Programming and Data Structure

Sudeshna Sarkar

Dept. of Computer Science & Engineering.

Indian Institute of Technology

Kharagpur

13 Jan 2012

Assignment Statement

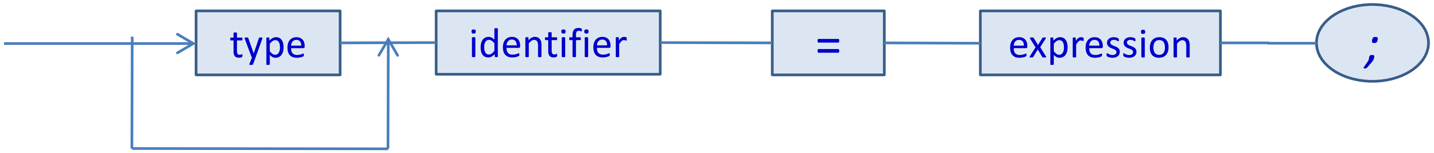
- Used to assign values to variables, using the assignment operator (=).
- General syntax:

```
        variable_name  =  expression;
  type  variable_name = expression;
```

- Examples:

```
int velocity = 20;
b = 15;  temp = 12.5;
A = A + 10;
v = u + f * t;
s = u * t + 0.5 * f * t * t;
```

lvalue and assignment operator



- Requires an **lvalue** as its left operand.
- l-value: represents an object stored in memory, which is neither a constant nor a result of computation.
- So a variable can be an lvalue, but neither any expressions nor any constant.

```
12 = i ;          // WRONG
i + j = 0 ;       // WRONG
–i = j ;            // WRONG
i++  = j ;         // WRONG
X+10 = Y∗2;  // WRONG
```

Assigning values to variables

- Lhs = rhs
- Rhs is an expression compatible with the type of the lhs

```
centigrade = 5*(fahrenheit – 32)/9;
```

- Assignment statement has value = rhs
- A value can be assigned to a variable at the time the variable is declared.

```
int   speed = 30;
char  flag = ‘y’;
```

Contd.

- Several variables can be assigned the same value using multiple assignment operators.

```
a = b = c = 5;
flag1 = flag2 = ‘y’;
speed = flow = 0.0;
```

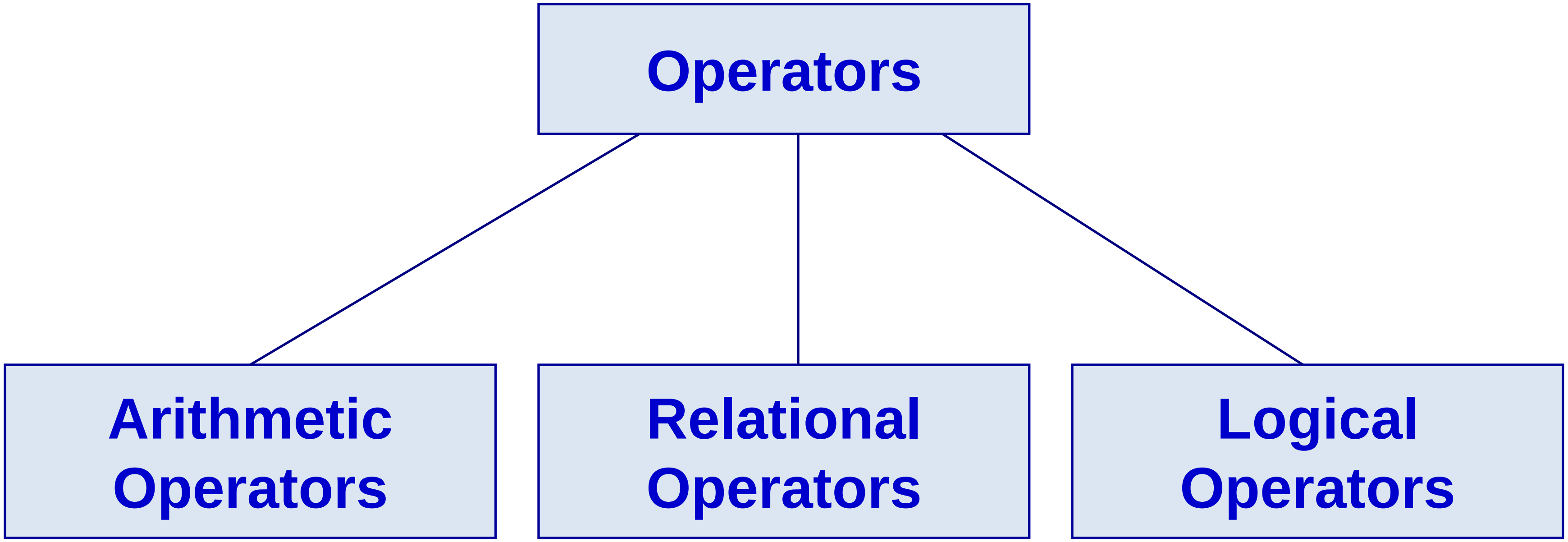
Example: swapping two numbers

```
float x = 5, y = 11;
float  temporary ;
temporary = x;
x = y;
y = temporary;
```

Can you swap without using a temporary variable?

x	y	temporary
5	11	
5	11	5
11	11	5
11	5	5

Operators in Expressions



Arithmetic Operators

• Addition ::	+
• Subtraction ::	–
• Division ::	/
• Multiplication ::	*
• Modulus ::	%

Examples

```
distance = rate ∗ time ;
netIncome = income – tax ;
speed = distance / time ;
area = PI ∗ radius  ∗ radius;
y = a ∗ x ∗ x + b ∗ x + c;
quotient = dividend / divisor;
remain =dividend % divisor;
```

Contd.

- Suppose x and y are two integer variables, whose values are 13 and 5 respectively.

x + y	18
x – y	8
x * y	65
x / y	2
x % y	3

Operator Precedence

- In decreasing order of priority
  1. Parentheses :: ( )
  2. Unary minus :: –5
  3. Multiplication, Division, and Modulus
  4. Addition and Subtraction
- For operators of the *same priority*, evaluation is from *left to right* as they appear.
- Parenthesis may be used to change the precedence of operator evaluation.

Examples: Arithmetic expressions

$$a + b \ast c - d \,/\, e \equiv a + (b \ast c) - (d \,/\, e)$$

$$a \ast - b + d \,\%\, e - f \equiv a \ast (- b) + (d \,\%\, e) - f$$

$$a - b + c + d \equiv (((a - b) + c) + d)$$

$$x \ast y \ast z \equiv ((x \ast y) \ast z)$$

$$a + b + c \ast d \ast e \equiv (a + b) + ((c \ast d) \ast e)$$

Integer Arithmetic

- When the operands in an arithmetic expression are integers, the expression is called *integer expression*, and the operation is called *integer arithmetic*.
- Integer arithmetic always yields integer values.

Real Arithmetic

- Arithmetic operations involving only real or floating-point operands.
- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.

  1.0 / 3.0 * 3.0 will have the value 0.99999 and not 1.0
- The modulus operator cannot be used with real operands.

Mixed-mode Arithmetic

- When one of the operands is integer and the other is real, the expression is called a *mixed-mode* arithmetic expression.
- If either operand is of the real type, then only real arithmetic is performed, and the result is a real number.

  25 / 10 → 2

  25 / 10.0 → 2.5
- Some more issues will be considered later.

- Mixing types may result in precision loss, overflow, underflow and ability to process full range.

Problem of value assignment

- Assignment operation

  variable= expression_value;

  or

  variable1 = variable2;

Data type of the RHS should be compatible with that of LHS.

If a floating point number is assigned to an integer variable, there will be truncation, may lead to loss.

Type Casting

```
int a=10, b=4, c;
float x, y;

c = a / b;
x = a / b;
y = (float) a / b;
```

The value of c will be 2

The value of x will be 2.0

The value of y will be 2.5

Type Casting

```
int x;
float r=3.0;

x= (int)(2*r);
```

Type casting of a floating point expression to an integer variable.

```
double perimeter;
float pi=3.14;
int r=3;

perimeter=2.0* (double) pi * (double) r;
```

Type casting to double

Relational Operators

- Used to compare two quantities.

<	**is less than**
>	**is greater than**
<=	**is less than or equal to**
>=	**is greater than or equal to**
==	**is equal to**
!=	**is not equal to**

Examples

$10 > 20$	is false
$25 < 35.5$	is true
$12 > (7 + 5)$	is false

- When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared.

  $a + b > c - d$ is the same as $(a+b) > (c+d)$

Examples

- Sample code segment in C

```
if  (x > y)
   printf (“%d is larger\n”, x);
else
  printf (“%d is larger\n”, y);
```

Logical Operators

- There are two logical operators in C (also called logical connectives).

  && ➔ Logical AND

  || ➔ Logical OR

  – They act upon operands that are themselves logical expressions.
  – The individual logical expressions get combined into more complex conditions that are true or false.

- Logical AND
  - Result is true if both the operands are true.
- Logical OR
  - Result is true if at least one of the operands are true.

X	Y	X && Y	X || Y
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

Input / Output

- printf
  - Performs output to the standard output device (typically defined to be the screen).
  - It requires a format string in which we can specify:
    - The text to be printed out.
    - Specifications on how to print the values.

```
printf ("The number is %d.\n", num) ;
```

    - The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d.
    - Output will appear as: **The number is 125.**

Input

- **scanf**
  - Performs input from the standard input device, which is the keyboard by default.
  - It requires a format string and a list of variables into which the value received from the input device will be stored.
  - It is required to put an ampersand (&) before the names of the variables.

```
scanf ("%d", &size) ;
scanf ("%c", &nextchar) ;
scanf ("%f", &length) ;
scanf (“%d  %d”, &a, &b);
```

Control Statements

What do they do?

- Allow different sets of instructions to be executed depending on the outcome of a logical test, whether TRUE or FALSE.
  - This is called **branching**.

- Some applications may also require that a set of instructions be executed repeatedly, possibly again based on some condition.
  - This is called **looping**.

How do we specify the conditions?

- Using relational operators.
  - Four relation operators: <, <=, >, >=
  - Two equality operations: ==, !=
- Using logical operators / connectives.
  - Two logical connectives: &&, ||
  - Unary negation operator: !

Examples

```
count <= 100

(math+phys+chem)/3  >= 60

(sex==‘M’) && (age>=21)

(marks>=80) && (marks<90)

(balance>5000) | | (no_of_trans>25)

! (grade==‘A’)

! ((x>20) && (y<16))
```

The conditions evaluate to …

- Zero
  - Indicates FALSE.
- Non-zero
  - Indicates TRUE.
  - Typically the condition TRUE is represented by the value ‘1’.

Branching: The if Statement

```
if (expression)
        statement;

if (expression) {
        Block of statements;
}
```

The condition to be tested is any expression enclosed in parentheses. The expression is evaluated, and if its value is non-zero, the statement is executed.

Branching: if-else Statement

```
if (expression) {
    Block of statements;
}
else {
    Block of statements;
}
```

Nesting of if-else Structures

- It is possible to nest if-else statements, one within another.
- All if statements may not be having the “else” part.
- Rule to be remembered:
  - An “else” clause is associated with the closest preceding unmatched “if”.

Dangling else problem

```
if (exp1) if (exp2) stmta else stmtb
```

```
if (exp1)
    if (exp2)
        stmta
    else
        stmtb
```

OR

```
if (exp1)
    if (exp2)
        stmta
else
    stmtb
```

?

Which one is the correct interpretation?

Dangling else problem

```
if (exp1) if (exp2) stmta else stmtb
```

```
if (exp1)
    if (exp2)
        stmta
    else
        stmtb
```

✓

```
if (exp1)
    if (exp2)
        stmta
else
    stmtb
```

✗

Which one is the correct interpretation?

```
if e1 s1
else if e2 s2
```

```
if e1 s1
else if e2 s2
else s3
```

?

```
if e1 if e2 s1
else s2
else s3
```

```
if e1 if e2 s1
else s2
```

```
if e1 s1                        if  e1 s1
else if e2 s2        →          else  if  e2  s2
```

```
if e1 s1                        if  e1  s1
else if e2 s2        →          else  if  e2  s2
else s3                               else s3
```

```
if e1 if e2 s1                if  e1  if  e2  s1
else s2            →                      else  s2
else s3                        else  s3
```

```
                   →
if e1 if e2 s1               if  e1  if  e2  s1
else s2                                else s2
```

```
int main()  {
    int  a,b,c;
    scanf (“%d %d %d”, &a, &b, &c);
    if (a>=b)
         if (a>=c)
                printf (“\n The largest number is: %d”, a);
         else  printf (“\n The largest number is: %d”, c);
    else
         if (b>=c)
                printf (“\n The largest number is: %d”, b);
         else    printf (“\n The largest number is: %d”, c);
    return 0;
}
```

```
int main( )
{
    int  a,b,c;
    scanf (“%d %d %d”, &a, &b, &c);
    if ((a>=b) && (a>=c))
         printf (“\n The largest number is: %d”, a);
    else
             if (b>c)
                  printf (“\n The largest number is: %d”, b);
             else
                  printf (“\n The largest number is: %d”, c);
}
```

```
int main ()  {
    int marks;
    scanf (“%d”, & marks) ;
     if (marks>= 80) {
        printf (“A”) ;
        printf (“Good Job!”) ;
     }
     else
        if (marks >= 60)
             printf (“B”) ;
         else
             if (marks >=60)
                 printf (“C”) ;
             else {
                 printf (“Failed”) ;
                 printf (“Study hard for the supplementary”) ;
              }
      printf (“\nEnd\n”) ;
}
```

Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
  - Does not ordinarily cause syntax errors.
  - Any expression that produces a value can be used in control structures.
  - Nonzero values are true, zero values are false.
- Example:

```
if ( marks == 100 )
   printf ( "You have aced!\n" );
```

```
if ( marks = 100 )
    printf( "You have aced!\n" );
```

WRONG

Generalization of expression evaluation in C

- Assignment (=) operation is also a part of expression.

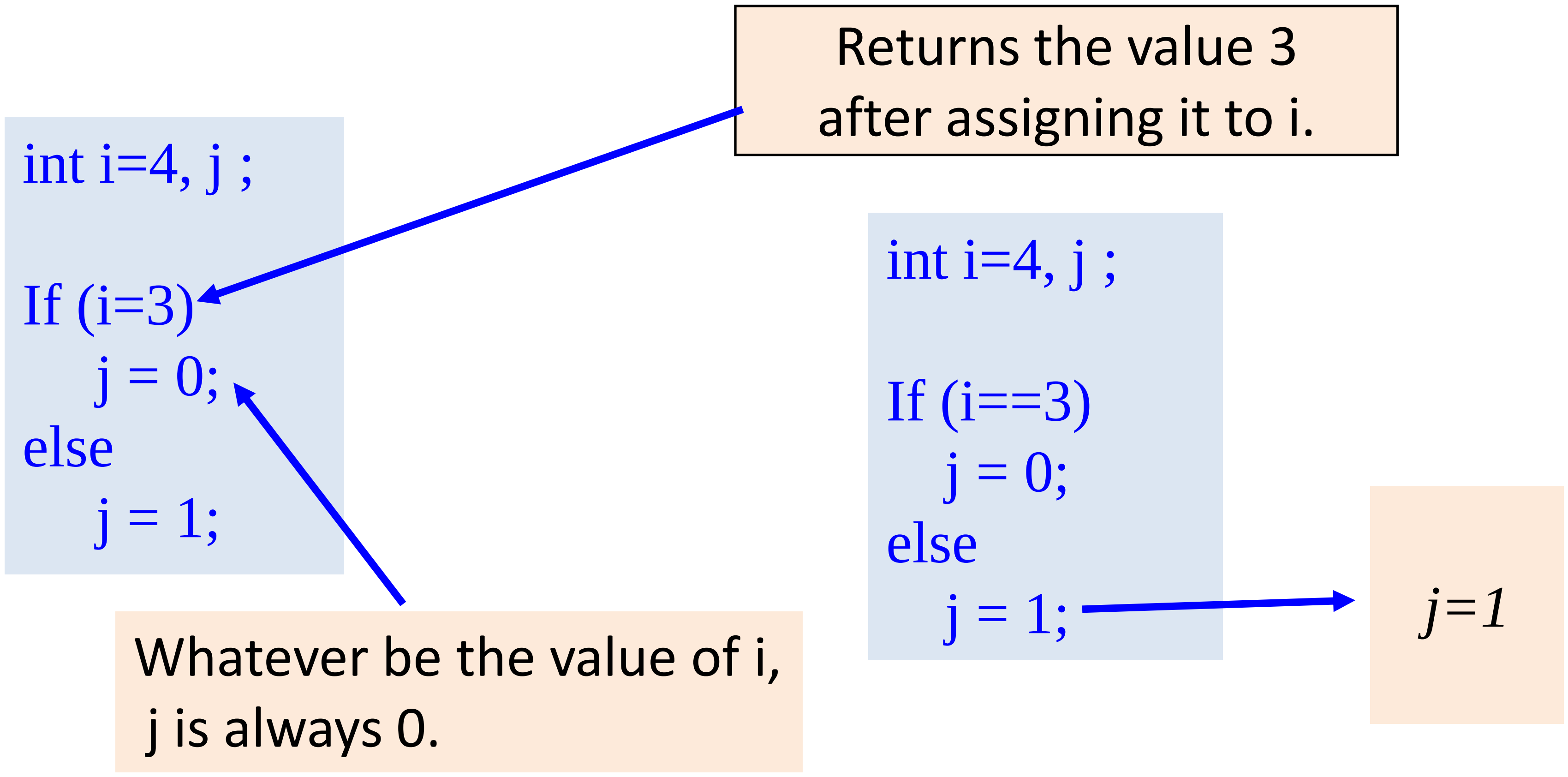
```
i=3;
```

Returns the value 3 after assigning it to i.

```
int i=4, j ;

If (i=3)
    j=0;
else
    j=1;
```

Generalization of expression evaluation in C



```
int i=4, j ;

If (i=3)
    j = 0;
else
    j = 1;
```

Returns the value 3 after assigning it to i.

Whatever be the value of i, j is always 0.

```
int i=4, j ;

If (i==3)
    j = 0;
else
    j = 1;
```

j=1

Increment (++) and Decrement (--)

- Both of these are unary operators; they operate on a single operand.
- The increment operator causes its operand to be increased by 1.
  - Example: a++, ++count
- The decrement operator causes its operand to be decreased by 1.
  - Example: i--, --distance

Prefix and postfix operator

- Operator written before the operand (++i, --i))
  - Called pre-increment operator.
  - Operator will be altered in value *before* it is utilized for its intended purpose in the program.
- Operator written after the operand (i++, i--)
  - Called post-increment operator.
  - Operator will be altered in value *after* it is utilized for its intended purpose in the program.

Examples

Initial values :: a = 10; b = 20;

x = 50 + ++a;	a = 11, x = 61
x = 50 + a++;	x = 60, a = 11
x = a++ + --b;	b = 19, x = 29, a = 11
x = a++ – ++a;	Undefined value (implementation dependent)

Ternary conditional operator (?:)

– Takes three arguments (condition, value if true, value if false)
– Returns the evaluated value accordingly.

```
(expr1)? (expr2) : (expr3);
```

```
grade >= 60 ? printf( “Passed\n” ) : printf( “Failed\n” );

interest = (balance>5000) ? balance*0.2 : balance*0.1;

x = ((a>10) && (b<5)) ? a+b : 0;
```

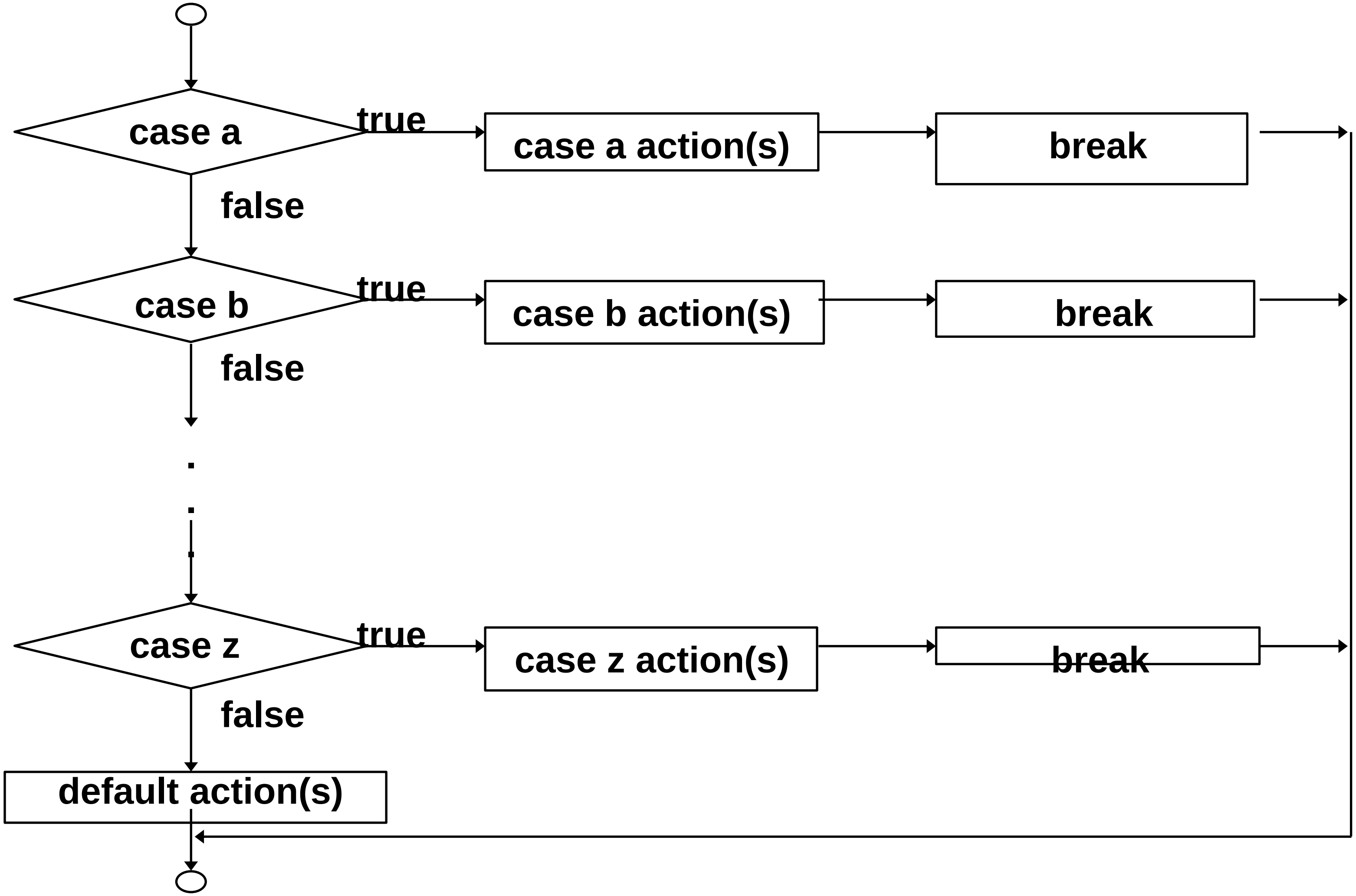
The switch Statement

- This causes a particular group of statements to be chosen from several available groups.
  - Uses “switch” statement and “case” labels.
  - Syntax of the “switch” statement:

```
switch (expression)  {
   case expression-1: { …….. }
   case expression-2: { …….. }

   case expression-m: { …….. }
   default: { ……… }
}
where “expression” evaluates to int or char
```

The switch Multiple-Selection Structure



case a

true

case a action(s)

break

false

case b

true

case b action(s)

break

false

.

.

.

case z

true

case z action(s)

break

false

default action(s)

Examples

```
switch ( letter ) {
    case 'A':
            printf ("First letter \n");
            break;
    case 'Z':
            printf ("Last letter \n");
            break;
    default :
            printf ("Middle letter \n");
            break;
}
```

Examples

```
switch (choice = getchar()) {
    case ‘r’ :
    case ‘R’: printf(“Red”);
               break;
    case ‘b’ :
    case ‘B’ : printf(“Blue”);
              break;
    case ‘g’ :
    case ‘G’: printf(“Green”);
              break;
    default:  printf(“Black”);
}
```

```
switch (digit)   {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4: printf (“Round down\n”);
                break;
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:printf(“Round up\n”);
}
```

```
int main () {

        int operand1, operand2;
        int result = 0;
        char operation ;

        /* Get the input values */
        printf (“Enter operand1 :”);
        scanf(“%d”,&operand1) ;

        printf (“Enter operation :”);
        scanf (“\n%c”,&operation);

        printf (“Enter operand 2 :”);
        scanf (“%d”, &operand2);
```

```
    switch (operation)    {
     case ‘+’ :
          result = operand1+operand2;
           break;
     case ‘-’ :
        result = operand1-operand2;
        break;
    case ‘*’ :
        result = operand1*operand2;
        break;
    case ‘/’ :
        if (operand2 !=0)
            result=operand1/operand2;
        else   printf(“Divide by 0 error”);
        break;
    default:
        printf (“Invalid operation\n”);
    }
    printf (“The answer is %d\n”,result);
}
```

The break Statement

- Used to exit from a switch or terminate from a loop.
  - Already illustrated in the switch examples.
- With respect to “switch”, the “break” statement causes a transfer of control out of the entire “switch” statement, to the first statement following the “switch” statement.

Control Flow: Looping

Types of Repeated Execution

- Loop: Group of instructions that are executed repeatedly while some condition remains true.

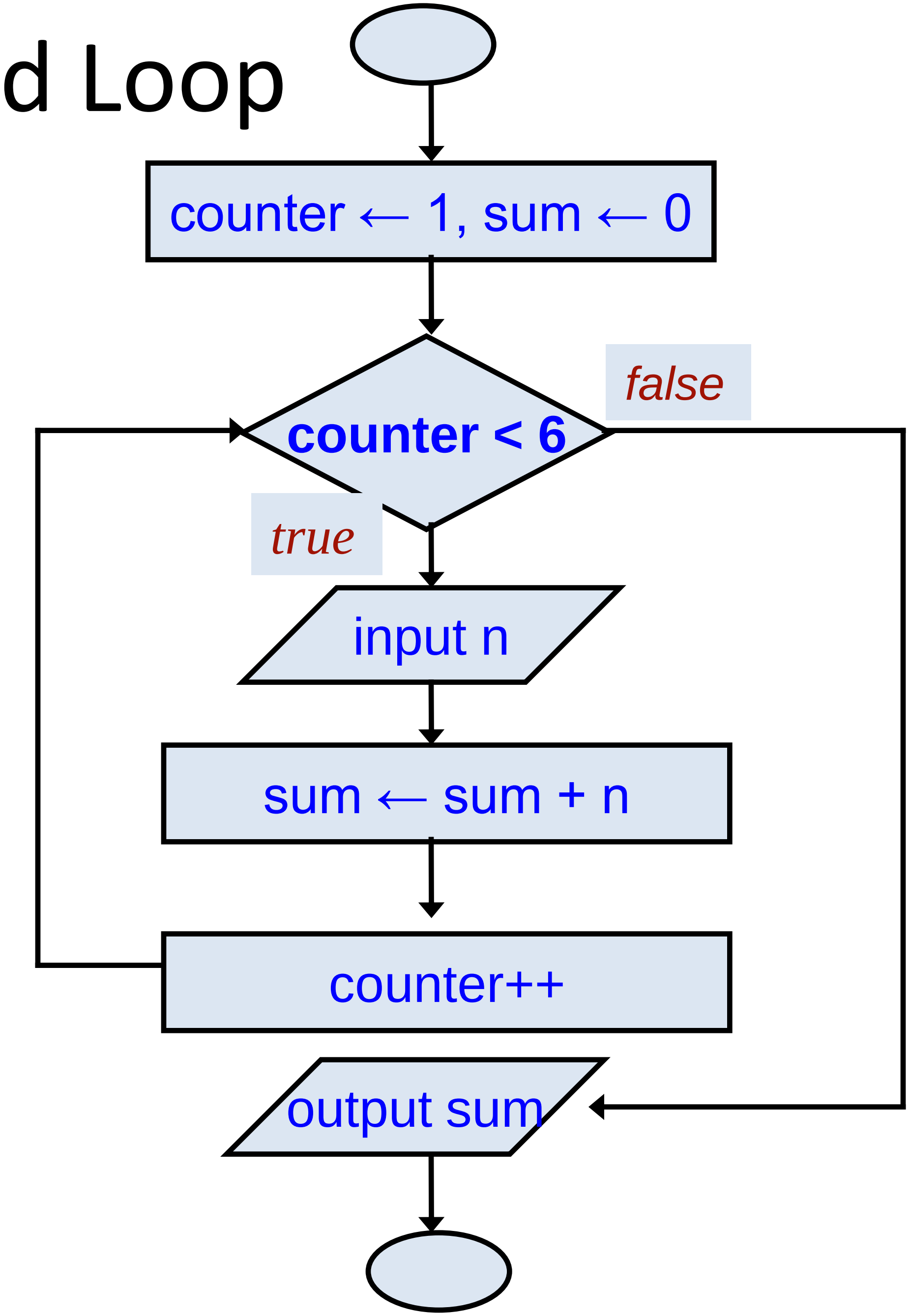
- How loops are controlled?

  By testing a condition

  - The condition may correspond to setting up a counter and checking its value
  - The condition may involve testing for a sentinel value
  - Or any general expression to be tested

Counter Controlled Loop

Read 5 integers and display the value of their summation.



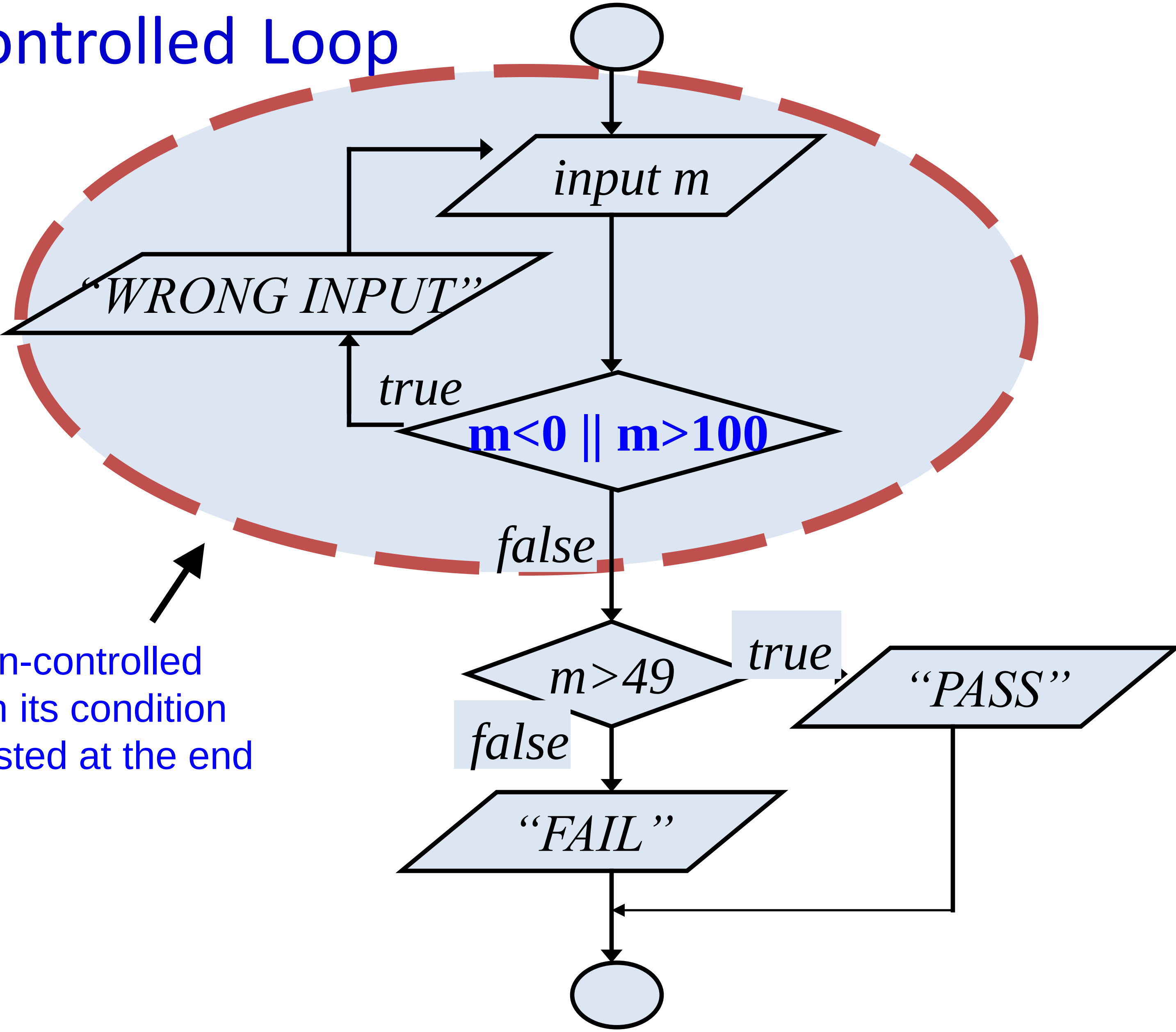
Condition-controlled Loop

Given an exam marks as input, display the appropriate message based on the rules below:

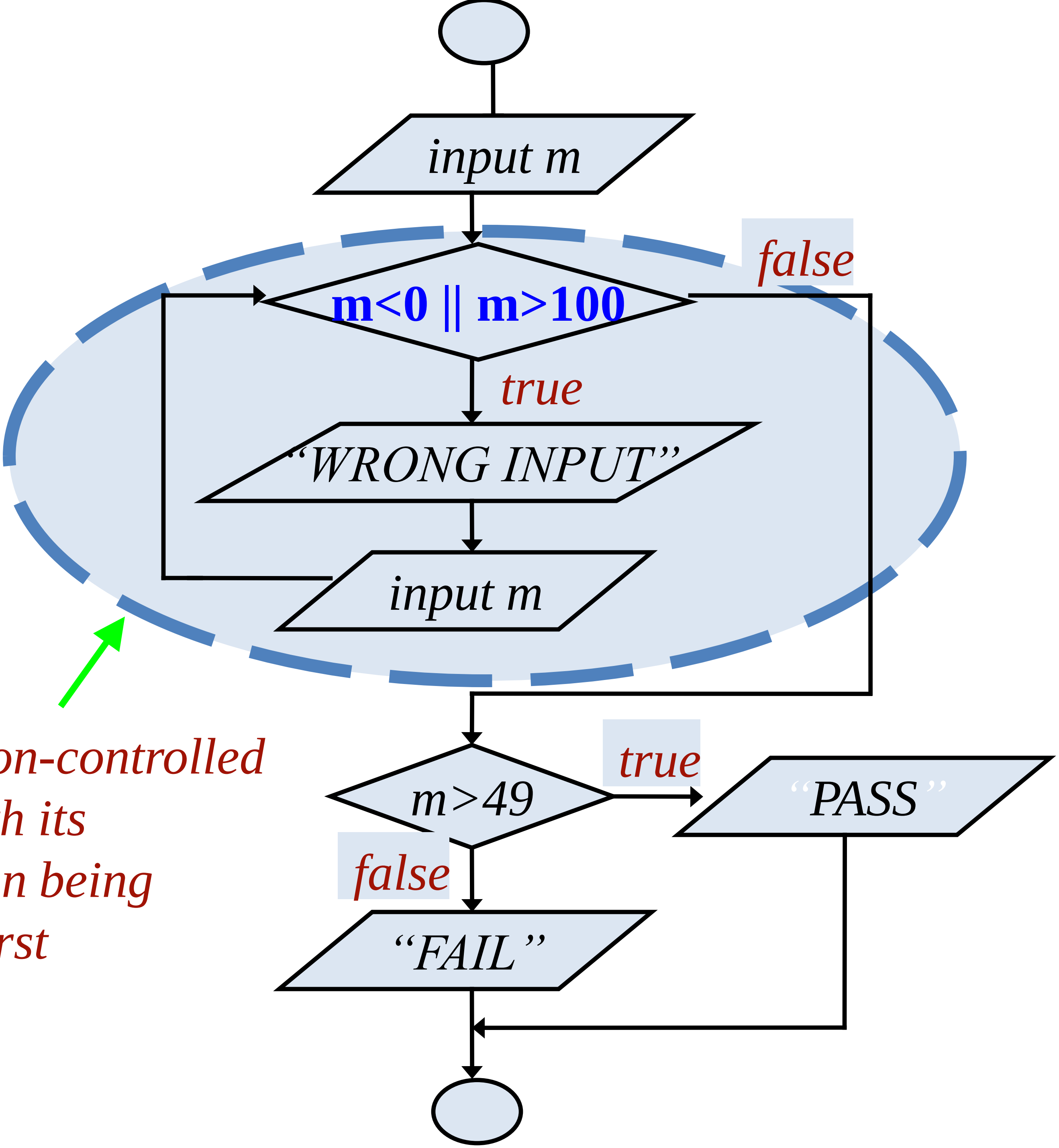
- ❑ If marks is greater than 49, display “PASS”, otherwise display “FAIL”

- ❑ However, for input outside the 0-100 range, display “WRONG INPUT” and prompt the user to input again until a valid input is entered

Condition-Controlled Loop



Condition-controlled loop with its condition being tested at the end



Condition-controlled loop with its condition being tested first

Sentinel-Controlled Loop

- Receive a number of positive integers and display the summation and average of these integers.
- A negative or zero input indicates the end of input process

Input: A set of integers ending with a negative integer or a zero

Output: Summation and Average of these integers

- Input Example:

  30 16 42 -9

  Sentinel Value

- Output Example:

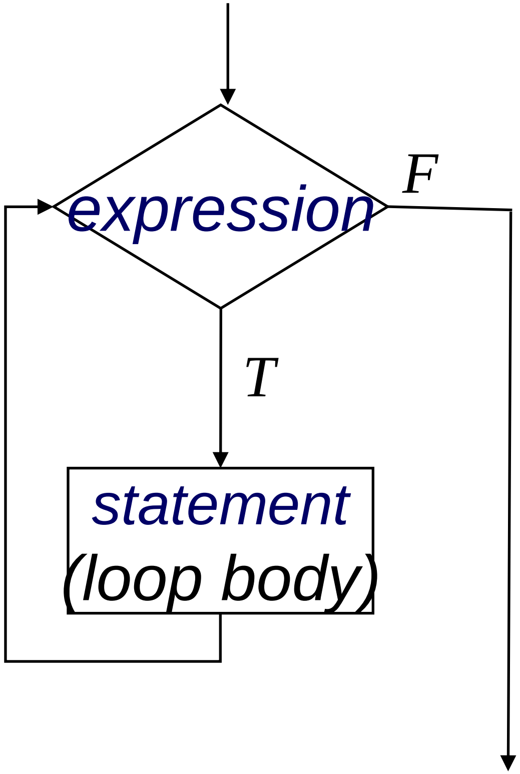
  Sum = 88

  Average = 29.33

while loop

```
while (expression)
        statement
```

```
while (i < n)  {
        printf ("Line no : %d.\n",i);
        i++;
}
```



while Statement

- The "while" statement is used to carry out looping operations, in which a group of statements is executed repeatedly, as long as some condition remains satisfied.

```
while (condition)
    statement_to_repeat;
```

```
while (condition) {
        statement_1;
           ...
        statement_N;
}
```

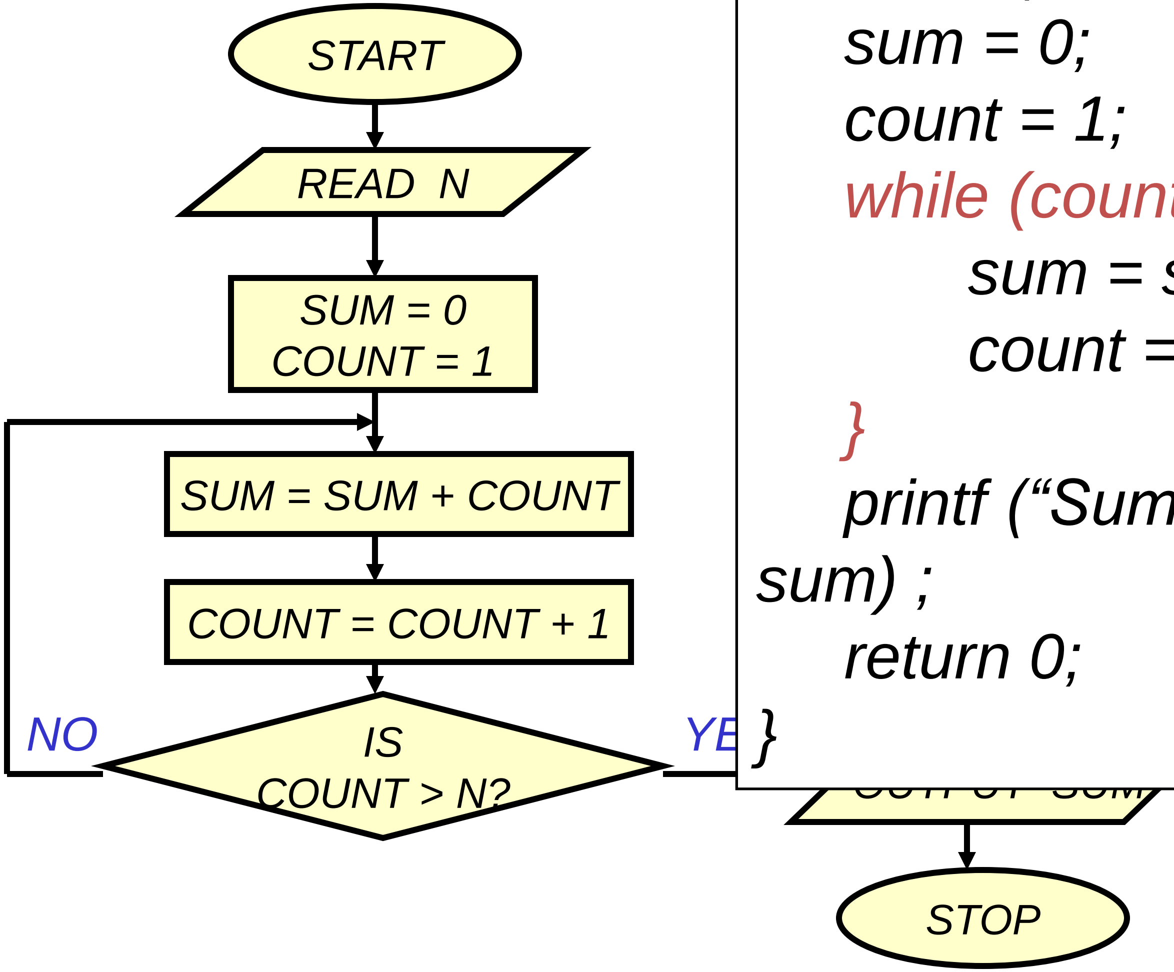
Note:
The while-loop will not be entered if the loop-control expression evaluates to false (zero) even before the first iteration.
break can be used to come out of the while loop.

while :: Examples

```
int  weight;

while ( weight > 65 ) {
      printf ("Go, exercise, ");
      printf ("then come back. \n");
      printf ("Enter your weight: ");
      scanf ("%d", &weight);
 }
```

Sum of first N natura



```
int main () {
     int N, count, sum;
     scanf (“%d”, &N) ;
     sum = 0;
     count = 1;
     while (count <= N)  {
            sum = sum + count;
            count = count + 1;
     }
     printf (“Sum = %d\n”,
sum) ;
     return 0;
}
```

Double your money

- Suppose your Rs 10000 is earning interest at 1% per month. How many months until you double your money ?

```
my_money=10000.0;
n=0;
while (my_money < 20000.0) {
        my_money = my_money*1.01;
        n++;
}
printf (“My money will double in %d months.\n”,n);
```

Maximum of inputs

```
printf (“Enter positive numbers to max, end with a negative number\n”);
max = 0.0;
count = 0;
scanf(“%f”, &next);
while (next >= 0)  {
    if (next > max)
        max = next;
    count++;
    scanf(“%f”, &next);
}
printf (“The maximum number is %f\n”, max) ;
```

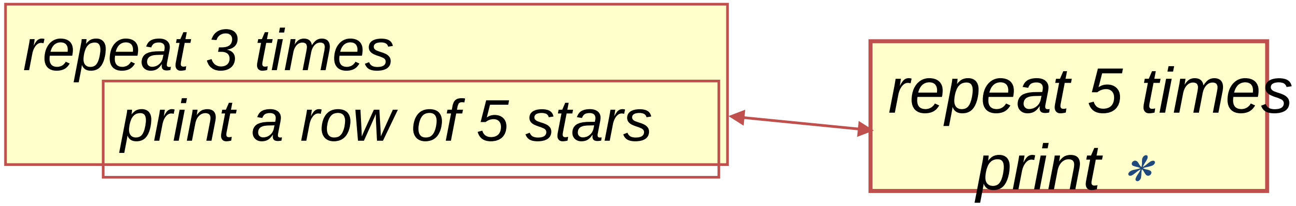
Printing a 2-D Figure

- How would you print the following diagram?

```
* * * * *

* * * * *

* * * *
```



```
repeat 3 times
    print a row of 5 stars
```

```
repeat 5 times
    print *
```

Nested Loops

```
#define ROWS 3
#define COLS 5
...
row=1;
while (row <= ROWS) {
    /* print a row of 5 *'s */
     ...
    row++;
}
```

```
row=1;
while (row <= ROWS) {
    /* print a row of 5 *’s */
    col=1;
    while (col <= COLS) {
        printf (“* “);
        col++;
    }
    printf(“\n”);
    row++;
}
```

outer loop

inner loop

while Loop Pitfall - 1

1

```
int product = 0;

while ( product < 500000 ) {
        product = product * 5;
}
```

2

```
int count = 1;

while ( count != 10 ) {
        count = count + 2;
}
```

Infinite Loops

Both loops will not terminate because the boolean expressions will never become false.

while Loop Pitfall - 2

Goal: Execute the loop body 10 times.

① ✗

```
count = 1;
while ( count < 10 ) {
        . . .
        count++;
}
```

② ✓

```
count = 1;
while ( count <= 10 ) {
        . . .
        count++;
}
```

③ ✗

```
count = 0;
while ( count < 10 ) {
        . . .
        count++;
}
```

④ ✓

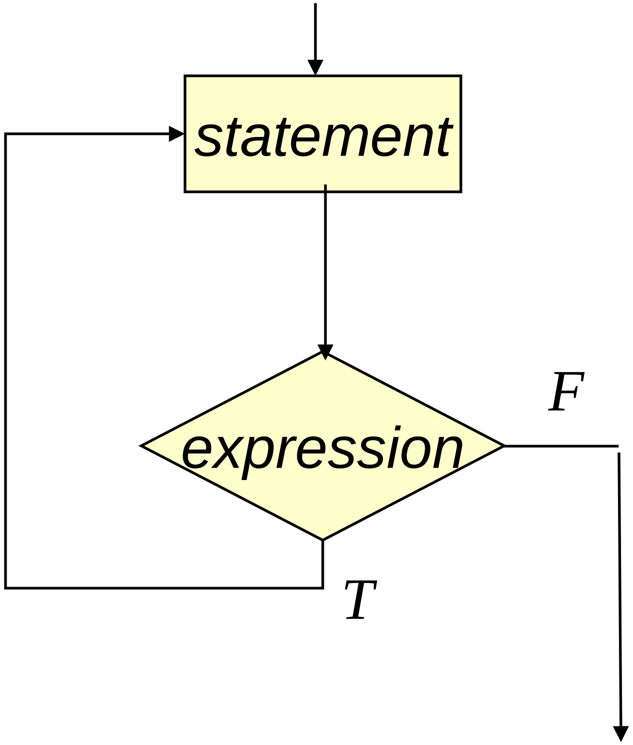
```
count = 0;
while ( count < 10 ) {
        . . .
        count++;
}
```

① and ③ exhibit *off-by-one* error.

do-while statement

do *statement* while (*expression*)

```
main () {
     int digit=0;
     do
          printf(“%d\n”,digit++);
      while (digit <= 9) ;
}
```



Example for do-while

Usage: Prompt user to input “month” value, keep prompting until a correct value of moth is input.

```
do {
        printf (“Please input month {1-12}”);
scanf (“%d”, &month);
} while ((month < 1) || (month > 12));
```

```
int main () {
    char echo ;
    do {
       scanf (“%c”, &echo);
            printf (“%c”,echo);
     }  while (echo != ‘\n’) ;
}
```

for Statement

- The “for” statement is the most commonly used looping structure in C.
- General syntax:

for *(expr1; expr2; expr3) statement*

expr1 (init) : initialize parameters

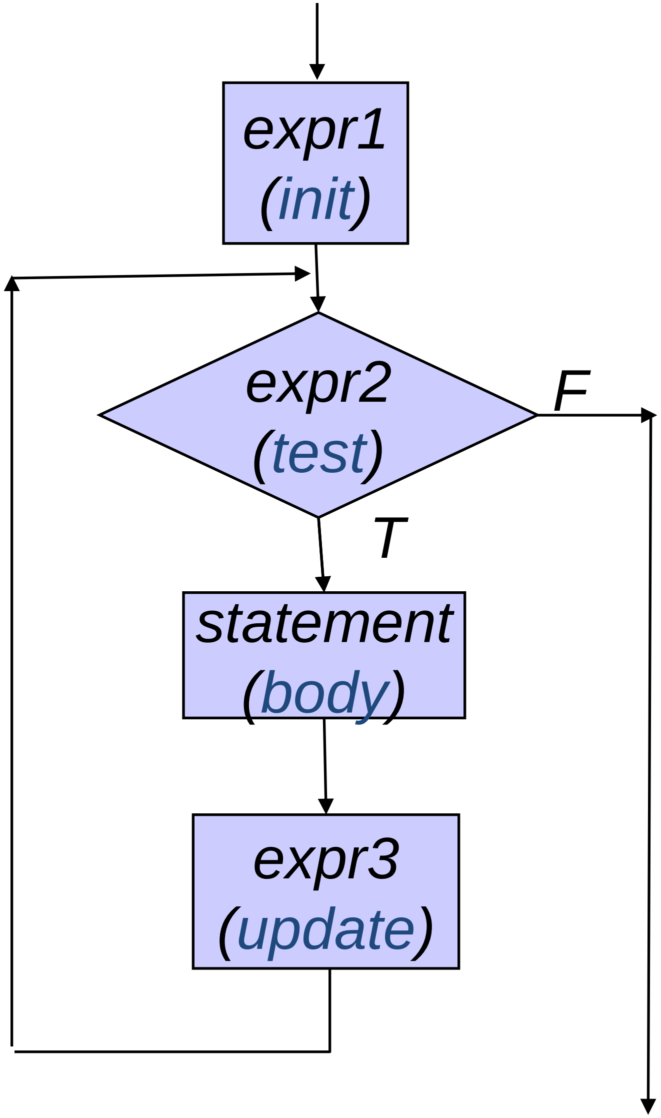
expr2 (test): test condition, loop continues if satisfied

expr3 (update): used to alter the value of the parameters after each iteration

statement (body): body of the loop

for *(expression1; expression2; expression3)*

statement



```
expr1;
while (expr2)  {
     statement
     expr3;
}
```

Sum of first N natural numbers

```
int main () {
    int N, count, sum;
    scanf (“%d”, &N) ;
    sum = 0;
    count = 1;
    while (count <= N)  {
        sum = sum + count;
        count = count + 1;
    }
    printf (“Sum = %d\n”, sum) ;
    return 0;
}
```

Sum of first N natural numbers

```
int main () {
     int N, count, sum;
     scanf (“%d”, &N) ;
     sum = 0;
     count = 1;
     while (count <= N)  {
            sum = sum + count;
            count = count + 1;
     }
     printf (“Sum = %d\n”, su
     return 0;
}
```

```
int main () {
     int N, count, sum;
     scanf (“%d”, &N) ;
     sum = 0;
     for (count=1; count <= N; count++)
            sum = sum + count;

     printf (“Sum = %d\n”, sum) ;
     return 0;
}
```

2-D Figure

Print

```
* * * * *

* * * * *

* * * * *
```

```
#define ROWS 3
#define COLS 5
....
for (row=1; row<=ROWS; row++) {
     for (col=1; col<=COLS; col++) {
              printf(“*”);
     }
     printf(“\n”);
}
```

Another 2-D Figure

Print

```
*
* *
* * *
* * * *
* * * * *
```

```
#define ROWS 5
....
int row, col;
for (row=1; row<=ROWS; row++) {
      for (col=1; col<=row; col++) {
                printf(“* ”);
      }
      printf(“\n”);
}
```

For - Examples

- Problem 1: Write a For statement that computes the sum of all odd numbers between 1000 and 2000.
- Problem 2: Write a For statement that computes the sum of all numbers between 1000 and 10000 that are divisible by 17.
- Problem 3: Printing square problem but this time make the square hollow.
- Problem 4: Print

```
* * * * *
 * * * *
  * * *
   * *
    *
```

Problem 4 : solution

Print

```
* * * * *
 * * * *
  * * *
   * *
    *
```

```
#define ROWS 5
....
int row, col;
for (row=0; row<ROWS; row++)  {
    for (col=1; col<=row; col++)
        printf("  ");
    for (col=1; col<=ROWS-row; col++)
        printf("* ");
    printf ("\n");
}
```

The comma operator

- *We can give several statements separated by commas in place of “expression1”, “expression2”, and “expression3”.*

```
for  (fact=1, i=1; i<=10; i++)
    fact = fact * i;


for (sum=0, i=1; i<=N, i++)
    sum = sum + i * i;
```

for :: Some Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.

```
for ( k = x;   k <= 4 * x * y;   k += y / x )
```

- "Increment" may be negative (decrement)

```
for  (digit=9; digit>=0; digit--)
```

- If loop continuation condition initially *false:*
  - Body of *for* structure not performed.
  - Control proceeds with statement after *for* structure.

Specifying “Infinite Loop”

```
while  (1)  {
   statements
}
```

```
for  (; ;)
{
      statements
}
```

```
do  {
   statements
}  while (1);
```

The break Statement

- Break out of the loop { }
  - can use with
    - while
    - do while
    - for
    - switch
  - does not work with
    - if
    - else

- Causes immediate exit from a *while*, *do/while*, *for* or *switch* structure.
- Program execution continues with the first statement after the structure.

An Example

```
#include  <stdio.h>
int main() {
      int  fact, i;

      fact = 1;  i = 1;

      while  ( i<10 )    {                     /* run loop –break when fact >100*/
                fact = fact * i;
                if ( fact > 100 )  {
                                printf ("Factorial of %d  above 100", i);
                                break;                         /* break out of the while loop */
                }
                i ++ ;
      }
}
```

The continue Statement

- Skips the remaining statements in the body of a *while*, *for* or *do/while* structure.
  - Proceeds with the next iteration of the loop.
- while and do/while
  - Loop-continuation test is evaluated immediately after the continue statement is executed.
- for structure
  - *expression3* is evaluated, then *expression2* is evaluated.

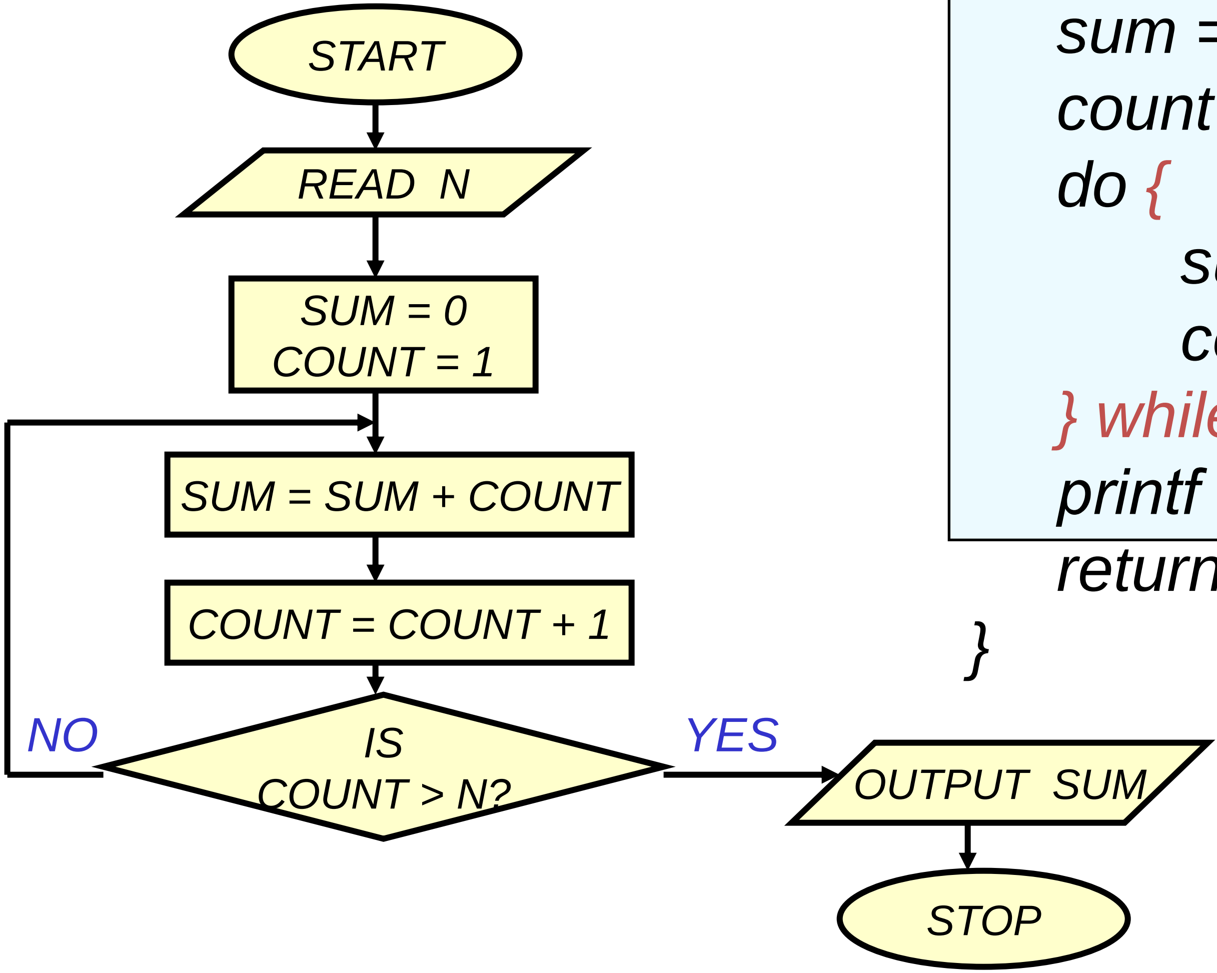
An Example with “break” & “continue”

```
fact = 1; i = 1;              /* a program segment  to calculate 10 !
while  (1)  {
    fact = fact * i;
    i ++ ;
    if  ( i<10 )
          continue;           /* not done yet ! Go to loop and
                              perform next iteration*/
    break;
}
```

Some Examples

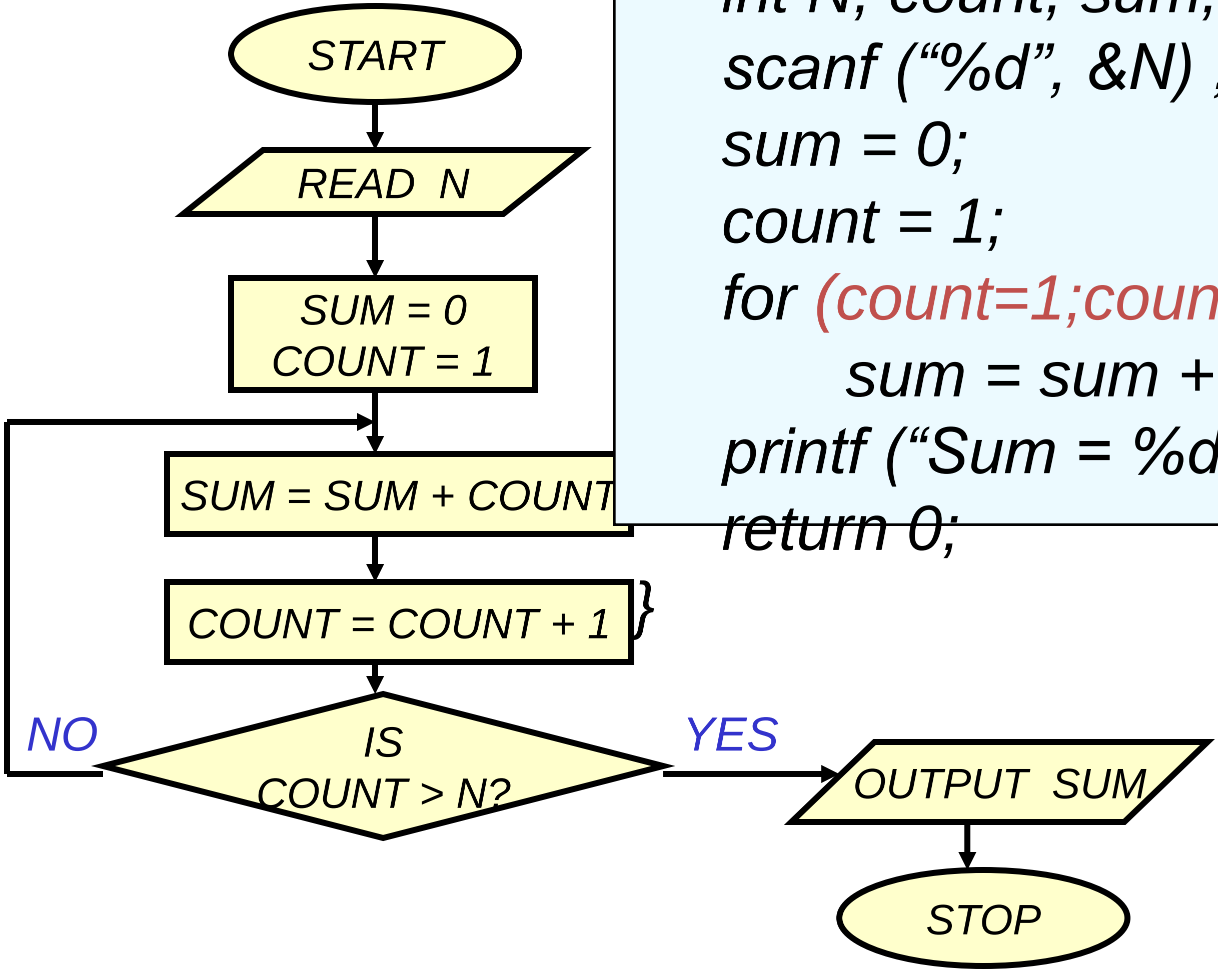
Sum of first N natura

```
int main () {
    int N, count, sum;
    scanf ("%d", &N) ;
    sum = 0;
    count = 1;
    do {
        sum = sum + count;
        count = count + 1;
    } while (count<=N) ;
    printf ("Sum = %d\n", sum) ;
    return 0;
}
```



Sum of first N natural numbers

```
int main () {
      int N, count, sum;
      scanf ("%d", &N) ;
      sum = 0;
      count = 1;
      for (count=1;count <= N;count+
              sum = sum + count;
      printf ("Sum = %d\n", sum) ;
      return 0;
}
```



START

READ N

SUM = 0
COUNT = 1

SUM = SUM + COUNT

COUNT = COUNT + 1

IS
COUNT > N?

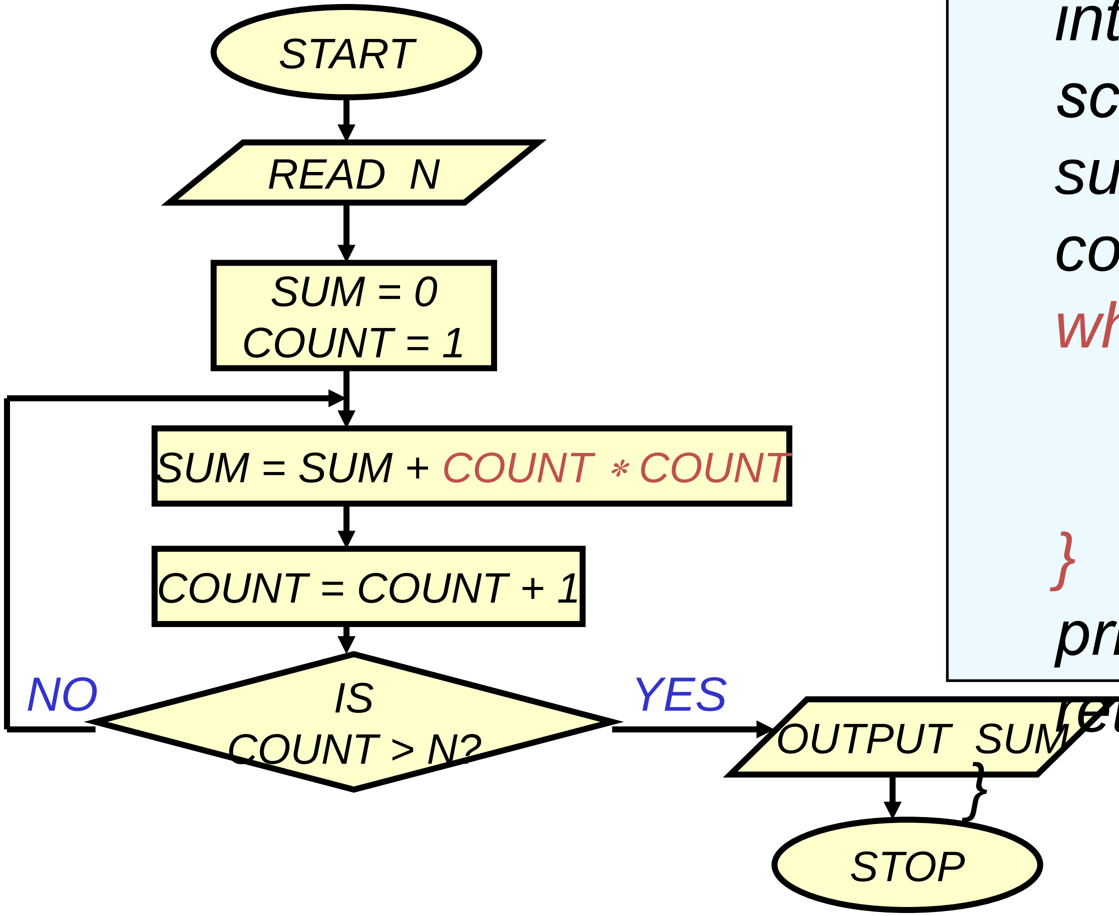
NO

YES

OUTPUT SUM

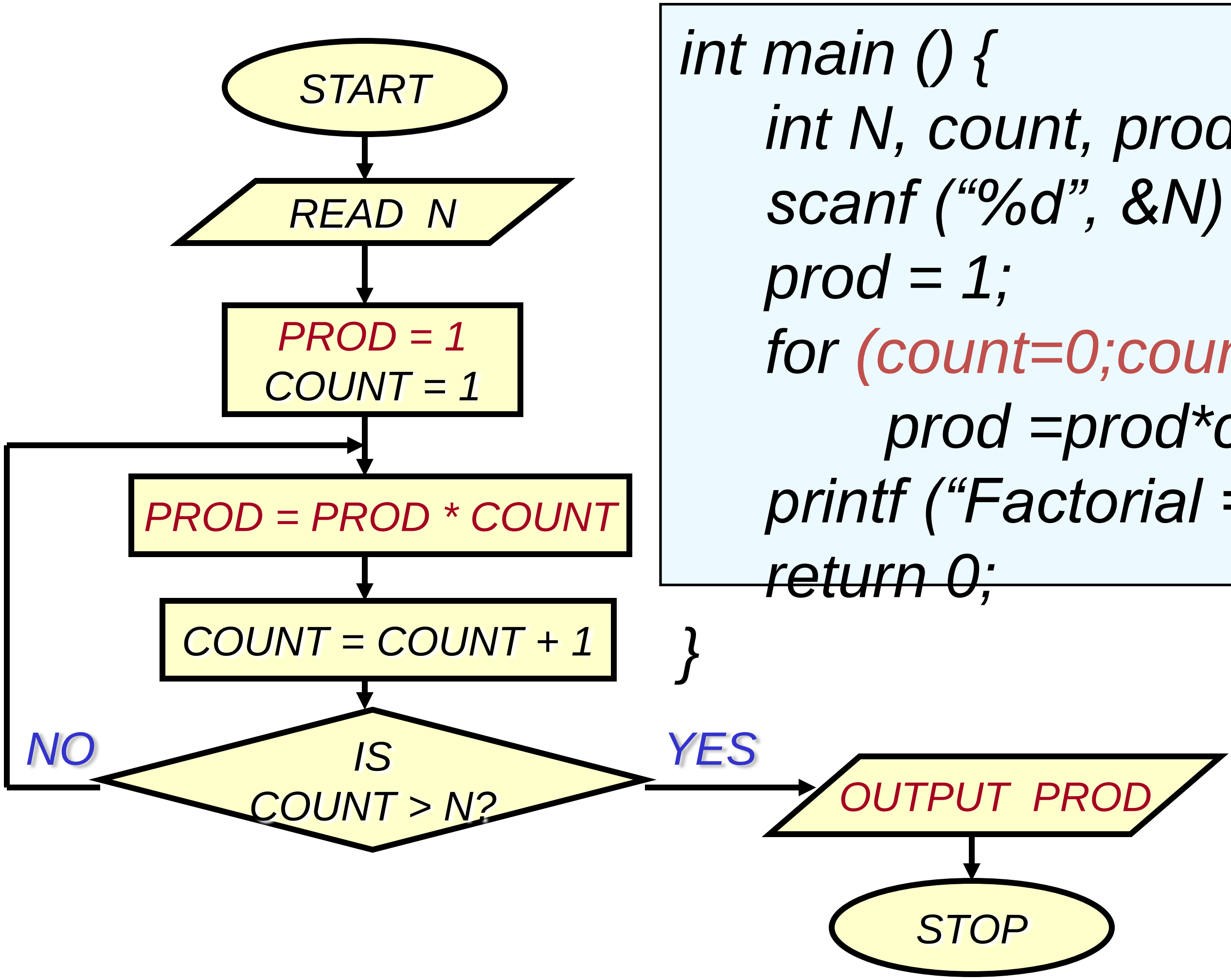
STOP

Example 5: $SUM = 1^2 + 2^2 + 3^2 + N^2$



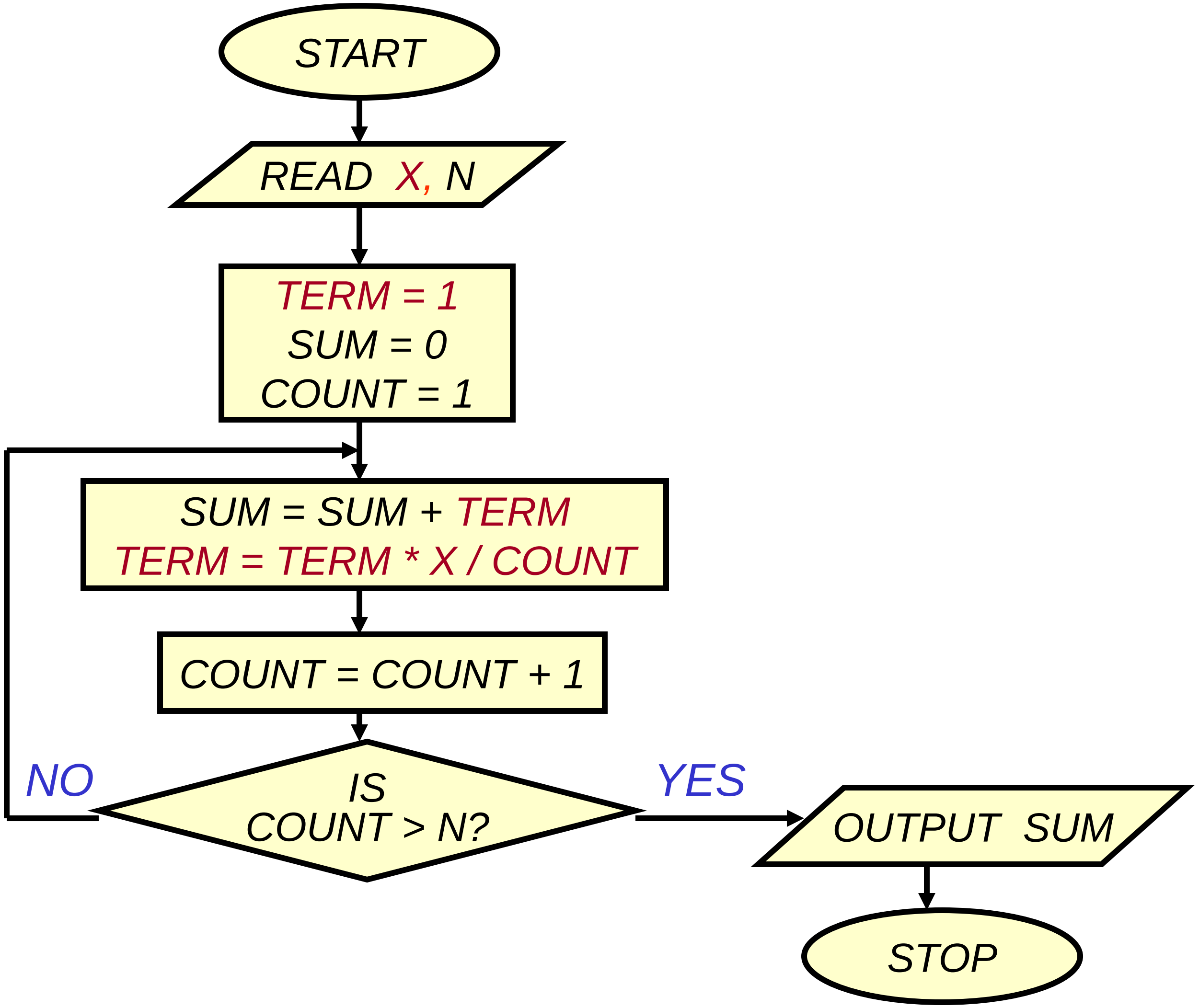
```
int main () {
    int N, count, sum;
    scanf (“%d”, &N) ;
    sum = 0;
    count = 1;
    while (count <= N)  {
        sum = sum + count∗cou
        count = count + 1;
    }
    printf (“Sum = %d\n”, sum) ,
    return 0;
}
```

Example: *Computing Factorial*



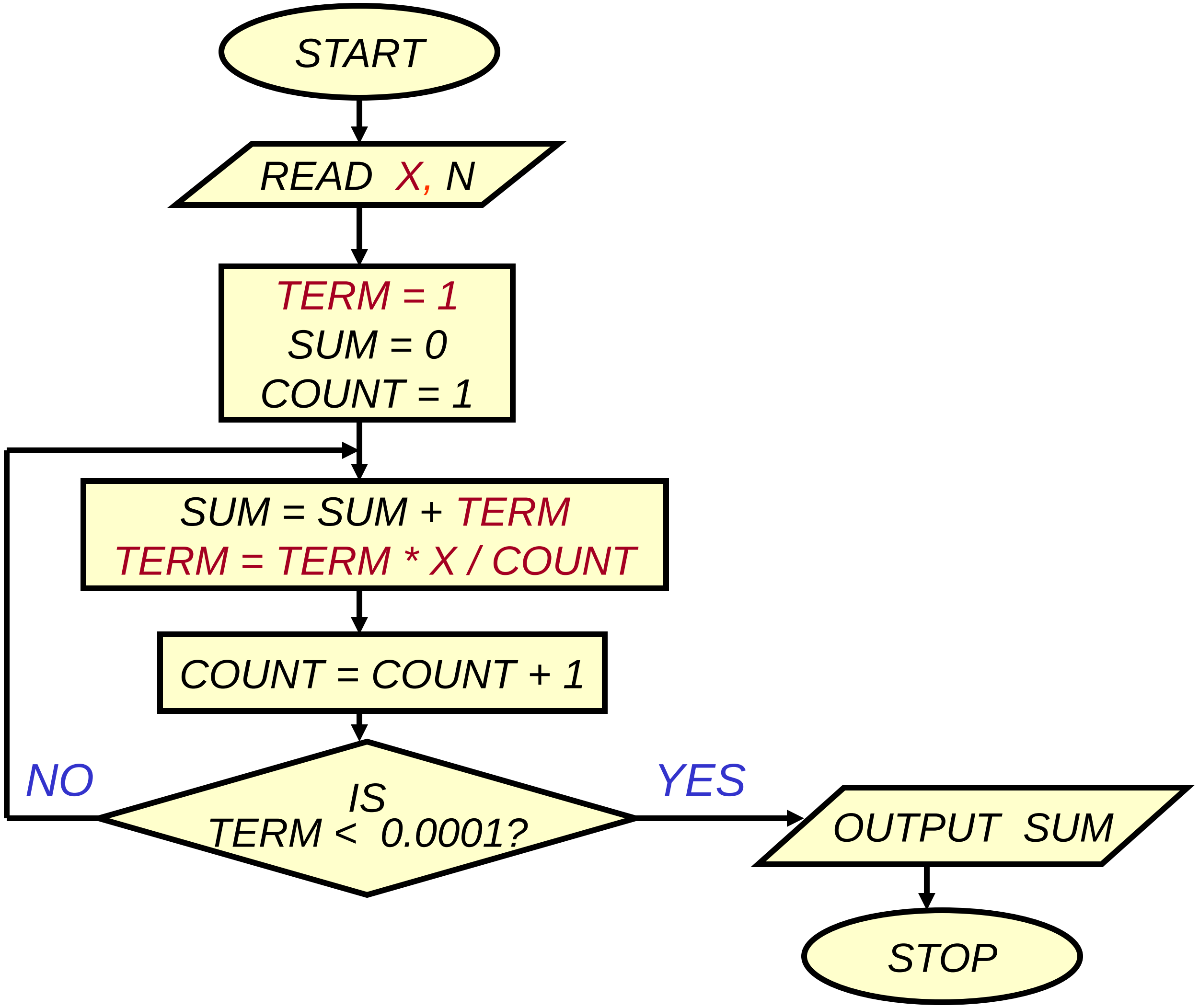
```
int main () {
     int N, count, prod;
     scanf (“%d”, &N) ;
     prod = 1;
     for (count=0;count < N; count++)  {
            prod =prod*count;
     printf (“Factorial = %d\n”, prod) ;
     return 0;
}
```

Example: *Computing e^x series up to N terms*



```
int main () {
    float x, term, sum;
     int n, count;
     scanf (“%d”, &x) ;
    scanf (“%d”, &n) ;
    term = 1.0; sum = 0;
    for (count = 0; count < n; count++)  {
          sum += term;
           term ∗= x/count;
     }
     printf (“%f\n”, sum) ;
}
```

Example 8: *Computing e^x series up to 4 decimal places*



```
int main () {
     float x, term, sum;
      int n, count;
      scanf (“%d”, &x) ;
     scanf (“%d”, &n) ;
     term = 1.0; sum = 0;
     for (count = 0; term<0.0001; count++)  {
            sum += term;
             term ∗= x/count;
      }
      printf (“%f\n”, sum) ;
}
```

Example 1: Test if a number is prime or not

```
#include <stdio.h>
int main() {
    int  n;
    scanf (“%d”, &n);
    i = 2;
    while (i < n)  {
          if (n % i == 0)  {
                    printf (“%d is not a prime \n”, n);
                    exit;
          }
          i++;
    }
    printf (“%d is a prime \n”, n);
}
```

More efficient??

```
#include <stdio.h>
int main()
{
    int  n, i=3;
    scanf (“%d”, &n);
    if (n%2 == 0) {
        printf (“%d is not a prime \n”, n);
        exit;
    }
    while (i < sqrt(n))  {
        if (n % i == 0)  {
            printf (“%d is not a prime \n”, n);
            exit;
        }
        i = i + 1;
    }
    printf (“%d is a prime \n”, n);
}
```

Example 2: Find the sum of digits of a number

```
#include  <stdio.h>
int main()  {
    int  n, sum=0;
    scanf (“%d”, &n);
    while (n != 0)  {
          sum = sum + (n % 10);
          n = n / 10;
    }
    printf (“The sum of digits of the number is %d \n”, sum);
}
```

Example 3: Decimal to binary conversion

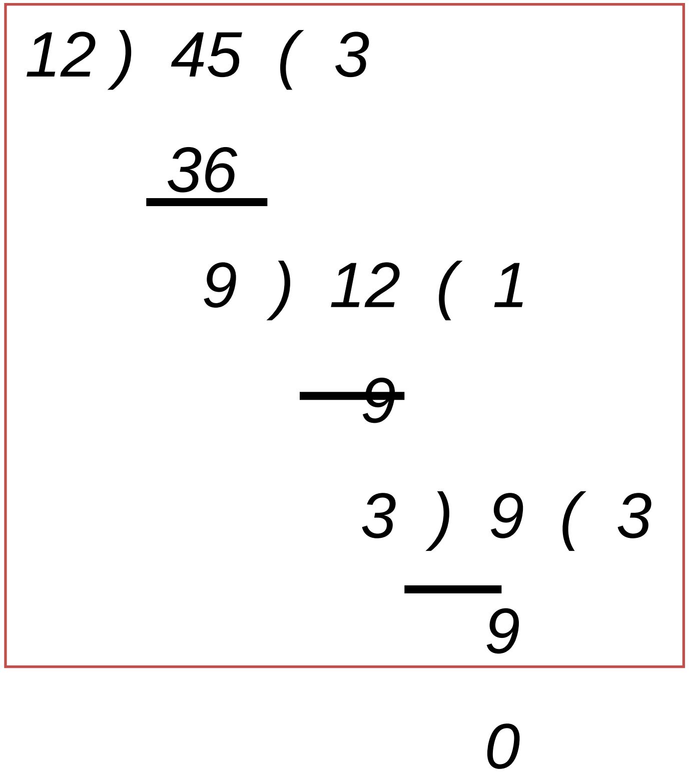
```
#include  <stdio.h>
int main()
{
    int  dec;
    scanf (“%d”, &dec);
    do
    {
           printf (“%2d”,  (dec % 2));
           dec = dec / 2;
    }  while (dec != 0);
    printf (“\n”);
}
```

Example 4:
Compute greatest common divisor (GCD) of two numbers

The standard gcd algorithm is based on successive Euclidean division.

Let us try to render it as a sequence of repetitive computations.

For the sake of simplicity, we assume that whenever we write gcd(a,b) we mean a>=b.



```
12 ) 45 ( 3
     36
      9 ) 12 ( 1
           9
           3 ) 9 ( 3
               9
               0
```

[*Euclidean gcd theorem*]

- Let a, b be positive integers and r = a % b. Then gcd(a,b) = gcd(b,r).
- If a is an integral multiple of b, we have r=0, and so by the theorem gcd(a,b)=gcd(b,0)=b.

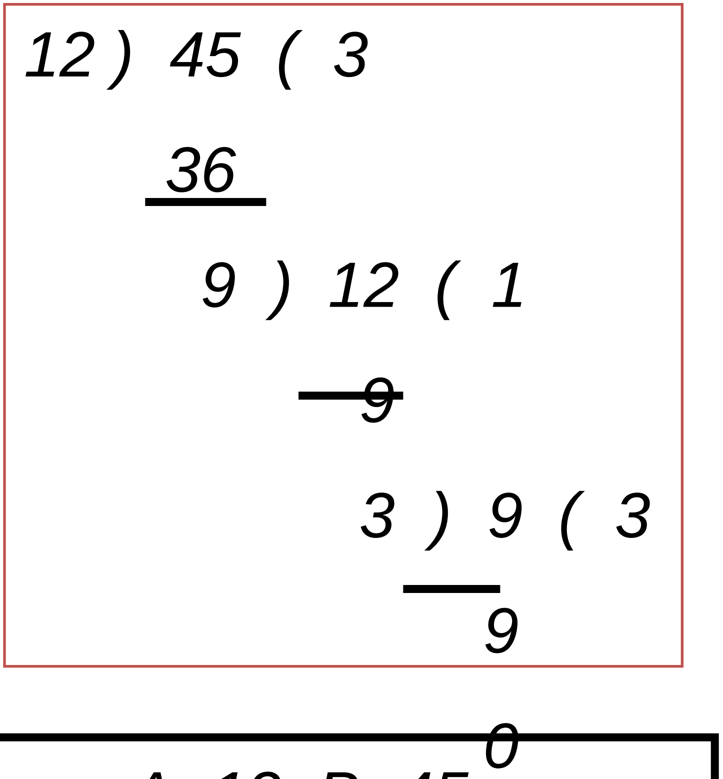
GCD algorithm

```
As long as b is not equal to 0 do the following:
    Compute the remainder r = a rem b.
    Replace a by b and b by r.
Report a as the desired gcd.
```

```
if  (a > b)  {
        temp = a;  a = b;  b = temp;
 }
 while (b != 0)  {
        rem = a % b;
        a = b;
        b = rem;
 }
```

Example 4: Compute GCD of two numbers

```
#include  <stdio.h>
int main() {
      int  a, b, rem, temp;
      scanf (%d %d”, &a, &b);
      if  (a > b)  {
                temp = a;  a = b;  b = temp;
      }
      while (b != 0)  {
                rem = a % b;
                a = b;
                b = rem;
      }
      printf (“The GCD is %d”, a);
}
```



12) 45 (3

36

9) 12 (1

9

3) 9 (3

9

0

Initial: A=12, B=45

Iteration 1: temp=9, B=12,A=9

Iteration 2: temp=3, B=9, A=3

B % A = 0   → GCD is 3

More about scanf and printf

Entering input data :: scanf function

- General syntax:

  scanf (control string, arg1, arg2, …, argn);

  - “control string refers to a string typically containing data types of the arguments to be read in;
  - the arguments arg1, arg2, … represent pointers to data items in memory.

    Example:

    scanf (%d %f %c”, &a, &average, &type);

- The control string consists of individual groups of characters, with one character group for each input data item.
  - ‘%’ sign, followed by a conversion character.

- Commonly used conversion characters:

c	single character
d	decimal integer
f	floating-point number
s	string terminated by null character
X	hexadecimal integer

- We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.

  Example: scanf (“%3d %5d”, &a, &b);

Writing output data :: printf function

- General syntax:

  printf (control string, arg1, arg2, …, argn);

  - “control string refers to a string containing formatting information and data types of the arguments to be output;
  - the arguments arg1, arg2, … represent the individual output data items.
- The conversion characters are the same as in scanf.

- Examples:

```
printf  (“The average of %d and %d is %f”, a, b, avg);
printf  (“Hello \nGood \nMorning \n”);
printf (“%3d %3d %5d”, a, b, a*b+2);
printf (“%7.2f  %5.1f”, x, y);
```

- Many more options are available:
  - Read from the book.
  - Practice them in the lab.
- String I/O:
  - Will be covered later in the class.

Exercise 1

sin() takes a value in radians and returns the sin of it. Use the sin function to plot a sin wave vertically using stars (it should look something like this):

```
*
             *
                *
                 *
                *
             *
         *
      *
  *
 *
  *
     *
         *
```

Hint: Obviously, sin returns a number between -1 and 1. Convert this to a number between 0 and 60 and print that many spaces before printing the * - then print a '\n'

```
int main () {
        int j;
        float x, v;
        for (x=0; x<6.3; x=x+0.2)  {
                v = sin (x) ;
                for (j=0; j<=30*v+30; j++)
                        printf (" ") ;
                printf ("*\n") ;
        }
}
```

```
[sudeshna@facweb temp]$ ./a.out
                                 *
                                      *
                                            *
                                                 *
                                                      *
                                                          *
                                                            *
                                                              *
                                                              *
                                                              *
                                                            *
                                                         *
                                                     *
                                                *
                                           *
                                     *
                               *
                        *
                  *
             *
         *
     *
   *
  *
  *
   *
     *
        *
             *
                  *
                       *
                              *
[sudeshna@facweb temp]$
```

Exercise 2

Write a C program to compute the following series:

x - x^2/(2*1) + 2*x^3/(3*2*1) -
3*x^4/(4*3*2*1) + ...

The value of x will be read from the user. The sum is to be computed over 10 terms. Print the partial sums as well as the final sum.

Exercise 3

It is known that the harmonic number H_n converges to $k + \ln n$ as n tends to infinity.

Here ln is the natural logarithm and k is a constant known as *Euler's constant*. In this exercise you are asked to compute an approximate value for Euler's constant.

Generate the values of H_n and $ln\ n$ successively for $n=1,2,3,\ldots,$ and compute the difference

Exercise 4

Write a C program that takes as input a number and computes and prints the following:

1. the sum of the digits of the number
2. the number reversed
3. the sum of the original number and the reversed number

Exercise 5

Write a program that find can find the roots of a mathematical function using the bisection method. Assume that the function has exactly one root in that interval.

The *bisection* method works as follows:

Check the value of the function at the middle of the interval: if it is positive, replace the left endpoint with the middle point; if it is negative, replace the right endpoint with the middle point. This halves the size of the interval. Stay in a loop doing this until the interval size is less than epsilon. The interval end points (xleft and xright) and the tolerance for the approximation (epsilon) are entered by the user.

For this lab, consider finding the root of the function

$p(x) = 5x^3 - 2x - 2$

over the interval [0,2] using epsilon = 0.0001.

Also print the number of iterations required for this value of epsilon. Print out all function evaluations to trace the execution of your program.