: S2	2

12.3 Program Testing and maintenance

- Choose appropriate test data for a test plan (Including normal, abnormal and extreme/boundary)
- Show understanding of the need for continuing maintenance of a system and the differences between each type of maintenance (Including perfective, adaptive, corrective)
- Analyze an existing program and make amendments to enhance functionality

12.3 Program Testing and maintenance

Programs need to be rigorously tested before they are released. Tests begin from the moment they are written; they should be documented to show that the program is robust and ready for general use. In order to clarify what tests need to be performed, a **test plan** is drawn up showing all the **stages of testing**.

During the program design stage, **pseudocode is written**. This can be tested using a dry run which the developer works thought the documents the results using a **trace table**.

For example, a procedure to perform a calculation could be tested as follows.

```
PROCEDURE calculation(number1, number2, sign)
CASE sign OF
   '+' : answer ← number1 + number2
   '-' : answer ← number1 + number2
   '*' : answer ← number1 * number2
   '/' : answer ← number1 / number2
  OTHERWISE answer ← 0
ENDCASE
IF answer <> 0
  THEN
    OUTPUT answer
ENDIF
ENDPROCEDURE
```

12.3 Program Testing and maintenance

The test data used could include 20 10 +, 20 10 −, 20 10 *, 20 10 /, 20 10 ? and 20 0 /.

The trace table below shows the value of each variable and any output.

number1	number2	sign	answer	OUTPUT
20	10	+	30	30
20	10	–	30	30
20	10	*	200	200
20	10	/	2	2
20	10	?	0	
20	0	/	undefined	

The errors found in the routine by performing the dry run have been highlighted in red. These can now be corrected before this routine is coded.

12.3 Program Testing and maintenance

Walkthrough: is a formalised version of a dry run using pre-defined test cases. This is where another member of the development team independently dry runs the pseudocode, or the developer takes the team members through the dry run process. This is often done as a demonstration. During the program development and testing, each module is tested as set out in the test plan. Test plans are often set out as a table; an example for the calculation procedure is shown below.

Test	Purpose	Test data	Expected outcome	Actual outcome
to test the + calculation	to ensure that the + calculation works as expected	normal data 20 10 +	30	30
		abnormal data twenty ten +	error message	incorrect calculation
to test the – calculation	to ensure that the – calculation works as expected	normal data 20 10 –	10	30
		abnormal data twenty ten –	error message	incorrect calculation

The results from this testing show that the error in the subtraction calculation has not been fixed and the routine is not trapping any abnormal data in the variables used by the calculation. These errors will need correcting and then the routine will need to be retested.

12.3 Program Testing and maintenance

Several types of test data need to be used during testing:

- **Normal test data:** that is to be accepted by a program and is used to show that the program is working as expected.
- **Abnormal test data:** that should be rejected by a program as it is unsuitable or could cause problems.
- **Extreme test data:** that is on the limit of that accepted by a program; for example, when testing a validation rule such as number >= 12 AND number <= 32 the extreme test data would be 12 at the lower limit and 32 at the upper limit; both these values should be accepted.
- **Boundary test data:** that is on the limit of that accepted by a program or data that is just outside the limit of that rejected by a program; for example, when testing a validation rule such as number >= 12 AND number <= 32 the boundary test data would be 12 and 11 at the lower limit and 32 and 33 at the upper limit; 12 and 32 should be accepted, 11 and 33 should be rejected.

12.3 Program Testing and maintenance

As the program is being developed the following types of testing are used:

- **White-box testing:** is the detailed testing of how each procedure works. This involves testing the structure and logic of every path through a program module.
- **Black-box testing:** tests a module's inputs and outputs.
- **Integration testing:** is the testing of any separately written modules to ensure that they work together, during the testing phase of the program development lifecycle. If any of the modules have not been written yet, this can include stub testing, which makes use of dummy modules for testing purposes.

When the program has been completed, it is tested as a whole:

- **Alpha testing:** is used first. The completed, or nearly completed, program is tested in-house by the development team.
- **Beta testing:** is then used. The completed program is tested by a small group of users before it is generally released.
- **Acceptance testing:** is then used for the completed program to prove to the customer that it works as required in the environment in which it will be used.

12.3.1 maintenance

Program maintenance is not like maintaining a piece of equipment by replacing worn out parts.

Programs do not wear out, but they might not work correctly in unforeseen circumstances. Logic or run-time errors that require correction may occur from time to time, or users may want to use the program in a different way.

Program maintenance can usually be divided into three categories:

- **Corrective maintenance** is used to correct any errors that appear during use, for example trapping a run-time error that had been missed during testing.
- **Perfective maintenance** is used to improve the performance of a program during its use, for example improving the speed of response.
- **Adaptive maintenance** is used to alter a program so it can perform any new tasks required by the customer, for example working with voice commands as well as keyboard entry.

12.3 Program Testing and maintenance

What is an IDE?

- An Integrated Developer Environment is a comprehensive software application for the development and testing of programs.
- An IDE is similar to a source code editor, but contains additional features specifically for programmers.

Eclipse is a fully featured Java coding IDE

Integration with NetBeans IDE | Jelastic

12.3 Program Testing and maintenance

What is an IDE?

Advantages:

- Faster development time, especially if you are unfamiliar with a language
- Easier and quicker to debug

Disadvantages:

- Often platform / language specific s you will have to install multiple IDEs
- Larger storage and RAM overhead
- Cluttered interfaces can be daunting for beginners

12.3 Program Testing and maintenance

What is an Code Editor?

Code editor is nothing but a text editor that is specialized for writing software. It may be a stand-alone program or part of an integrated development environment (IDE). They **make writing and reading the source code easier by differentiating the elements, so the programmers can view their code**.

Advantages:

- Single Editor for all languages
- Low RAM / HDD overhead
- Simpler Interface

Disadvantages:

- Limited coding / debugging assistance
- Need to run/compile script separately

Notepad ++ is popular code editor. It contains some features of an IDE, such as syntax highlighting and collapsible blocks

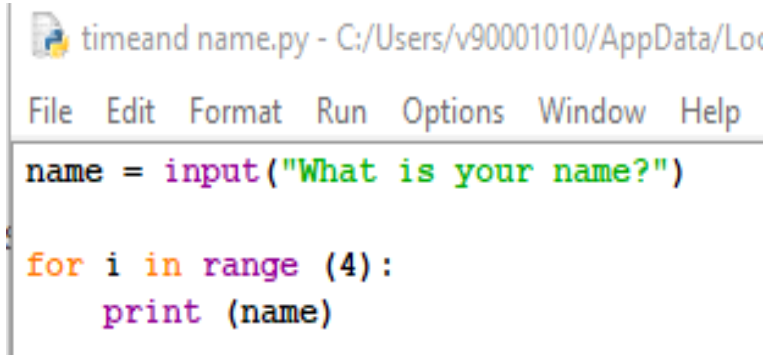
12.3 Program Testing and maintenance

Code Formatting

Syntax Highlighting

- Syntax highlighting is where different types of syntax are displayed in different colors, making the code easier to read see below .

```
#Show today date

from datetime import date
today = date.today()
print("Today is date:", today, "\n")
```



12.3 Program Testing and maintenance

Code Formatting

Pretty Print:

- Pretty Print is the use of formatting rules (such as **indent sizes** and **white-space**) to make code easier to read.

```
>>> from datetime import datetime
>>> from prettyprinter import cpprint, set_default_style
>>> set_default_style('light')
>>> cpprint({'text': 'lorem ipsum dolor sit amet' * 10, 'created': datetime.now()})
{
    'created': datetime.datetime(
        year=2017,
        month=12,
        day=13,
        hour=21,
        minute=21,
        second=21,
        microsecond=318412
    ),
    'text':
        'lorem ipsum dolor sit ametlorem ipsum dolor sit ametlorem ipsum '
        'dolor sit ametlorem ipsum dolor sit ametlorem ipsum dolor sit '
        'ametlorem ipsum dolor sit ametlorem ipsum dolor sit ametlorem ipsum '
        'dolor sit ametlorem ipsum dolor sit ametlorem ipsum dolor sit amet'
}
>>>
```

12.3 Program Testing and maintenance

Code Formatting

Expandable Collapsible code blocks:

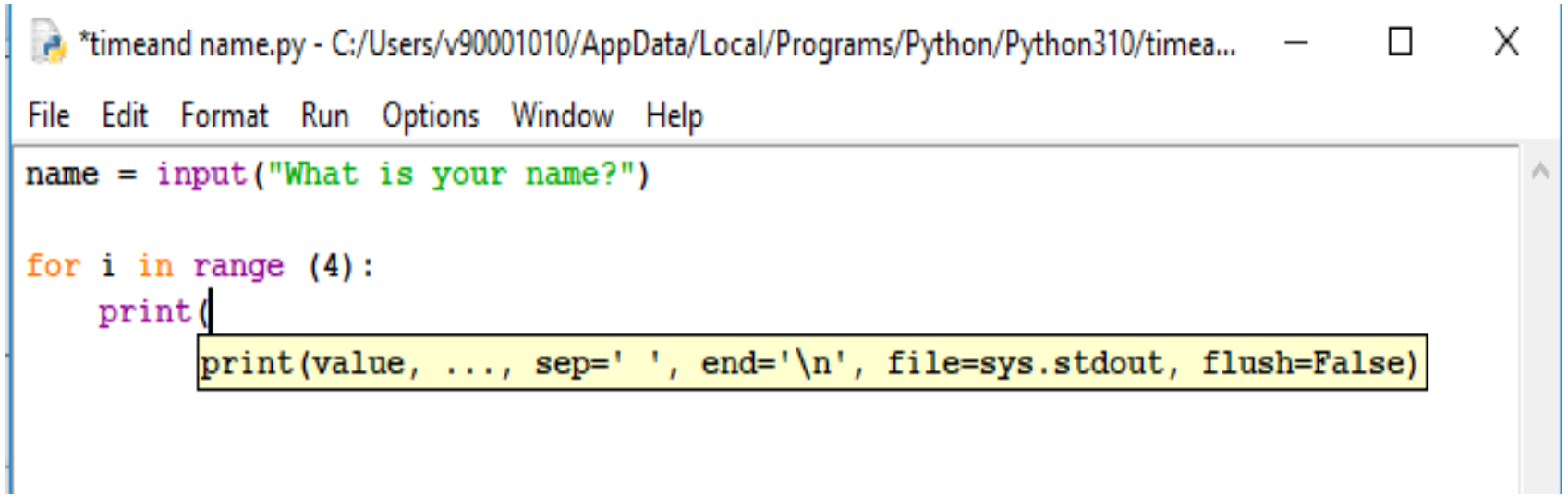
Content of functions and classes can be hidden when viewing is not needed. Makes code easier to read when working on larger projects.

```
class Solver:

    def calculate(self):
            while True:
                a = int(input("a "))
                b = int(input("b "))
                c = int(input("c "))
                d = b ** 2 - 4 * a * c
                if d>=0:
                    disc = math.sqrt(d)
                    root1 = (-b + disc) / (2 * a)
                    root2 = (-b - disc) / (2 * a)
                    print(root1, root2)
                else:
                    print('error')
    pass
```

12.3 Program Testing and maintenance

Auto-completion / Context Sensitive Prompts

- When typing in code the IDE may suggest prompt you to add code(such as the function arguments in the example below). In some instances it may even autocomplete code (for instance adding closing parenthesis when you open them.



12.3 Program Testing and maintenance

Error Detection

- The IDE can often spot mistakes as you type them, such is spelling mistakes or referencing variables before you have assigned them, or referencing them outside of their scope.

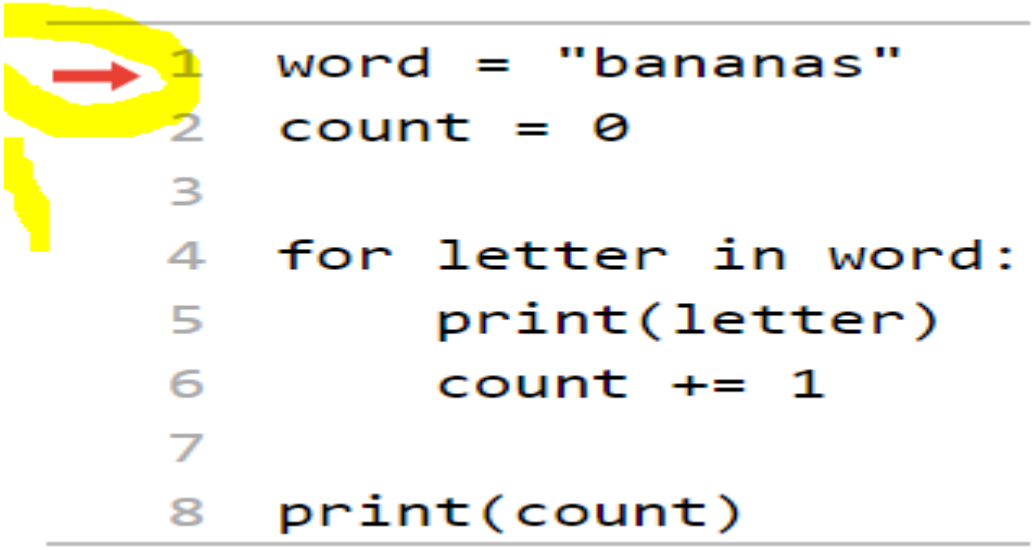
```
= RESTART: C:/Users/v90001010/AppData/Local/Programs/Python/Python310/timeand na
me.py
What is your name?noureddine
Traceback (most recent call last):
  File "C:/Users/v90001010/AppData/Local/Programs/Python/Python310/timeand name.
py", line 4, in <module>
    pritn(name)
NameError: name 'pritn' is not defined. Did you mean: 'print'?
```

12.3 Program Testing and maintenance

Debugging

Single Stepping:

- This is where a program is executed line by line and paused after each line so you can inspect variables



```
1  word = "bananas"
2  count = 0
3
4  for letter in word:
5      print(letter)
6      count += 1
7
8  print(count)
```

12.3 Program Testing and maintenance

Debugging

Breakpoints:

- You can often set a break point in a your IDE to stop the program at a particular point, rather than to step through slowly line by line. This is useful if you are pretty sure where an error is occurring, but not sure why. Whenever your code does not work as expected and you want to find out why, you can utilize the Python debugger to detect the bug. It's very simple to get started. You can insert a breakpoint with the **breakpoint()** function at any position in your code.

With breakpoint no error

timeand name.py - C:/Users/v90001010/AppData/Loca

File Edit Format Run Options Window Help

```
# Create a loop over 5 integers
for i in range(5):
    # Stream i to stdout
    print(i)
    # Create breakpoint at # 3
    if i == 3:
        breakpoint()
```

```
= RESTART: C:/Users/v90001010/App
me.py
0
1
2
3
> c:\users\v90001010\appdata\loca
module>()
-> for i in range(5):
(Pdb)
```

Without using a breakpoint you will get an error

File Edit Format Run Options Window Help

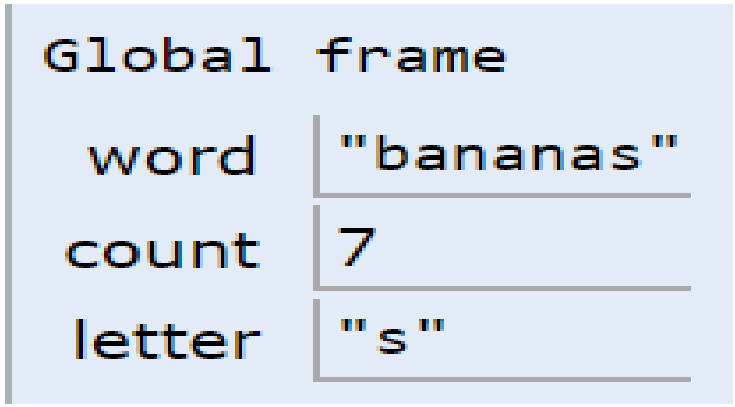
```
# Create a loop over 5 integers
for i in range(5):
    # Stream i to stdout
    print(i)
    # Create breakpoint at # 3
    if i == 3:
        #breakpoint()
```

12.3 Program Testing and maintenance

Debugging

Variable reports window:

The IDE will often have a variable window that displays the current contents of variables and objects. This can be used with single-stepping and breakpoints to debug your code.



12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Question 1:

3 A 1D array, `Product`, of type `STRING` is used to store information about a range of products in a shop. There are 100 elements in the array. Each element stores one data item.

The format of each data item is as follows:

```
<ProductID><ProductName>
```

- `ProductID` is a four-character string of numerals
- `ProductName` is a variable-length string

12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Question 1:

The following pseudocode is an initial attempt at defining a procedure, `ArraySort`, which will perform a bubble sort on `Product`. The array is to be sorted in ascending order of `ProductID`. Line numbers have been added for identification purposes only.

```
01   PROCEDURE SortArray
02     DECLARE Temp : CHAR
03     DECLARE FirstID, SecondID : INTEGER
04     FOR I ← 1 TO 100
05       FOR J ← 2 TO 99
06          FirstID ← MODULUS(LEFT(Product[J], 6))
07          SecondID ← MODULUS(LEFT(Product[J + 1], 6))
08          IF FirstID > SecondID
09             THEN
10                Temp ← Product[I]
11                Product[I] ← Product[J + 1]
12                Product[J + 1] ← Temp
13        ENDFOR
14          ENDIF
15     ENDFOR
16   ENDPROCEDURE
```

12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Question 1:

The pseudocode on page 8 contains a number of errors. Complete the following table to show:

- the line number of the error
- the error itself
- the correction that is required.

Note:

- If the same error occurs on more than one line, you should only refer to it ONCE.
- Lack of optimisation should not be regarded as an error.

Line number	Error	Correction
01	Wrong procedure name – "SortArray"	`PROCEDURE ArraySort`

[8]

12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Answer 1:

Question	Answer	Marks
3		Max 8

Line number	Error	Correction
01	Wrong procedure name – "SortArray"	PROCEDURE ArraySort
02	Wrong data type - CHAR	`DECLARE Temp: STRING`
03	Variables undefined	`DECLARE FirstID, SecondID, I, J : INTEGER`
04	Wrong 'Value2' of 100	`FOR I ← 1 TO 99`
05	Wrong range	`FOR J ← 1 TO (100 - I)`
06/07	Wrong function - MODULUS	Replace MODULUS with TONUM: `FirstID ← TONUM(LEFT(Product[J],`
06/07	Wrong value of 6	Should be 4: `FirstID ← TONUM(LEFT(Product[J],`
10	Assigning wrong value to Temp	`Temp ← Product[J]`
11	Assigning wrong value to Product[I]	`Product[J] ← Product[J + 1]`
13/14	Lines reversed	`13 ENDIF` `14 ENDFOR`

One mark for each correct row

12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Question 2:

A company creates two new websites, Site X and Site Y, for selling bicycles.

Various programs are to be written to process the sales data.

These programs will use data about daily sales made from Site X (using variable SalesX) and Site Y (using variable SalesY).

Data for the first 28 days is shown below.

12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Question 2:

	SalesDate	SalesX	SalesY
1	03/06/2015	0	1
2	04/06/2015	1	2
3	05/06/2015	3	8
4	06/06/2015	0	0
5	07/06/2015	4	6
6	08/06/2015	4	4
7	09/06/2015	5	9
8	10/06/2015	11	9
9	11/06/2015	4	1
...			
28	01/07/2015	14	8

a) Name the data structure to be used in a program for SalesX.

[2]

12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Question 2:

(b) The programmer writes a program from the following pseudocode design.

```
x ← 0
FOR DayNumber ← 1 TO 7
   IF SalesX[DayNumber] + SalesY[DayNumber] >= 10
      THEN
         x ← x + 1
         OUTPUT SalesDate[DayNumber]
   ENDIF
ENDFOR
OUTPUT x
```

(i) Trace the execution of this pseudocode by completing the trace table below.

x	DayNumber	OUTPUT
0		

12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Question 2:

(ii) Describe, in detail, what this algorithm does.

.....

.....

.....

.....[3]

12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Answer 2:

(a) • 1D Array // List [1]

• INTEGER [1]

(b) (i)

x	DayNumber	OUTPUT
0	1	
	2	
1	3	5/6/2015
	4	
2	5	7/6/2015
	6	
3	7	9/6/2015
		3

Note: 'x' and 'output' entries must be on or below the relevant 'DayNumber' entry
Mark as above [4]

12.3 Program Testing and maintenance

AS/A level Past Paper 2 Exam Answer 2:

(ii)
- ... Sales for the first seven days (1)
- ... the number of days on which the total sales were **10 or over** (1)
- Outputs the corresponding dates (1)
- Output the final value/total (of x) (1)