MENOUFIA UNIVERSITY
FACULTY OF COMPUTERS AND INFORMATION

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

PRINCIPLES OF PROGRAMMING

LECTURE TWO

Dr. Hamdy M. Mousa

Computers and Programming

Input/Output

Most computer operations require data to be entered and results to be printed, displayed, or recorded.

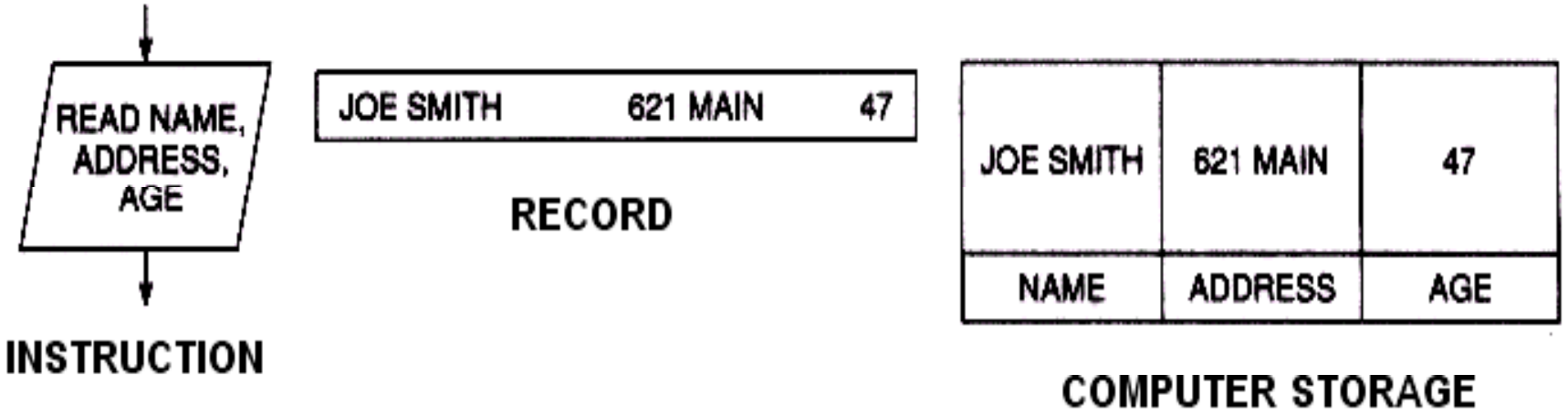
CUSTOMER FILE			
JOE SMITH	621 MAIN	47	CUSTOMER RECORD
TOM ADAMS	418 SOUTH	26	
MARK BARNS	149 CREST	44	
MARY THOMAS	31 6 EAST	37	
FRED JONES	947 ADA	52	
JOE SMITH	621 MAIN	47	
/*			EOF MARKER
NAME FIELD	ADDRESS FIELD	AGE FIELD	

A single input **instruction** (**READ**) obtains all the appropriate data from a single **record.**

Input/Output

Instruction READ NAME, ADDRESS, AGE

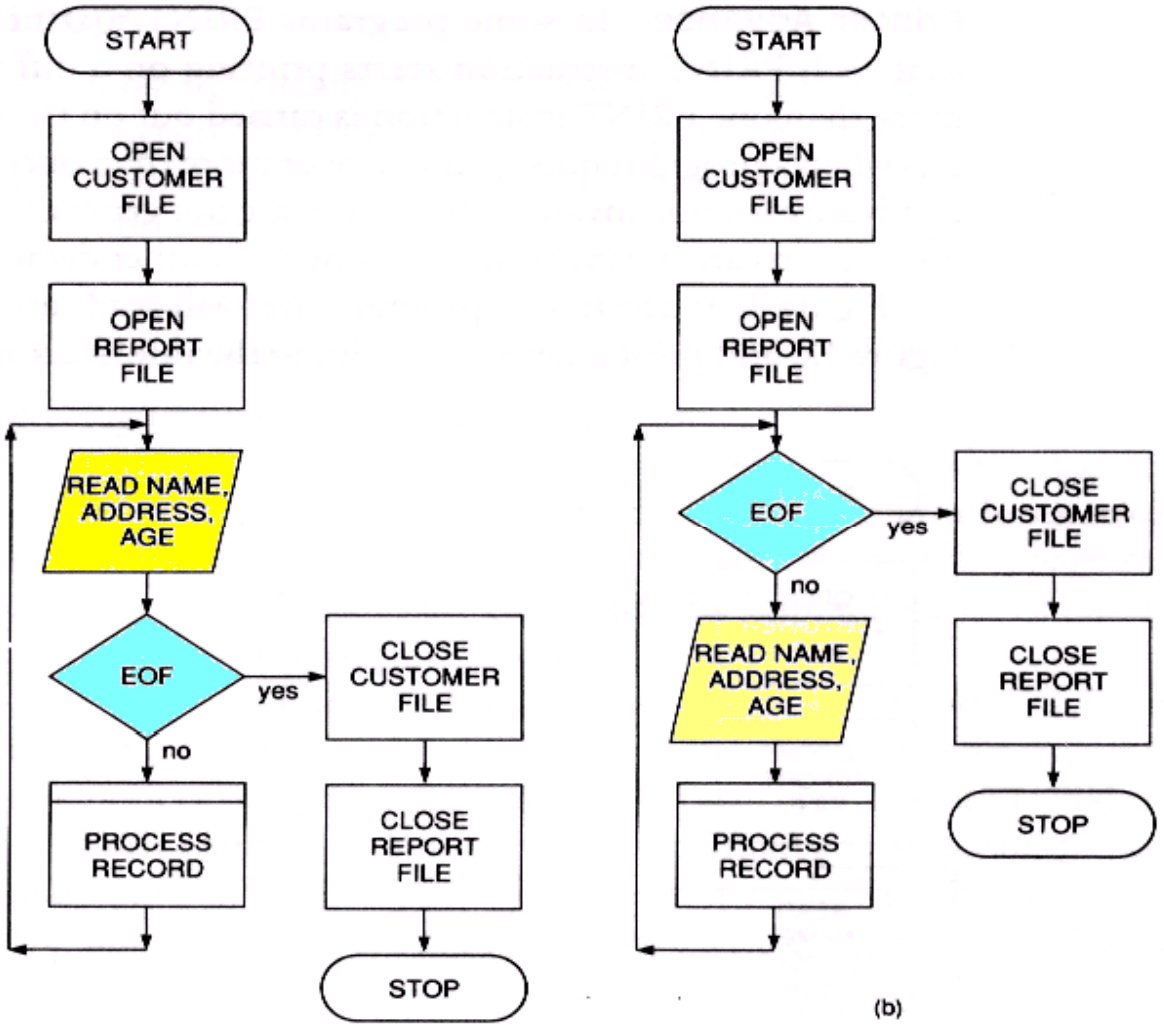
directs the computer to store data from three fields of the next record at three memory locations called NAME, ADDRESS, and AGE



Once data is stored in memory, it remains until the instruction to read is executed again.

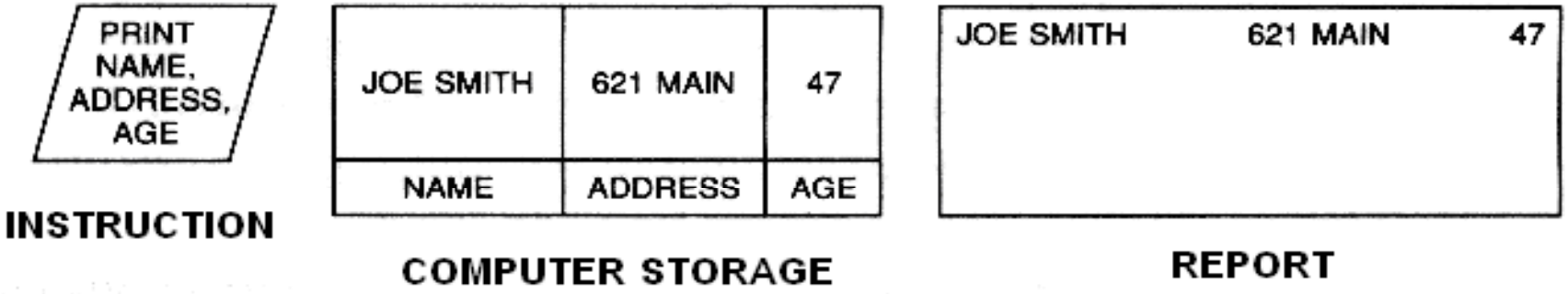
Place the EOF decision AFTER the corresponding READ.

Tests for EOF BEFORE executing the READ instruction.



PRINT

- To print a line on a report, we will use the instruction PRINT.
- The PRINT instruction directs the computer to print information stored at one or more of its memory locations.

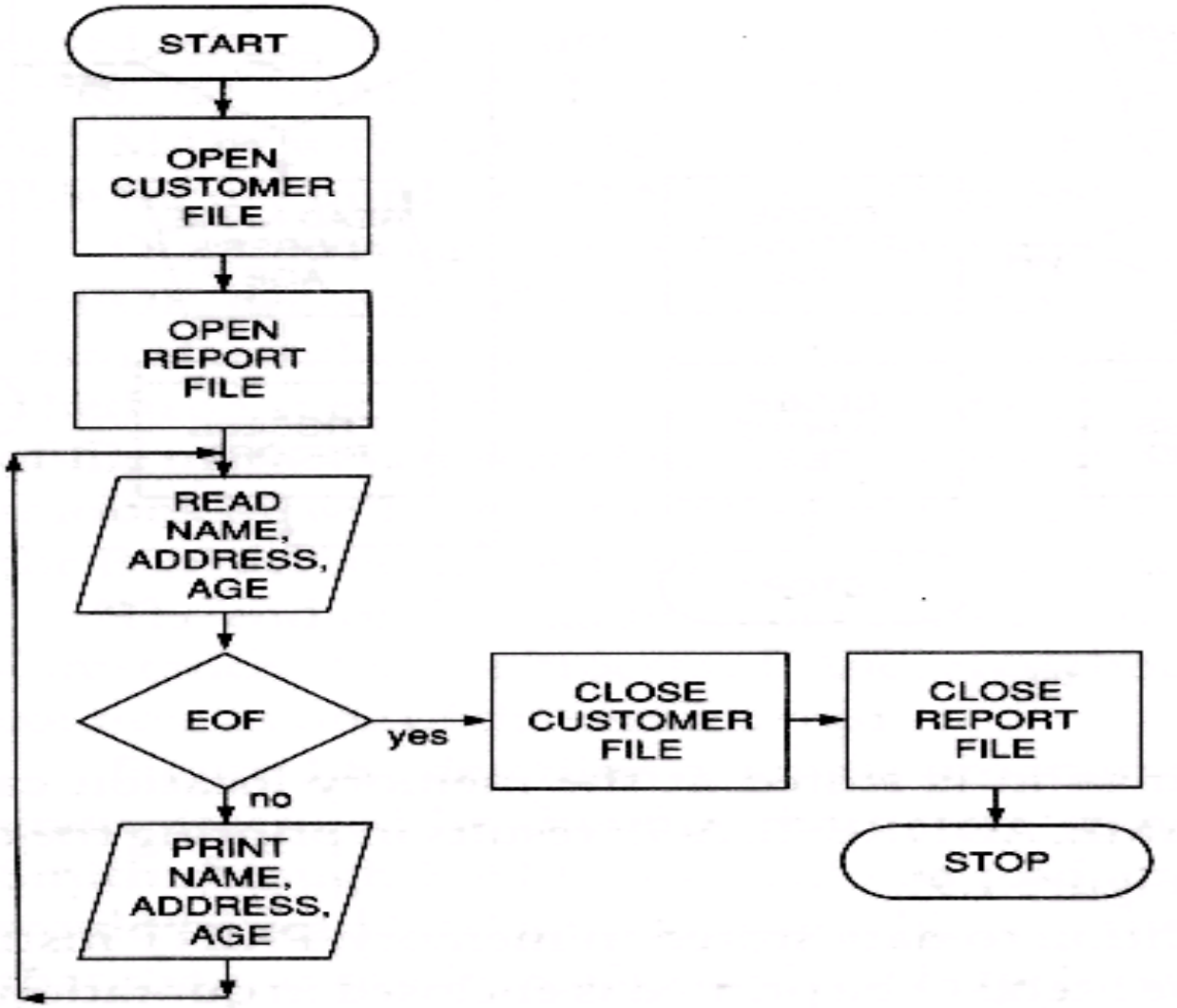


PRINT

In addition to data stored in memory, **PRINT instructions** may be used to print **literals**.

Instruction	**Line Printed**
PRINT NAME PRINT "NAME" PRINT "NAME", NAME PRINT "NAME", NAME, "AGE", AGE PRINT "MY NAME IS", NAME PRINT NAME, "IS", AGE, "."	JOE SMITH NAME NAME JOE SMITH NAME JOE SMITH AGE 47 MY NAME IS JOE SMITH JOE SMITH IS 47.

Figure describes a program that will **read records** from the customer file and **print a report** containing the information from each record.



OPEN & CLOSE Files

Note:

Files must be opened and closed.

Because OPEN and CLOSE instructions are not universally required—and especially because they pose no interesting problems in the logic of program design

PSEUDOCODE

Write pseudocode to:

- read name and gross pay from each record.
- print the name and fed W/H

If Gross Pay > 400 >>> Fed W/H = Gross Pay X 25%

Else

Fed W/H = Gross Pay X 17%

```
START
READ NAME, GROSS PAY
DO CALCULATE FED W/H LOOP WHILE NOT EOF
    IF GROSS PAY > 400
        FED W/H = GROSS PAY X 25 %
    ELSE
        FED W/H = GROSS PAY X 17 %
    END IF
    PRINT NAME, FED W/H
    READ NAME, GROSS PAY
END DO
STOP
```

Performing Arithmetic Operations

Precedence and associativity of five common operators

Operator symbol	Operator name	Associativity	Precedence (compared to other operators)
=	Assignment	Right-to-left	Lowest
+	Addition	Left-to-right	Medium
-	Subtraction	Left-to-right	Medium
*	Multiplication	Left-to-right	Highest
/	Division	Left-to-right	Highest

Example:

answer = a + b + c * d / e – f

answer = 7 + 18/6 - 2 * 5

answer = a + b + ((c * d) / e) – f

Sequence

A sequence is represented by listing a series of instructions at the same margin

```
STOTAL = 0
GTOTAL = 0
READ NAME, SALES
```

Routine Components in Programs

I. Loop

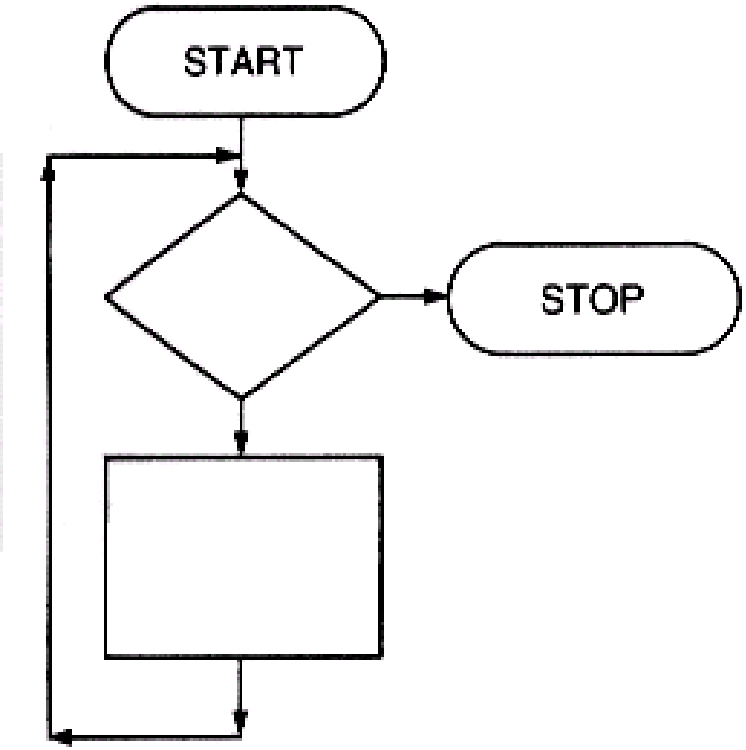
Loop is a circular logical structure.

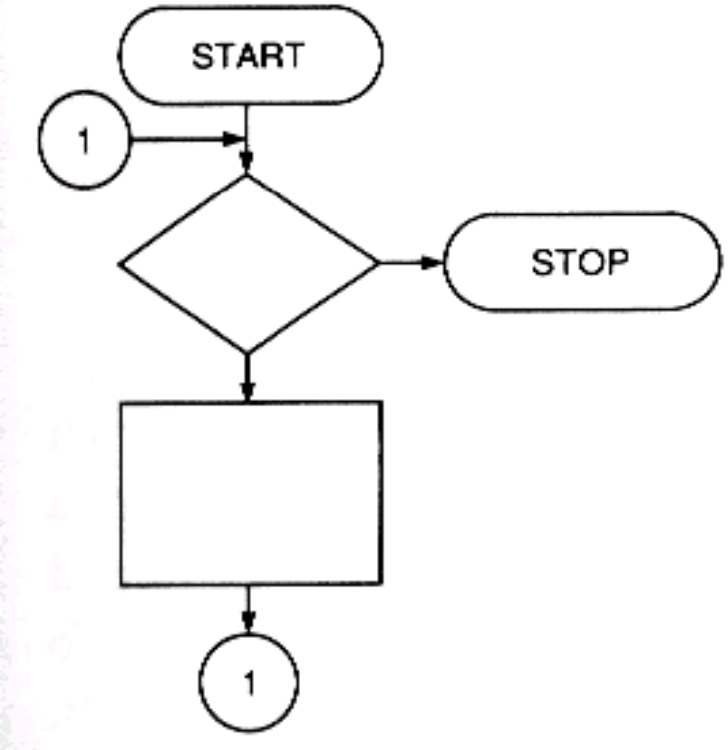
It causes a series of steps to be continuously repeated until a decision directs program flow to some other part of the program.

Loops are diagrammed in either of two ways:

1. By an upward flow direction line.
2. By a connector that returns program flow to a previous point on the chart.

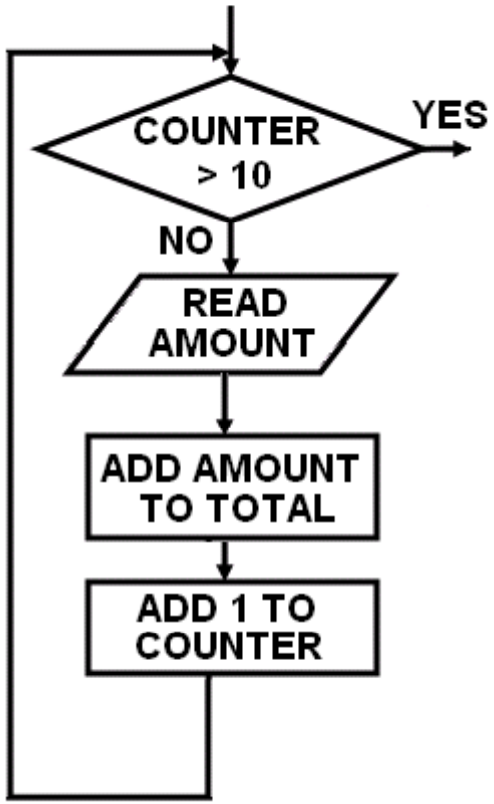
I. Loop





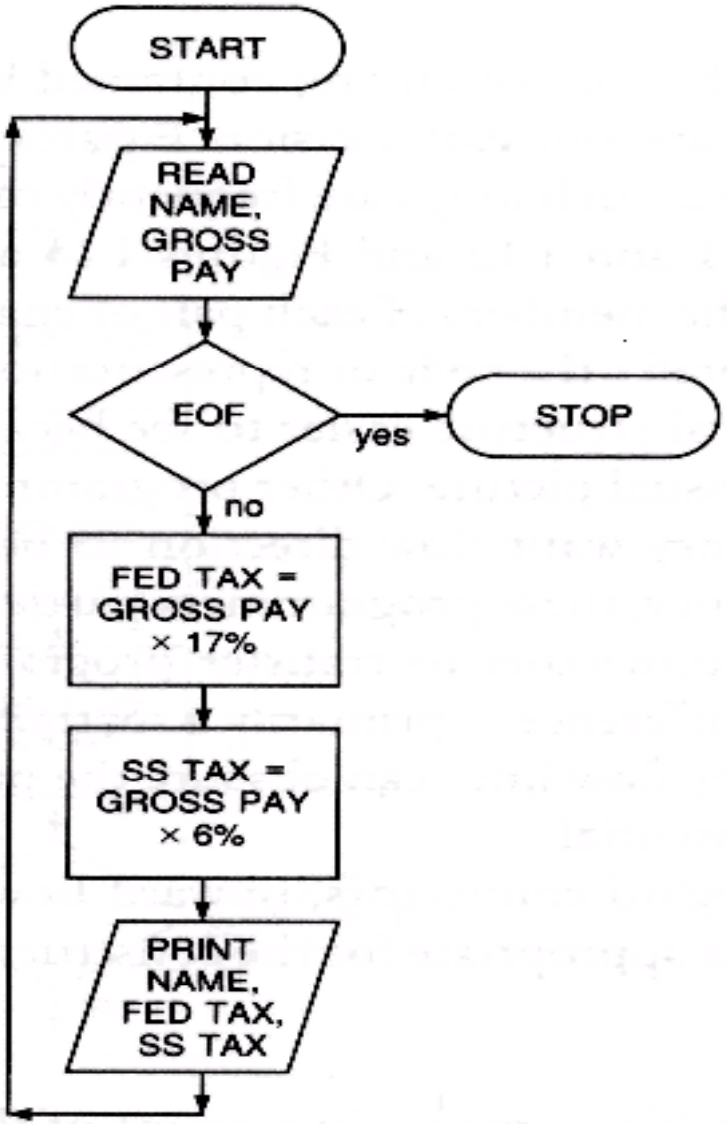
LOOPS

Loops are represented by statements that indicate the name of the loop, the type of loop (while or until), and the loop exit decision.



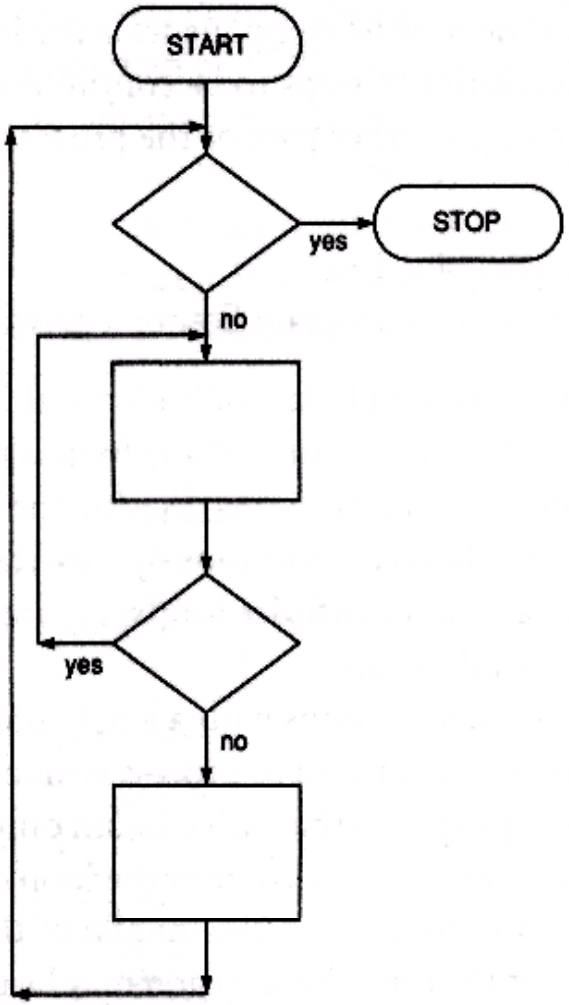

```
DO COUNT LOOP WHILE NOT (COUNT > 10)
    READ AMOUNT
    ADD AMOUNT TO TOTAL
    ADD 1 TO COUNT
END DO
```

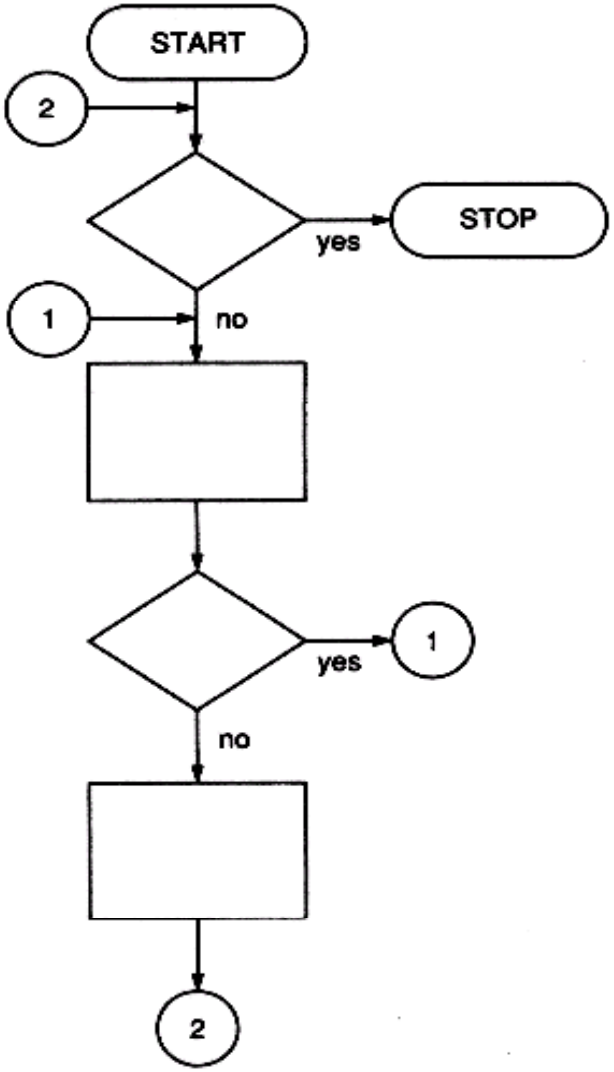
EXAMPLE of single loop containing a variety of symbols



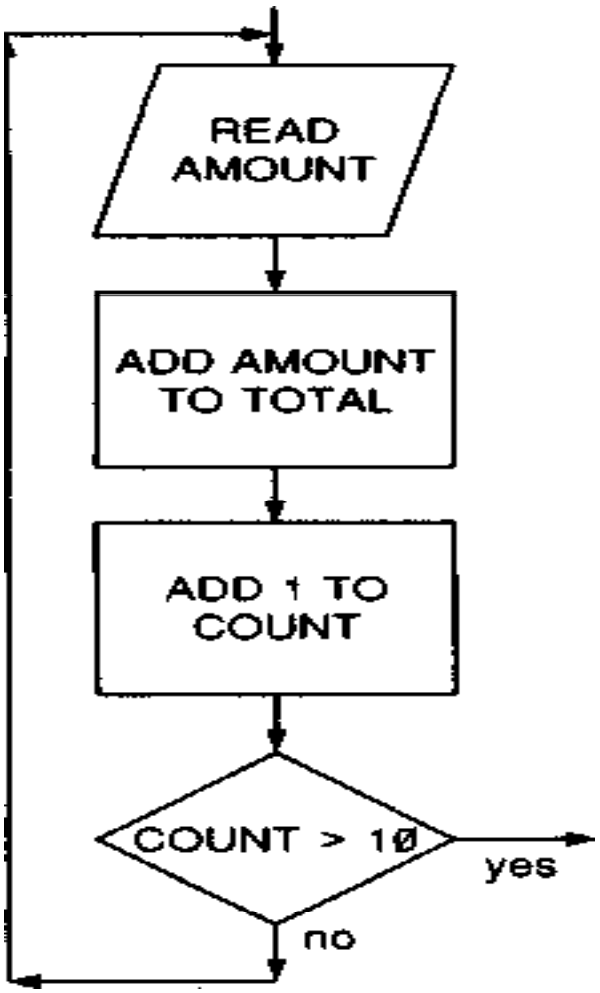
Start
Read Name, Gross Pay
DO Calc LOOP WHILE NOT EOF
Read Name, Gross Pay
Fed Tax= Gross Pay X 17%
SS Tax= Gross Pay X 6%
Print Name, Fed Tax, SS Tax
END DO

Nested loops



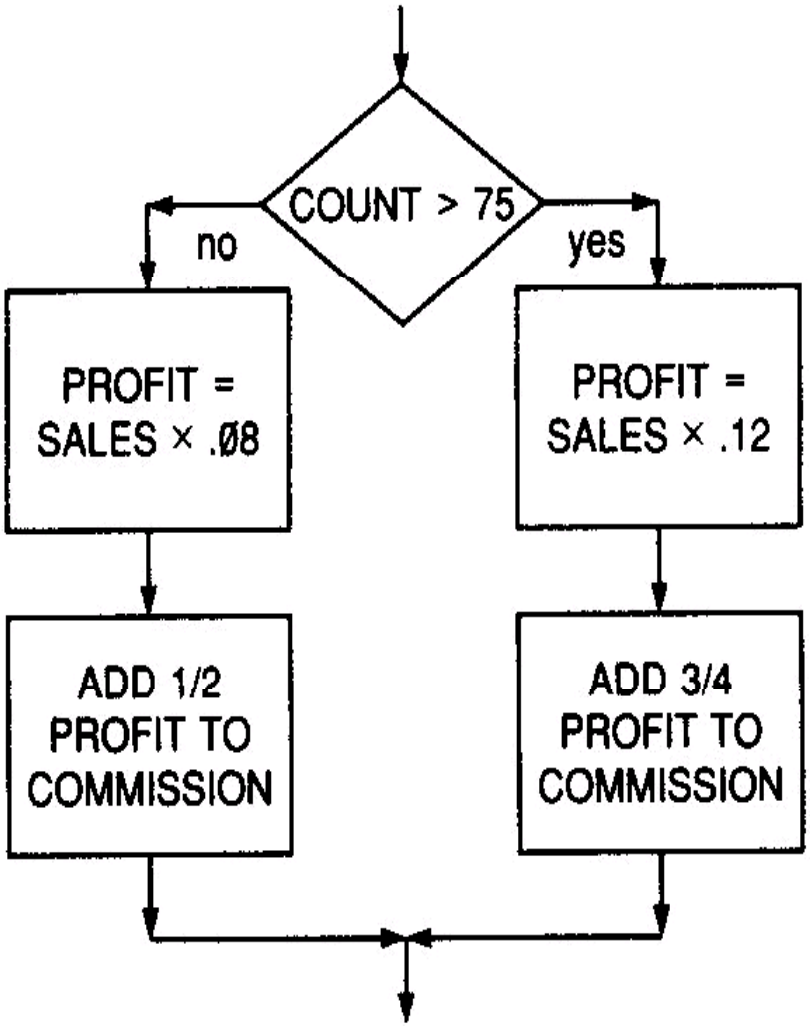


LOOPS



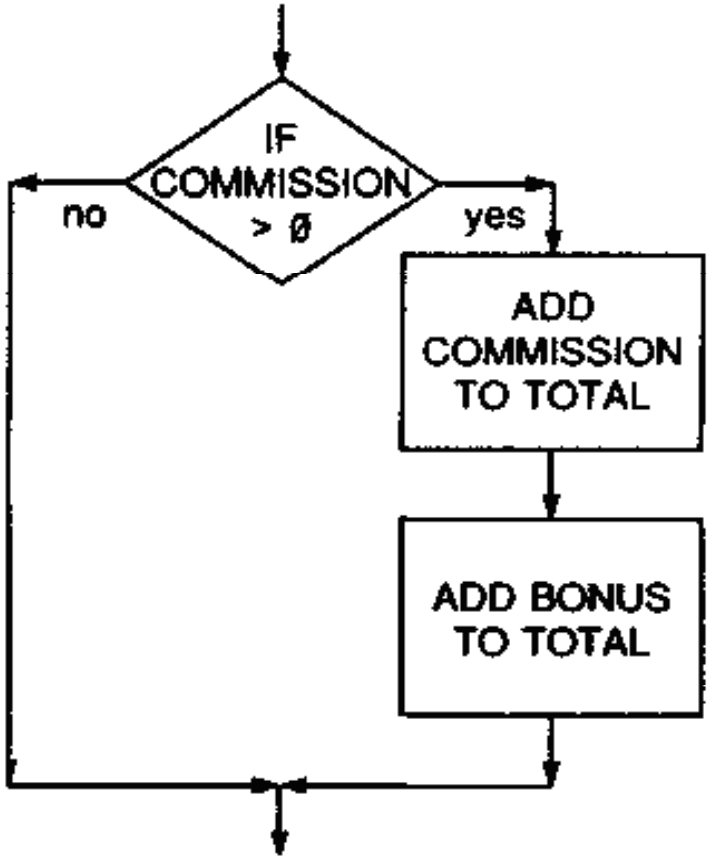

```
DO COUNT LOOP UNTIL COUNT > 10
    READ AMOUNT
    ADD AMOUNT TO TOTAL
    ADD 1 TO COUNT
END DO
```

If-Then-Else



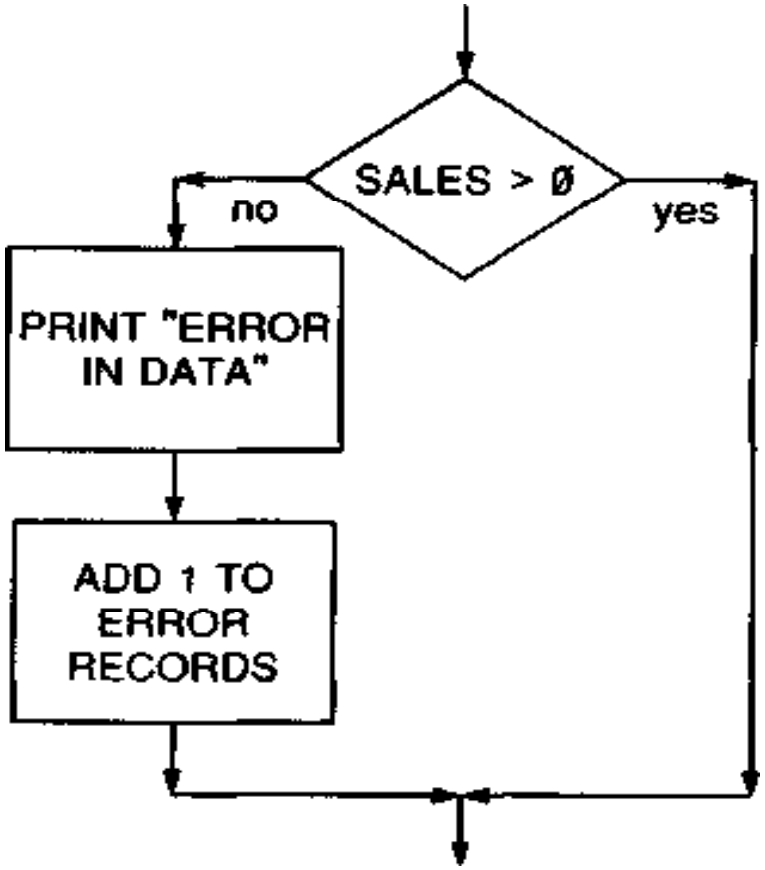

```
IF COUNT > 75
      PROFIT = SALES × .12
      ADD 3/4 PROFIT TO COMMISSION
ELSE
      PROFIT = SALES × .Ø8
      ADD 1/2 PROFIT TO COMMISSION
END IF
```

If-Then



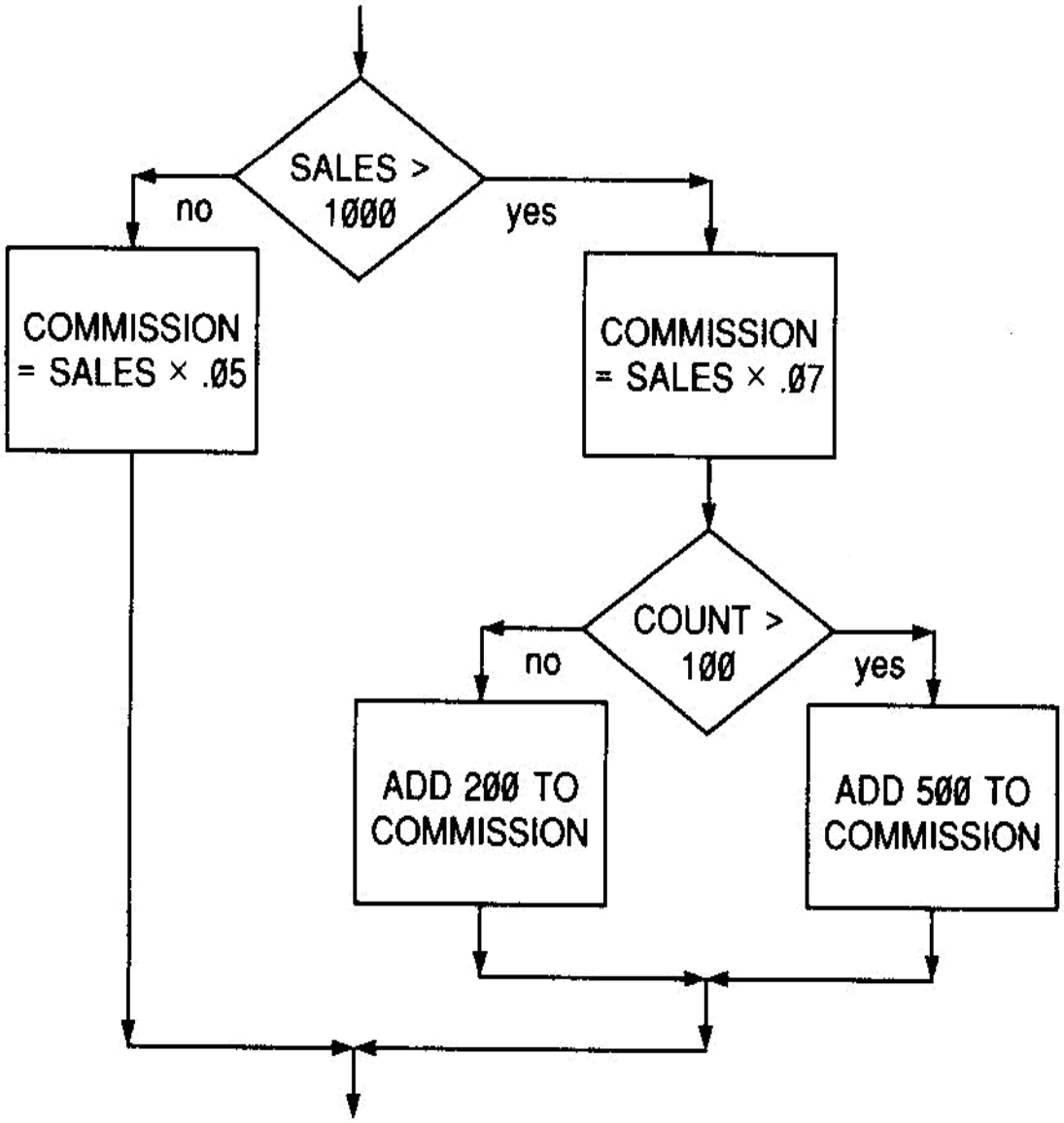

```
IF COMMISSION > 0
    ADD COMMISSION TO TOTAL
    ADD BONUS TO TOTAL
END IF
```

If-Then-Else




```
IF SALES > 0
   SKIP
ELSE
   PRINT "ERROR IN DATA"
   ADD 1 TO ERROR RECORDS
END IF
```

Nested If-Then-Elses




```
IF SALES > 1000
      COMMISSION = SALES × .07
      IF COUNT > 100
            ADD 500 TO COMMISSION
      ELSE
            ADD 200 TO COMMISSION
      END IF
ELSE
      COMMISSION = SALES × .05
END IF
```

Routine Components in Programs

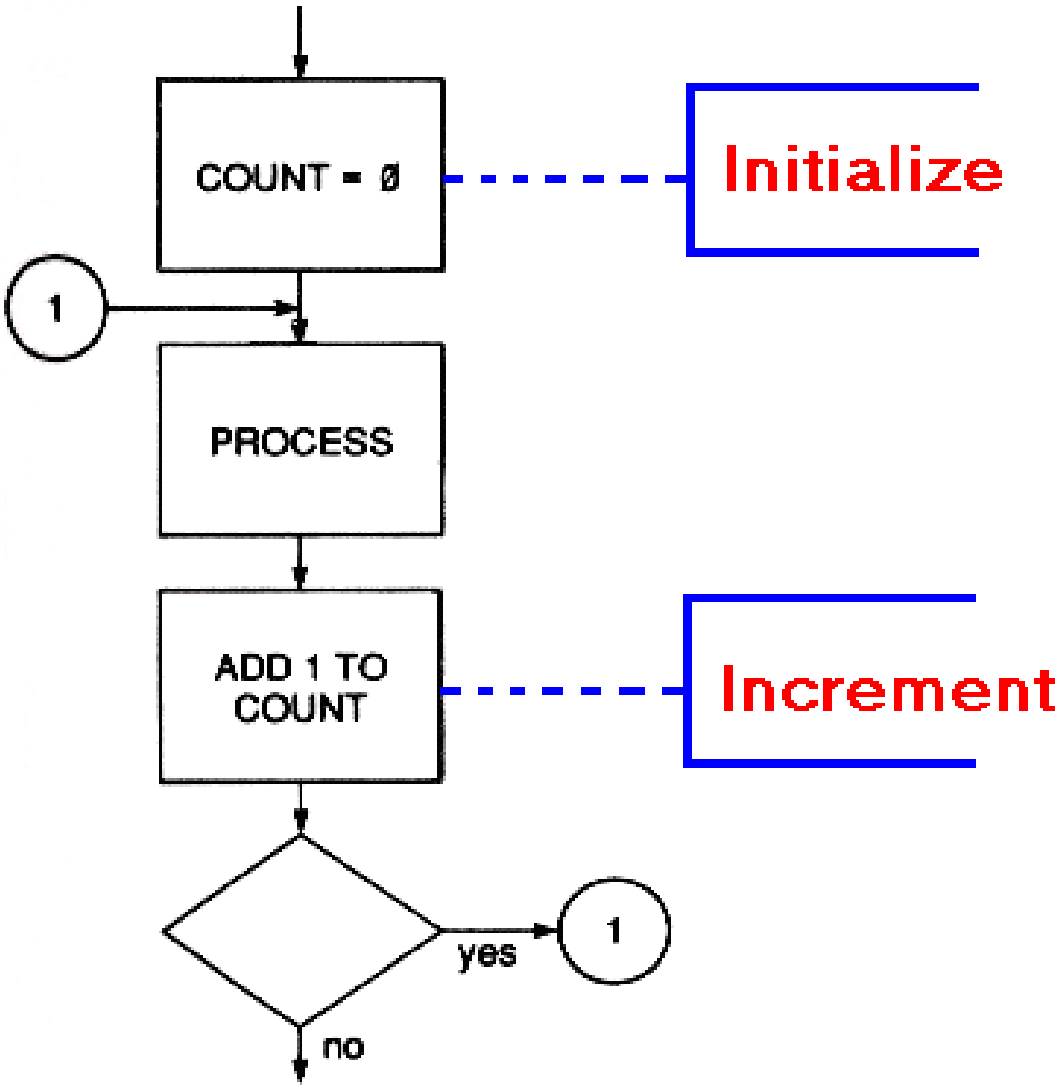
II. COUNTER

count of the number of times a process within a loop has been completed.

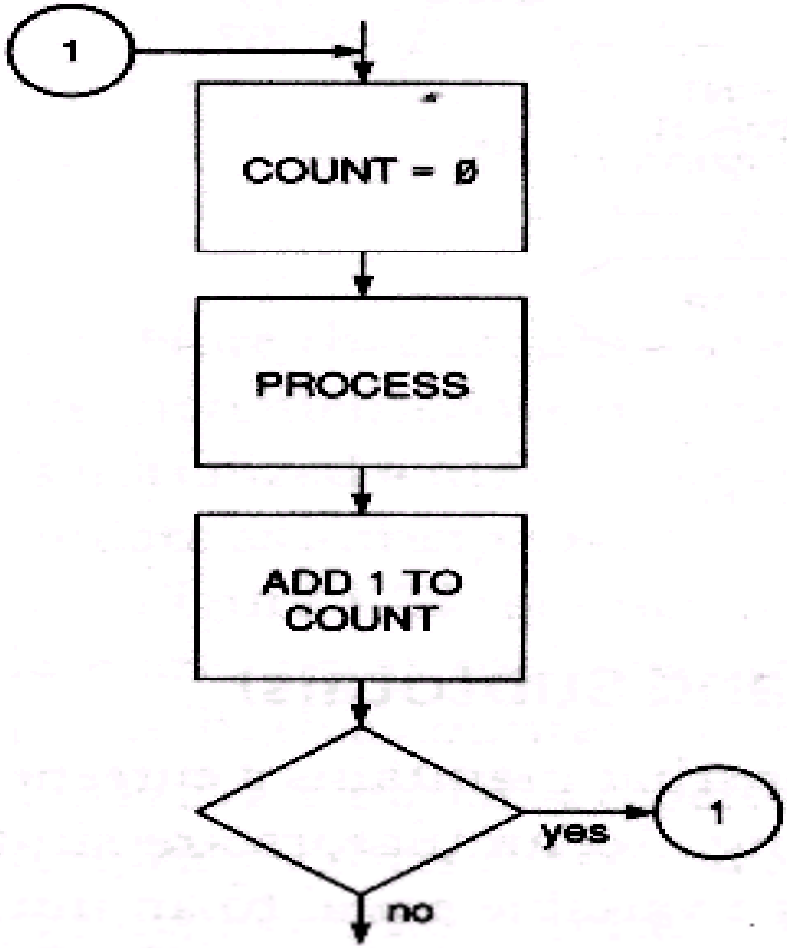
A **counter consists of two processes** that manipulate a variable:

- initializes the variable
- The second adds/subtract a constant to (or **increments/decrement**) the variable each time the loop is completed.

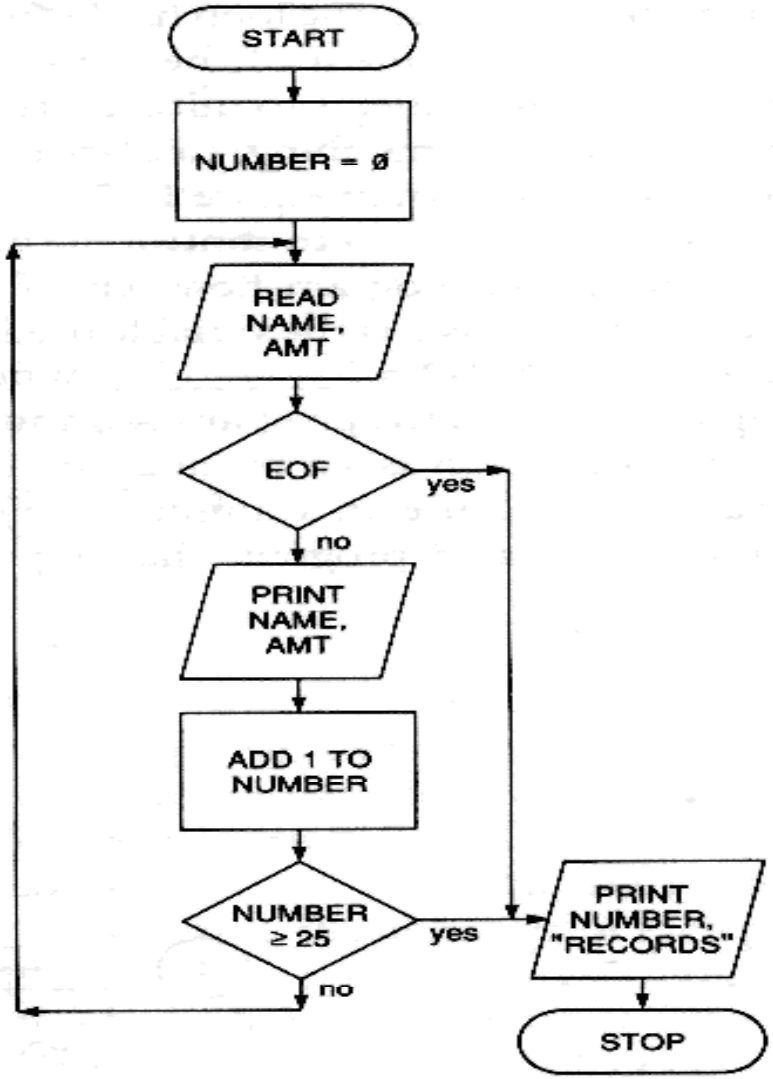
Example of flowchart contains a counter



The flowchart illustrates the error of initializing the variable within the loop itself.



The flowchart prints a report of records in a file called "SALES FILE."



FLOWCHART

SALES FILE	
JILL JOHNSON	2110.ØØ
FRANK JONES	4117.ØØ
THOMAS JACOBS	695.65
JAMES FORD	18319.3Ø
EDWARD EDISON	987.65
MARY SMITH	2745.6Ø
SUSAN CRAMER	3165.ØØ
/*	

SAMPLE INPUT DATA

JILL JOHNSON	2,110.ØØ
FRANK JONES	4,117.ØØ
THOMAS JACOBS	695.65
JAMES FORD	18,319.3Ø
EDWARD EDISON	987.65
MARY SMITH	2,745.6Ø
SUSAN CRAMER	3,165.ØØ
7 RECORDS	

SAMPLE OUTPUT

Routine Components in Programs

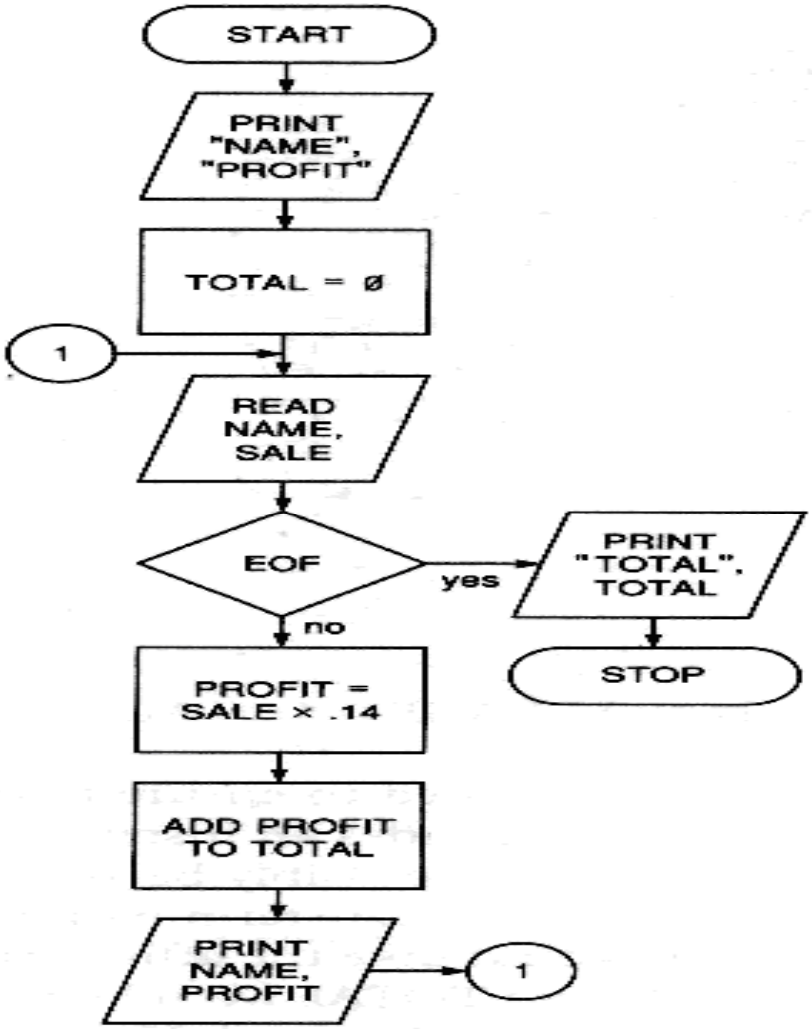
III. An accumulator

is a structure that maintains a current total without affecting ongoing calculations or processing.

- Accumulators are similar to counters in that both consist of a process that sets a variable equal to an initial value and another process that, increments that variable.

- Accumulators differ from counters in that they are incremented by variable amounts rather than by constants.

The flowchart represents a simple program that contains an accumulator called TOTAL. In this flowchart, the profit for each item is accumulated in TOTAL. When EOF is reached, this total is printed.



FLOWCHART

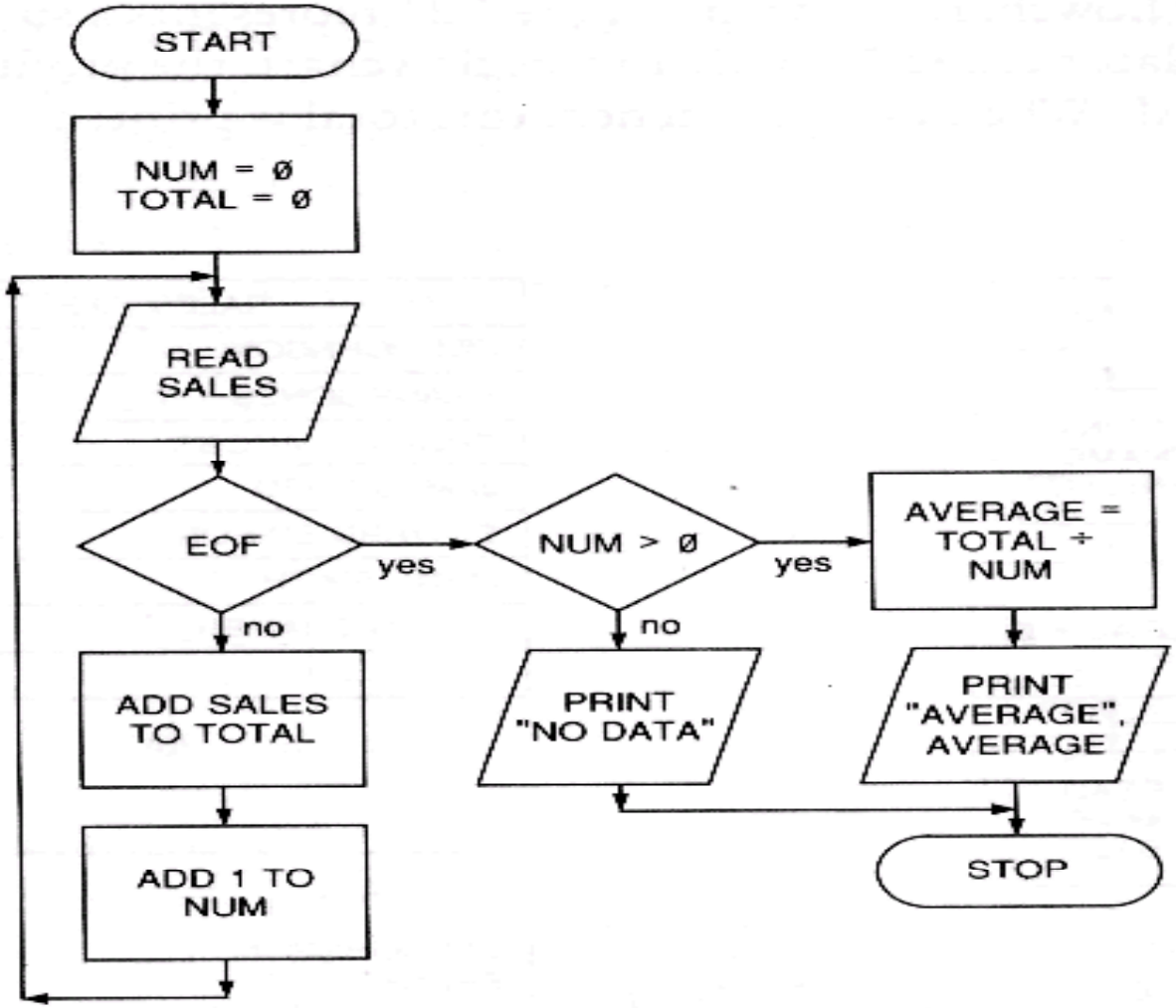
SALES FILE	
JILL JOHNSON	2110.00
FRANK JONES	4117.00
THOMAS JACOBS	695.65
JAMES FORD	18319.30
EDWARD EDISON	987.65
MARY SMITH	2745.60
SUSAN CRAMER	3165.00

SAMPLE INPUT DATA

NAME	PROFIT
JILL JOHNSON	295.40
FRANK JONES	576.38
THOMAS JACOBS	97.39
JAMES FORD	2,564.70
EDWARD EDISON	138.27
MARY SMITH	384.38
SUSAN CRAMER	443.10
TOTAL	4,499.62

SAMPLE OUTPUT

Flowchart (uses both a **counter and an accumulator**) represents a program that will read a series of sales and calculate and print their average.



Routine Components in Programs

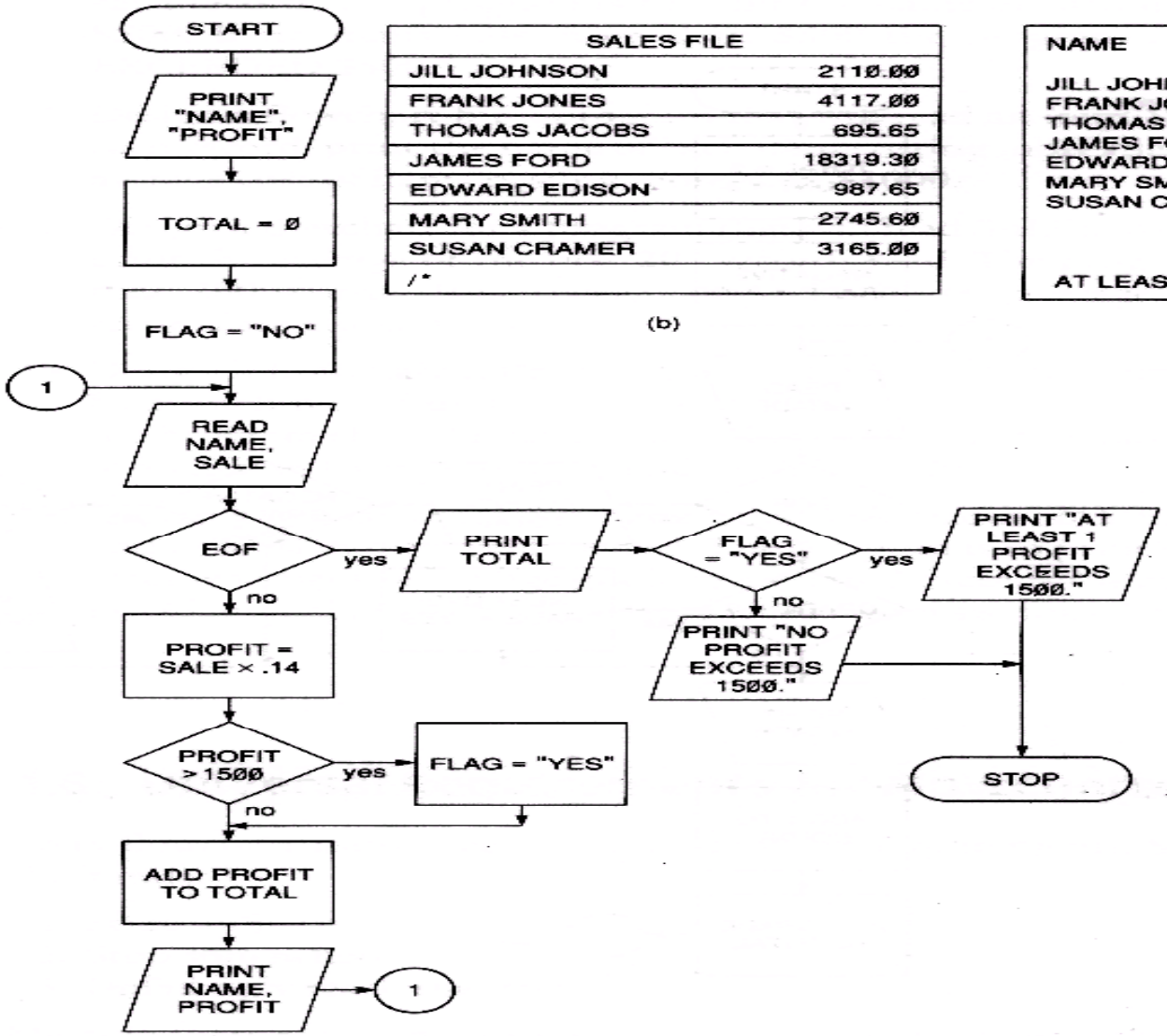
IV. indicators (Switches or Flags)

Indicators, often called **switches or flags,** are used to record some event or circumstance within a program.

An indicator consists of two elements:

- A process that initializes a variable (sets that variable at a predetermined value).
- A process that stores a different predetermined value as the value of that variable when the particular event occurs.

A message would be printed at the end of the program to indicate whether or not *any* of the profits exceeded 1500.



SALES FILE	
JILL JOHNSON	211ø.øø
FRANK JONES	4117.øø
THOMAS JACOBS	695.65
JAMES FORD	18319.3ø
EDWARD EDISON	987.65
MARY SMITH	2745.6ø
SUSAN CRAMER	3165.øø
/*	

(b)

NAME	PROFIT
JILL JOHNSON	295.4ø
FRANK JONES	576.38
THOMAS JACOBS	97.39
JAMES FORD	2,564.7ø
EDWARD EDISON	138.27
MARY SMITH	384.38
SUSAN CRAMER	443.1ø
	4,499.62

AT LEAST 1 PROFIT EXCEEDS 1,5øø.

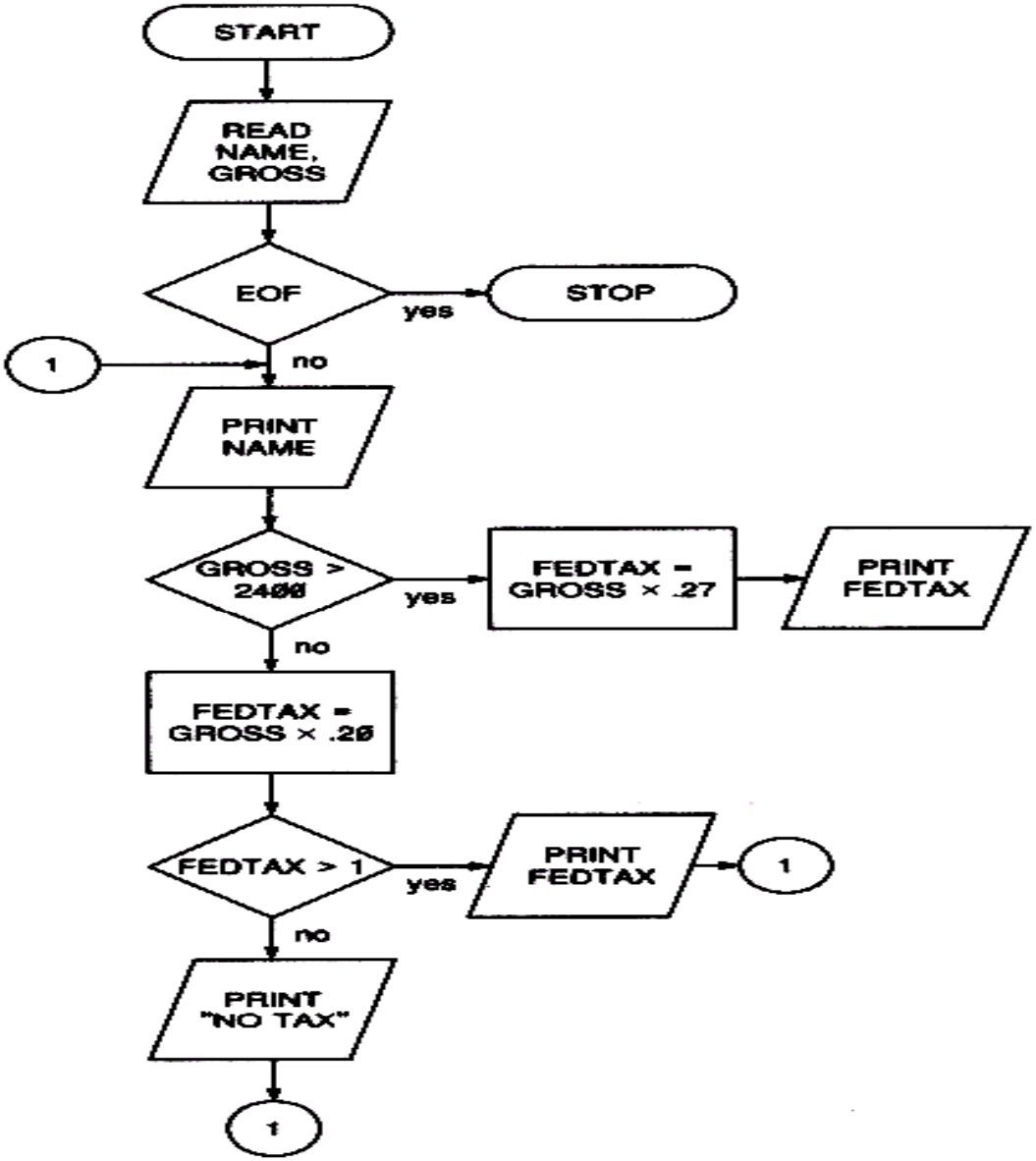
(c)

Universal Requirements for Flowcharts

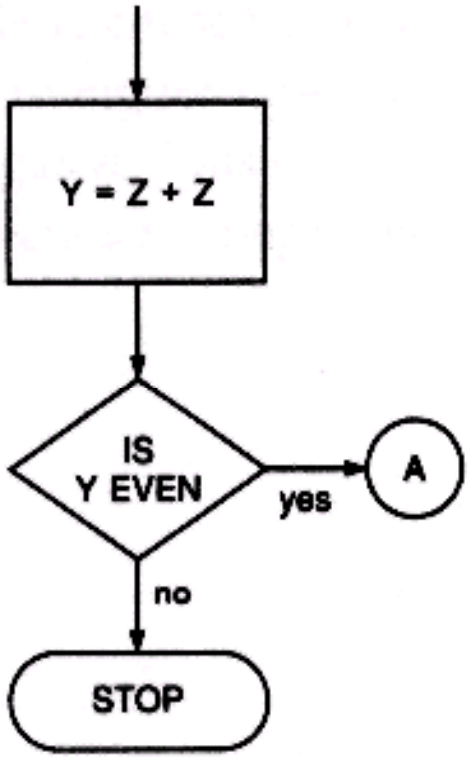
Every flowchart must

1. Start somewhere.
2. Stop somewhere.
3. Unfailingly reach the stop.

This flowchart **illustrates two common errors** in flowcharting.



This flowchart contains an endless loop that results from its content

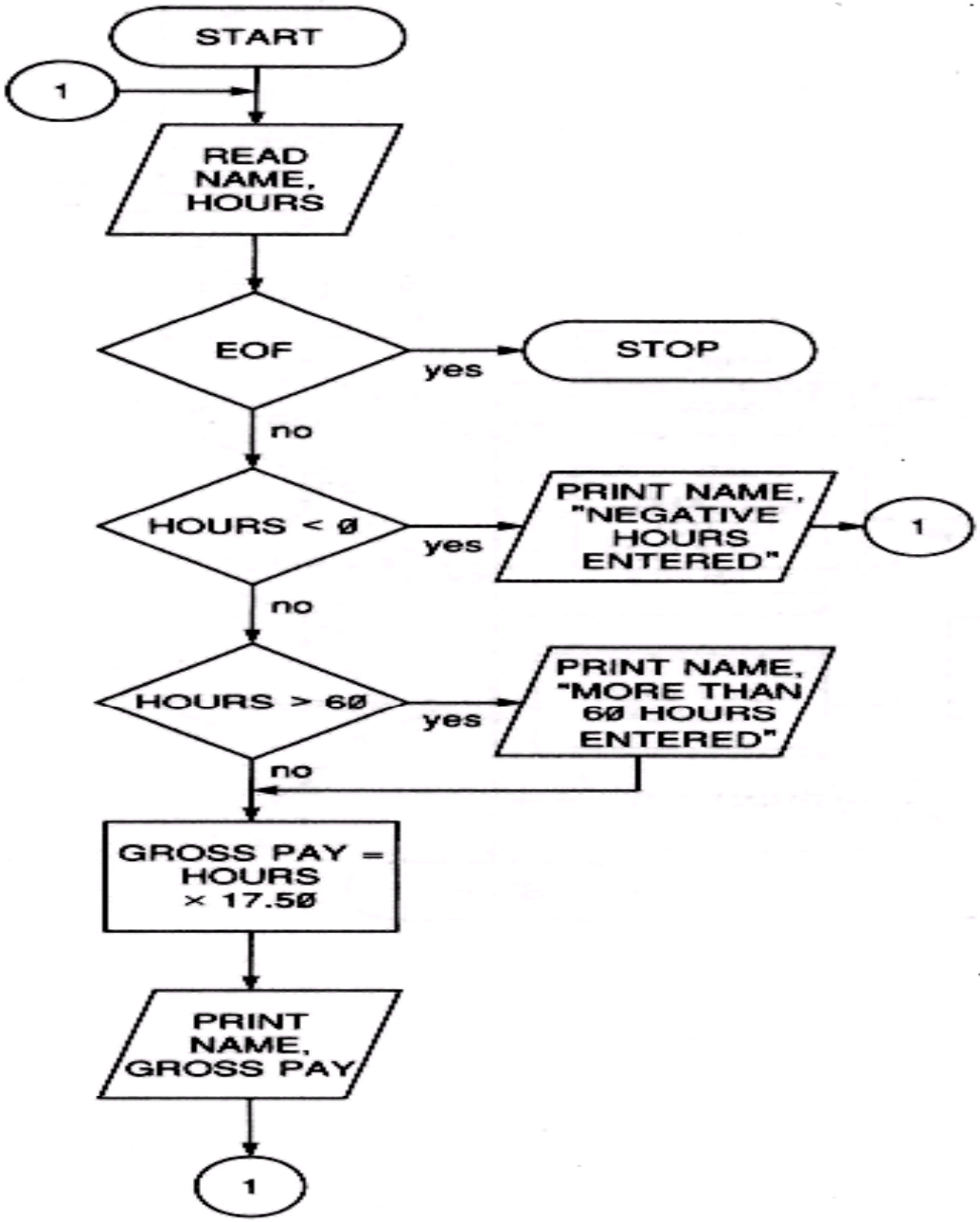


Error Messages

When the data used does not meet the programmer's expectations, the program may not run. Or, if it runs, it produces output that suggests a programming error.

EXAMPLE

Figure will read a name and a number of hours worked from each record. It will also print the name and gross pay and will print an error message if the number of hours worked is less than zero or greater than 60.


START
1
READ NAME, HOURS
EOF
yes
STOP
no
HOURS < ø
yes
PRINT NAME, "NEGATIVE HOURS ENTERED"
1
no
HOURS > 6ø
yes
PRINT NAME, "MORE THAN 6ø HOURS ENTERED"
no
GROSS PAY = HOURS × 17.5ø
PRINT NAME, GROSS PAY
1

Program Design

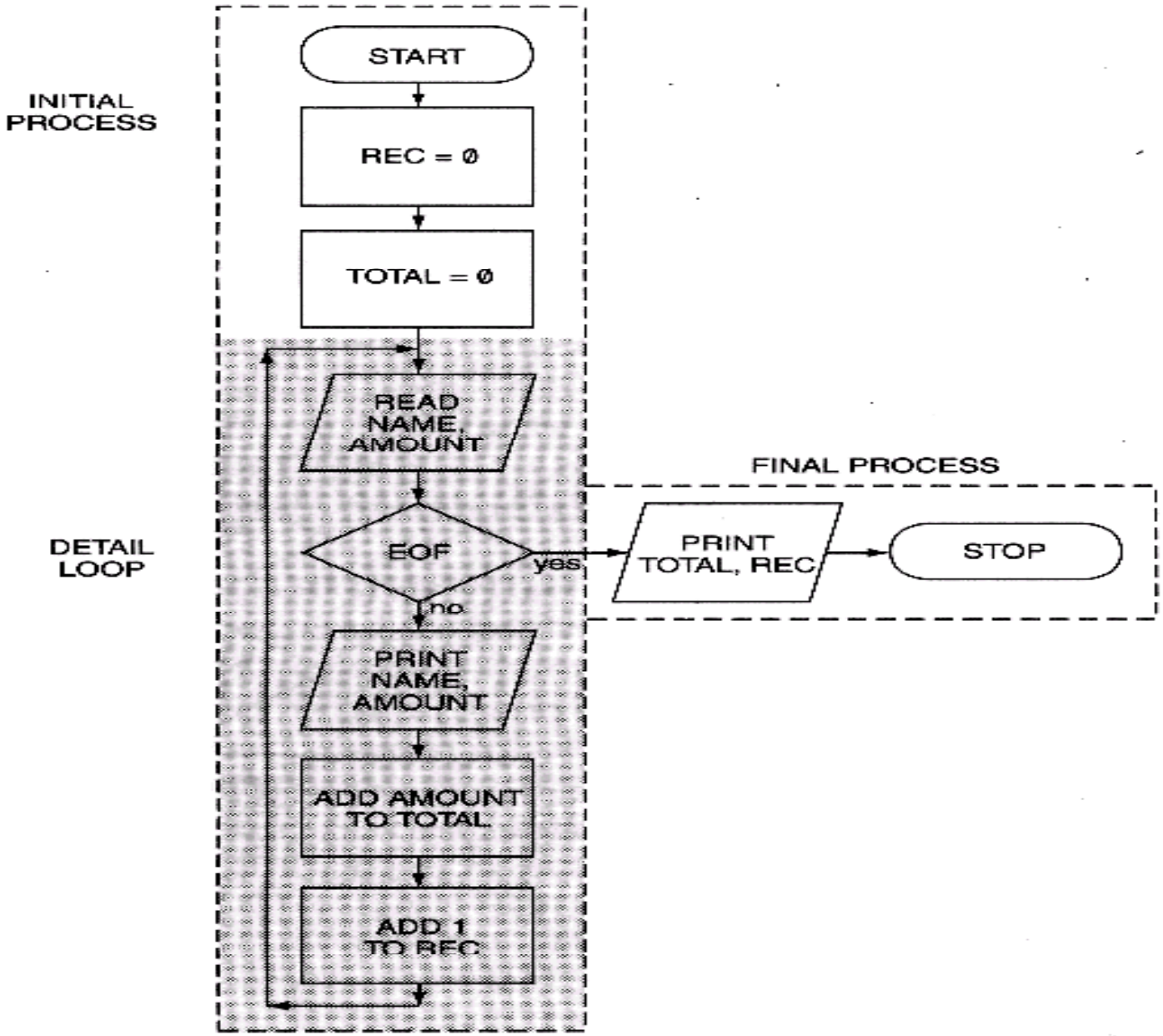
Programs of this sort can be analyzed into three parts: **initial process, detail loop, and final process.**

INITIAL PROCESS	OPEN FILES PRINT HEADINGS (TITLES AND DATES) INITIALIZE COUNTERS INITIALIZE ACCUMULATORS INITIALIZE INDICATORS
DETAIL LOOP	READ AND LOOP EXIT INCREMENT COUNTERS INCREMENT ACCUMULATORS PERFORM CALCULATIONS PRINT DETAIL LINES (INDIVIDUAL NAMES, ETC.)
FINAL PROCESS	PERFORM FINAL CALCULATIONS PR I NT TOTALS CLOSE FILES

EXAMPLE

Design a flowchart for a program that will

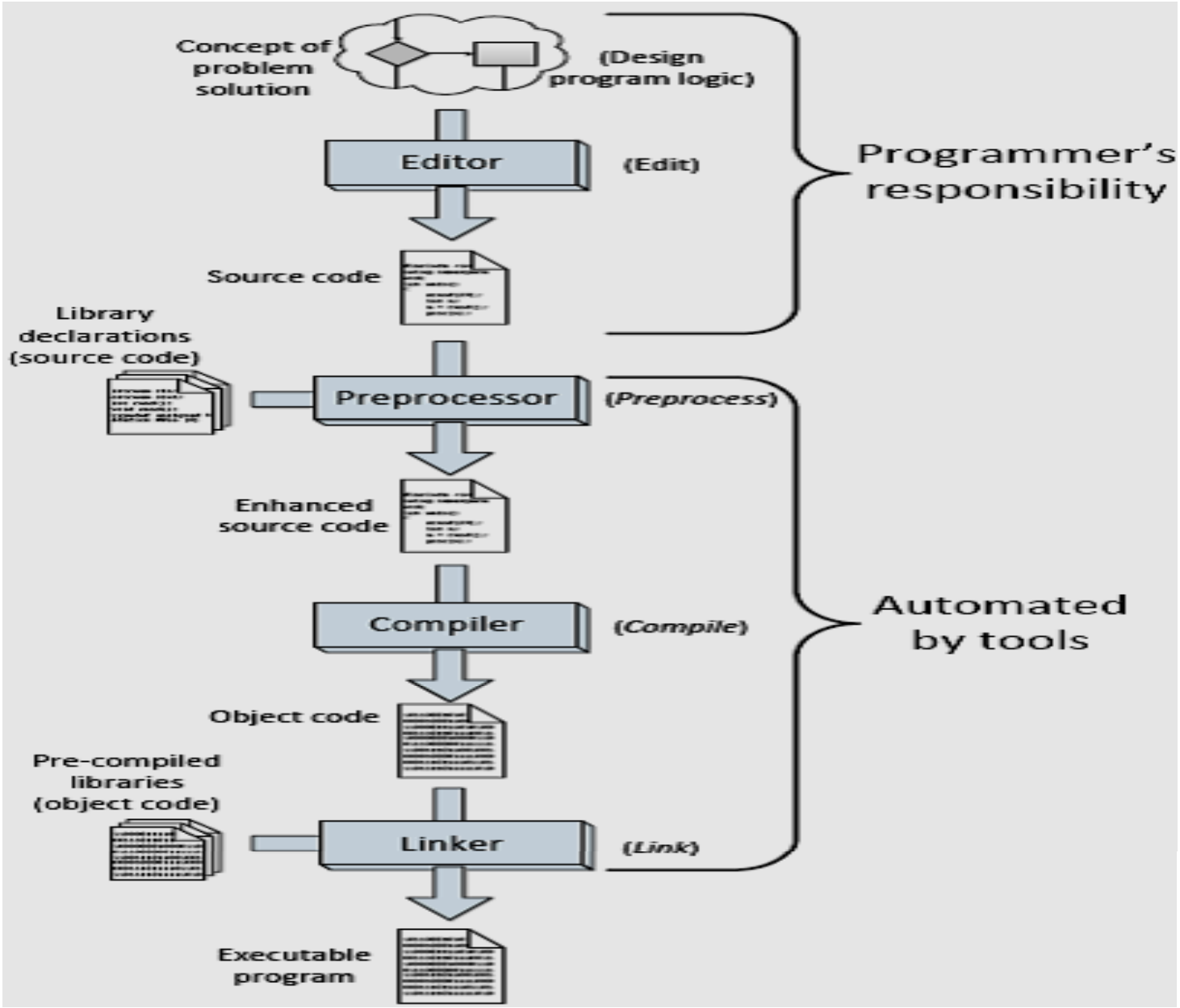
- Read names and amounts.
- Print each name and amount.
- Print a total of the amounts and the number of records.


INITIAL PROCESS
START
REC = 0
TOTAL = 0
READ NAME, AMOUNT
DETAIL LOOP
EOF
yes
no
PRINT NAME, AMOUNT
ADD AMOUNT TO TOTAL
ADD 1 TO REC
FINAL PROCESS
PRINT TOTAL, REC
STOP

Writing a C++ Program

- Many developers use integrated development environments (IDEs).
- An IDE includes editors, debuggers, and other programming aids in one comprehensive program.

IDEs for C++


Concept of problem solution
(Design program logic)
Editor
(Edit)
Source code
Library declarations (source code)
Preprocessor
(Preprocess)
Enhanced source code
Compiler
(Compile)
Object code
Pre-compiled libraries (object code)
Linker
(Link)
Executable program
Programmer's responsibility
Automated by tools

Writing a C++ Program

```
#include <iostream>

using namespace std;

int main() {
    cout << "This is a simple C++ program!" << endl;
}
```

```
This is a simple C++ program!
```

Writing a C++ Program

- **#include <iostream>**
  - This line is a preprocessing directive. All preprocessing directives within C++ source code begin with a **#** symbol. This one directs the preprocessor to add some predefined source code to our existing source code before the compiler begins to process it.
- The **iostream** library contains routines that handle input and output (I/O)
  - that include functions such as printing to the display, getting user input from the keyboard, and dealing with files.
  - The **#include** directive specifies a file, called a header, that contains the specifications for the library code.

Writing a C++ Program

- **using namespace std;**
  - The two items our program needs to display a message on the screen, **cout** and **endl**, have longer names: **std::cout** and **std::endl**.
  - This **using namespace std** directive allows us to omit the **std::** prefix and use their shorter names.
- **int main() {**
  - This specifies the real beginning of our program. Here we are declaring a function named **main**. All C++ programs must contain this function to be executable.
- The opening curly brace **{** at the end of the line marks the beginning of the body of a function.
- **cout << "This is a simple C++ program!"<< endl;**
  - All statements in C++ end with a semicolon (;).

Template for simple C++ programs

```
#include <iostream>

using namespace std;

int main()  {

        program statements

}
```

- The **namespace std** directive isn't used in the following C++ program

```
#include <iostream>

int main() {
    std::cout << "This is a simple C++ program!" << std::endl;
}
```

Expressions

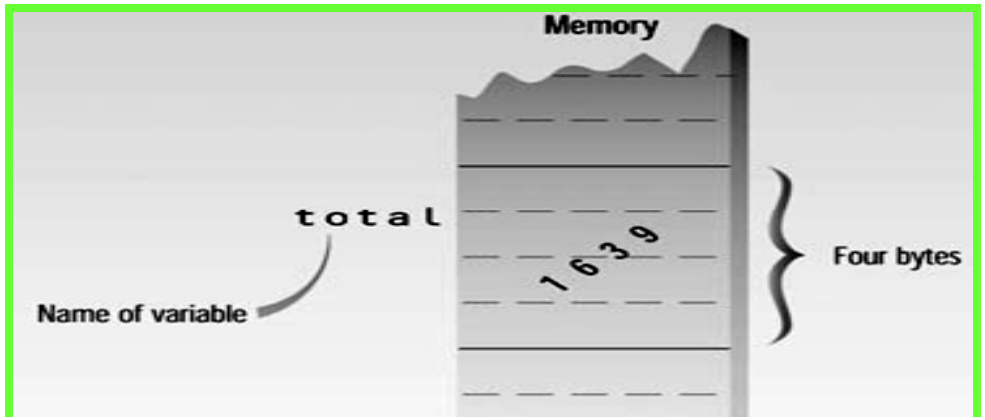
```
#include <iostream>

using namespace std;

int main() {
    int value1, value2, sum;
    cout << "Please enter two integer values: ";
    cin >> value1 >> value2;
    sum = value1 + value2;
    cout << value1 << " + " << value2 << " = " << sum << endl;
}
```

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;
```

Example

- C++ program to convert a temperature from degrees Fahrenheit to degrees Celsius using the formula

$$°C = \frac{5}{9} \times (°F - 32)$$

```
//  File tempconv.cpp

#include <iostream>

using namespace std;

int main() {
    double degreesF, degreesC;
    //  Prompt user for temperature to convert
    cout << "Enter the temperature in degrees F: ";
    //  Read in the user's input
    cin >> degreesF;
    //  Perform the conversion
    degreesC = 5/9*(degreesF - 32);
    //  Report the result
    cout << degreesC << endl;
}
```

```
Enter the temperature in degrees F: 32
Degrees C = 0
```

CMATH Header Files

- # include a header file that contains the declaration of any library functions you use.
  - In the documentation for the sqrt() function, you'll see that the specified header file is CMATH.
- In SQRT the preprocessor directive

  #include <cmath>

Library Function

```
/*  ------------------------------------------------
demonstrates sqrt() library function
Date : 26/9/2016
I / P : number
O/P: Square root  of i/p No.
------------------------------------------------- */
#include <iostream>                    //for cout, etc.
#include <cmath>                       //for sqrt()
using namespace std;
int main() { double number, answer;     //sqrt() requires type double
    cout << "Enter a number: ";
    cin >> number;                              //get the number
    answer = sqrt(number);                      //find square root
    cout << "Square root is " << answer << endl;        //display it
return 0;
}
```

```
Enter a number: 1000
Square root is 31.622777
```