(5) Chapter 2

Program Design Tools and Techniques.

● Write down the nature of the design process

⇒ Very personalized and highly interactive process

⇒ Analyst uses mixture of knowledge and intuition.

⇒ Apply any known design tools to analyze, compare with specification.

⇒ Iterate if the solution does not meet the specification.

⇒ Included analysis, synthesis, conceptualization and trial & error.

● Why synthesis is seldom possible in practical engineering design?

Synthesis uses a clear-cut and straight-forward algorithm to evolve a design which exactly meets the assumed specification. This is not the actual picture.

In general, synthesis is not possible and we are forced to evolve our design via a trial-and-error process.

That is why, synthesis is seldom possible in practical engineering design.

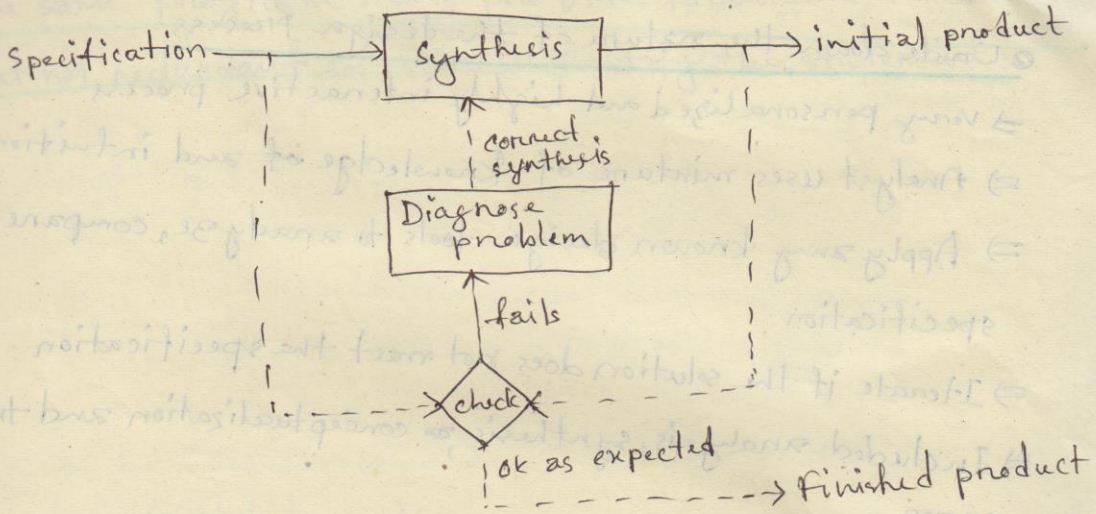
● compare a synthesis procedure with an iterative design procedure along with proper diagram.

I Synthesis Procedure:

→ Uses a clear-cut and straight-forward algorithm.

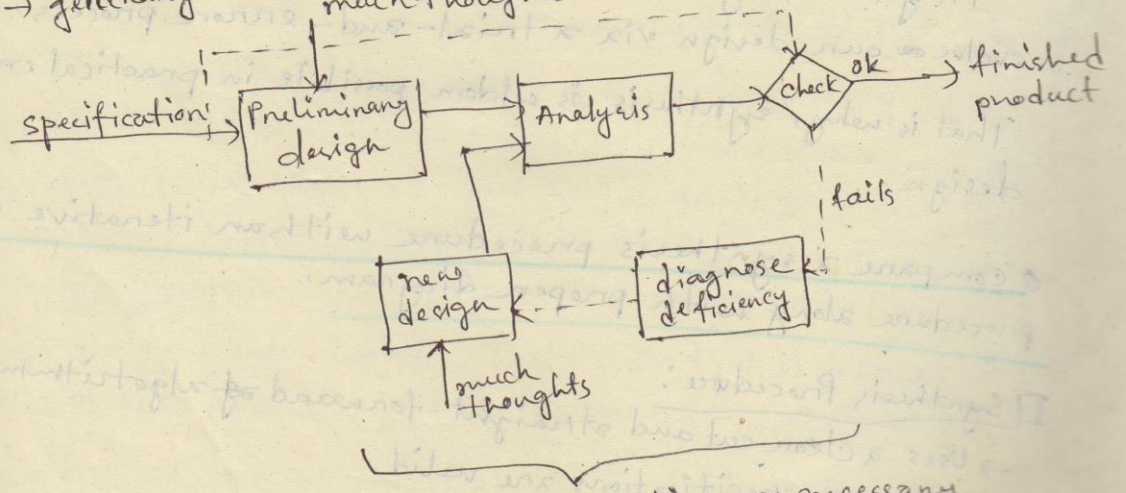
→ Assumes specifications are valid.

→ It is a "one-shot" open loop design.



☐ Iteration:

→ Assumes analysis technique exists.
→ Propose intuitively some hypothetical design and subject it to analysis.
→ generally checked several times. much thoughts



iterate many times as necessary

→ to meet specifications, evolves an improved version of original design if required.

Modern Design Techniques:

1. Top-down design & programming
2. Bottom-up design & programming
3. Modular design and programming
4. Structured design and programming
5. Defensive design and programming
6. Redundant design and programming
7. Automatic Programming.

Top-down Design:

Top-down design is a decomposition process which focuses on the flow of control or on the control structure of the program.

Decomposition Steps:

1. To study the overall aspects of the task at hand and to break it into a number of independent consitituent functions or modules.
2. to break each one of these modules further into independent submodules.
3. These processes are repeated until one obtains modules which are small enough to grasp mentally & to code at one sitting in a straight-forward, uncomplicated manner.

Feature:

→ At each level the details of the design at lower levels are hidden

Bottom-Up Design:

→ First visualizes a typical system design and decides which part or parts of design are the most difficult or limiting ones.

→ These crucial parts are investigated first and necessary design decisions are made.

→ Remainder of the design is tailored to fit around the designs for crucial path.

→ It vaugely represents synthesis.

→ postponed details of decision until the last stages of design.

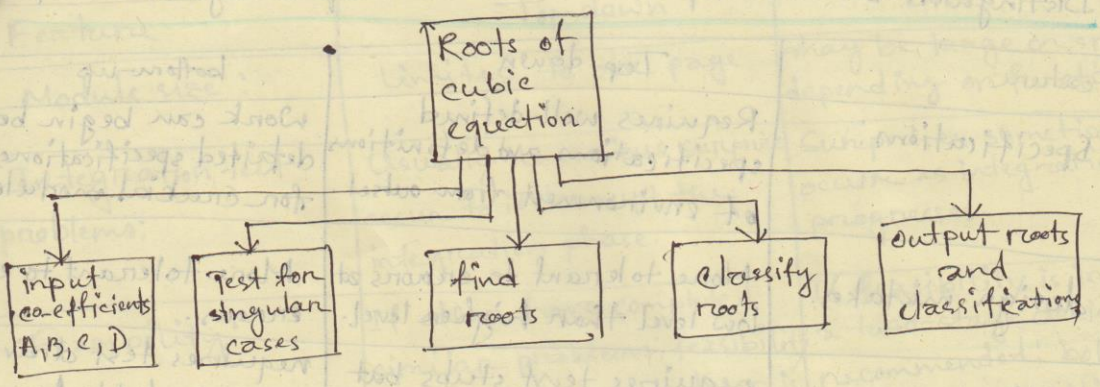
* Consider the following cubic equation: " $Ax^3 + Bx^2 + Cx + D = 0$ ". Illustrate the top-down and bottom-up solutions for finding and classifying the roots of this equation with all levels of decomposition

* State the difference between top-down and bottom-up design with relevant example.

Difference between top-down and bottom-up design with the simple exp example $Ax^3 + Bx^2 + Cx + D = 0$.

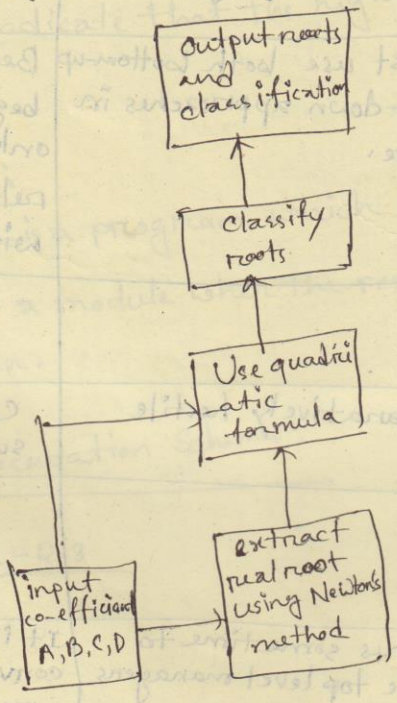
In top-down approach, solution does not commit the design to a specific approach at finest decomposition level. design will come when we go to the second level of detail when the singular cases and the root modules appear.

change in algorithm to be used will change in the find roots module only.



But in case of bottom-up approach, algorithm is already specified; we use newton's method for one root and quadriatic formula for the other roots.

A change in algorithm to be used will change entire structure of bottom-up design.



Distinguish between top-down and bottom-up design techniques

Feature	Top-down	bottom-up
1. Specifications:	Requires well-defined specifications and definitions of environment from outset.	Work can begin based on detailed specifications only for crucial module(s).
2. design mistakes.	More tolerant to errors at low level than higher level.	More tolerant to early errors.
3. Testing:	requires test stubs but not test drivers.	requires test drivers but not a test stubs.
4. Standardization:	standandized structure can be developed for a particular application.	standandized modules can be devised for critical functions which are often used
5. Data Structure:	Can delay details of data structure to lower levels, during initial stages of design	One must decide on all data structures which the module interfaces with before proceeding.
6. Crucial Storage Problem:	We must use both bottom-up and top-down approaches in this case.	Best philosophy is to begin bottom-up design only on crucial storage related problem in parallel with a top-down design
7. Critical real-time problems:		
8. Impact on existing personnel:	Comparatively hostile	Comfortable with no surprises
9. Impact on new personnel:		
10. Management acceptance:	requires some time to educate top level managers	It is easy to accept convince top level manage - ment by showing them the codes

Feature	Top-down	bottom-up
11. Module size:	Limited to one page	may be large or small depending on functions.
12. Integration test problems:	Usually no serious surprise occur to interrupt the integration phase.	Surprises sometime occur as integration progresses.
13. Feasibility:	If other have completed similar problems, feasibility is proved, then top-down is recommended.	If feasibility is in question a two-stage decision is recommended: bottom-up to prove feasibility, top-down for final design
14. Added features:	Easy to include	May cause major redesign

Test stub:

Test stub is a dummy program inserted in place of code not as yet written, to indicate that the right point in the program has been reached.

Test driver:

Test driver is a program which provides control and input data for testing a module when the remainder of the code has not yet been written.

Design Representation Schemes:

Table-2.3

2.3.4. Hipo diagrams:

⇒ Flowchart = a graphical design tool, give the programmer a picture to which he or she can related the design.

⇒ Hierarchy plus Input-Process-Output.

◍ Define HIPO diagrams.

◍ What is HIPO diagram.

⊞ HIPO diagrams:

HIPO techniques is a tool for planning and/or documenting a computer program.

It consists of -

① a hierarchy chart that graphically represents the program's control structure (Top-down structure)

② a set of IPO (Input-Process-Output) charts that describes the inputs to, outputs from and function performed by each module on hierarchy chart.

◍ Discuss the strength, and limitations of HIPO diagrams.

⊟ Strength:

→ Designers can evaluate and refine a program's design, and correct flaws prior to implementation.

→ Users and managers can easily follow a program's structu -re from its graphic nature.

→ Hierachy charts serves as a useful planning and visualizati -on document for ~~planning~~ managing the program's development process.

→ IPO charts defines for the programer each module's inputs, outputs and algorithms.

☐ Limitations:

→ "Text plus flowchart" nature of IPO charts make them difficult to maintain.

→ Documentation often does not represent the current state of the program.

→ Hierarchically structure representation plan and for program documentation is best for it to use.

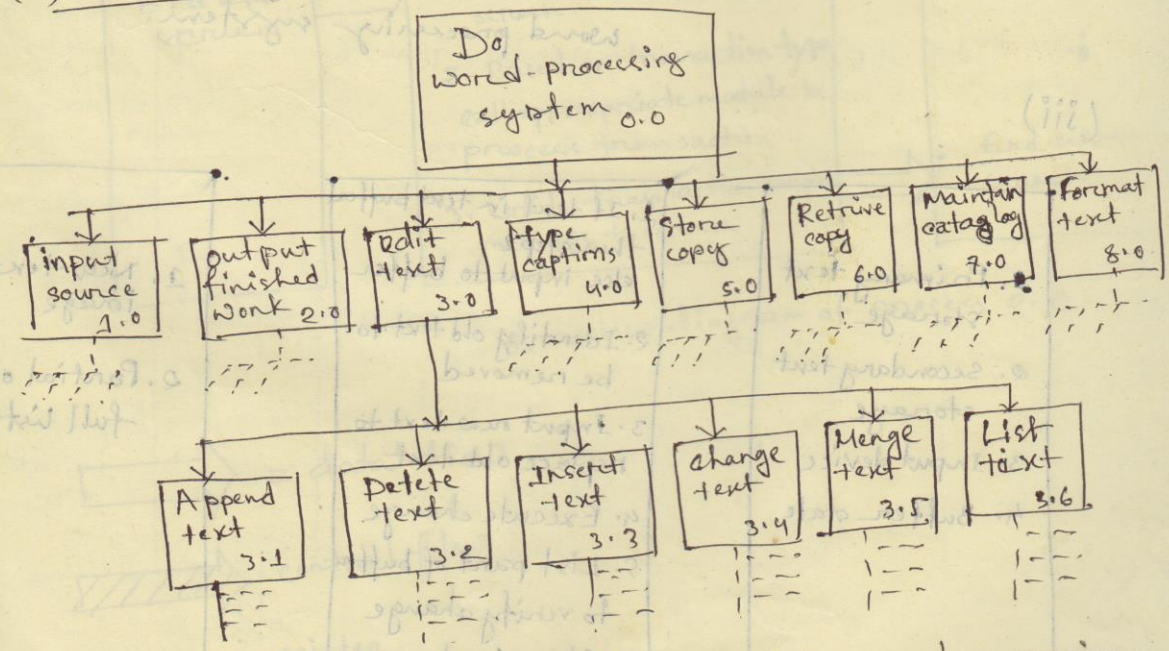
Draw the following diagrams for a word processing system.

(i) hierarchy diagram (H charts)

(ii) Overview IPO diagram for "edit text" process.

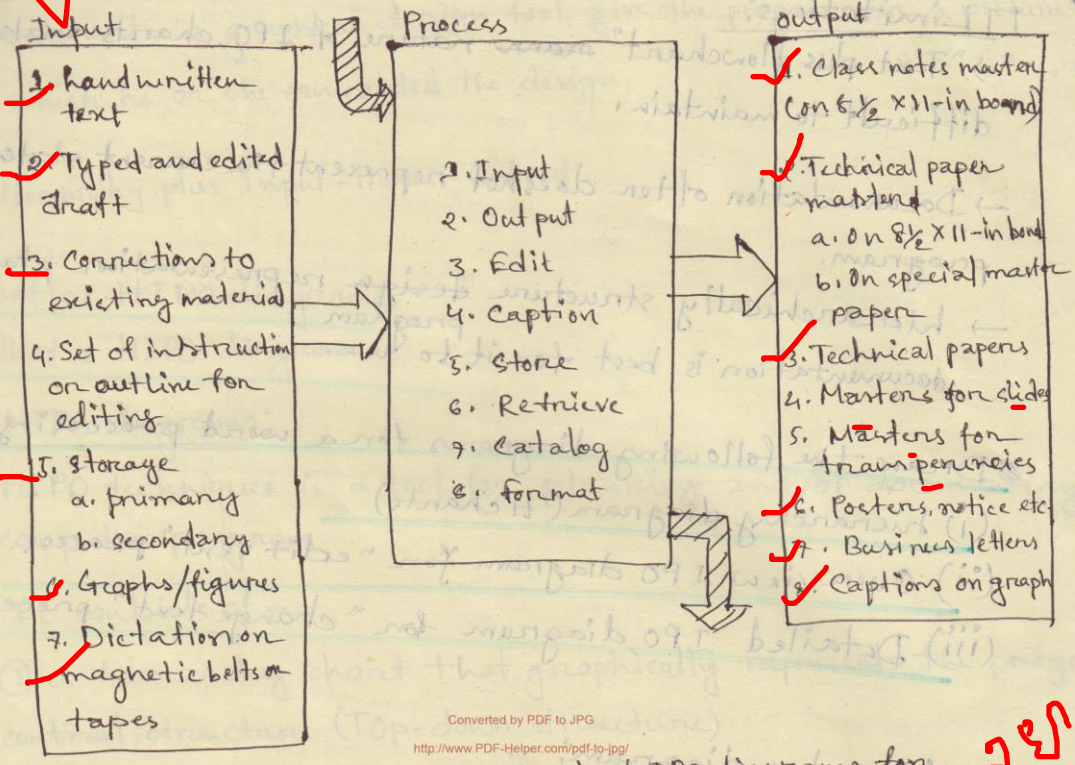
(iii) Detailed IPO diagram for "change text" process.

(i) hierarchy diagram:



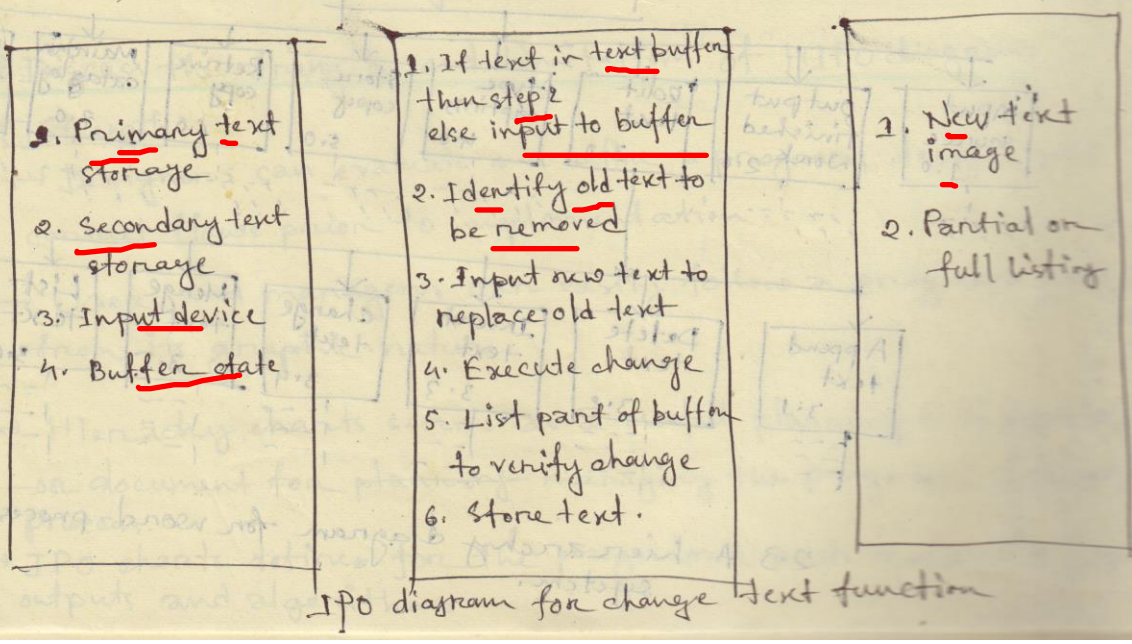
2.3 A hierarchy diagram for word-processing system.

(iii)



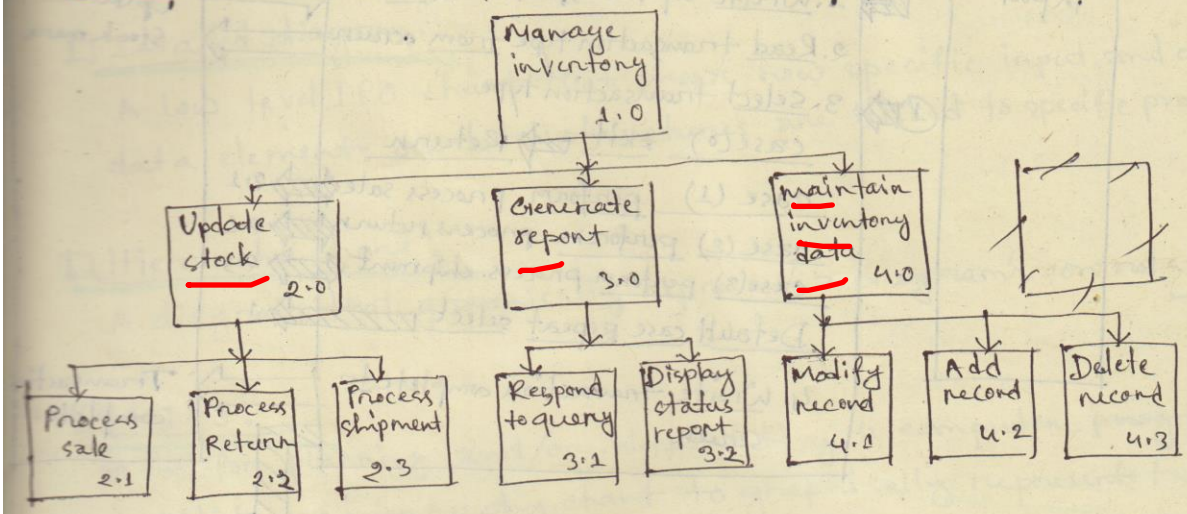
2.6. Overview-level IPO diagrams for word processing system.

(iii)

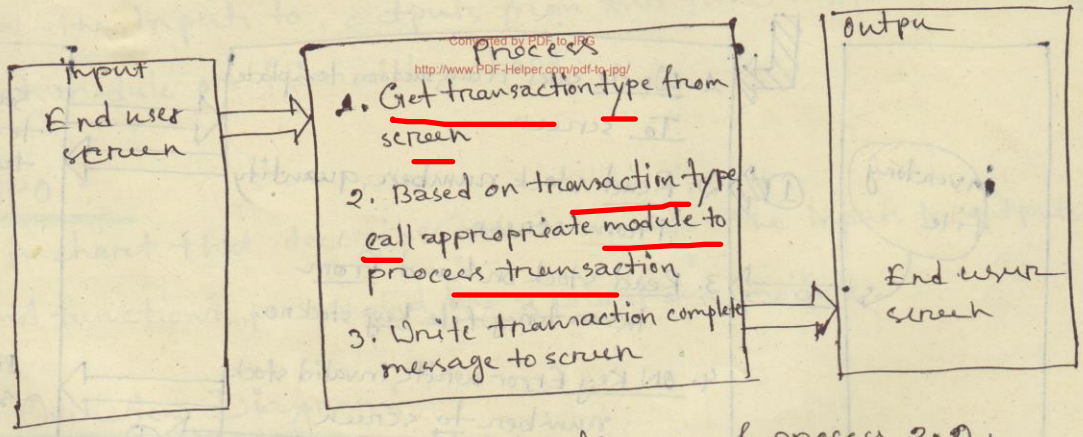


IPO diagram for change text function

An interactive Inventory Program:



hierarchy

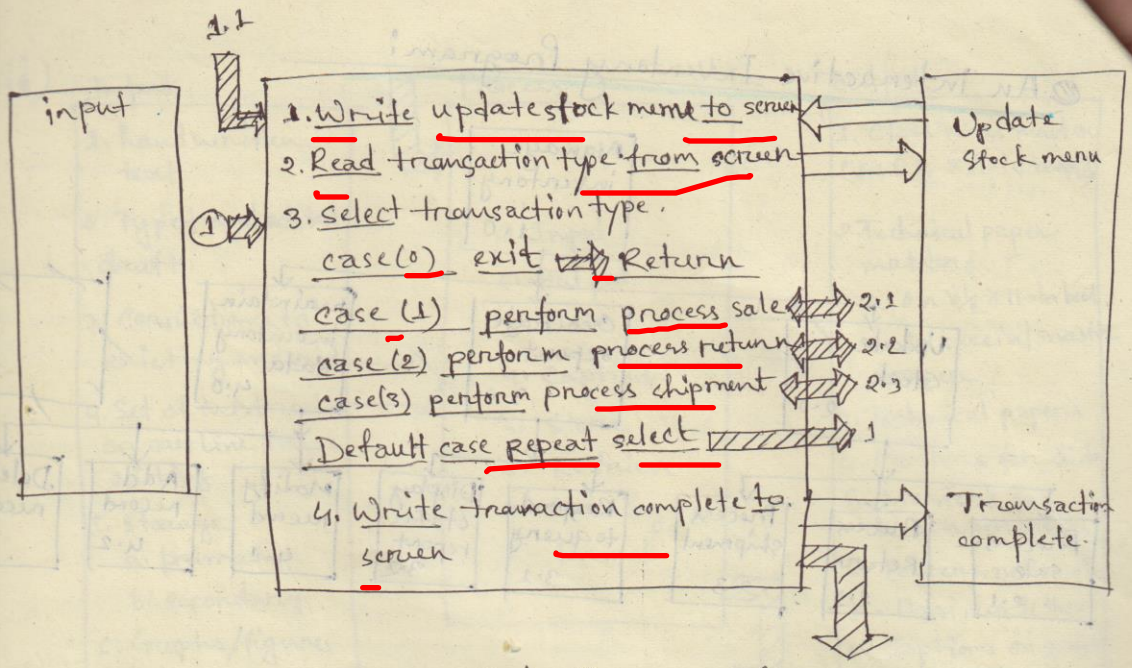


overview diagram of process 2.0.

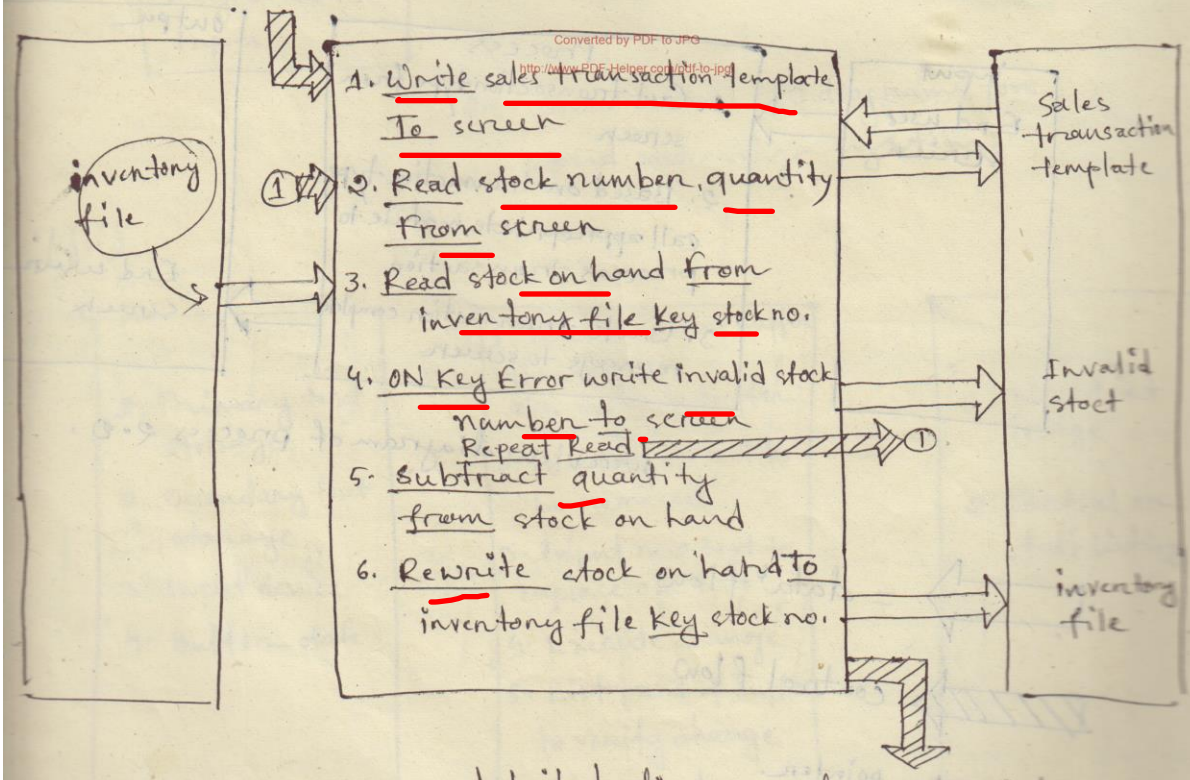
= data flow

control flow

pointer.



detailed diagram of process 2.0.



detailed diagram of process 2.1.

614.5 Key terms:

☐ Detailed diagram:

A low level IPO chart that shows how specific input and output data elements or data structures are linked to specific processes

☐ Hierarchy chart:

A diagram that graphically represents a program's control structure.

☐ HIPO:

A tool for planning and/or documenting a computer program that utilizes a hierarchy chart to graphically represent the programs control structure and a set of IPO charts to describe the inputs to, outputs from and functions performed by each module on hierarchy chart

☐ IPO:

A chart that describes or documents the inputs to, outputs from and functions performed by a program module.

☐ Overview Diagram:

A high-level IPO ~~diagram~~ chart that summarizes the inputs to, outputs from, processes or task performed by etc. by a program module.

☐ Visual Table of Contents (VTOC) —

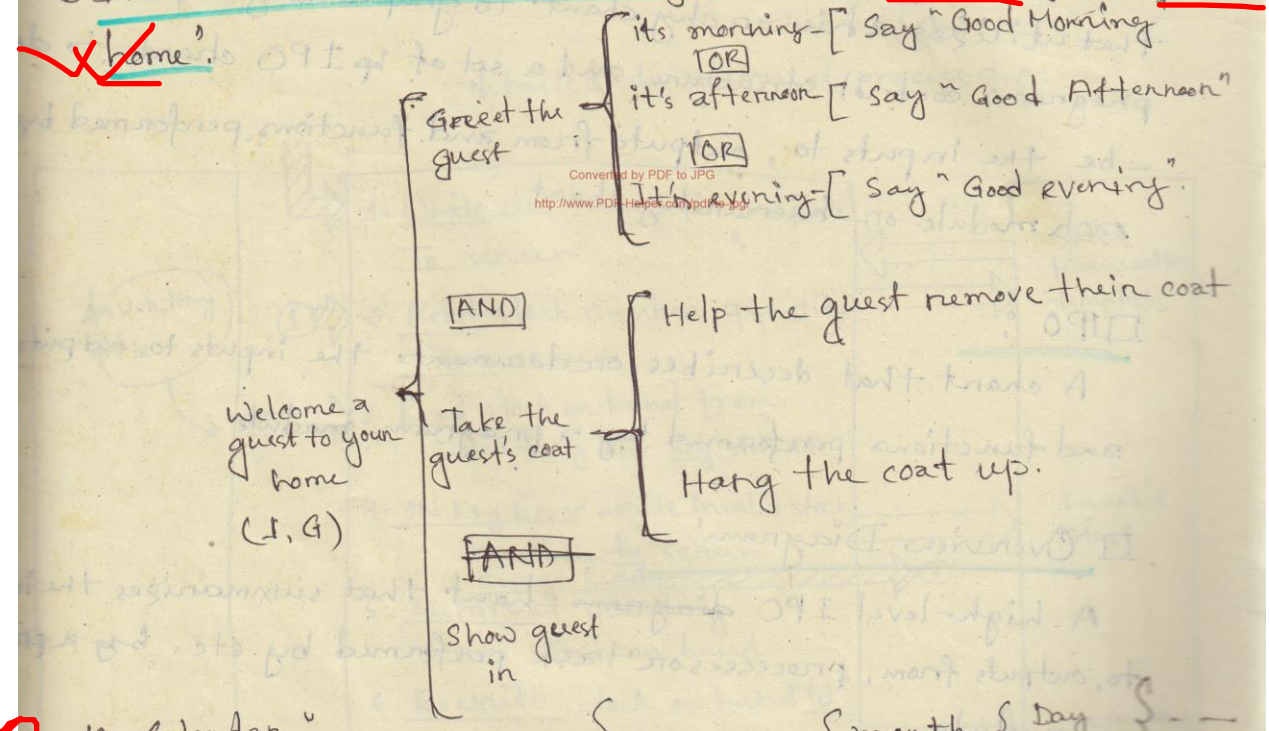
A more formal name for a hierarchy charts.

Warnier-Orr diagram:

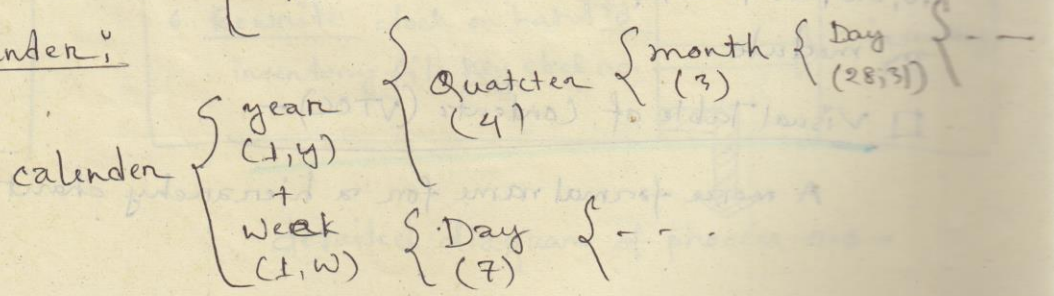
Warnier-Orr diagram is a style of diagram which is extremely useful for describing complex processes like computer programs, business processes, objects, etc.

It is
- elegant
- easy to understand
- easy to create.

Draw the Warnier-Orr diagram to "Welcome a guest to your home".



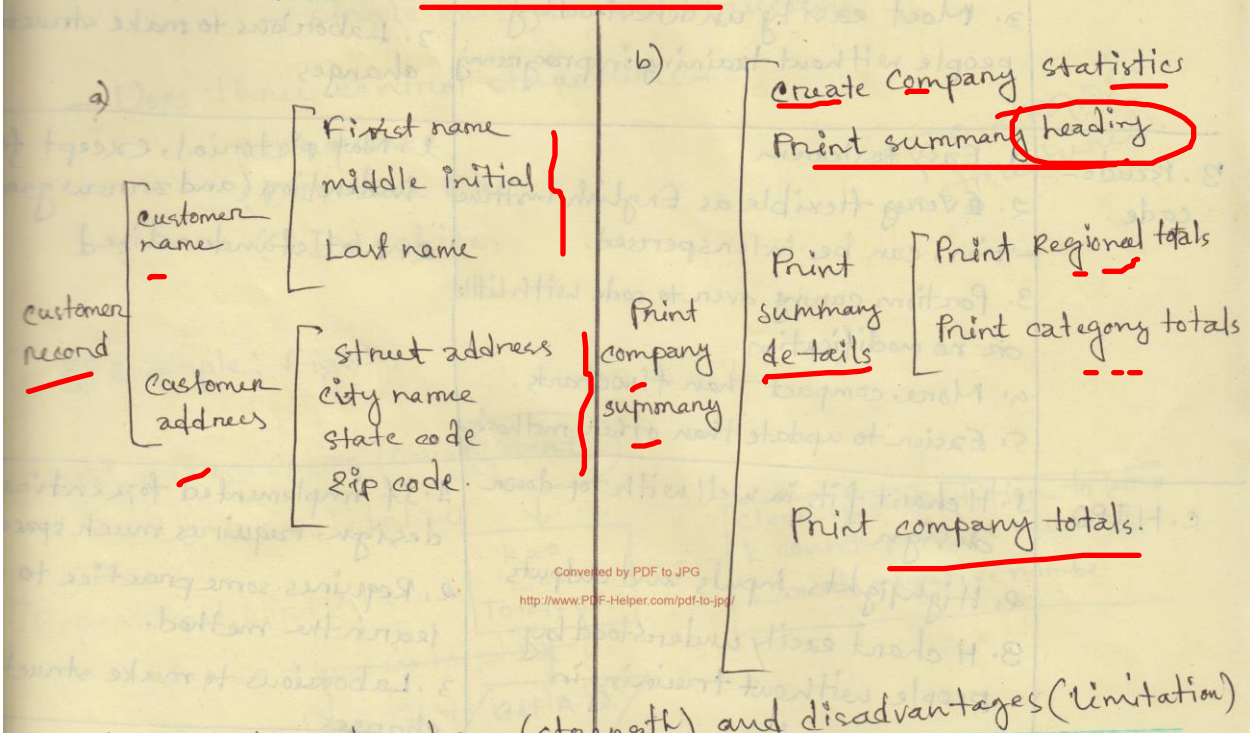
Calender;



● Using a Warnier/Orr diagram show:

(a) a data structure

(b) a processing structure



● What are the advantages (strength) and disadvantages (limitation) of Warnier-ORR diagram.

☐ Advantages:

- Easy to learn
- Standardized
- Slightly compact than pseudo-code

☐ Disadvantage:

- Not pictorial
- Require more practice.

● To make a phone call:

To make a phone call
- find a phone
- pick up the handset
- Dial the number

Technique	Advantages	Disadvantages
1. Flowcharts	1. Universally known 2. Pictorial representation 3. Most easily understood by people without training in programming	1. Require much space 2. Clumsy for depicting sub-routines, interrupts. 3. Laborious to make structural changes
2. Pseudo-code.	1. Easy to learn 2. Very flexible as English instructions can be interspersed. 3. Portions carry over to code with little or no modification. 4. More compact than flowcharts. 5. Easier to update than other methods.	1. Not pictorial, except for indention (and arrow grouping) 2. Not standardized
3. HIPO	1. H chart fits in well with top-down design 2. Highlights inputs and outputs. 3. H chart easily understood by people without training in programming	1. If implemented for entire design, requires much space. 2. Requires some practice to learn the method. 3. Laborious to make structural changes
4. Warnier Orr	1. Easy to learn 2. Standardized 3. Slightly more compact than Pseudo-code.	1. Not pictorial, except for indentation (and bracket groupings) 2. Requires more practice to learn than pseudo-code.

- Structured programming: characteristics
- What are rules of structured programming?

→ aims to provide a well-defined, clean, and simple approach to program design.

- easifer to understand, evaluate and modify.
- it requires the use of
  - single entry
  - single exit
  } control structure.
- Uses three control structures -
  (a) Sequence
  (b) If then Else
  (c) Do while.

GO TO ကြောင့် unstructured.

example: fig 2.12.

```
count : Proc Option (Main);
    K=0, k=0, Total=0
Repeat; Get List(A);
    If A=0 then Go to print;
    If A>0 then Go to Update;
    L=L+1;
    Go to Repeat;
Update: k=k+1
    Total=Total+A
    if Total >= 1000, then Go to Repeat;
Print: Put List(k,L,total);
End Counts
```

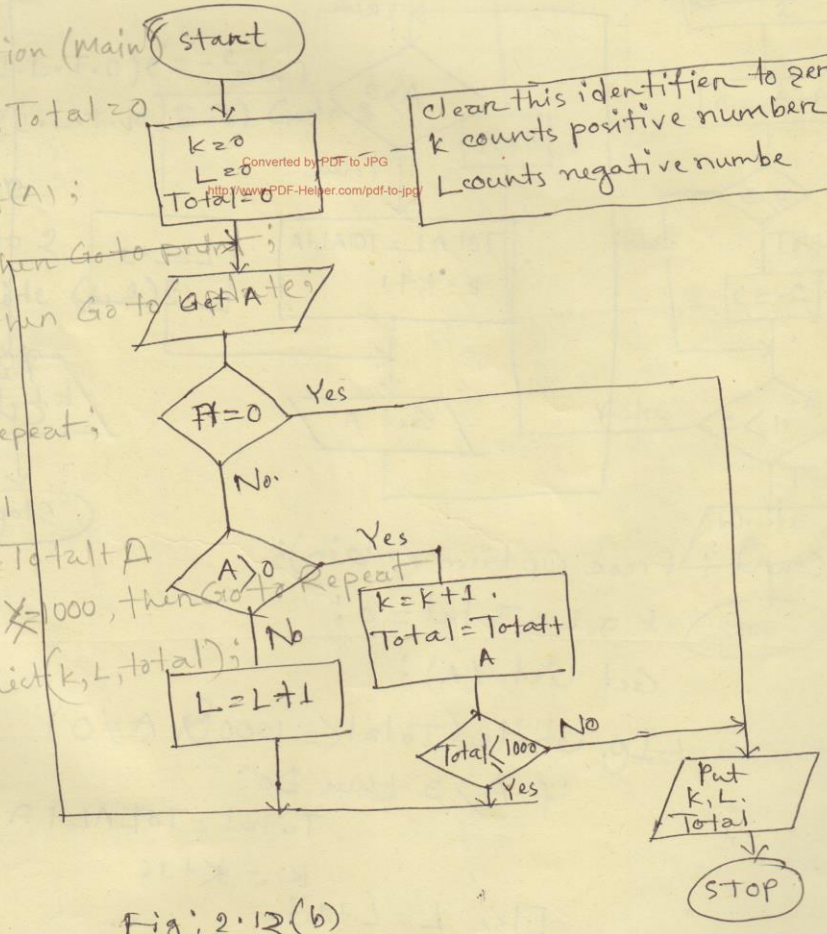
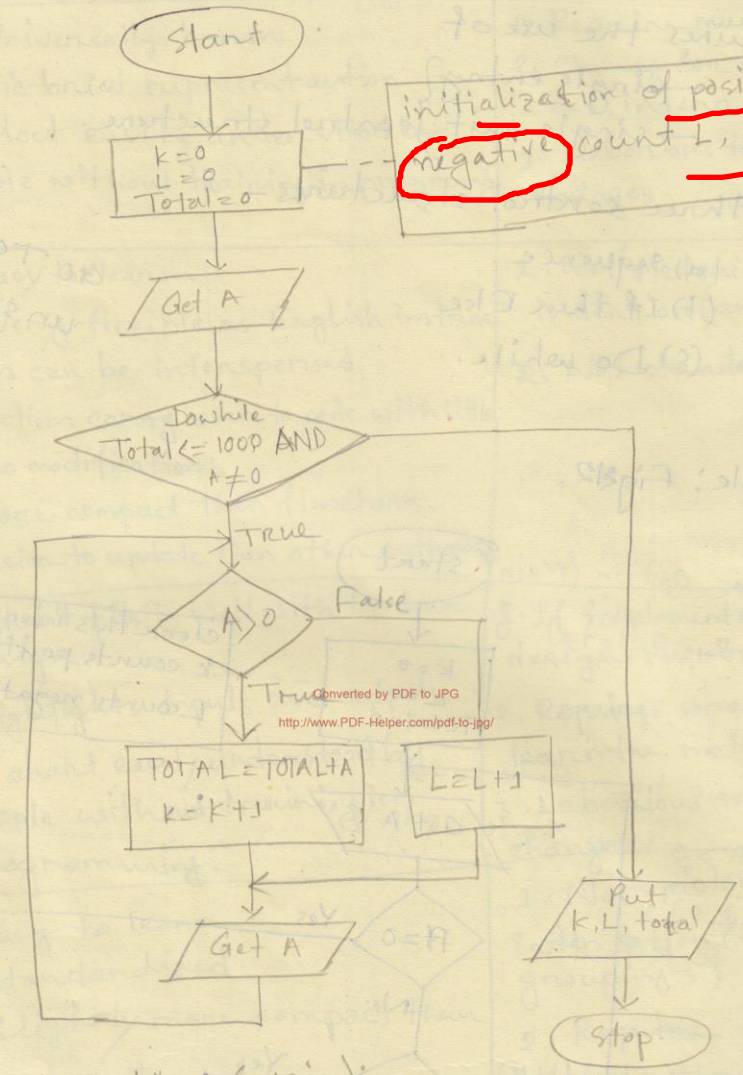


Fig: 2.12(b)

Fig: 2.13 (a) Structured

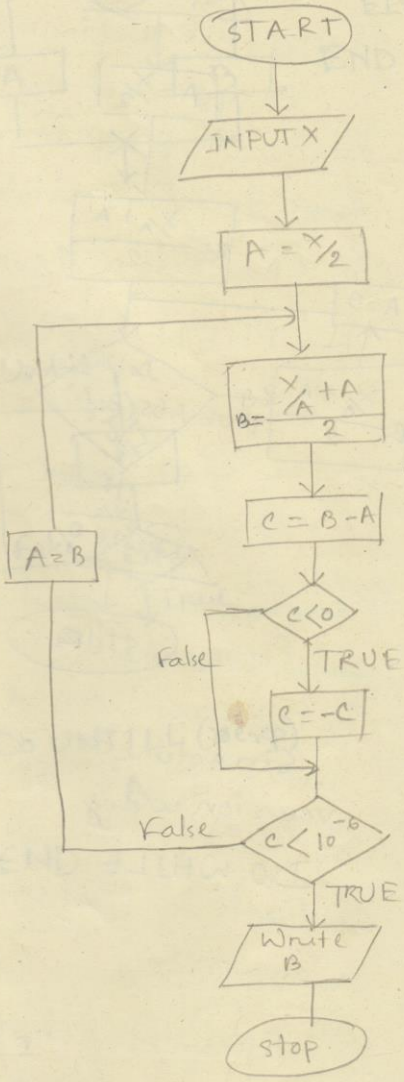



```
Count : Proc Optimi (main);
    K, L, Total = 0;
    Get list (A);
    Do while (Total<= 1000 & A≠0)
        if A>0 then Do:
                Total = TOTAL + A;
                K = K+1;
            Else L = L+1;
        Get list (A);
    End
Put list (K, L, total);
```

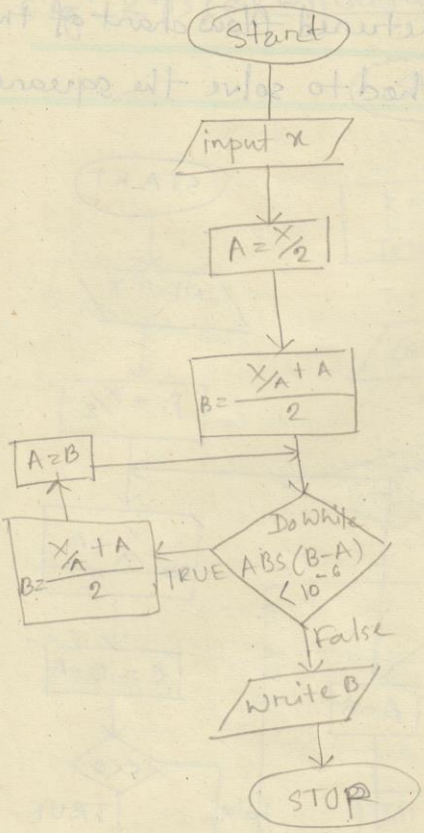
Draw the unstructured and structured flow chart of the program that uses Newton's method to solve the squeare root of number.

P' root(B) of a number(X):

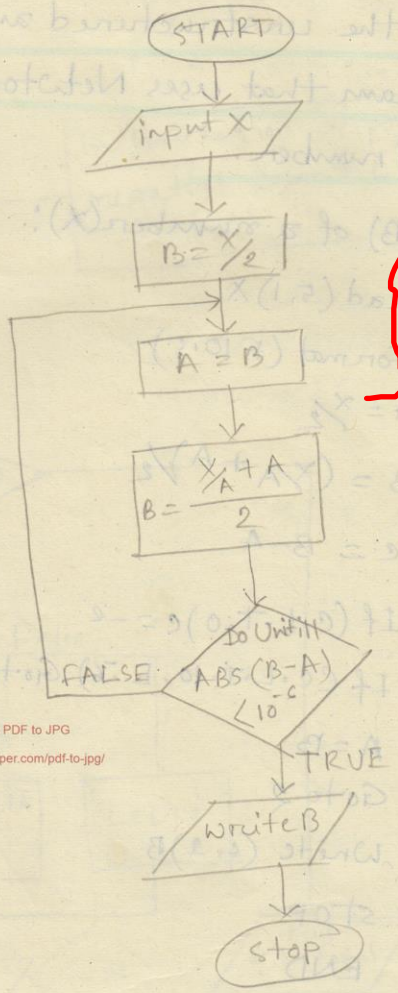
```
      Read (5,1) X
1.    Format (F10.5)
      A = X/2
2     B = (X/A + A)/2
      C = B - A
      If (C.L.T.0) C = -C
      If (C.L.T.10.E-6) Goto 3
      A = B
      Goto 2
3     Write (6,1) B
      STOP
      END
```

Unstructured

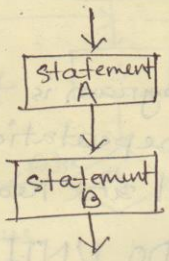


Structured version using DO WHILE



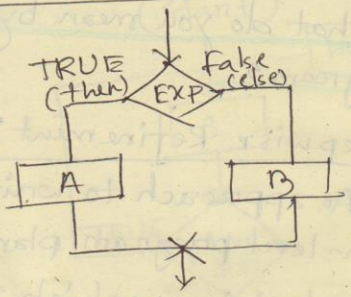
structured version using DO UNTILL

(A) Sequence:



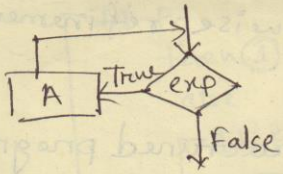
Statement A
:
statement B
:

(B) IF-THEN-ELSE:



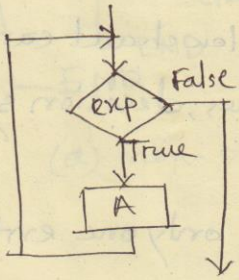
If (exp)
THENA
ELSE B
END

(C) DO WHILE

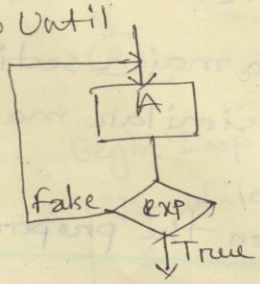


Do While(exp)
A
END

or

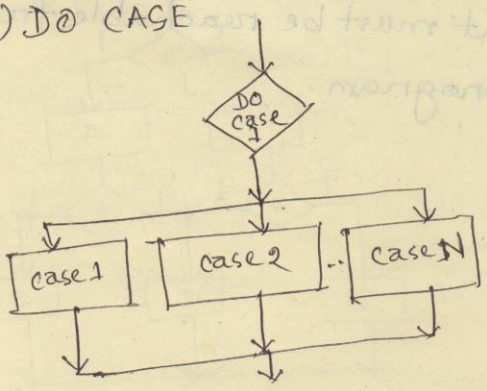


(D) Do Until



Do UNTILL (exp)
A
END

(E) DO CASE



Do CASE I
CASE 1
CASE 2
:
CASE N
END.

2.4.4 Techniques of Structured Programming

⑨ What do you mean by "stepwise refinement" of a structured program?

☐ Stepwise Refinement:

An approach to writing a structured program is to begin with high-level program plan where any loop repetitions and loops decisions which appear at this Level are realised in the design by using DO WHILE (plus DO UNTIL) and IF-THEN-ELSE (plus DO CASE).

Each major section of the design can then be expanded in the similar manner is called stepwise refinement.

⑩ Mention the properties of a proper structured program.

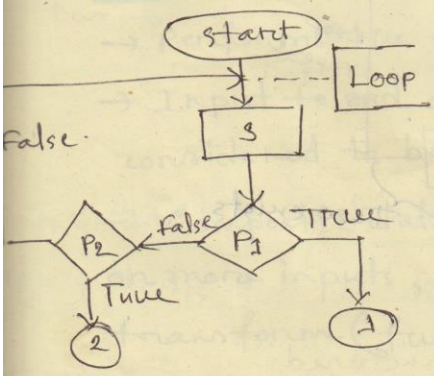
☐ Properties:

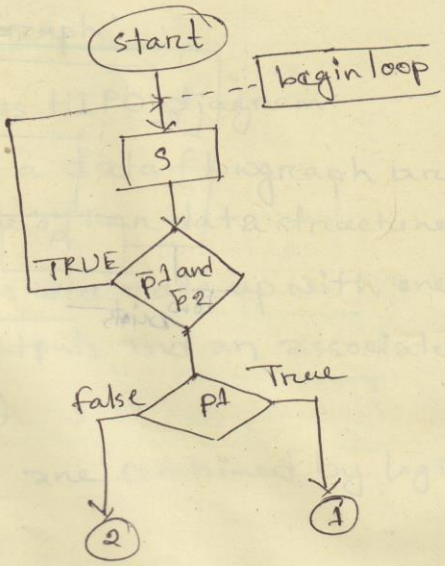
① Program must be such that a flowchart can be drawn using only processing blocks, flow lines, decision symbols, and merges.

② Each control structure must have only one entry and one exit.

③ Each flowchart element must be reachable from the starting point of the program.

⑲ Transformation:



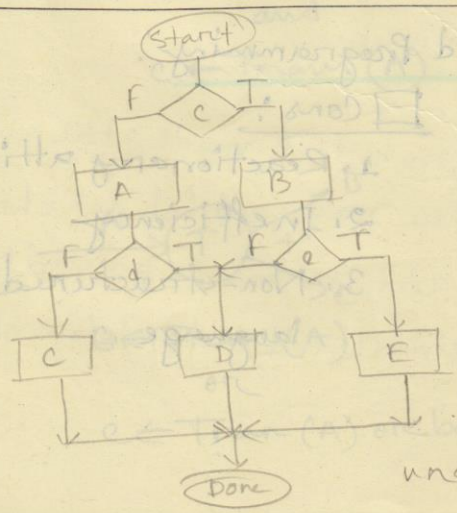


Loop!
S
If p1
then (1)
else
IF P2
Then (2)
else go to loop
END
END

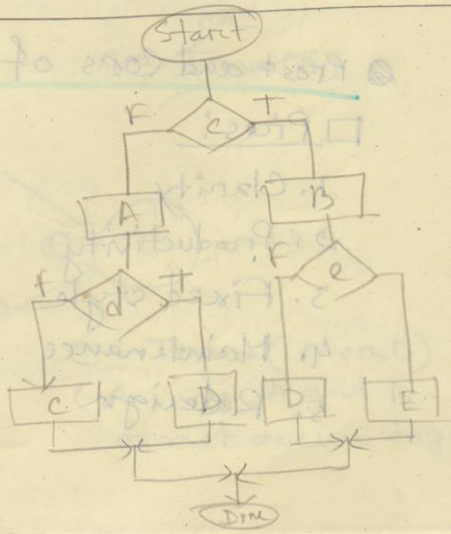
(a) non-structured

Begin Loop
Do while (p̄1 and p̄2)
S
END
IF p1
Then (1)
else (2)
END

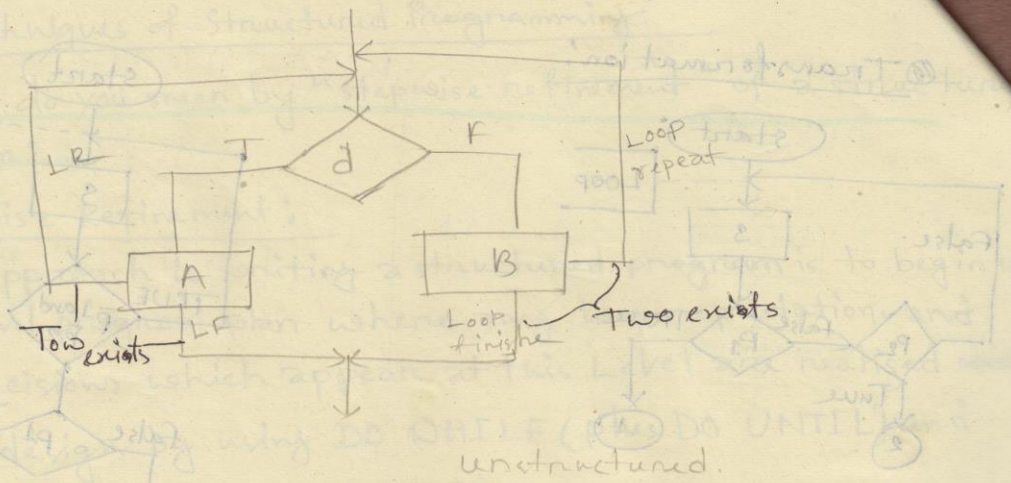
(b) structured equivalent

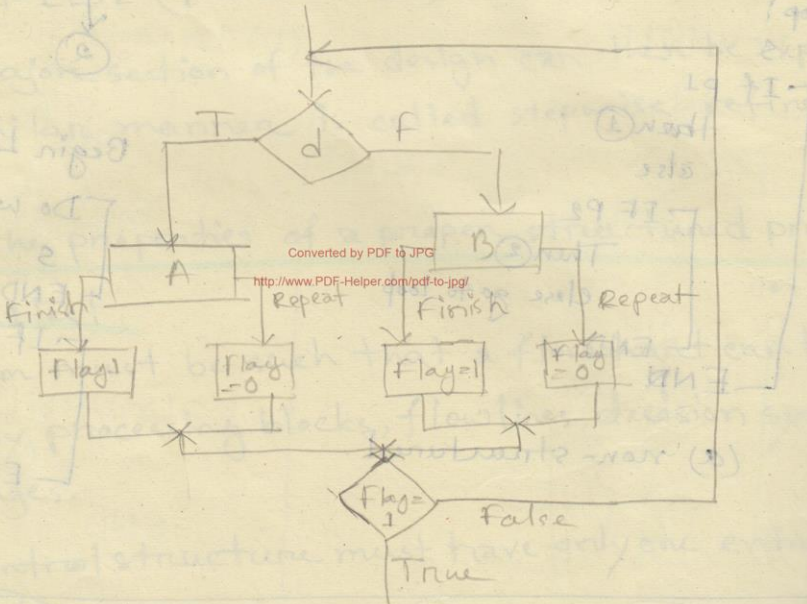


≃



unstructured





Pros and Cons of structured Programming:

Pros:

1. Clarity
2. Productivity
3. Fixed style
4. Maintenance
5. Redesign

Cons:

1. Reactionary attitude
2. Inefficiency
3. Non-structured language.

Data Flow Diagram / Data Flow Graph:

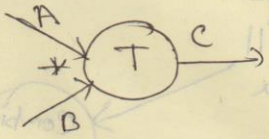
→ Performs the same function as HIPO diagram.

→ Input to and output from a data flow graph are considered to be a file or some other data structure

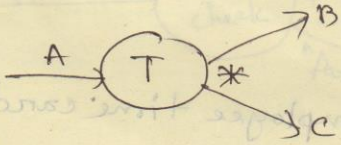
→ Composite data flow diagrams are made up with one or more inputs, one or more outputs and an associated transform (function, process)

→ More than one input or output are combined by logic relationship (AND, OR, etc.)

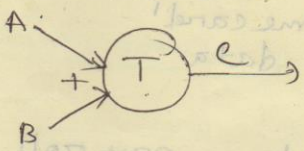
B ← Transform(A)



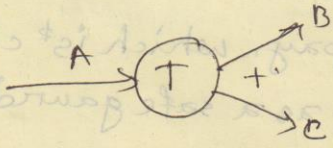
C ← T (A and B)



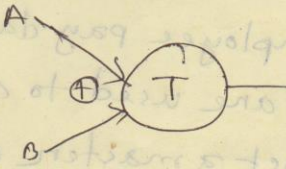
B ⇔ Transform(A)
and
C ← Trans (A)



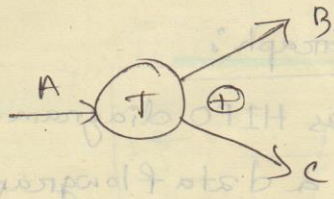
C ← Tran (A) or B or both
or
C ← Tran (B)



B ← Tran (A)
or
C ← Tran (A) or both



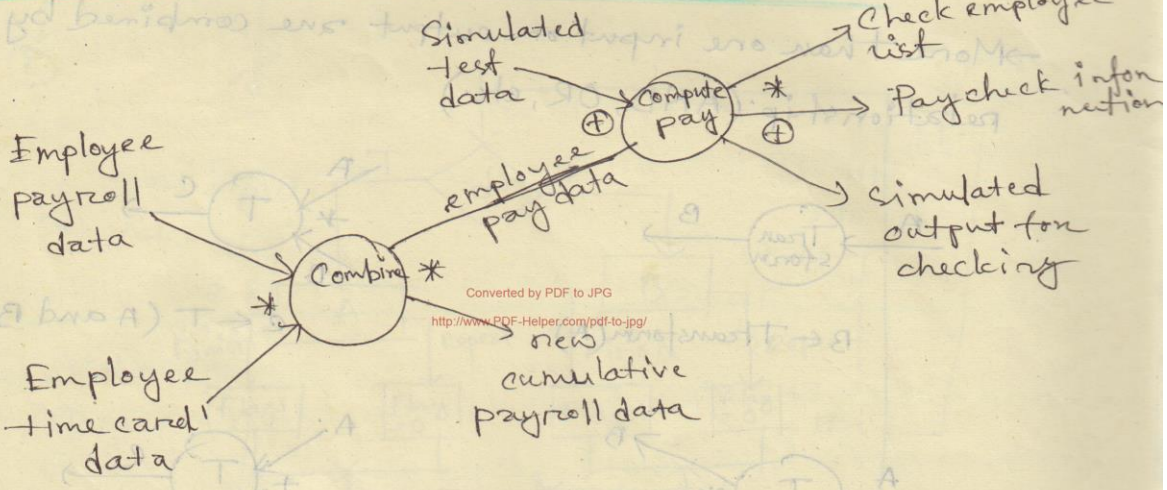
C ← Trans (A or B)
(Assume A and B
do not occur together)



B ← Trans (A) or
C ← Trans (A)
(Assume B and C do not appear together)

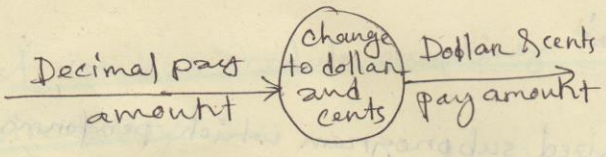
10

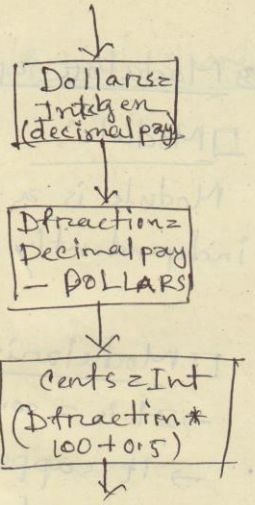
⑥ Draw the data flow graph for two typical functions namely "combine" and "compute pay" in a payroll program. Give verbal description of this flow graph.

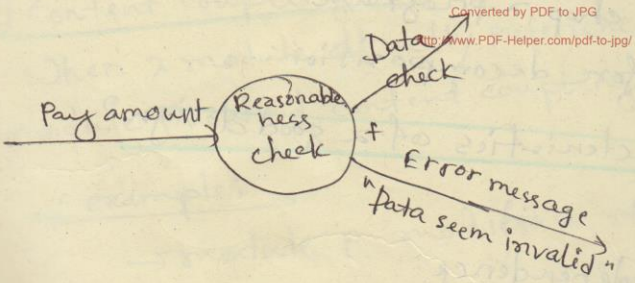


Employee payroll data and employee time card data are combined to yeild both the employee pay and a new new cumulative payroll number.

Employee pay data (or, in a test mode, the simulated data) are used to compute the pay, which is checked against a master employee list, as a safe gaurd and paycheck information (or simulated output for checking) is produced.







Modular approach:

Module:

Module is a modest-sized subprogram which performs independently one specific function.

Modularity:

→ It is a concept closely tied to abstraction.
→ it supports independence of modules.
→ supports abstraction in software.
→ Supports hierarchical structuring of programs.
→ It enhances design clarity, eases implementation.
→ Reduces cost of testing, debugging and maintenance.
→ Cannot simply chop a program into modules, need some some criteria for decomposition.

What are the characteristics of a good design?

Characteristics:

⇒ Component independence.
  → High cohesion
  → Low coupling.
⇒ exception identification and handling
⇒ fault-prevention and fault tolerance.

Coupling:

Degree of dependencies among components.

Q. List various module coupling in order of their decreasing desirability.

1. No direct coupling (best)
2. Data coupling
3. Stamp coupling
4. Control coupling
5. External coupling
6. Common coupling
7. Content coupling (worst)

☐ Content Coupling:

When a module directly references the content of another module is called content coupling.

example:

→ module P modifies a statement in module Q.
→ module P refers a local data of module Q.
(in terms of numerical displacement)
→ Module P branches to a local label of module Q

☐ Common Coupling: (Global coupling)

Common coupling occurs when two or more modules references the same global data structure.

- Modules exchange data using a global data block. (instead of arguments)
- single module with write access where all other modules have read access is not common coupling.

Control Coupling:

Control coupling occurs when one module passes switch, flag or module name i.e. control parameters to another.

- May be either good or bad depending on the situation.
  - → Bad, when component must be aware of internal structure and logic of another module
  - → good, if parameters allow factoring and reuse of functionality.

External Coupling:

External coupling is possible in some languages where we can control the variable coupling and limit it to those variables which are formally declared as external.

Stamp Coupling:

component passes a data structure to another component that does not have access to the entire structure.

- Requires a second component to know how to manipulate the data structure

example:

II Data Coupling:

Two components are data coupled if there are homogeneous data items.

- Every elements arguments is simple argument or data structure in which all elements are used.
- Good if it can be achieved.
- Easy to write contracts for this and modify component independently

- Key Idea in Object-Oriented programming:
  → object oriented designs tend to have low coupling.

- Differentiate between content and common coupling with proper examples.

Cohesion:

Degree to which all elements of a components are directed towards a single task and all elements directed towards that task are contained in a single component.

- Cohesion of a unit, of a module, of an object, or a component addresses the attributes of "degree of relatedness within that unit, module, object or component.
- Internal glues with which component is constructed.
- All elements in a component are directed towards and essential for performing the same task.
- High is good.

List various module cohesion in order of their decreasing desirability!

1. Functional (best)
2. Informational
3. Sequential
4. Communicational /
5. Procedural
6. Classical/temporal
7. Logical
8. Coincidental (worst)

☐ Coincidental cohesion:

Parts of the component performs multiple completely unrelated actions.

- May be based on the factors outside of the design.
  - → skill set or interest of developers.
  - → avoidance of small modules.
- No reusibility
- Difficult corrective maintenance or enhancement.
- Elements needed to achieve some functionality are scattered through out the system.
- Accidental worst form.

example: a "Utilities" class.

☐ Logical Cohesion:

Element of components are related logically and not functionally.

- Several logically related elements are in the same component and one of the elements is selected by caller.
- May include both high and low level actions in the same class.
- May include unused parameters for certain uses.
- Interface is difficult to understand.

example:
grouping all mouse and keyboard input handling routines.

☐ Temporal Cohesion:

Elements of a component are related by timing.

- Difficult to change because you may have to look at numerous components when a change in a data structure is made.
- Increases chances of regression of fault.
- Component unlikely to be reusable.
- Often happens in initialization or shutdown.

example:

a function which is called after catching an exception which closes open files, creates an error log and notifies the user.

☐ Procedural Cohesion:

Elements of a component are related only to ensure a particular order of execution.

- Actions are weakly connected and unlikely to be reusable.
- Changes to the order of steps or purpose of steps requires changing the module abstraction.

example:

a function that checks file permissions and then open the file.

☐ Communicational Cohesion:

Module performs a series of actions related by a sequence of steps to be followed by the product and all actions are performed on the same data.

- Actions based on ordering of steps on all the same data.
- Actions are related ~~and~~ but still not completely separated.
- Module cannot be reused.

☐ Sequential Cohesion:

- Methods are together in a class because the output from one part is the input to ~~another~~ part.

- Output of one component is input to another.
- Occurs naturally in functional programming languages.
- Good situation

example:

function that reads data from a file and process the data.

☐ Informational Cohesion:

Module performs a number of actions, each with its own entry point, with independent code for each action, all performed on the same data.

Ø Different from logical cohesion

→ Each piece of code has single entry and single exit.

→ In logical cohesion, actions of module intermingled.

O ADT and object-oriented paradigm promote.

[1] Functional Cohesion:

Every essential element to a single computation is contained in the component.

- Every element in the component is essential to the computation.
- Ideal situation.

example:
tokenizing a string of XML.

② Why functional cohesion is the best?

③ Differentiate between coupling and cohesion of a module?

- coupling is "degree of inter dependency" where cohesion is "degree of relativeness"
- coupling performs minimization of external interaction where cohesion performs maximization of internal interaction.

⑦ Desined Class / Object Interaction:

- Maximize internal interaction (cohesion)
  - → Easier to understand.
  - → Easier to test.
- Minimize external interaction (coupling)
  - → can be used independently
  - → Easier to test
  - → Easier to replace.
  - → Easier to understand.

⑧ What do you mean by automatic and defensive programming

☐ Automatic programming:

Automatic programming is the concept of writing one program that can itself write another program.

☐ Defensive Programming:

If programs can be written without errors, it makes little sense to build checks in the software. Conversely if one believes that there will always be residual software errors, then built-in errors checking is an important strategy.

These strategy is called defensive programming,

- A passive technique checks information at an appropriate point in a program when the checking code is reached.
- An active technique searches throughout the program or data base periodically or during slack periods looking for unusual conditions.