MENOUFIA UNIVERSITY
FACULTY OF COMPUTERS AND INFORMATION

First Year (First Semester)
CS Dept., (CS131)

PRINCIPLES OF PROGRAMMING

Lecture Three

Dr. Hamdy M. Mousa

Computer Programming

Computer Programming

- In this lecture, you will learn about:
  - Declaring and Using Variables
  - Arithmetic Operations
  - Programming and user environments
  - **C++ Program**
  - **Structured Flowcharts**

اعلم دائماً ان غداً افضل

Always know that tomorrow is better

First Program in C++

- ***Printing a Line of Text***

```
1 // Fig. 2.1: fig02_01.cpp
2 // Text-printing program.
3 #include <iostream> // allows program to output data to the screen
4
5 // function main begins program execution
6 int main()
7 {
8 std::cout << "Welcome to C++!"; // display message
9
10 return 0; // indicate that program ended successfully
11
12 } // end function main
```

- Output:

Welcome to C++!

Escape sequence Description

Escape sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
\a	Alert. Sound the system bell.
\\	Backslash. Used to print a backslash character.
\'	Single quote. Use to print a single quote character.
\"	Double quote. Used to print a double quote character.

Printing a line of text with multiple statements.

```
1 // Fig. 2.1: fig02_01.cpp
2 // Text-printing program.
3 #include <iostream> // allows program to output data to the screen
4
5 // function main begins program execution
6 int main()
7 {
8 std::cout << "Welcome ";
9 std::cout << "to C++!"; // display message
10
11 return 0; // indicate that program ended successfully
12
13 } // end function main
```

Output:

Welcome to C++!

Printing multiple lines of text with a single statement.

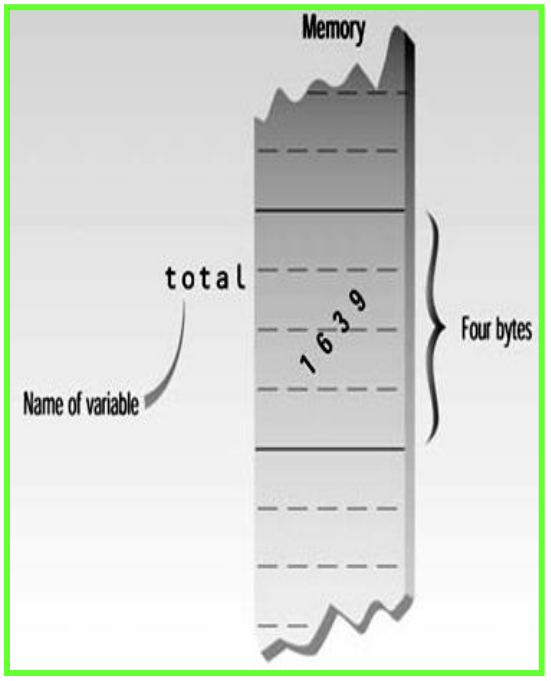
```
1 // Fig. 2.1: fig02_01.cpp
2 // Text-printing program.
3 #include <iostream> // allows program to output data to the screen
4
5 // function main begins program execution
6 int main()
7 {
8 std::cout << "Welcome \nto\n\n C++!\n"; // display message
9
10 return 0; // indicate that program ended successfully
11
12 } // end function main
```

Output:

```
Welcome
 to

 C++!
```

Variables

- Variables are the most fundamental part of any language.
  - A variable has a symbolic name and can be given a variety of values.
  - Variables are located in particular places in the computer's memory.
  - When a variable is given a value, that value is actually placed in the memory space assigned to the variable.

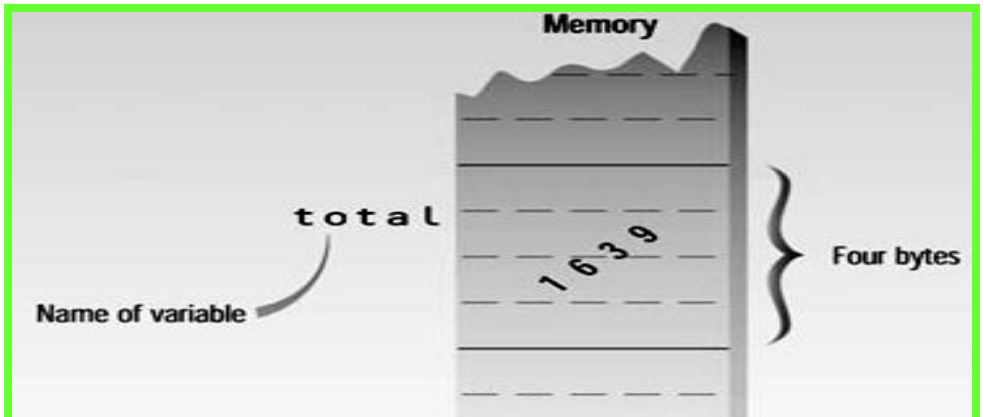


Declarations

```
// intvars.cpp
// demonstrates integer variables
#include <iostream>
using namespace std;
int main()
{
    int var1;          //define var1
    int var2;         //define var2
    var1 = 20;      //assign value to var1
    var2 = var1 + 10;                 //assign value to var2
    cout << “var1+10 is “;      //output text
    cout << var2 << endl;       //output value of var2
 return 0;
}
```

Basic C++ Variable Types

Keyword	Numerical Range Low	Numerical Range High	Digits of Precision	Bytes of Memory
`bool`	false	true	n/a	1
`char`	–128	127	n/a	1
`short`	–32,768	32,767	n/a	2
`int`	–2,147,483,648	2,147,483,647	n/a	4
`long`	–2,147,483,648	2,147,483,647	n/a	4
`float`	3.4×10^{-38}	3.4×10^{38}	7	4
`double`	1.7×10^{-308}	1.7×10^{308}	15	8



Unsigned Integer Types

Keyword	*Numerical Range* *Low*	*High*	*Bytes of Memory*
`unsigned char`	0	255	1
`unsigned short`	0	65,535	2
`unsigned int`	0	4,294,967,295	4
`unsigned long`	0	4,294,967,295	4

- To change an integer type to an unsigned type, precede the data type keyword with the keyword unsigned. For example, an unsigned variable of type char would be defined as:

```
unsigned char ucharvar;
```

Structured Flowcharts

Structured Flowcharts

A **structure** is a basic unit of programming logic; each structure is a sequence, selection, or loop; with these can diagram any event.

Designing **structured programs** is a matter of dividing tasks into logically independent units (or modules) and then joining these units in a limited number of predefined ways.

Advantages of Structured programs

- **They are easier to understand**
  - The modules can be considered individually.
  - Need not comprehend an entire program before understanding its individual parts.
  - The modules can be arranged makes it easier to identify the structure

- **Structured programs are easier to maintain or change**, because the individual modules can be changed or corrected without changing the entire program.

Structured flowcharts

structured flowcharts must meet the <u>following additional conditions</u>:

1. Each part of the chart must be one of three to five **specified structures**.
2. Program flow must enter each part at exactly **one place** and must **leave** each part at exactly **one place**.

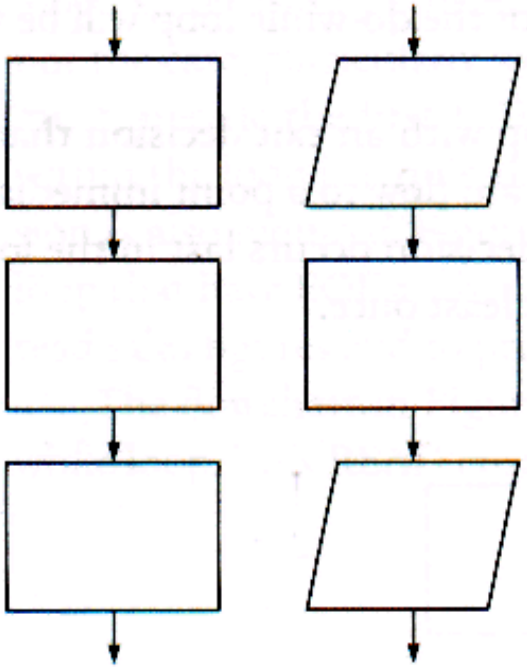
Structured Flowcharts

Structured flowcharts include one or more of:

1. Sequence.
2. If-then (or if-then-else).
3. Do-while loop.
4. Do-until loop.
5. Case.

Requirements for Structured Flowcharts

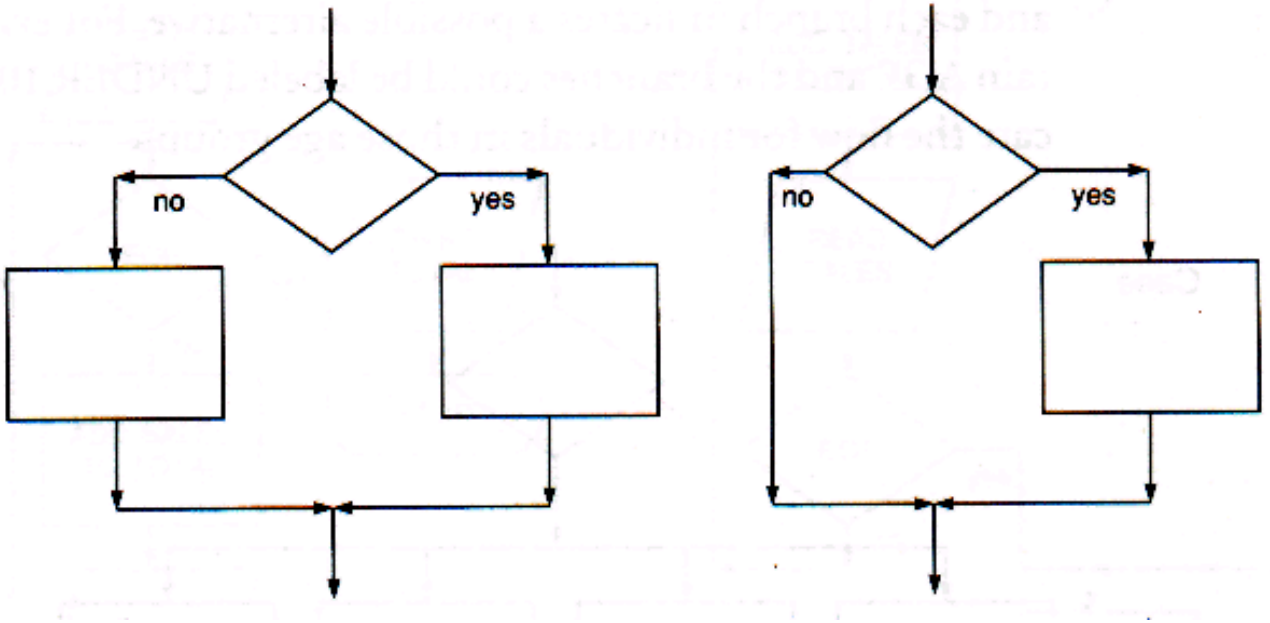
- **Sequence** represents any series of individual processes.



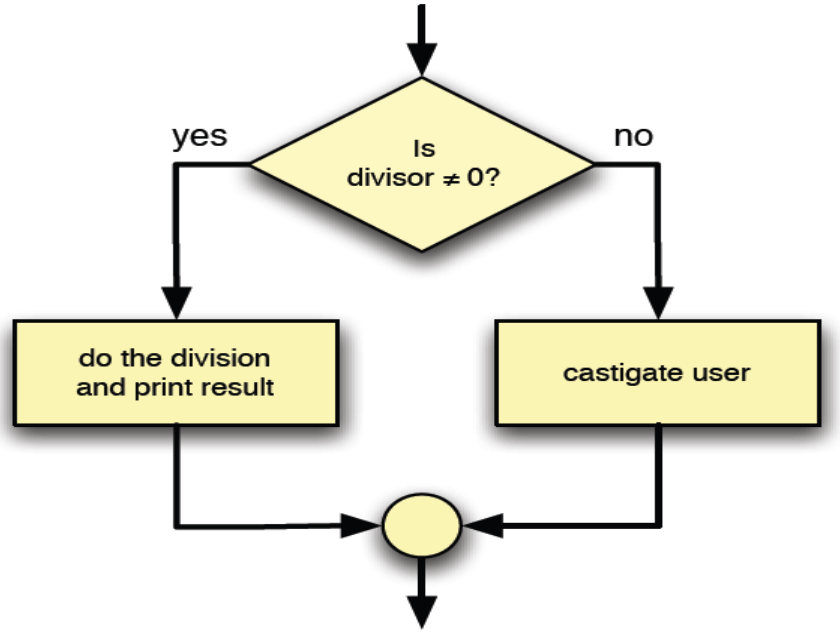
```
cout << "a = " << a << endl;
// Mix integers and Booleans
a = 1;
b = 1;
int x = a, y = true;
b = (x != 10 && y == 20);
cout << ", x = " << x << ", y = " << y <<
endl;
```

Requirements for Structured Flowcharts

- **If-then (or if-then-else)** represents a decision or selection.
- The important feature of if-then that distinguishes it from unstructured decisions is that flow lines from **both sides of the decision meet before program** flow passes from this structure.



Example



If divisor = 0 then
  Do the division and
  Print result
Else
  Castigate user
End if

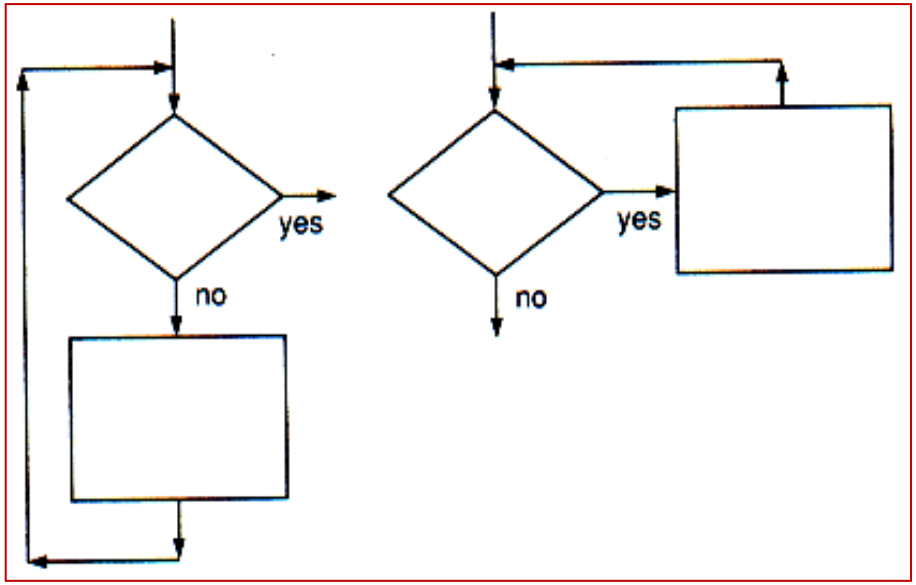
C++ Relational operators

Operator	Meaning
==	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to

```
    //  Get two integers from the user
    cout << "Please enter two integers to divide:";
    cin >> dividend >> divisor;
    //  If possible, divide them and report the result
    if (divisor != 0)
        cout << dividend << "/" << divisor << " = "
             << dividend/divisor << endl;
    else
        cout << "Division by zero is not allowed" << endl;
}
```

Requirements for Structured Flowcharts

- **Do-while loop** is a loop with an **exit** decision that occurs **before processing** within the loop.
- After processing, program flow **returns** to a point immediately **before** the exit decision.



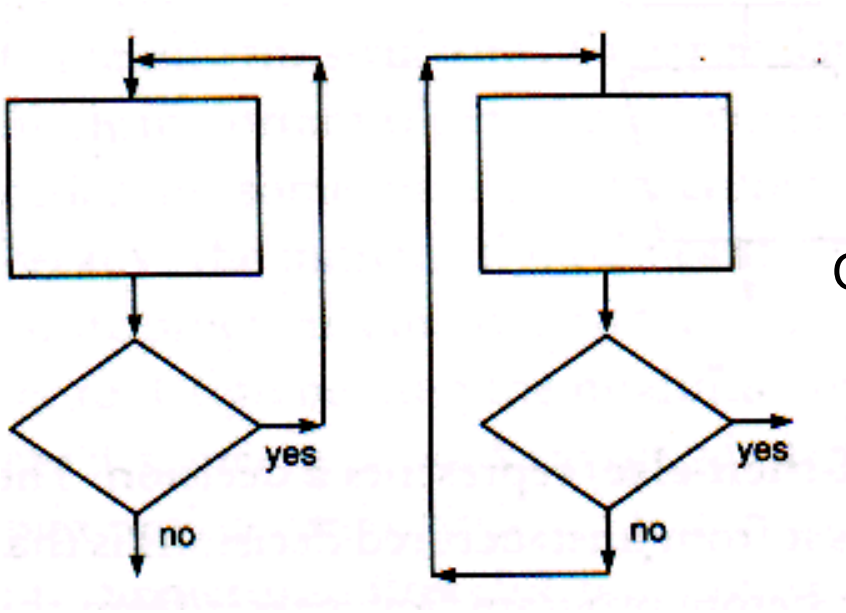

```
do PROCESS while (condition)
  {
      Statements
  }
   enddo
```

OR:

```
while (condition)
        Statements
endwhile
```

Requirements for Structured Flowcharts

- **Do-until loop** is a loop with an **exit decision** that occurs **after processing**.
- The do-until loop returns program flow to a point immediately before the first symbol in the loop.



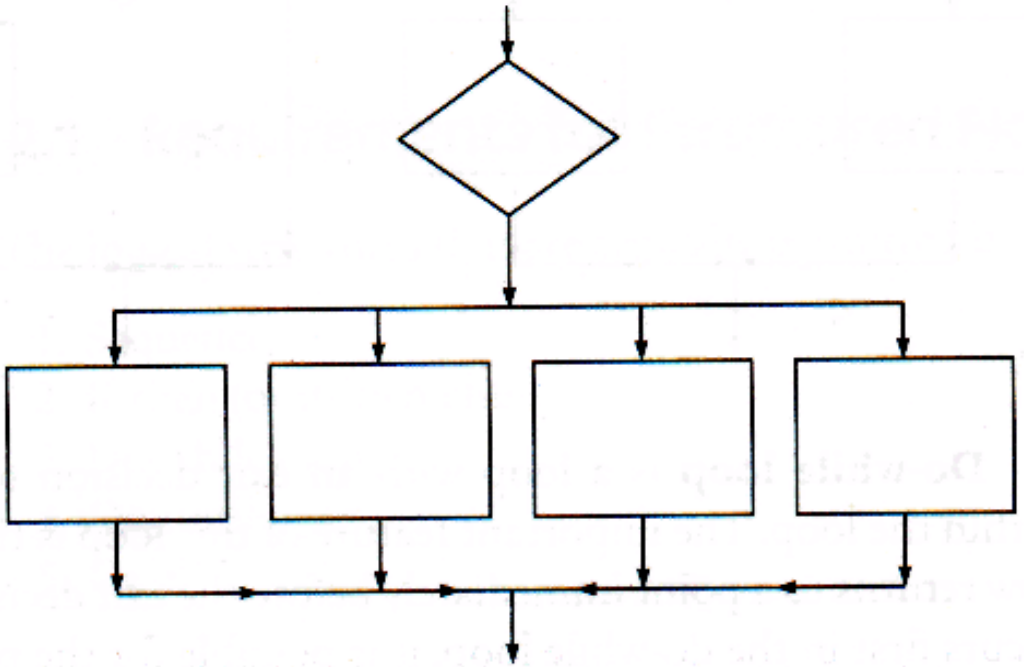

```
do PROCESS until (condition)
        Statement(s)
  enddo
```

OR:

```
do {
        Statement(s)
   } while (condition)
```

Requirements for Structured Flowcharts

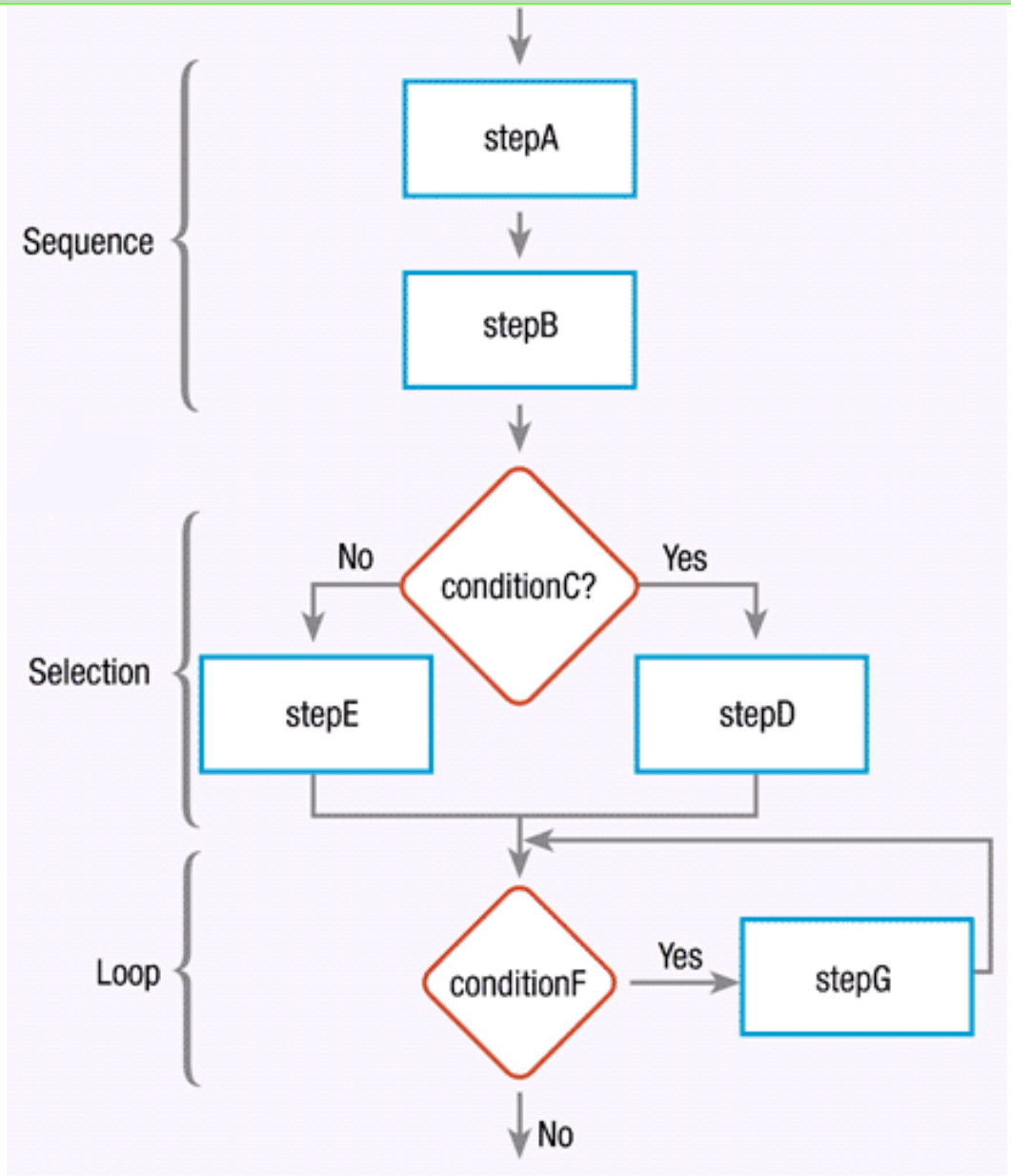
- **Case** represents a decision with more than two possible outcomes.



Structured Loops and the EOF Decision

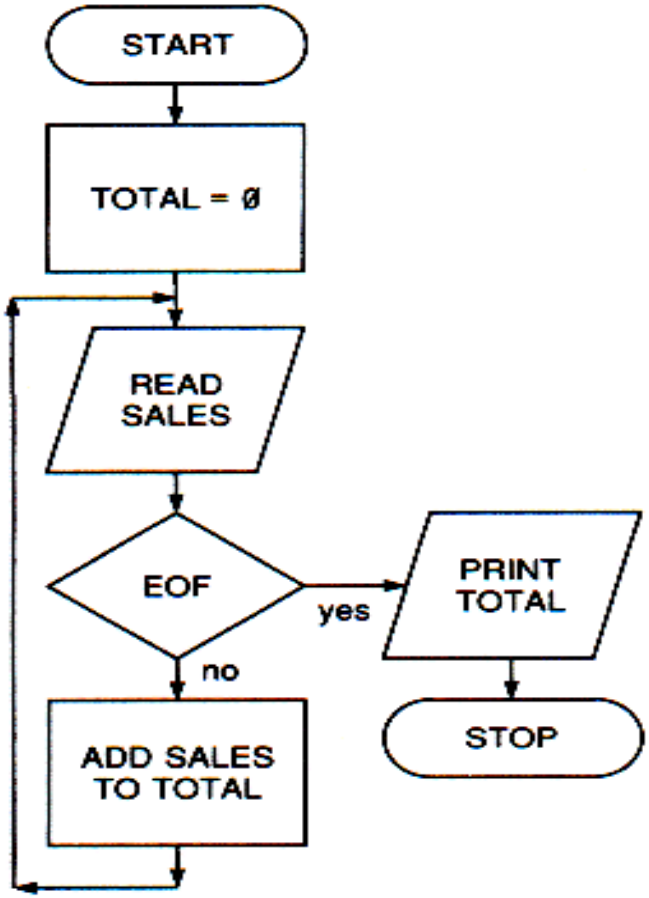
- **Loop structure** depends upon the placement of the ***loop exit decision***.
- ***Do-while*** loops require that loop exit decisions occur **first**.
- ***do-until*** loops require that they occur ***last***.
- Many unstructured loops can be redrawn as structured loops by *moving the loop exit decision to the first or last position* in the loop.

Structured Flowcharts

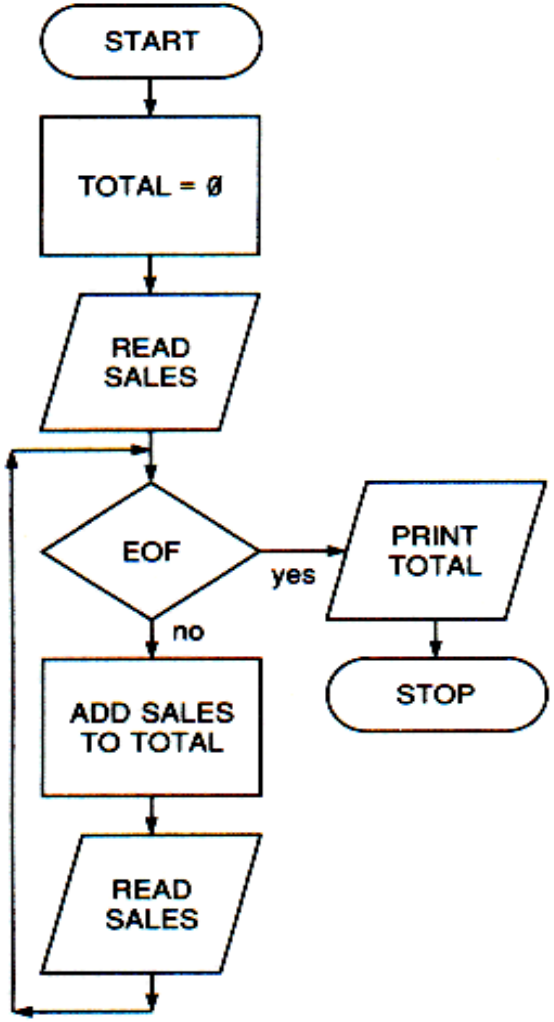


Priming Read

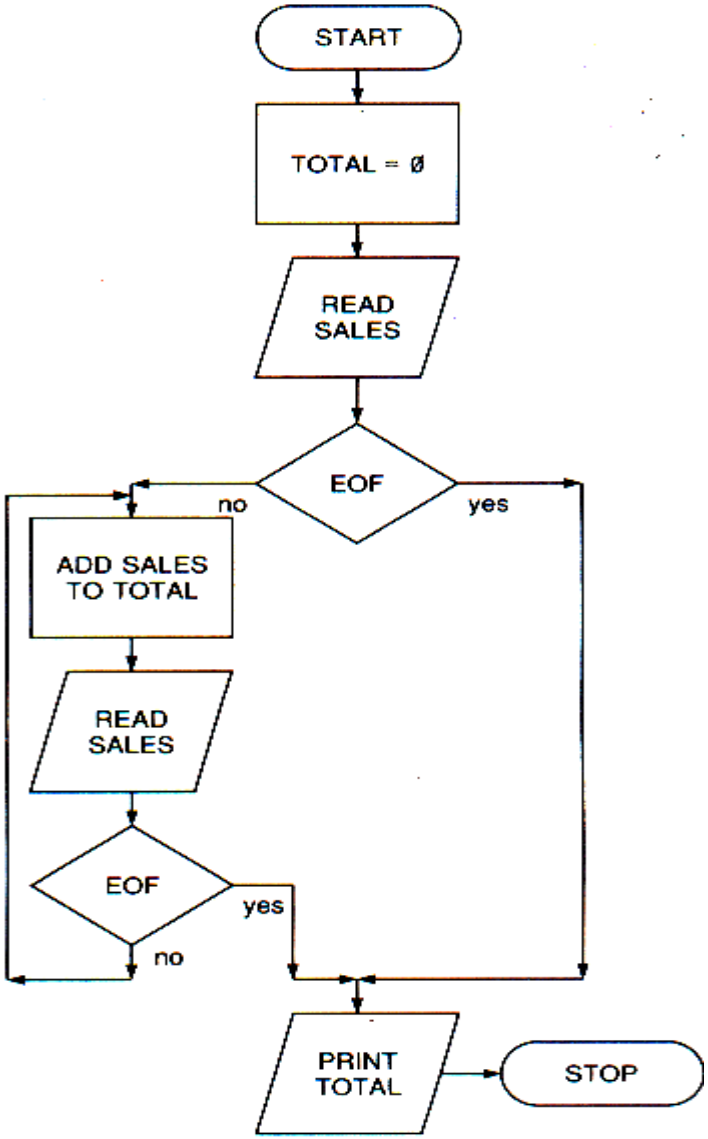
- **A priming read or priming input is the first read or data input statement in a program**
- **When EOF is the exit decision. Because a READ instruction must precede the first EOF decision,**
  - **a prime READ as well as a READ instruction within the loop is required.**



This flowchart reads SALES and prints a total sales figure. It contains a do-while loop.

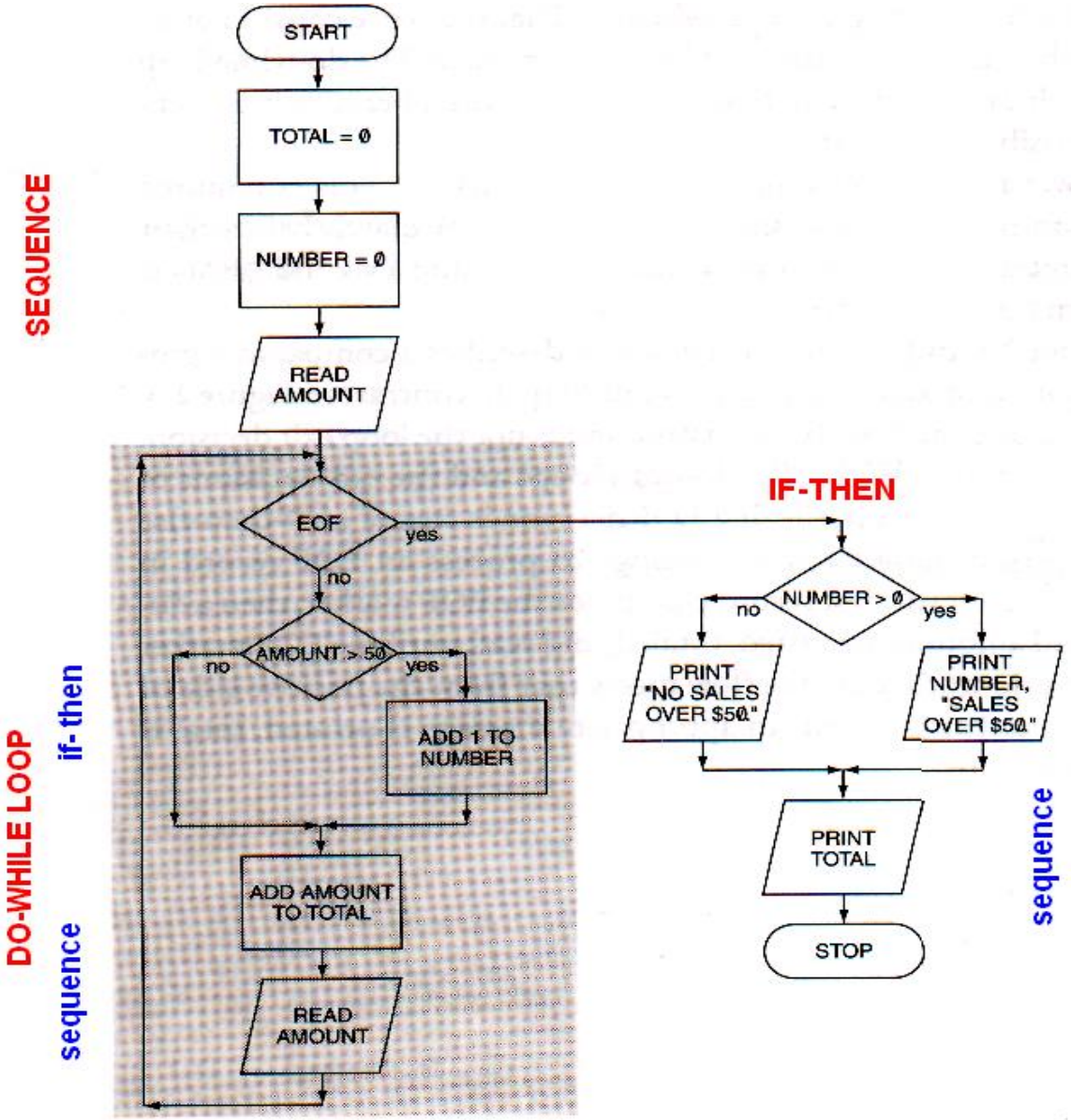


This flowchart uses a do-until loop to accomplish the same task as previous flowchart.



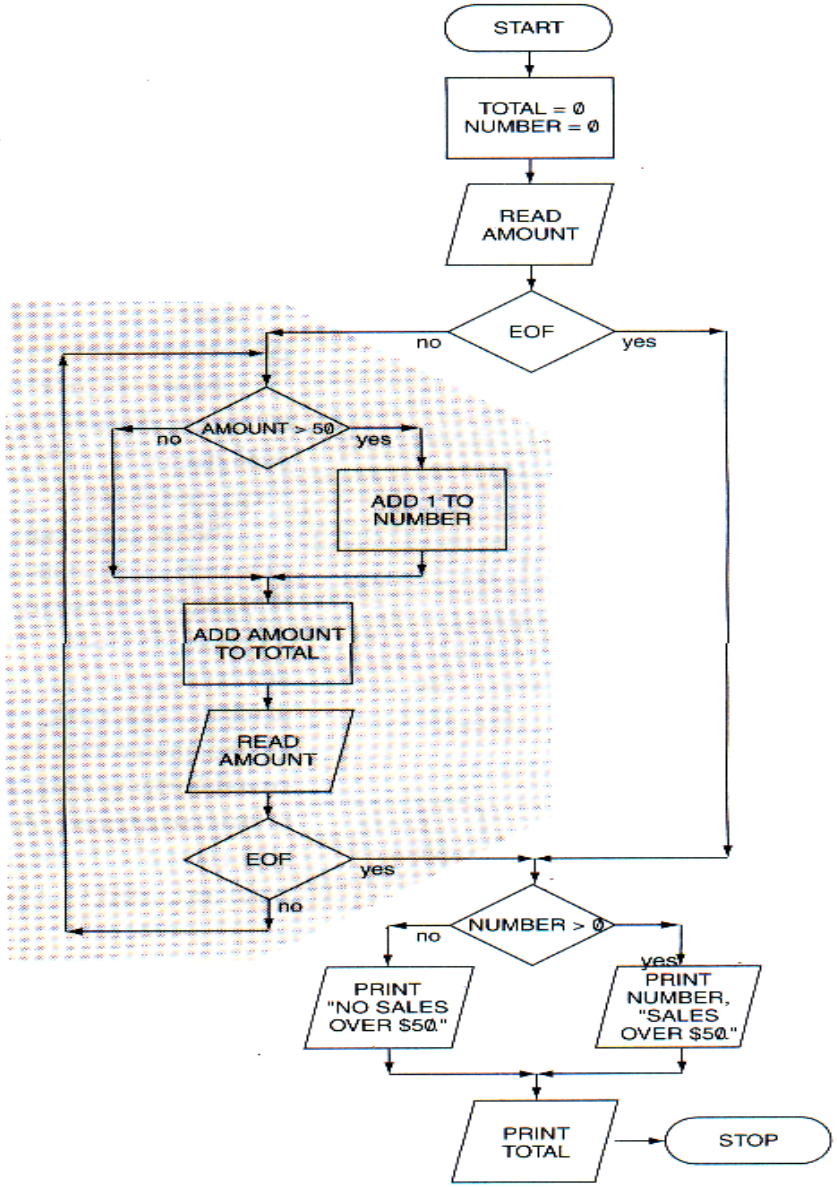
Combining the Structures

Design program flowchart that will read amounts, keep track of the number of amounts greater than 50, print the number of amounts over 50, and print the total of all amounts.



The flowchart is structured because it is composed of only the permissible logical structures.

- Redesign the previous program flowchart by using **do-until** loop rather than a do-while loop.



Reason to use Structured Flowcharts

1. Clarity: small doubling method
2. Professionalism: It's the way things are done Professionally
3. Efficiency: modern computer languages are structured languages
4. Maintenance: easier
5. Modularity: routines assigned to number of programmers

An "Unofficial" Explanation of structure

Structured programs

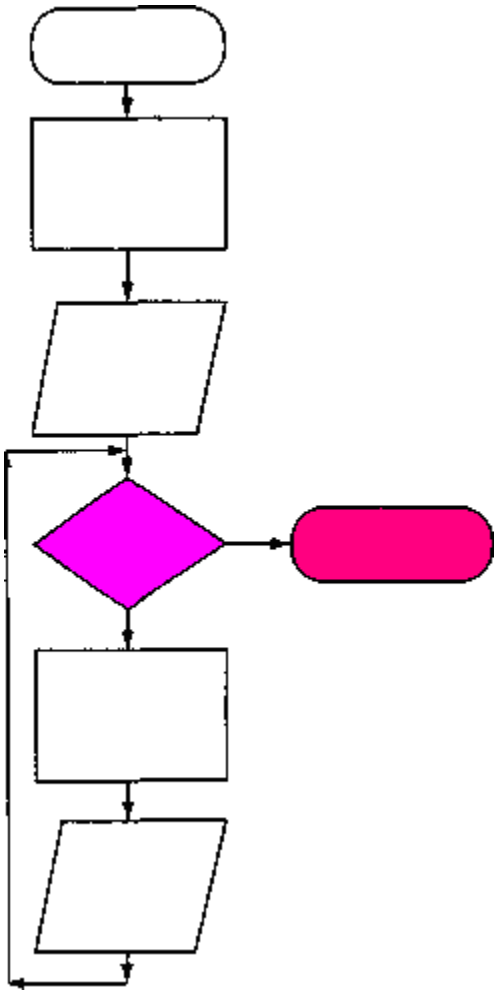
- *If a decision is a loop exit decision it must*
  - *be the only loop exit decision in its loop*
  - *be either the first or the last symbol in the loop.*

- If a decision is an if-then-else decision, the flow lines from each side of the decision must meet.

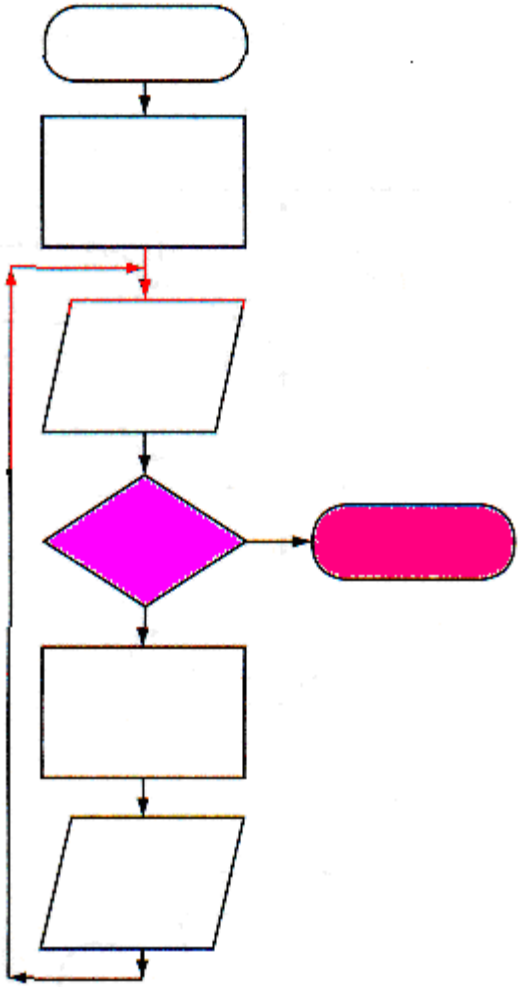
Conventions Facilitate Constructing Structured Flowcharts

1. Draw flow lines so that flow enters each decision at the top.
2. Place the affirmative branch of every decision on the right.
3. Place the negative decision of an if-then-else to the left.
4. Place the negative decision of a loop exit decision at the bottom.

Structured and unstructured flowcharts

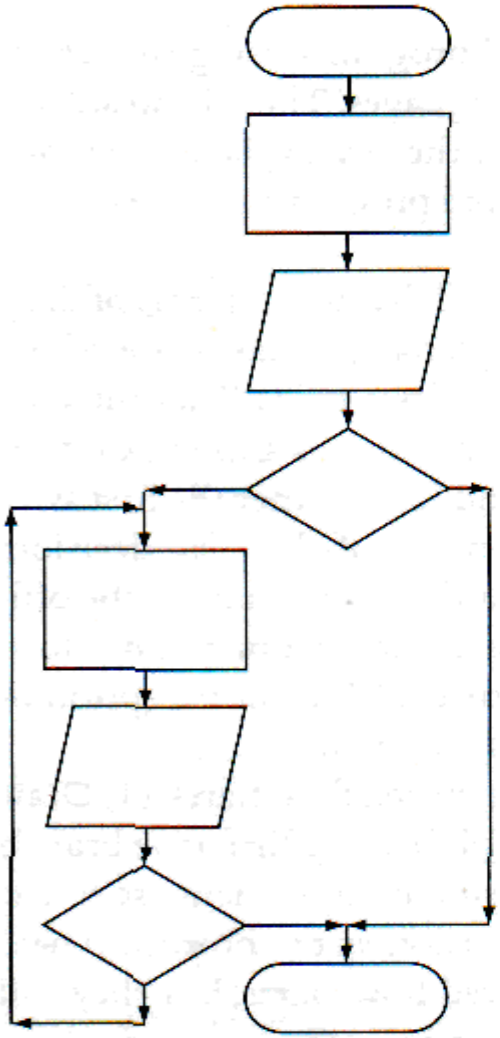


Structured
loop is a do-while

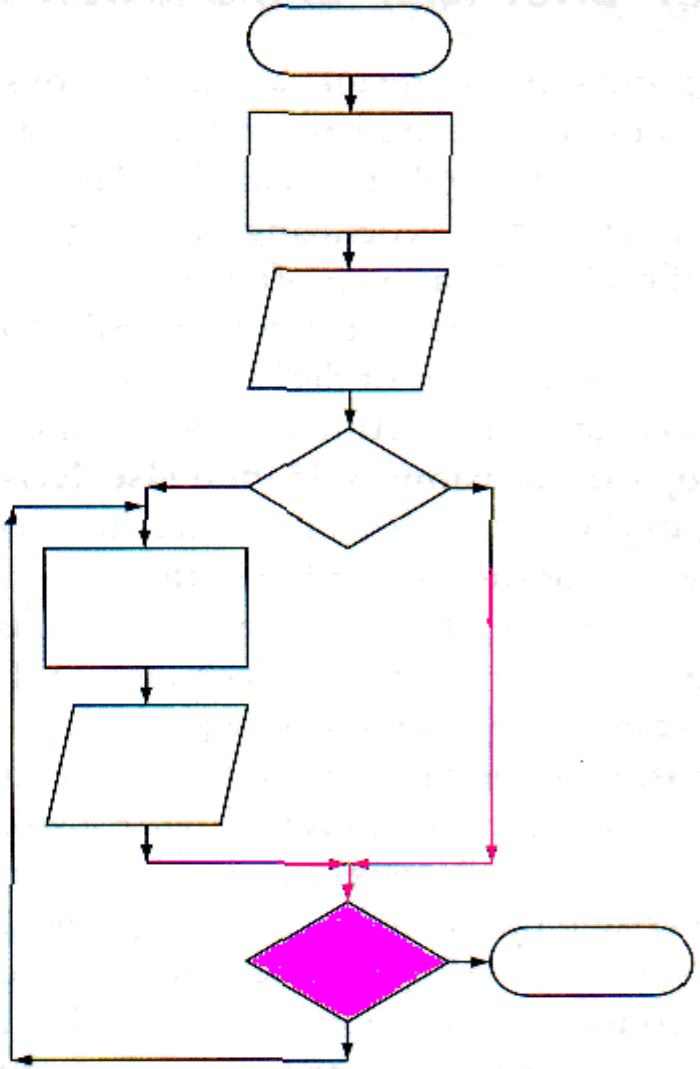


Unstructured
loop exit is not first or last

Structured And Unstructured Flowcharts

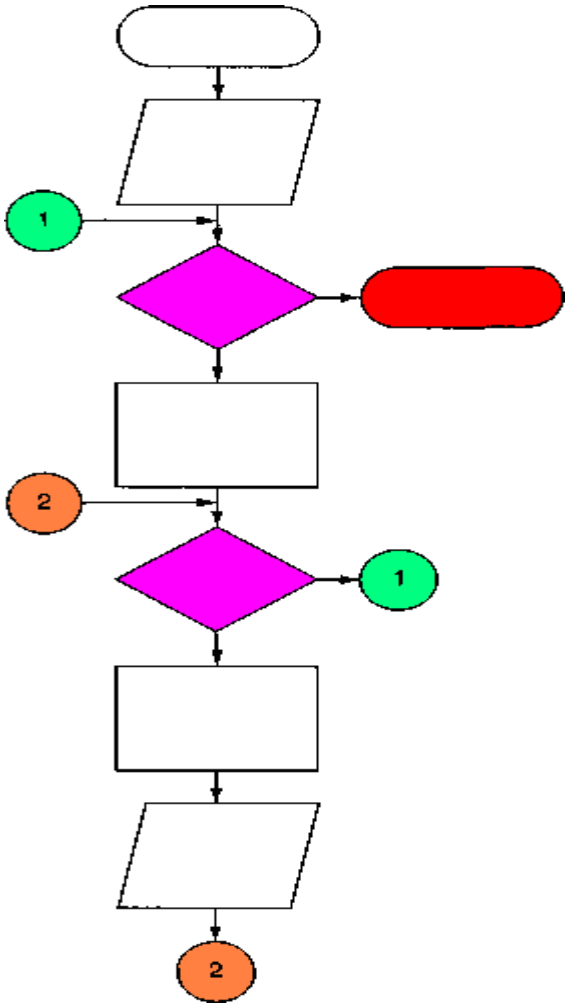


Structured -do-until loop contained within an if-then-else

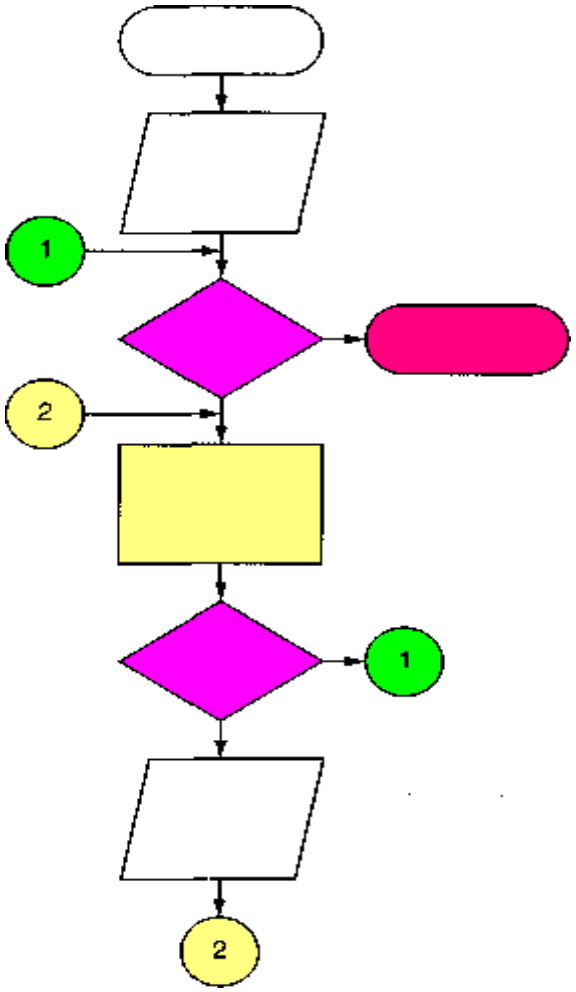


Unstructured - flow enters if-then-else at two locations

Structured and unstructured flowcharts

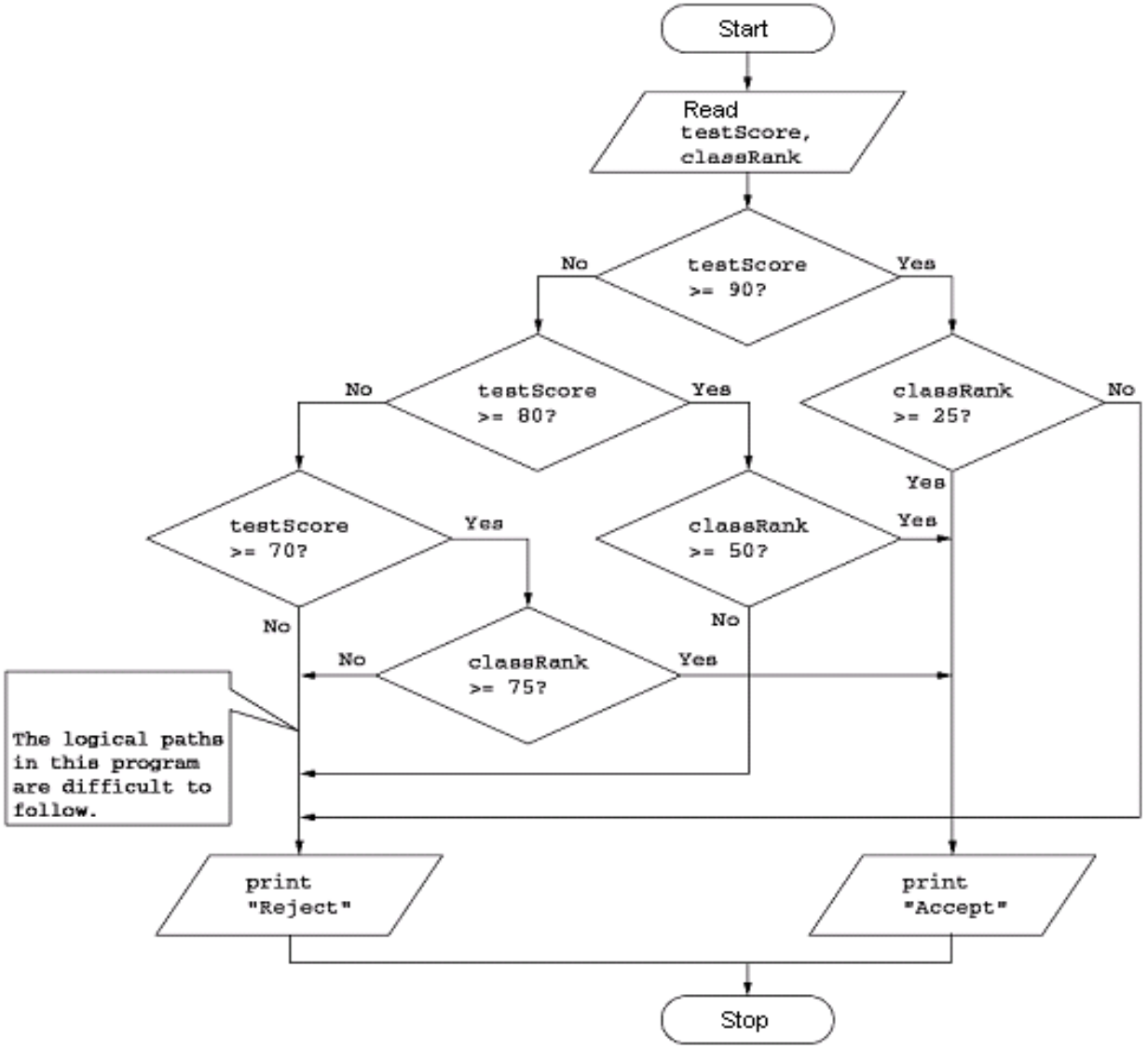


Structured
nested do-while loops

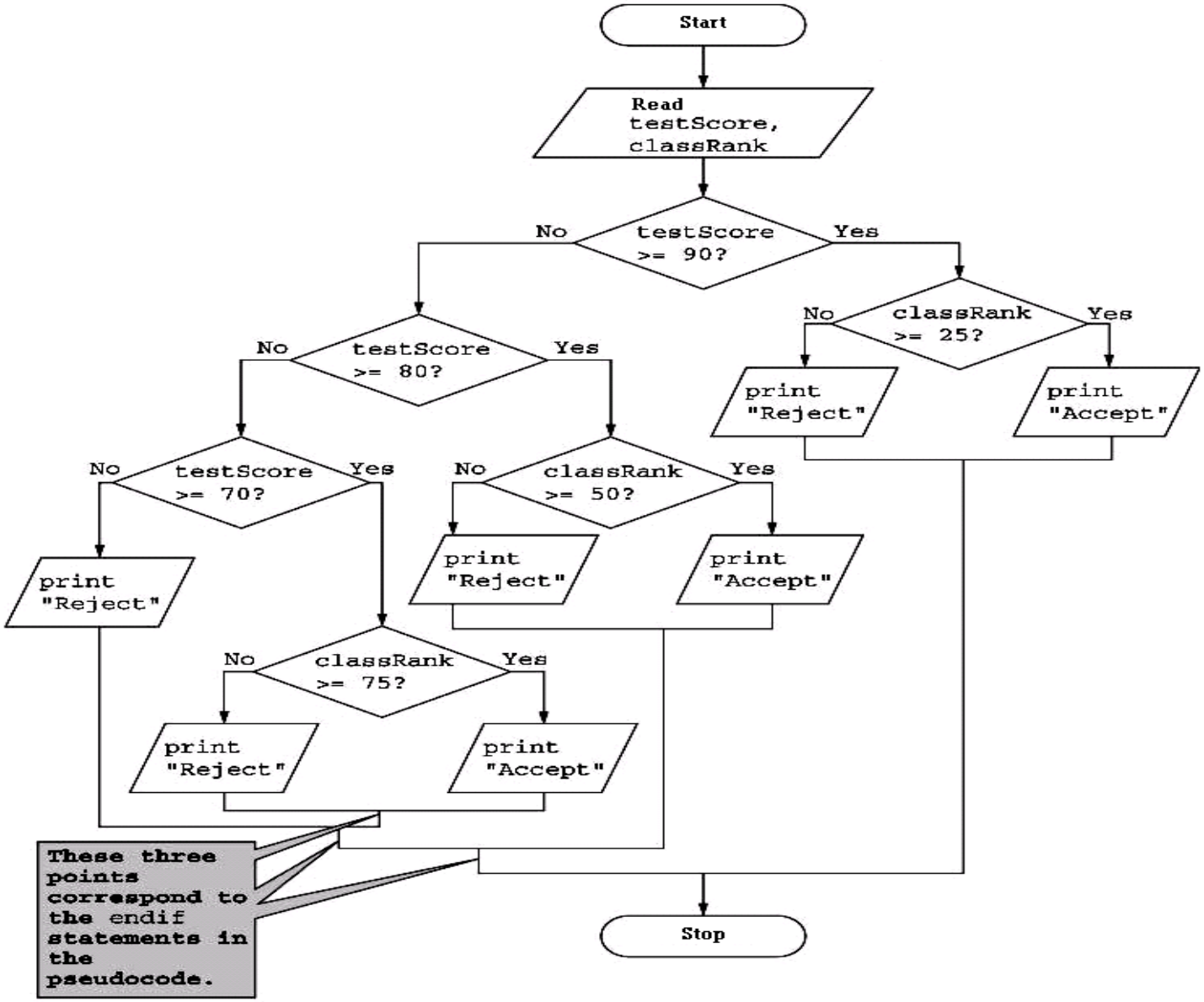


Unstructured - exit decision for
second loop is not first or last

Structured and unstructured flowcharts?



Structured and unstructured flowcharts



Indicators in Loop Exit Decisions

Indicators are often used in the loop exit decisions of structured programs when flow must exit a loop for more than one reason.

Example:

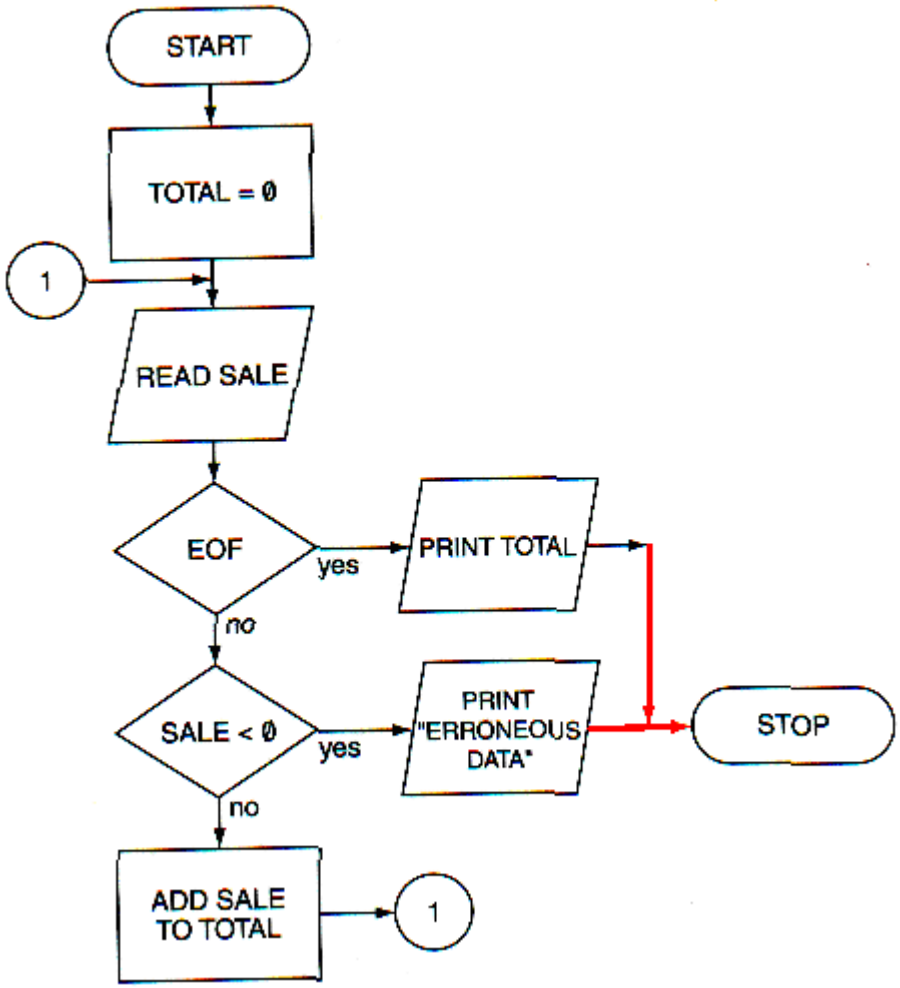
Require that a loop end either when EOF is reached or when a negative sale is read

Solution:

One decision tests for EOF; the other, for a negative sale. This program could run as described, but it is not structured as shown in Figure.

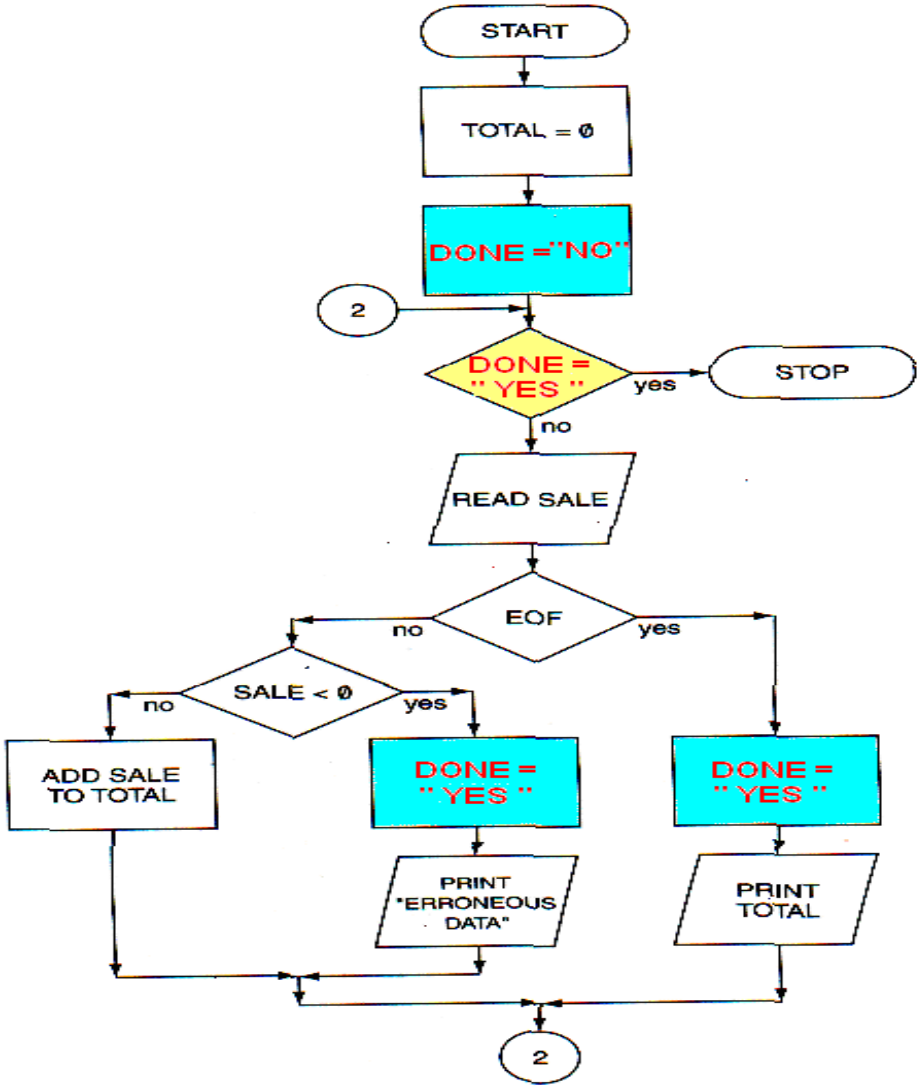
Indicators in Loop Exit Decisions

Ex., exit a loop for more than one reason



Indicators in Loop Exit Decisions

- Use **DONE indicator** to stops the program when it is yes. Thus, two exit decisions of previous Figure are replaced by a single loop exit decision, and the program is structured.



The following characteristics of the flowcharts

1. Structured flowcharts with EOF exit decisions contain two READ instructions.
2. Structured loops with some other exit decision contain a single READ within the loop.
3. Do-until loops with EOF exit decisions are preceded by a preliminary EOF if-then-else decision.
4. Do-until loops with some other exit decision contain a single EOF decision, and it is an if-then-else decision.
5. When some decision other than EOF is the loop exit decision, the only difference between the do-while and do-until is the location of the exit decision within the loop.

Modular Flowcharts

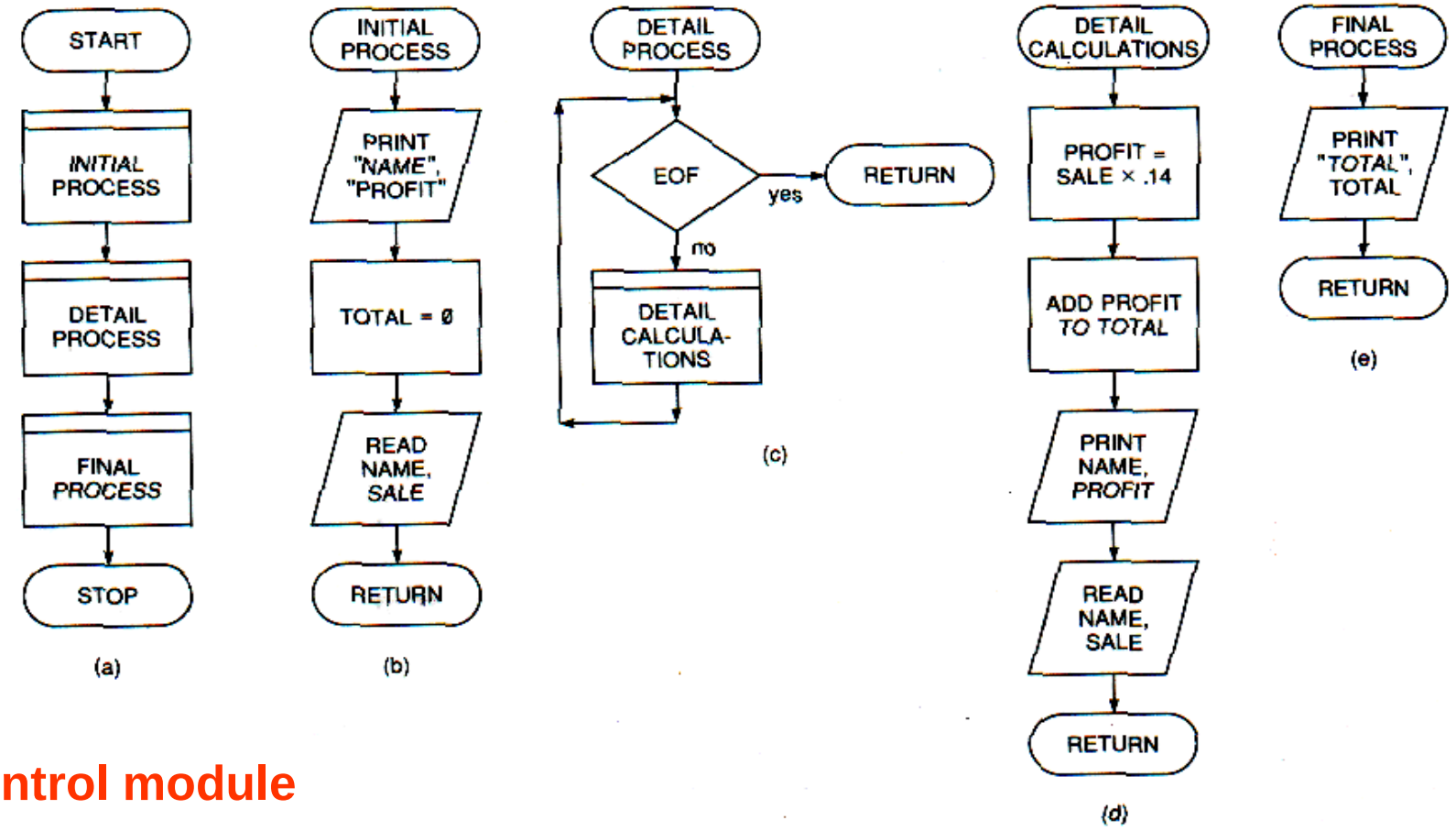
- **Module**:
  - Unit of code that performs one small task
  - Called a subroutine, procedure, function, or method
- **Modularization**: breaking a large program into modules
- **Advantages of modularization:**
  - Provides abstraction
  - Allows multiple programmers to work simultaneously
  - Allows code reuse
  - Makes identifying structures easier

FUNCTION

- A **function** is a group of statements that together perform a task.
  - Every C++ program has at least one function, **(main())**
  - Programs can define additional functions.
- A **function** can also be referred as a method or a sub-routine or a procedure, etc.
  - You can divide up your code into separate functions.
    - How you divide up your code among different functions is up to you,
      - each function performs a specific task.
- A **function declaration** tells the compiler about a function's name, return type, and parameters.
  - A function definition provides the actual body of the function.

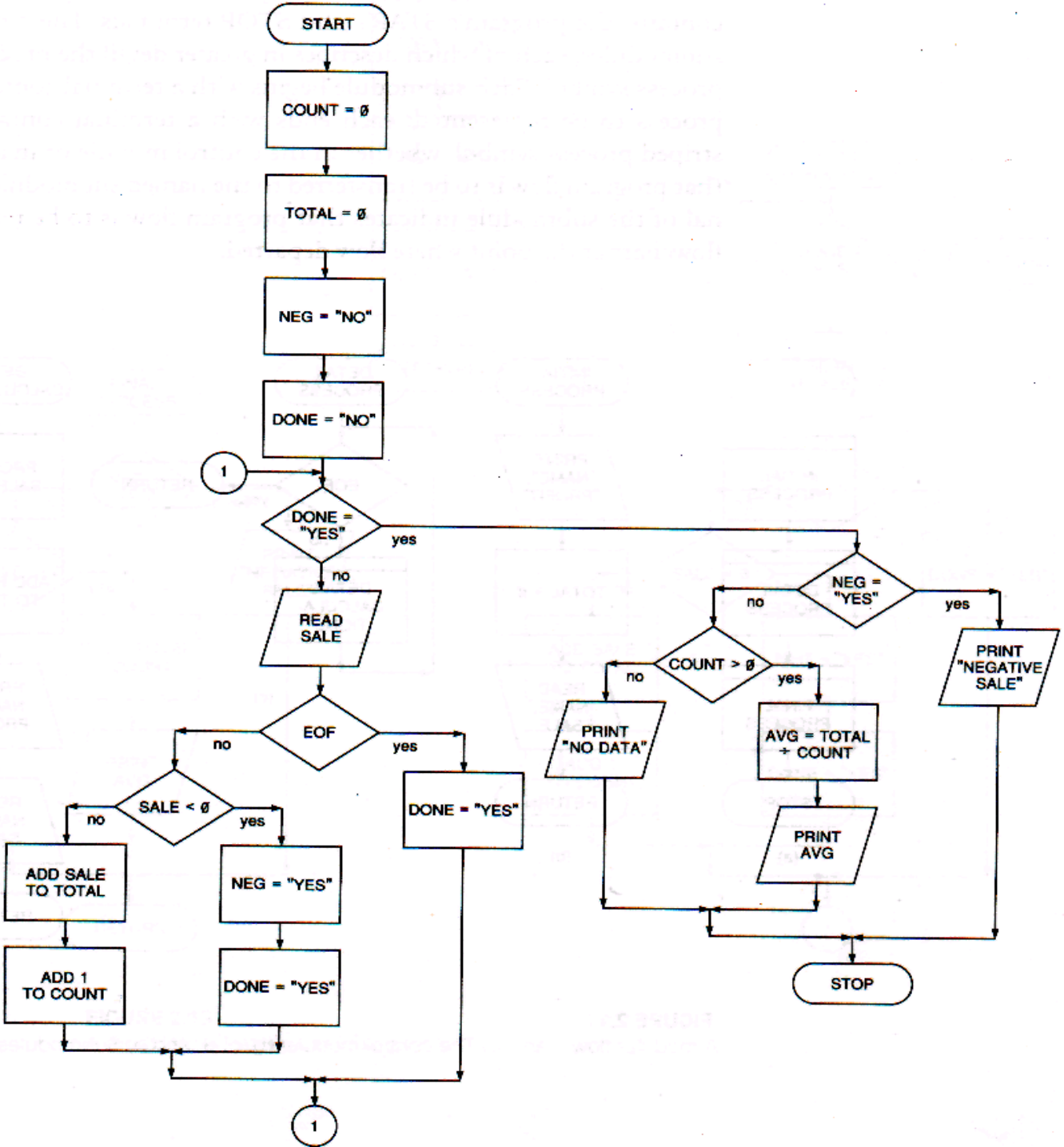
Modular Flowcharts

Each of the submodules in Figure represents a single structure from list of permissible logical structures. It is always possible to modularize a program to this degree

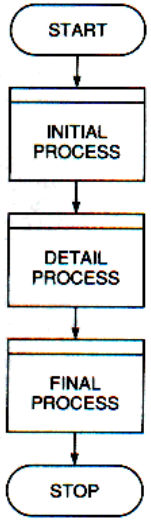


Control module

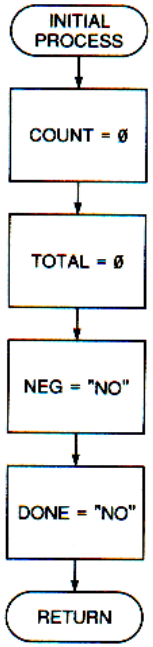
Submodules


START
COUNT = ø
TOTAL = ø
NEG = "NO"
DONE = "NO"
1
DONE = "YES"
yes
no
READ SALE
EOF
no
yes
SALE < ø
no
yes
ADD SALE TO TOTAL
ADD 1 TO COUNT
NEG = "YES"
DONE = "YES"
DONE = "YES"
1
NEG = "YES"
no
yes
COUNT > ø
no
yes
PRINT "NO DATA"
AVG = TOTAL ÷ COUNT
PRINT AVG
PRINT "NEGATIVE SALE"
STOP

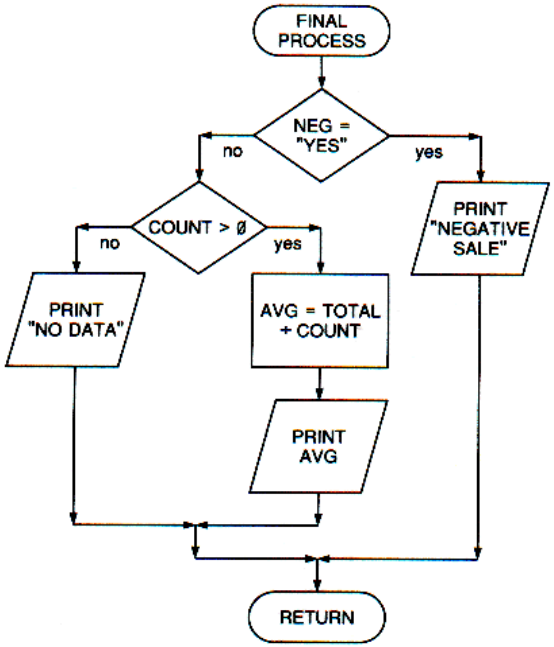
modularized version of previous Figure

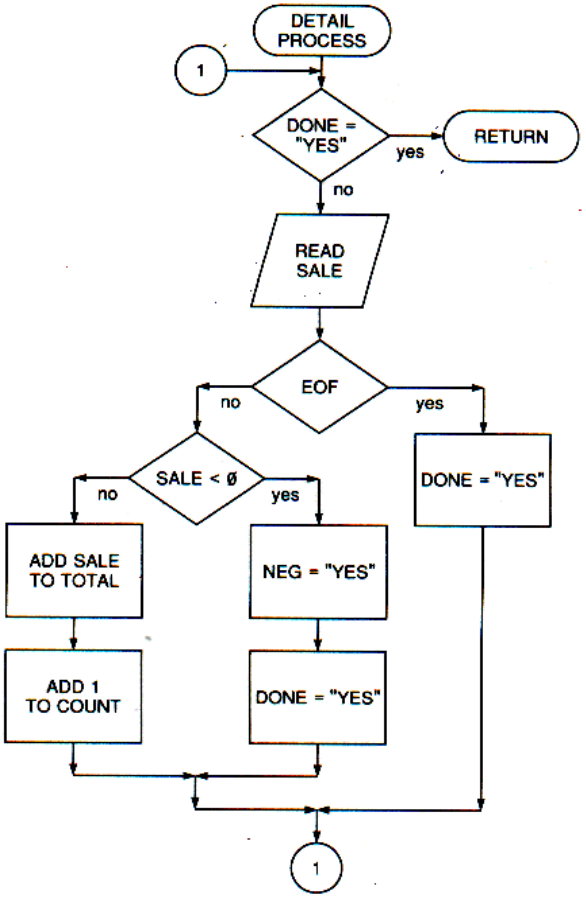


Control module



Submodules





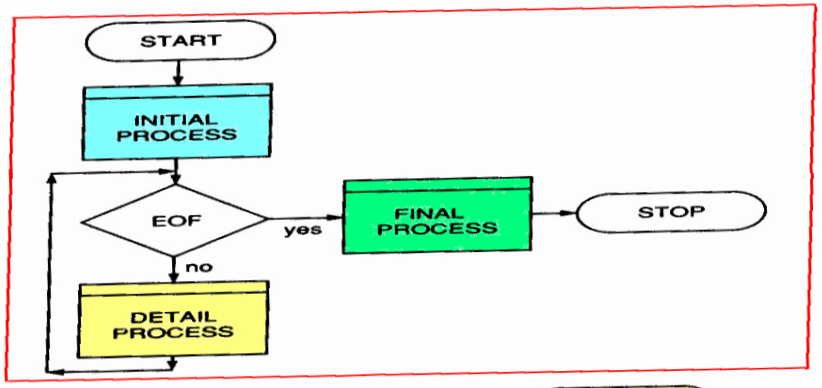
Modular Pseudocode

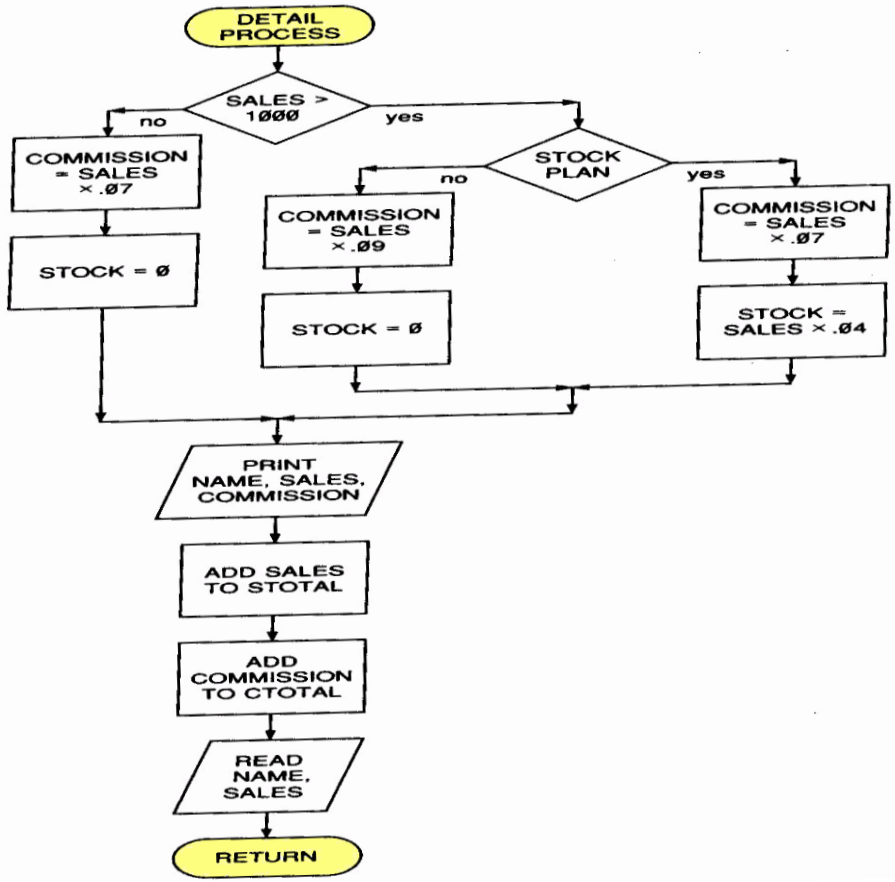
Modular pseudocode allows the program designer to quickly sketch the broad outlines of a program and complete the details later.

Example

```
START
DO INITIAL PROCESS
DO DETAIL PROCESS WHILE NOT EOF
DO FINAL PROCESS
STOP

INITIAL PROCESS
      STOTAL = Ø
      CTOTAL = Ø
      READ NAME, SALES
END INITIAL PROCESS

DETAIL PROCESS
      IF SALES > 1ØØØ
            IF STOCK PLAN
                  COMMISSION = SALES × .Ø7
                  STOCK = SALES × .Ø4
            ELSE
                  COMMISSION = SALES × .Ø9
                  STOCK = Ø
            END IF
      ELSE
            COMMISSION = SALES × .Ø7
            STOCK = Ø
      END IF
      PRINT NAME, SALES, COMMISSION
      ADD SALES TO STOTAL
      ADD COMMISSION TO CTOTAL
      READ NAME, SALES
END DETAIL PROCESS

FINAL PROCESS
      PRINT STOTAL
      PRINT CTOTAL
END FINAL PROCESS
```

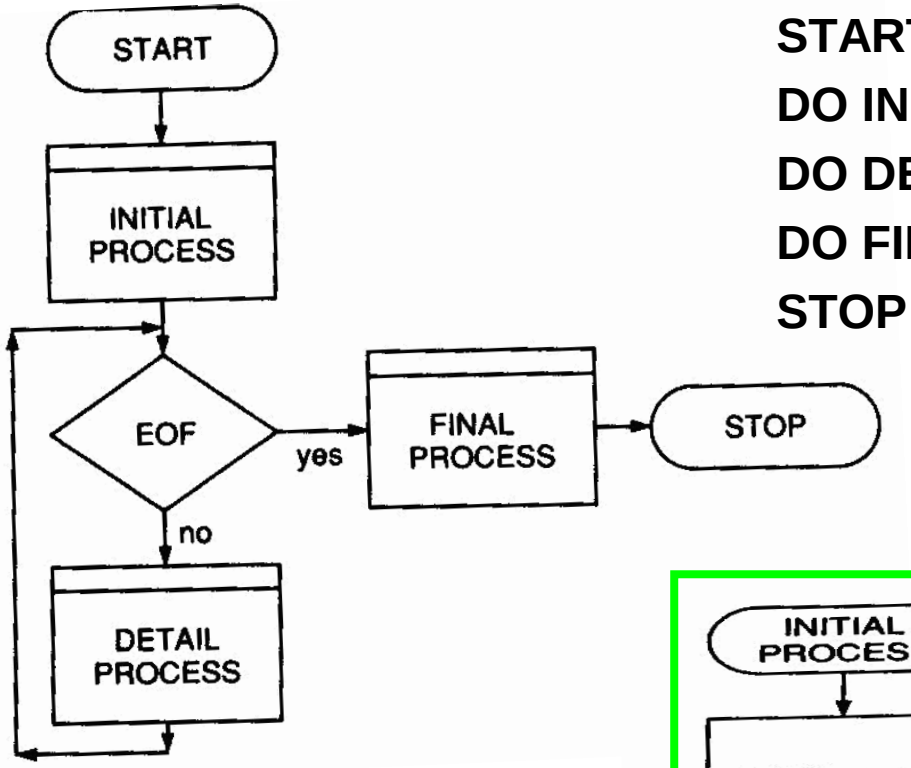
Example



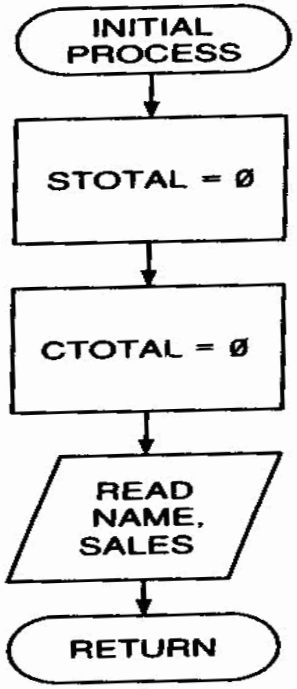



```
START
DO INITIAL PROCESS
DO DETAIL PROCESS WHILE NOT EOF
DO FINAL PROCESS
STOP

INITIAL PROCESS
      STOTAL = Ø
      CTOTAL = Ø
      READ NAME, SALES
END INITIAL PROCESS

DETAIL PROCESS
      IF SALES > 1000
            IF STOCK PLAN
                  COMMISSION = SALES × .07
                  STOCK = SALES × .04
            ELSE
                  COMMISSION = SALES × .09
                  STOCK = Ø
            END IF
      ELSE
            COMMISSION = SALES × .07
            STOCK = Ø
      END IF
      PRINT NAME, SALES, COMMISSION
      ADD SALES TO STOTAL
      ADD COMMISSION TO CTOTAL
      READ NAME, SALES
END DETAIL PROCESS

FINAL PROCESS
      PRINT STOTAL
      PRINT CTOTAL
END FINAL PROCESS
```

Cont., Example

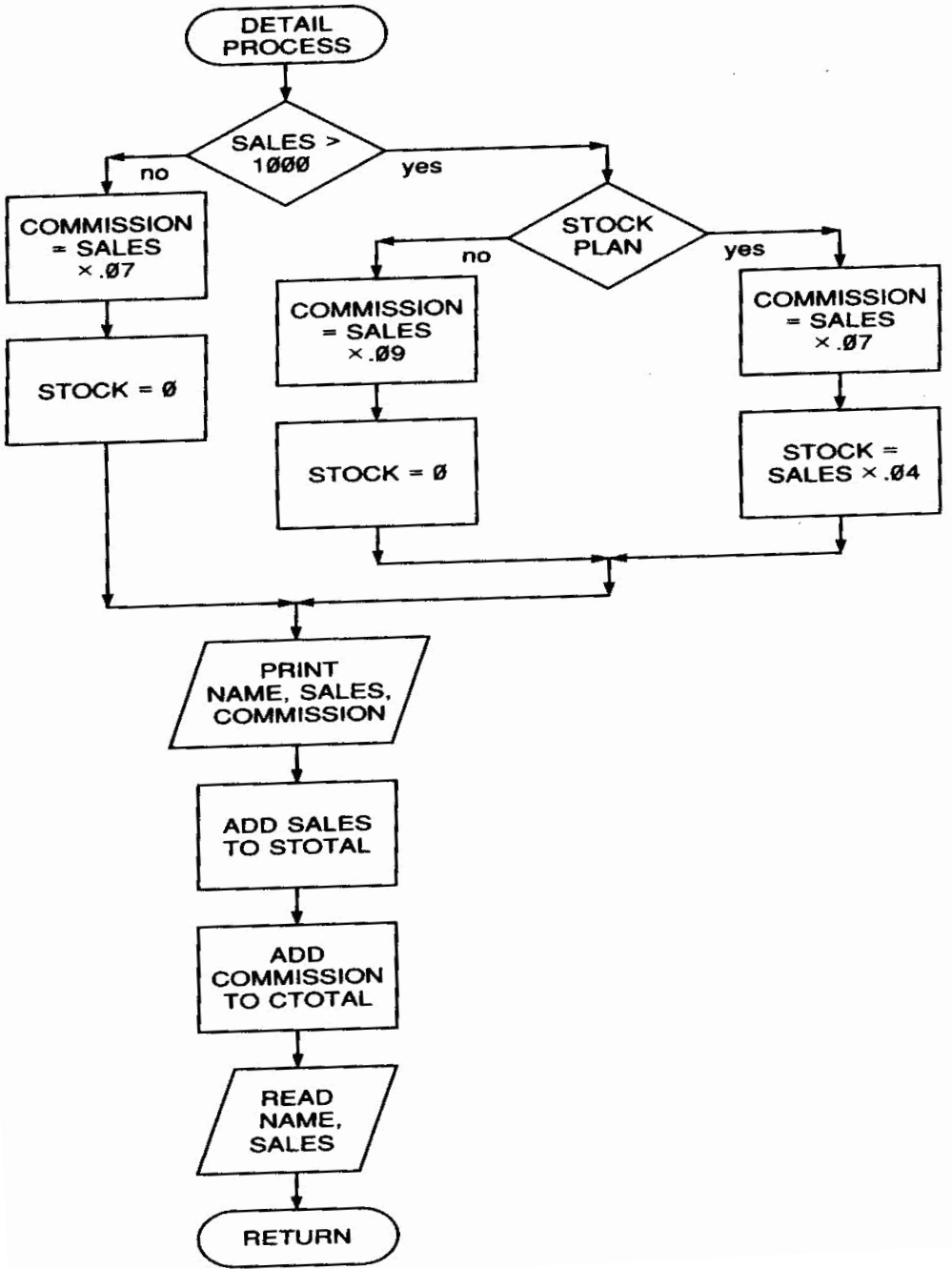


START
DO INITIAL PROCESS
DO DETAIL PROCESS WHILE NOT EOF
DO FINAL PROCESS
STOP



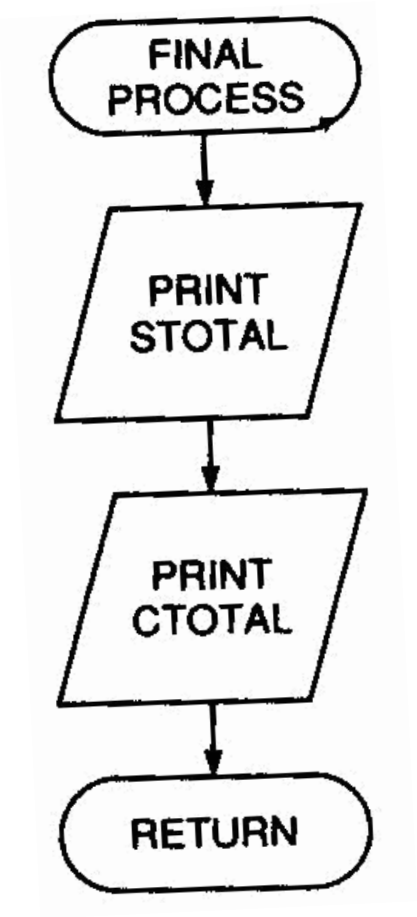
INITIAL PROCESS
    STOTAL = 0
    CTOTAL = 0
    READ NAME, SALES
END INITIAL PROCESS

Cont., Example




```
DETAIL PROCESS
IF SALES > 1000
IF STOCK PLAN
        COMMISSION = SALES X .07
        STOCK = SALES X .04
ELSE
        COMMISSION = SALES X .09
        STOCK = 0
END IF
ELSE
        COMMISSION = SALES X .07
        STOCK = 0
END IF
PRINT NAME, SALES, COMMISSION
ADD SALES TO STOTAL
ADD COMMISSION TO CTOTAL
READ NAME, SALES
END DETAIL PROCESS
```

Cont., Example




```
FINAL PROCESS
    PRINT STOTAL
    PRINT CTOTAL
END FINAL PROCESS
```

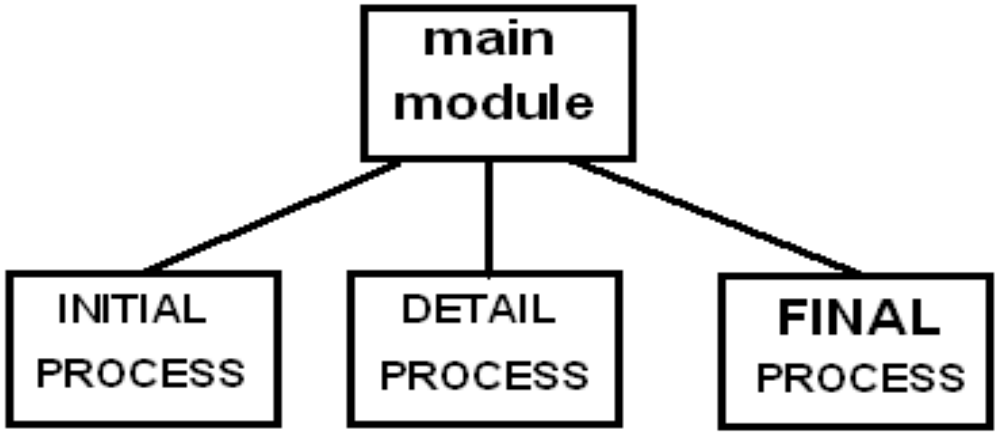
Hierarchy Charts

When pseudocode is modularized, it is frequently accompanied by hierarchy charts.

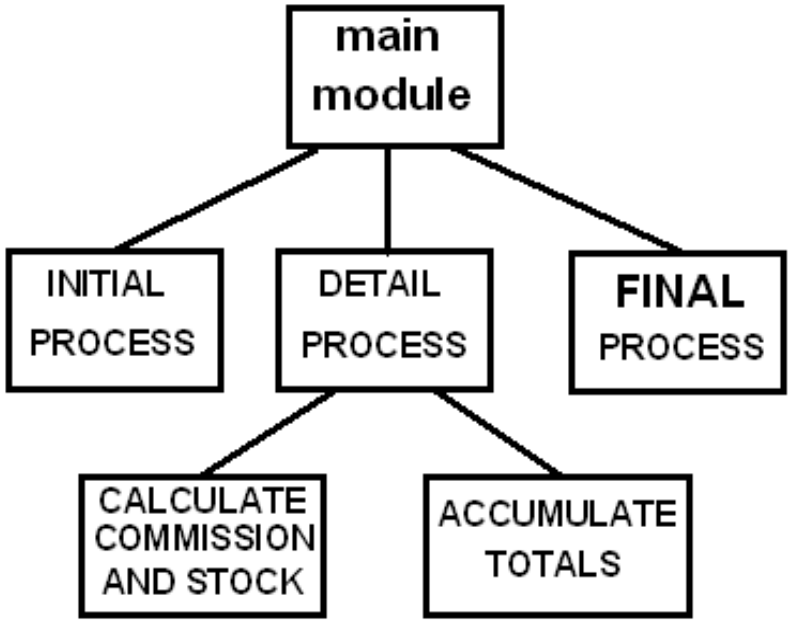
- **These charts present a picture of a program's hierarchy and of the organization of its predefined processes.**

Hierarchy charts begin with a rectangle that represents a program's main module.

- **Below this rectangle on a single horizontal level other rectangles represent each process mentioned in the main module.**

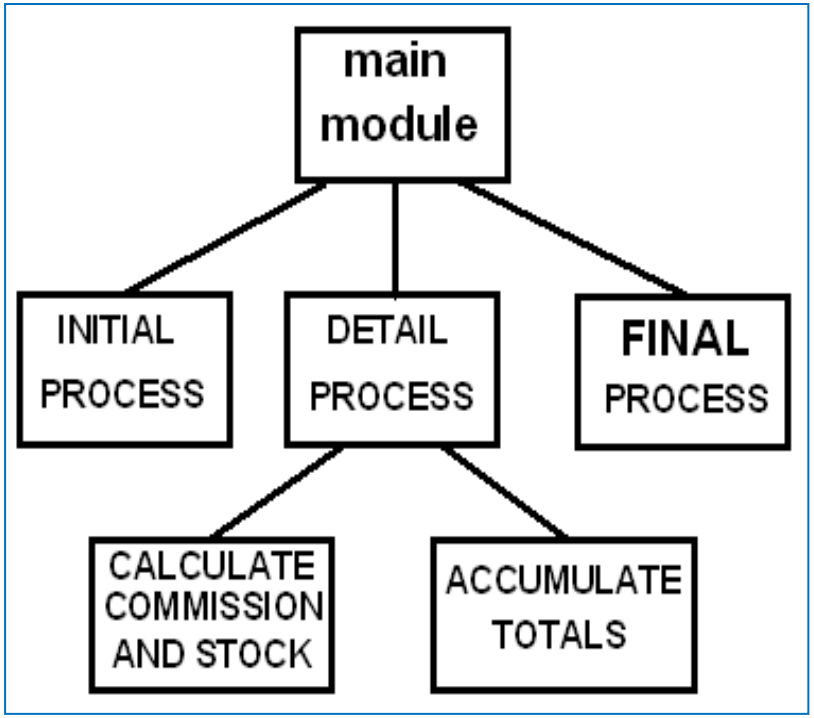


Example




```
START
DO INITIAL PROCESS
DO DETAIL PROCESS WHILE NOT EOF
DO FINAL PROCESS
STOP
INITIAL PROCESS
  STOTAL = 0
  CTOTAL = 0
  READ NAME, SALES
END  INITIAL PROCESS
DETAIL PROCESS
  DO CALCULATE COMMISSION AND STOCK
  PRINT NAME, SALES, COMMISSION
  DO ACCUMULATE TOTALS
  READ NAME, SALES
END DETAIL PROCES
FINAL PROCESS
  PRINT STOTAL
  PRINT CTOTAL END
FINAL PROCESS
```

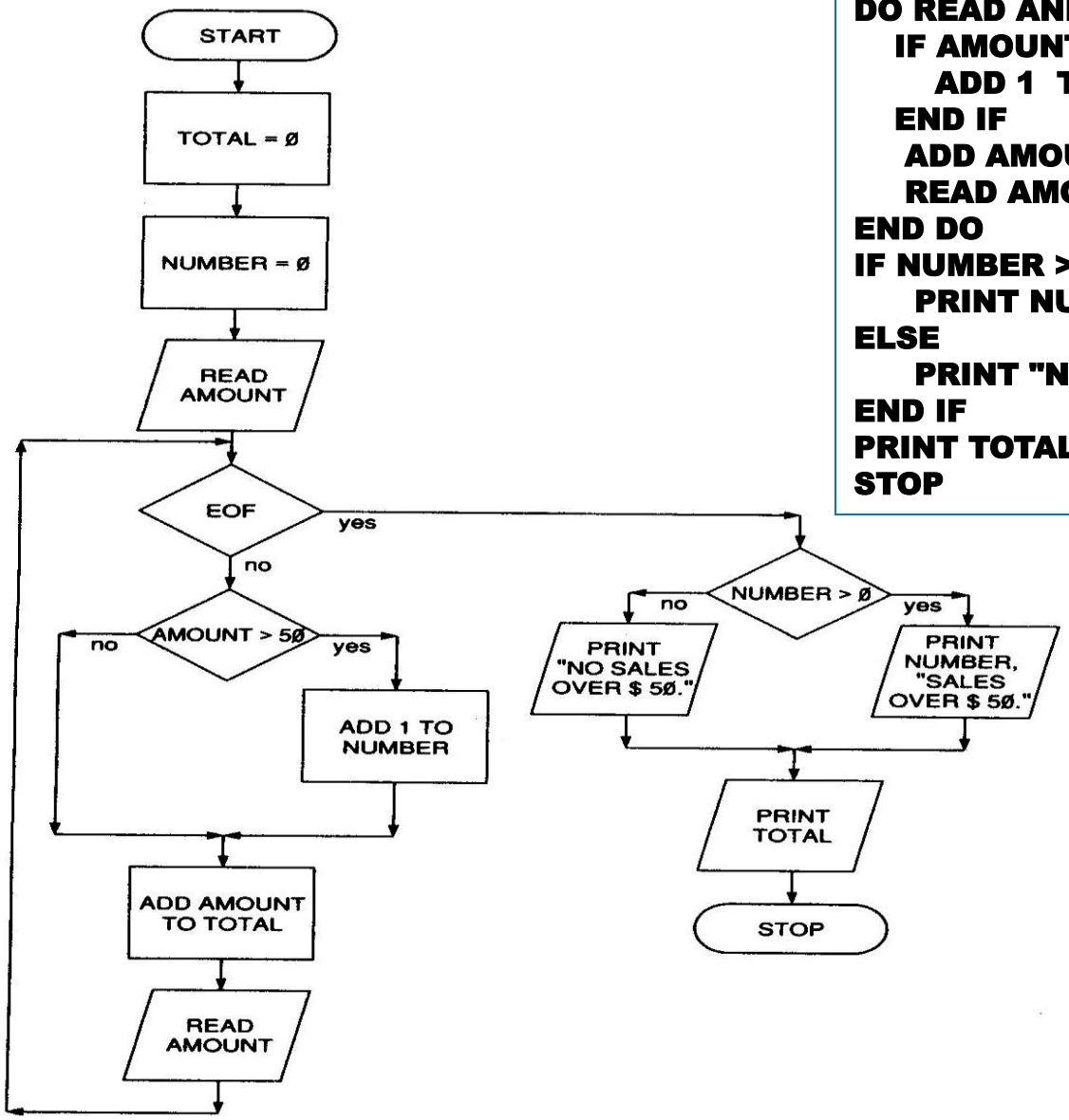
Cont., Example




```
CALCULATE COMMISSION AND STOCK
IF SALES > 1000
    IF STOCK PLAN
      COMMISSION = SALES x .07
      STOCK = SALES x .04
    ELSE
      COMMISSION = SALES x .09
       STOCK = 0
    END IF
ELSE
  COMMISSION = SALES x .07
   STOCK = 0
END IF
END CALCULATE COMMISSION AND STOCK
```

```
ACCUMULATE TOTALS
    ADD SALES TO STOTAL
    ADD COMMISSION TO CTOTAL
END ACCUMULATE TOTALS
```

EXAMPLE




```
START
   TOTAL =0
    NUMBER = 0
    READ AMOUNT
DO READ AND CALCULATE LOOP WHILE NOT EOF
    IF AMOUNT > 50
        ADD 1  TO NUMBER
    END IF
     ADD AMOUNT TO TOTAL
     READ AMOUNT
END DO
IF NUMBER > 0
      PRINT NUMBER, "SALES OVER $50."
ELSE
      PRINT "NO SALES OVER $50."
END IF
PRINT TOTAL
STOP
```

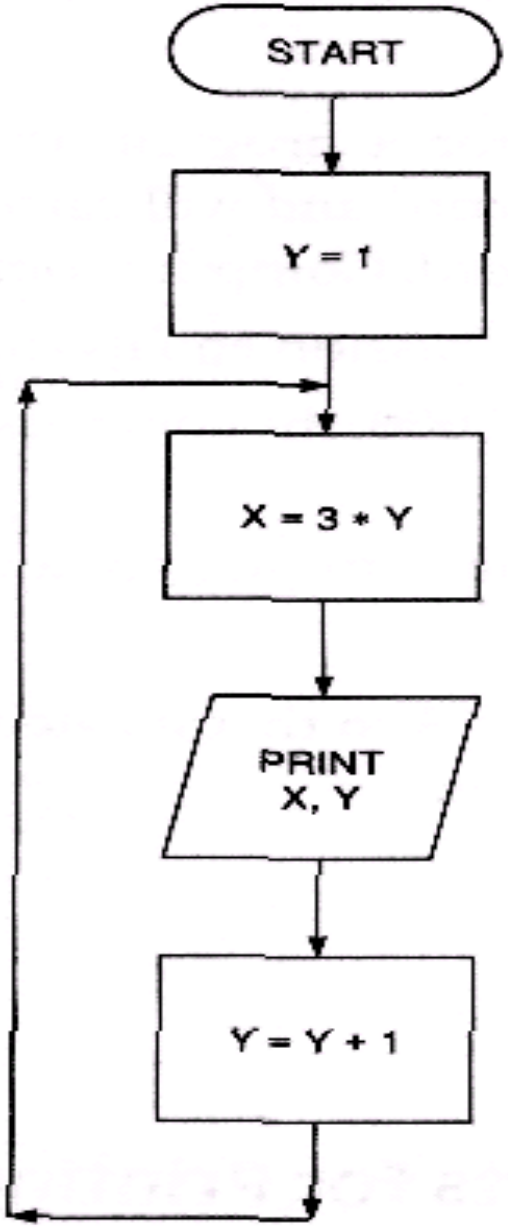
Structured Flowcharts for Printing Tables

- The programs to be designed create tables by calculating and printing the value of one variable of an equation, given the values of other variables.

Example:

Draw flowchart to creates a table for the two-variable equation X =3Y by calculating and printing the value of X for each positive-integer value of Y.

Flowchart for the two-variable equation X = 3Y



X	Y
3	1
6	2
9	3
.	.
.	.
.	.
.	.

Two-dimensional Tables

Design flowchart for program that will generate the values required to complete the tables for the following equations: X=Y+Z

Y \ Z	1	2	3	4	5
1					
2					
3					
4					
5					

Program flowchart & table for equation X=Y+Z

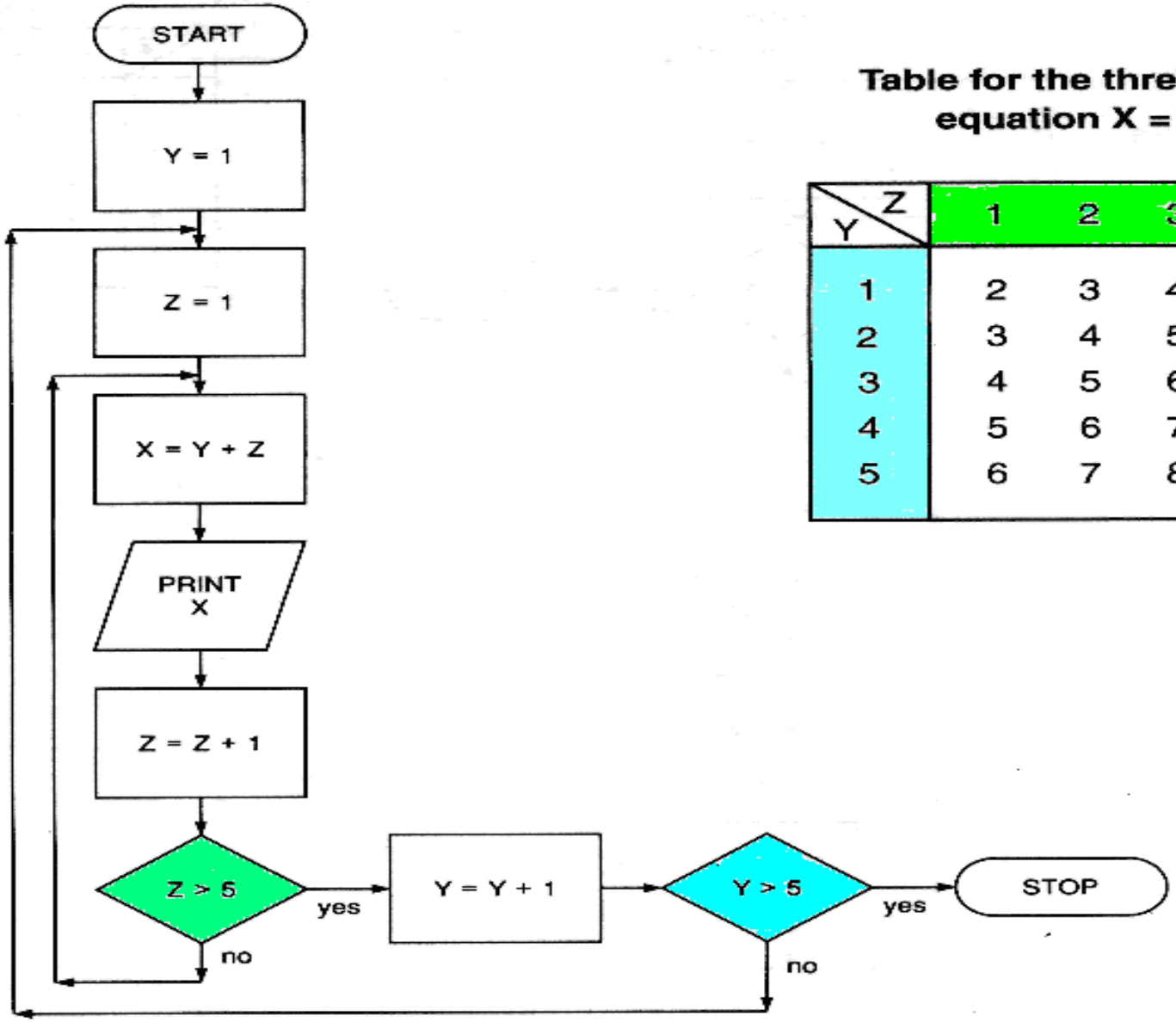


Table for the three-variable equation X = Y + Z.

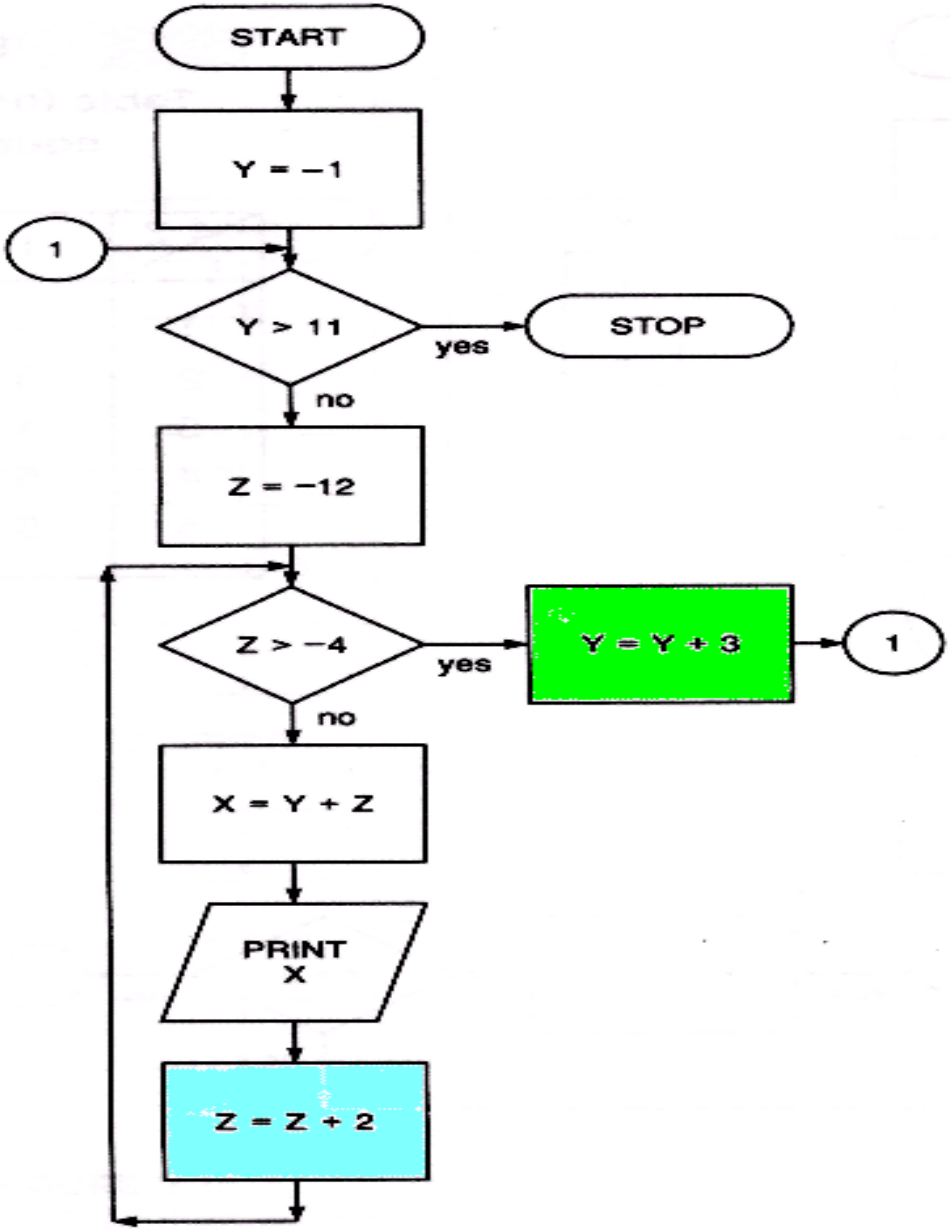
Y \ Z	1	2	3	4	5
1	2	3	4	5	6
2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9
5	6	7	8	9	10

Two-dimensional tables for equations with three variables

Design flowchart for program that will generate the values required to complete the tables for the following equations: X=Y+Z

Y \ Z	- 12	- 10	- 8	- 6	- 4
- 1					
2					
5					
8					
11					

Program flowchart & table for equation X=Y+Z



X = Y + Z

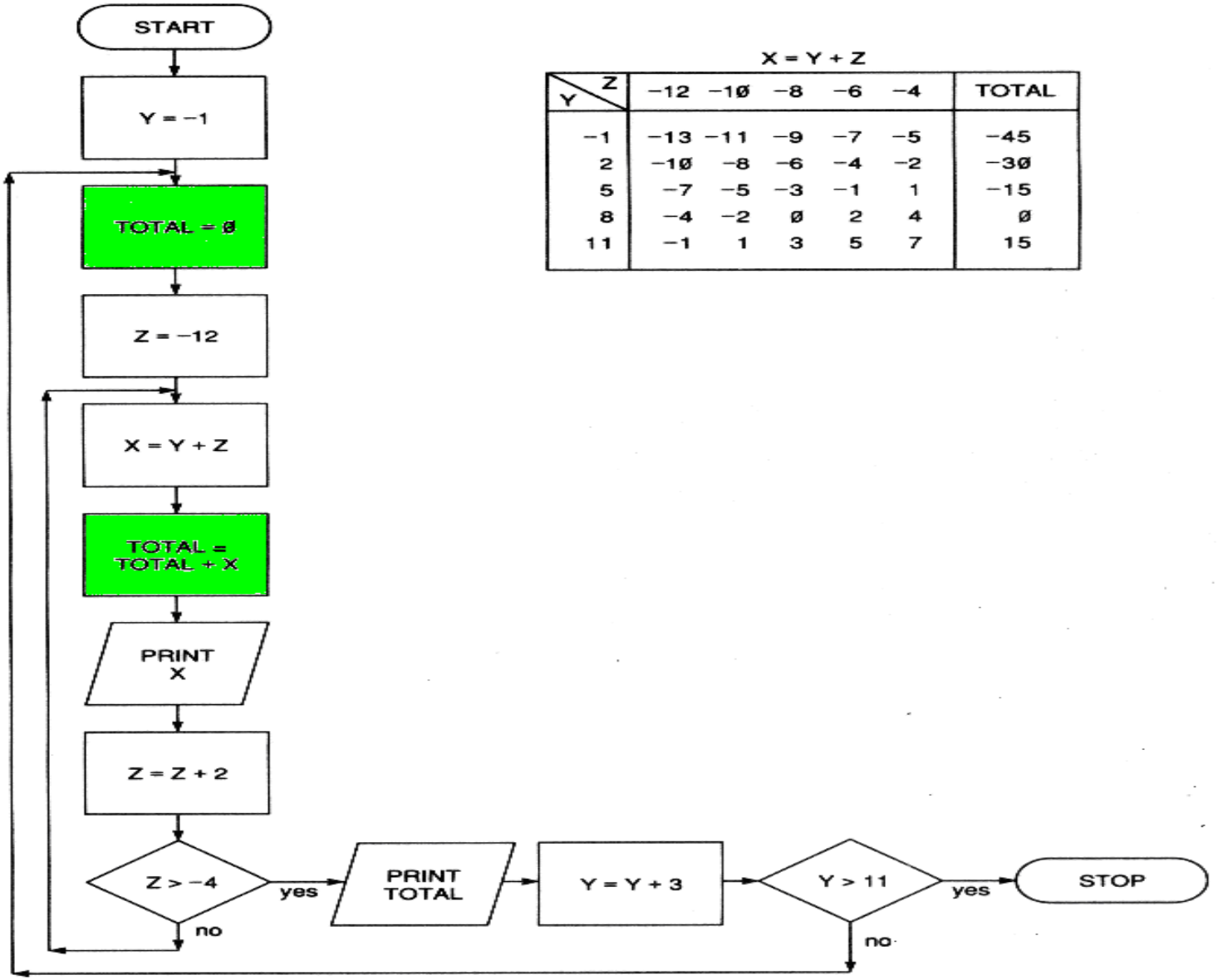
Y \ Z	−12	−10	−8	−6	−4
−1	−13	−11	−9	−7	−5
2	−10	−8	−6	−4	−2
5	−7	−5	−3	−1	1
8	−4	−2	0	2	4
11	−1	1	3	5	7

Two-dimensional Tables

- Design structured flowchart for program that will generate the values required to complete the tables for the following equations: X=Y+Z and the total for each row

Y \ Z	- 12	- 10	- 8	- 6	- 4	TOTAL
- 1						
2						
5						
8						
11						

Program flowchart & table for equation X=Y+Z & Row Total



X = Y + Z

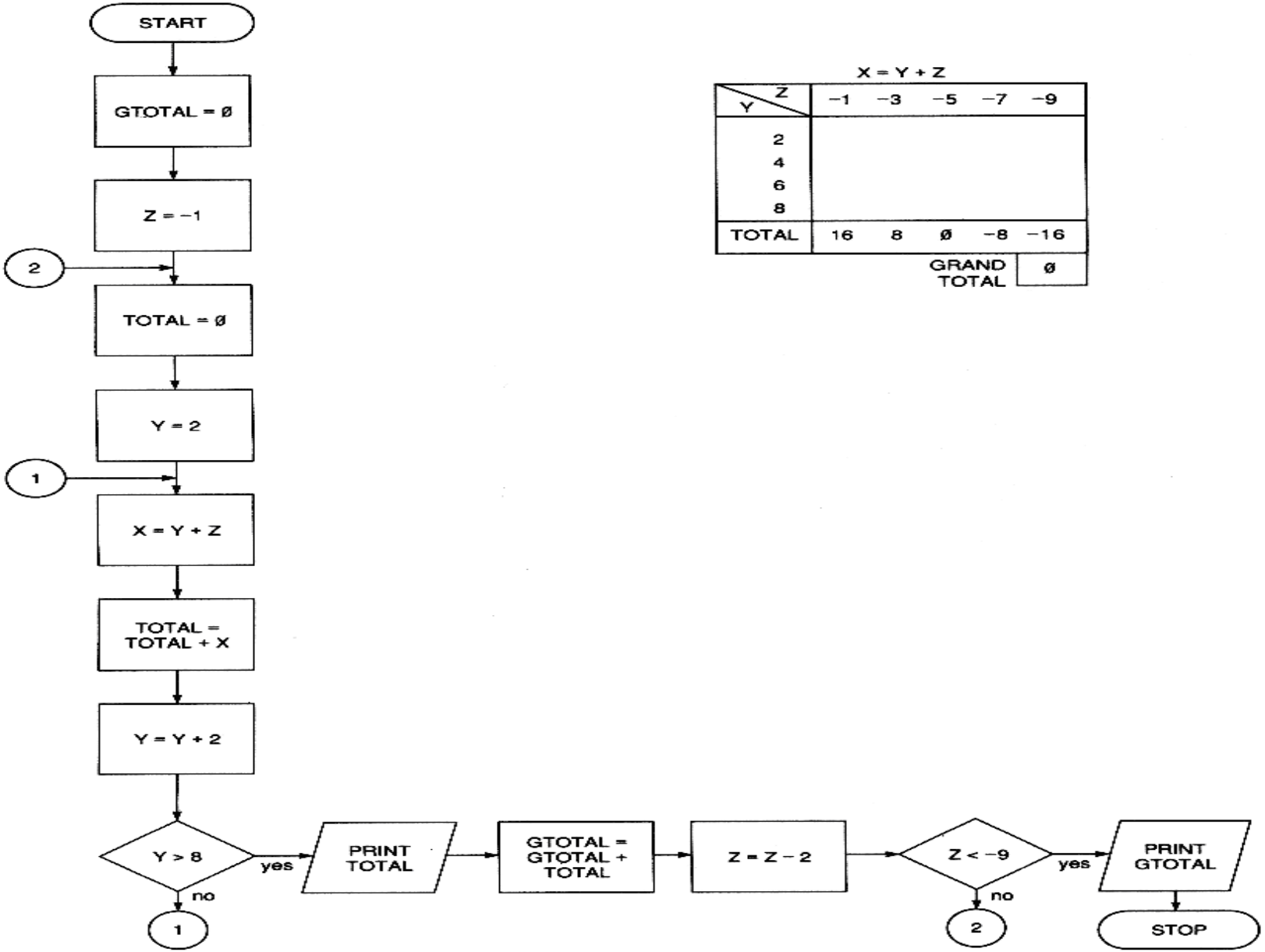
Y \ Z	−12	−1ø	−8	−6	−4	TOTAL
−1	−13	−11	−9	−7	−5	−45
2	−1ø	−8	−6	−4	−2	−3ø
5	−7	−5	−3	−1	1	−15
8	−4	−2	ø	2	4	ø
11	−1	1	3	5	7	15

Two-dimensional Tables

Design a structured flowchart with do-until loops for a program that will calculates and prints both the column totals and a grand total for the following equation: X=Y+Z. The values for Y, Z are:

Y	2	4	6	8	
Z	-1	-3	-5	-7	-9

structured flowchart that uses do-until loops to calculate column totals and a grand total



X = Y + Z

Y \ Z	−1	−3	−5	−7	−9
2					
4					
6					
8					
TOTAL	16	8	ø	−8	−16

GRAND TOTAL	ø