Control Statements

Palash Dey
Department of Computer Science & Engg.
Indian Institute of Technology
Kharagpur

Slides credit: Prof. Indranil Sen Gupta

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 operators: ==, !=
- 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 …

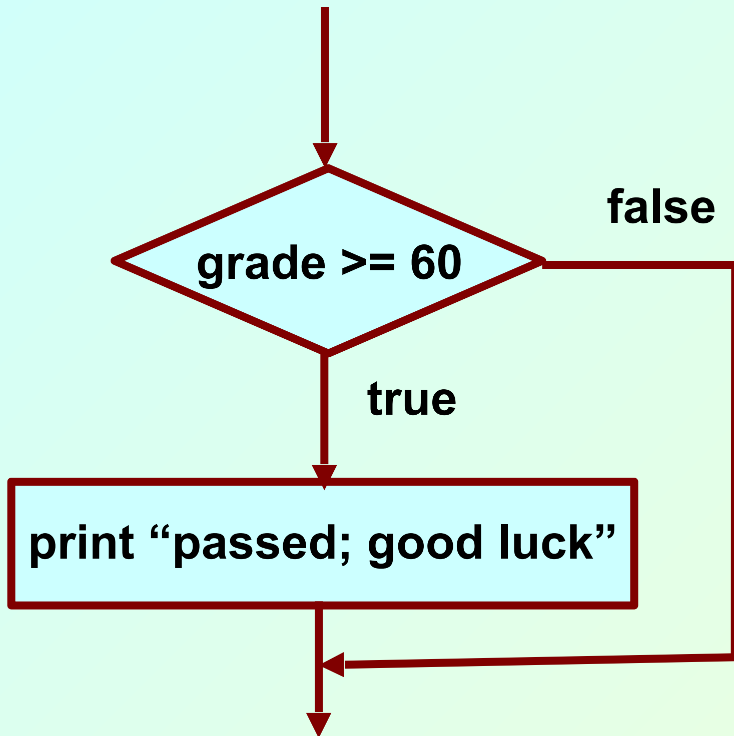
- Zero
  - Indicates *FALSE*.
- Non-zero
  - Indicates *TRUE*.
  - Typically the condition TRUE is represented by the value '1'.

Branching: The if Statement

- Diamond symbol (decision symbol) - indicates decision is to be made.
  - Contains an expression that can be TRUE or FALSE.
  - Test the condition, and follow appropriate path.
- Single-entry / single-exit structure.
- General syntax:

```
if (condition)  { ……..}
```

  - If there is a single statement in the block, the braces can be omitted.



A decision can be made on any expression.

zero - false

nonzero - true

```
if (grade>=60)
{
  printf(“Passed \n”);
  printf(“Good luck\n”);
}
```

Example

```
#include <stdio.h>
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);
    if ((b>=a) && (b>=c))
        printf ("\n The largest number is: %d", b);
   if ((c>=a) && (c>=b))
        printf ("\n The largest number is: %d", c);
}
```

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 (payCode == 4)
  printf("You get a bonus!\n");
```

```
if (payCode = 4)
  printf("You get a bonus!\n");
```

Some Examples

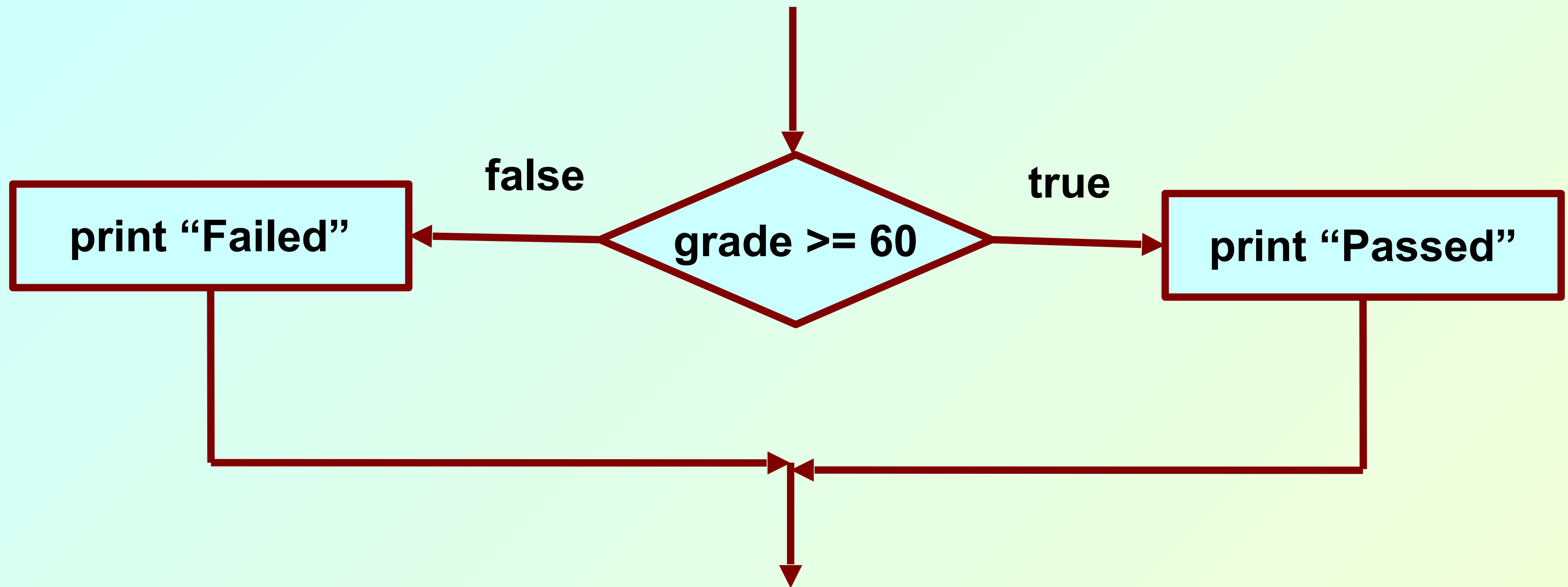
```
if (10<20)  { a = b + c;  printf ("%d", a);  }

if ((a>b) && (x=10))  { ............... }

if (1)  { .................. }

if (0)  { .................. }
```

Branching: The if-else Statement

- Also a single-entry / single-exit structure.
- Allows us to specify two alternate blocks of statements, one of which is executed depending on the outcome of the condition.
- General syntax:

```
if  (condition)   { …… block 1 …… }
else  { …… block 2 …… }
```

  – If a block contains a single statement, the braces can be deleted.




```
if ( grade >= 60 )
      printf ("Passed\n");
else
       printf ("Failed\n");
```

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.
  - Confusion??
- Rule to be remembered:
  - An "else" clause is associated with the closest preceding unmatched "if".
  - Some examples shown next.

```
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

else if e2 s2

if e1 s1

else if e2 s2

if e1 s1

else if e2 s2

else s3

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

else s3

if e1 if e2 s1

else s2

if e1 if e2 s1

else s2

Example

```
#include <stdio.h>
main()
{
    int  a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if (a>=b)
          if (a>=c)  printf ("\n The largest is: %d", a);
       else     printf ("\n The largest is: %d", c);
    else
             if (b>=c)
          printf ("\n The largest is: %d", b);
             else     printf ("\n The largest  is: %d",
  c);
}
```

Example

```
#include <stdio.h>
main()
{
    int  a,b,c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>=b) && (a>=c))
        printf ("\n Largest number is: %d", a);
    else if (b>c)
             printf ("\n Largest number is: %d", b);
         else
             printf ("\n Largest number is: %d", c);
}
```

The Conditional Operator ? :

- This makes use of an expression that is either true or false. An appropriate value is selected, depending on the outcome of the logical expression.
- Example:

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

- Examples:

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

(marks>=60) ? printf("Passed \n") : printf("Failed \n");
```

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

Example

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

The break Statement

- Used to exit from a switch or terminate from a loop.
  - Already illustrated in the previous example.
- 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 block.

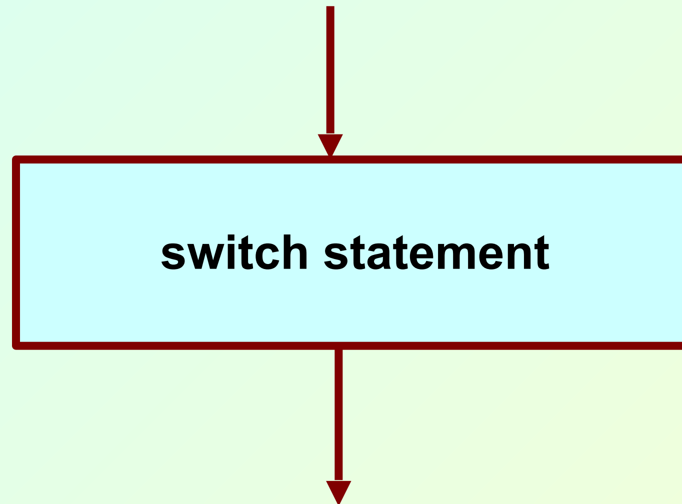
Example

```
switch  (choice = getchar())  {

   case 'r':
   case 'R':  printf ("RED \n");
              break;
   case 'g':
   case 'G':  printf ("GREEN \n");
              break;
   case 'b':
   case 'B':  printf ("BLUE \n");
              break;
   default:   printf ("Invalid choice \n");

}
```

Example

```
switch  (choice = toupper(getchar()))  {

   case 'R':  printf ("RED \n");
              break;
   case 'G':  printf ("GREEN \n");
              break;
   case 'B':  printf ("BLUE \n");
              break;
   default:   printf ("Invalid choice \n");
}
```

- The "switch" statement also constitutes a single-entry / single-exit structure.



A Look Back at Arithmetic Operators: the Increment and Decrement

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`

- 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.
  - Operand will be altered in value ***after*** it is utilized for its intended purpose in the program.

Examples

<u>`Initial values :: a = 10; b = 20;`</u>

`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)`

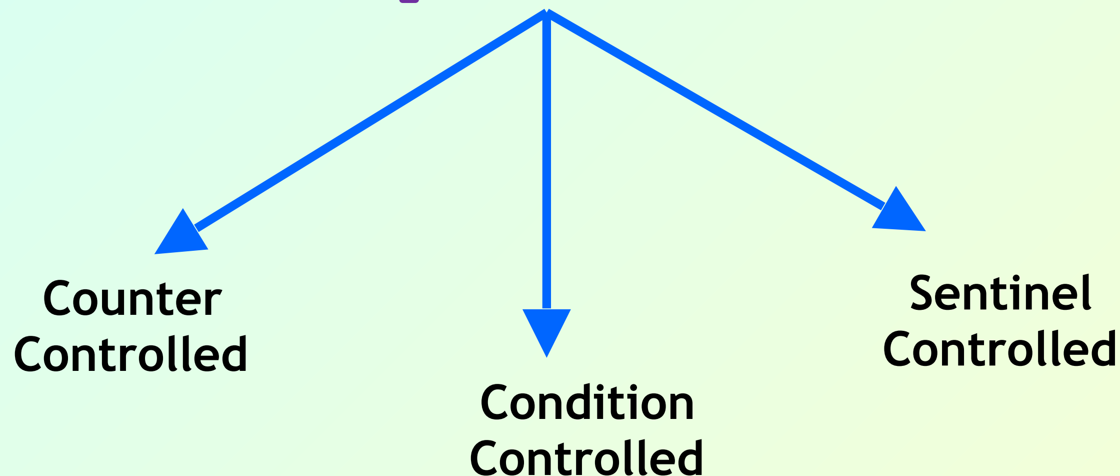
Called side effects:: while calculating some values, something else get changed.

Control Structures that Allow Repetition

Types of Repeated Execution

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

How loops are controlled?



- Counter-controlled repetition
  - Definite repetition – know how many times loop will execute.
  - Control variable used to count repetitions.
- Condition-controlled repetition
  - Loop executes as long as some specified condition is true.
- Sentinel-controlled repetition
  - Indefinite repetition.
  - Used when number of repetitions not known.
  - Sentinel value indicates "***end of data***".

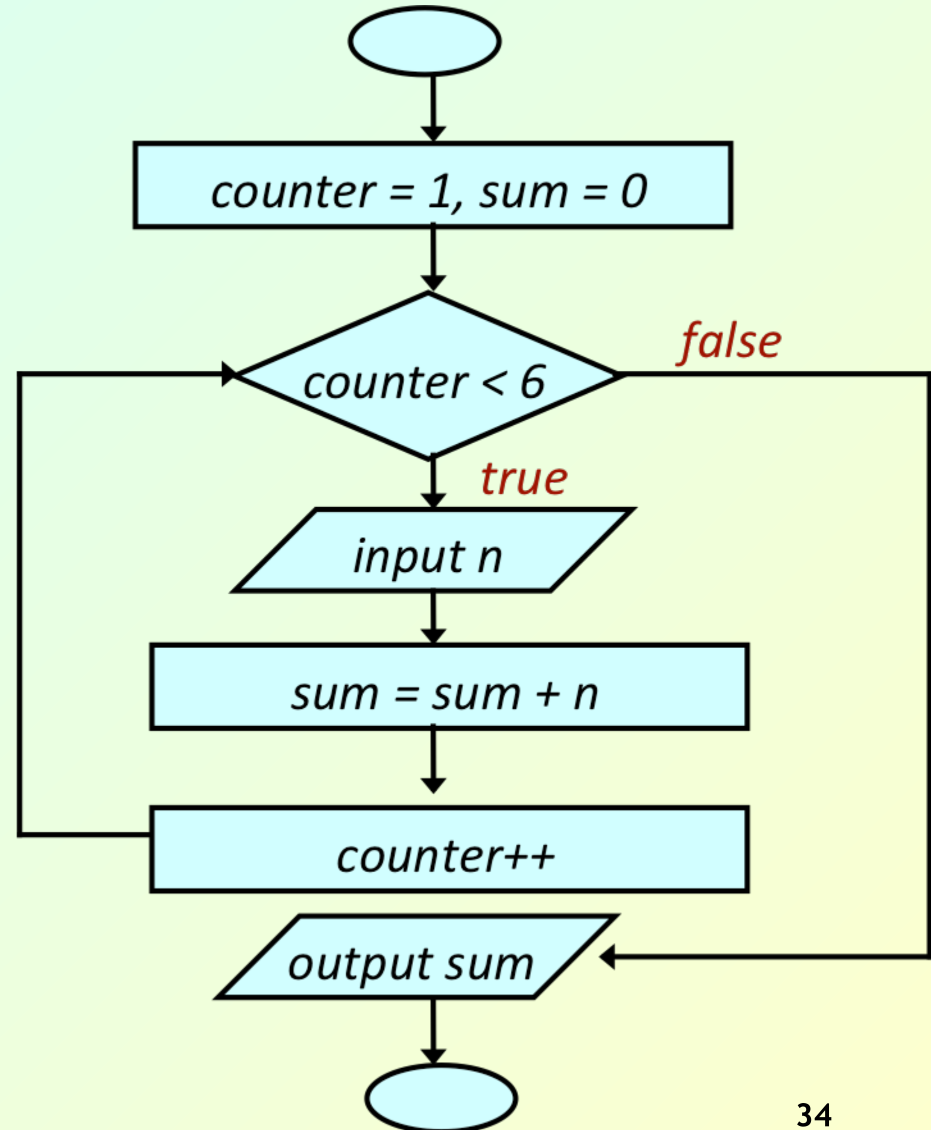
Counter-controlled Repetition

- Counter-controlled repetition requires:
  - *name* of a control variable (or loop counter).
  - *initial value* of the control variable.
  - *condition* that tests for the final value of the control variable (i.e., whether looping should continue).
  - *increment (or decrement)* by which the control variable is modified each time through the loop.

Counter Controlled Loop

Read 5 integers and display the value of their sum.

```
int counter=1, sum=0, n;

while (counter <6 ) {
  scanf ("%d", &n);
  sum = sum + n;
  counter++;
}
```



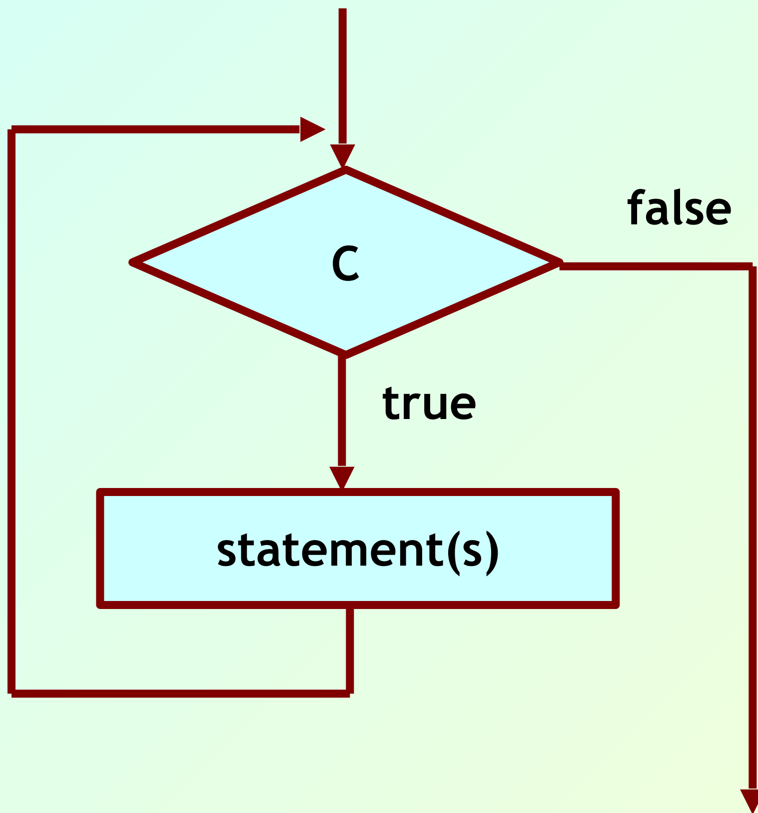
```
int counter, sum=0, n;

for (counter=1; counter<6; counter++)
{
      scanf ("%d", &n);
      sum = sum + n;
}
   printf ("\nSum is: %d", sum);
```

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;
}
```



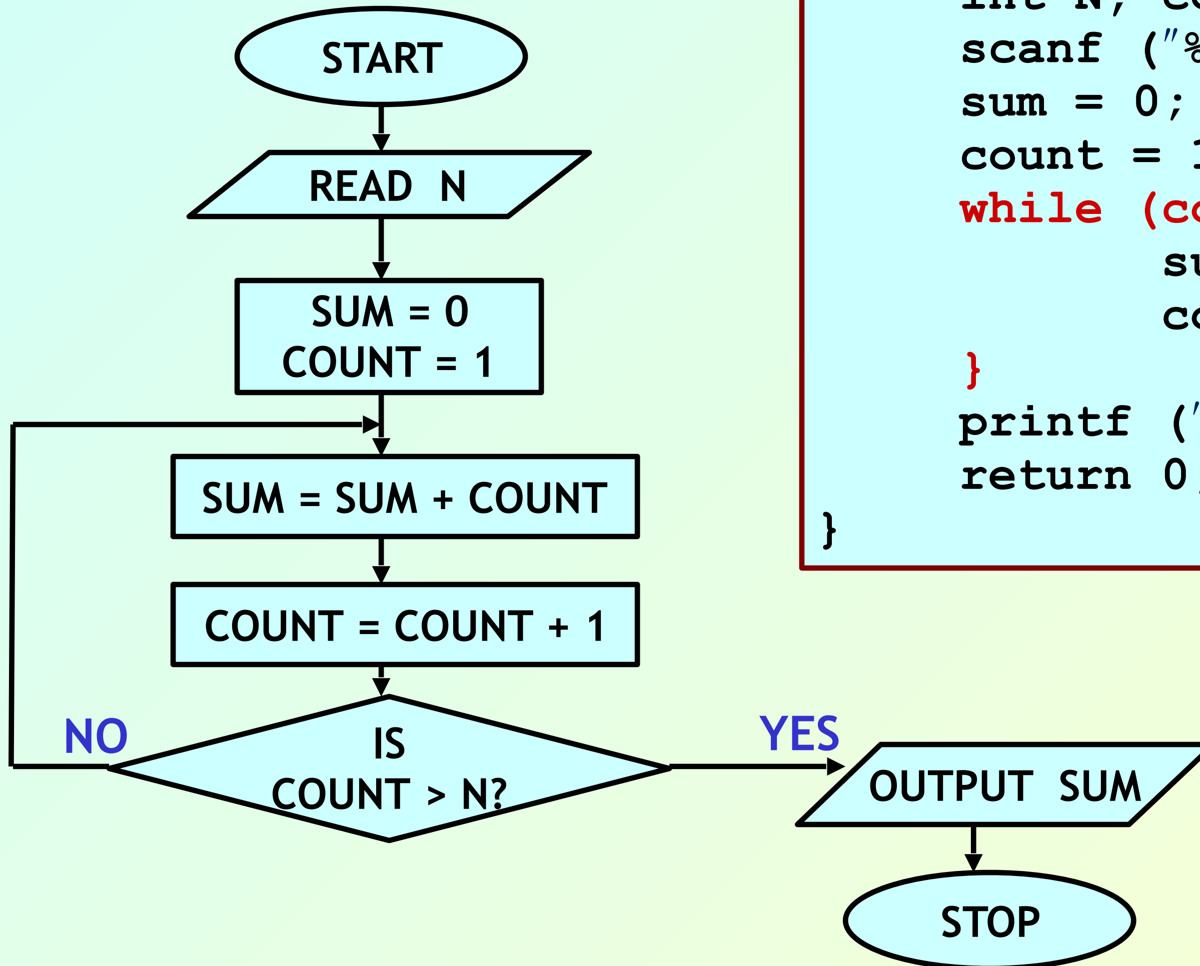
Single-entry / single-exit structure

while :: Examples

```
int  digit = 0;

while  (digit <= 9)
  printf ("%d \n", digit++);
```

```
int  weight=100;

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

Example: Compute 1+2+…+N




```
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;
}
```

Example: Maximum of inputs

```
printf ("Enter positive numbers, end with -1.0\n");
max = 0.0;
scanf("%f", &next);

while (next != -1.0f)   {
       if (next > max)
             max = next;
       scanf("%f", &next);
}
printf ("The maximum number is %f\n", max) ;
```

Example of Sentinel-controlled loop

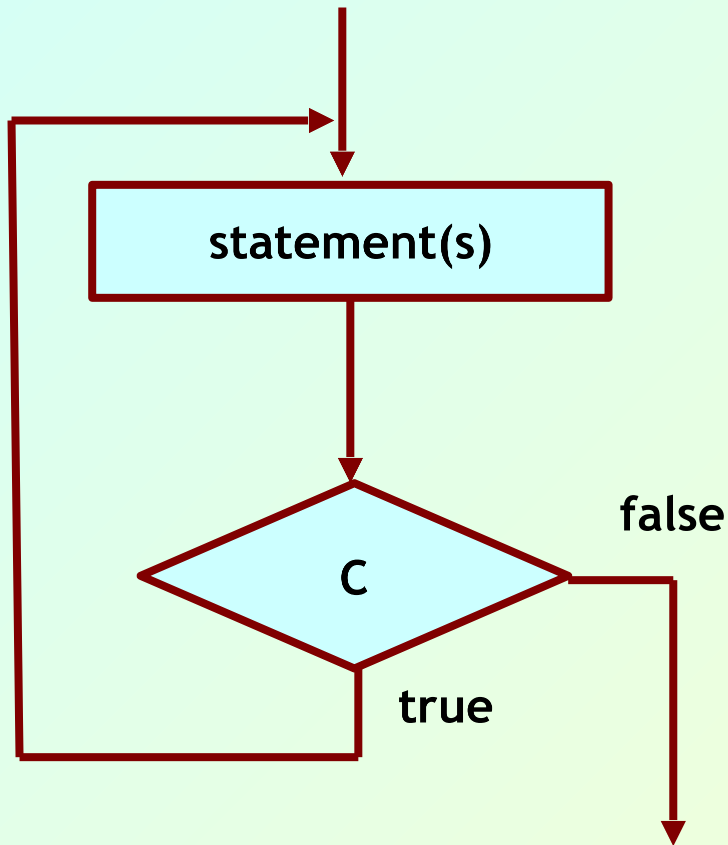
Inputs: 10   5   100   25   68   -1

do-while Statement

- Similar to "while", with the difference that the check for continuation is made at the ***end*** of each pass.
  - In "while", the check is made at the ***beginning***.
- Loop body is executed ***at least once***.

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

```
do {
        statement-1;
        statement-2;

        statement-n;
      }   while (condition );
```



Single-entry / single-exit structure

do-while :: Examples

```
int  digit = 0;

do
   printf ("%d \n", digit++);
while (digit <= 9);
```

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

for Statement

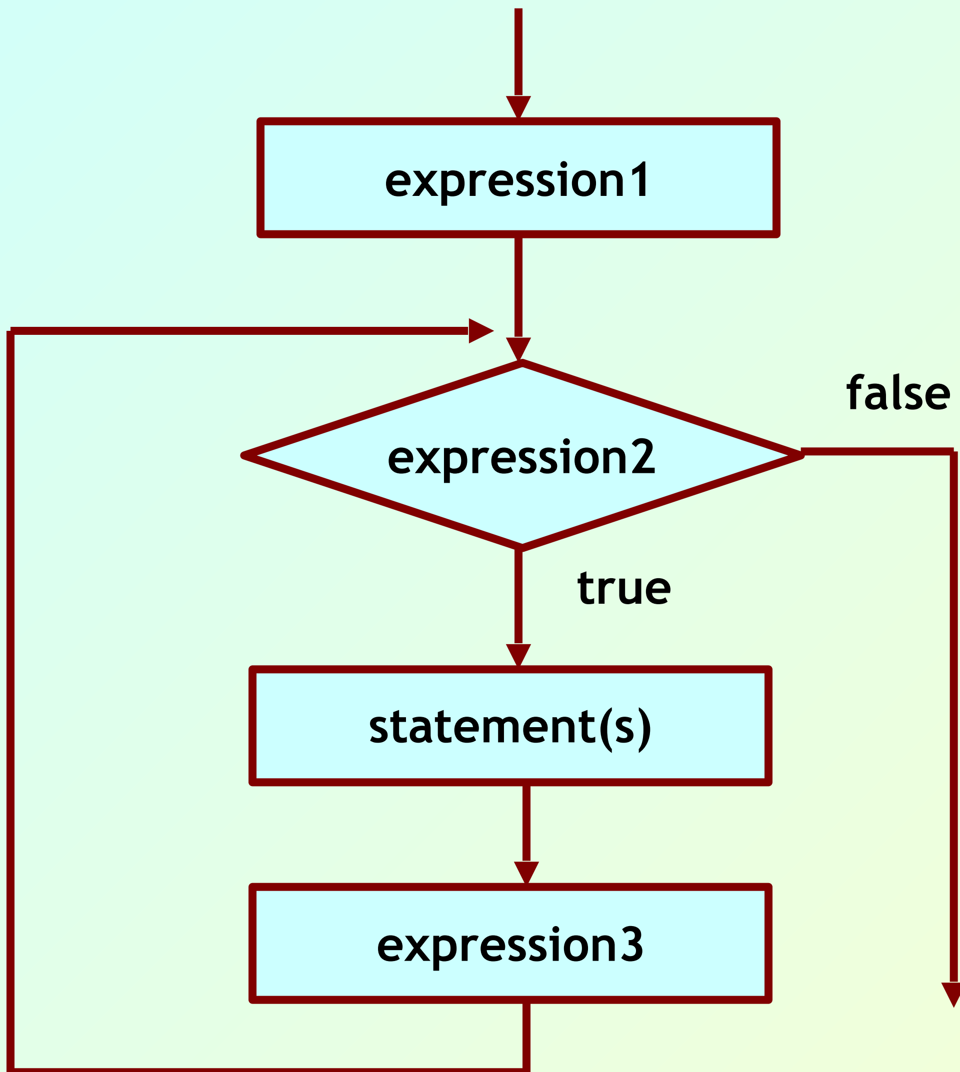
- The "for" statement is the most commonly used looping structure in C.
- General syntax:

```
for (expression1; expression2; expression3)
    statement-to-repeat;

for (expression1; expression2; expression3)
{
    statement_1;
     :
    statement_N;
}
```

- How it works?
  - "expression1" is used to ***initialize*** some variable (called ***index***) that controls the looping action.
  - "expression2" represents a ***condition*** that must be true for the loop to continue.
  - "expression3" is used to ***alter*** the value of the ***index*** initially assigned by "expression1".

```
int  digit;
for (digit=0; digit<=9;digit++)
  printf ("%d \n", digit);
```

```
int  digit;
for (digit=9;digit>=0;digit--)
   printf ("%d \n", digit);
```



Single-entry / single-exit structure

for :: Examples

```
int  fact = 1, i, N;

scanf ("%d", &N);

for  (i=1; i<=N; i++)
   fact = fact * i;
printf ("%d \n", fact);
```

Compute factorial

```
int  sum = 0, N, i;

scanf ("%d", &N);

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

printf ("%d \n", sum);
```

Compute $1^2+2^2+\ldots+N^2$

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

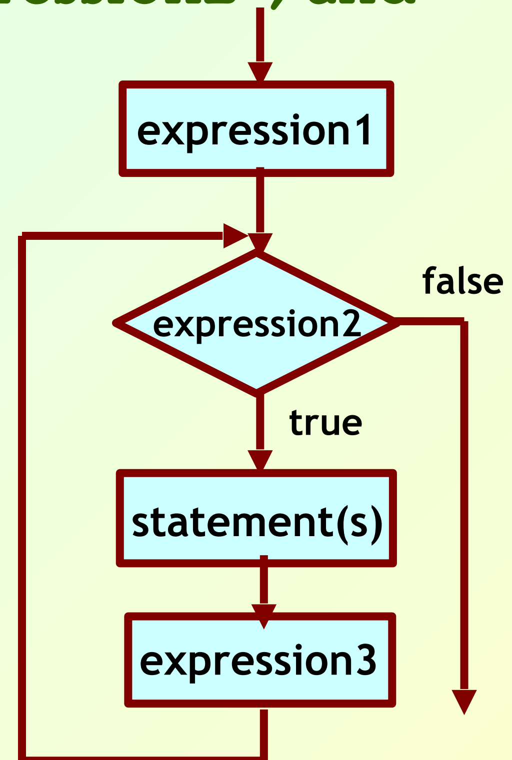
Print

```
*

* *

* * *

* * * *

* * * * *
```

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

- 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)
```

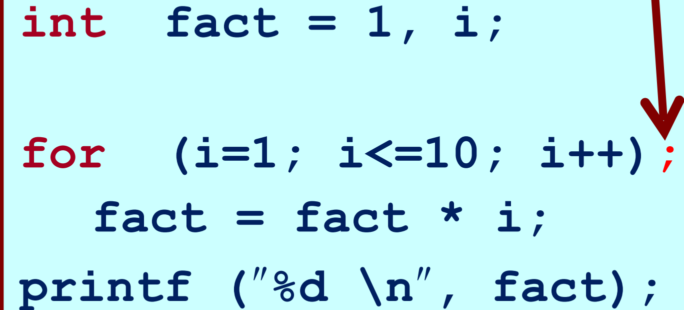
- "Increment" may be negative (decrement)

```
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.

A common mistake (; at the end)

```
int  fact = 1, i;

for  (i=1; i<=10; i++)
   fact = fact * i;
printf ("%d \n", fact);
```



"Loop body" will execute only once!

Specifying "Infinite Loop"

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

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

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

The "break" Statement Revisited

- 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 with "*break*"

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

   fact = 1;  i = 1;

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

The "continue" Statement

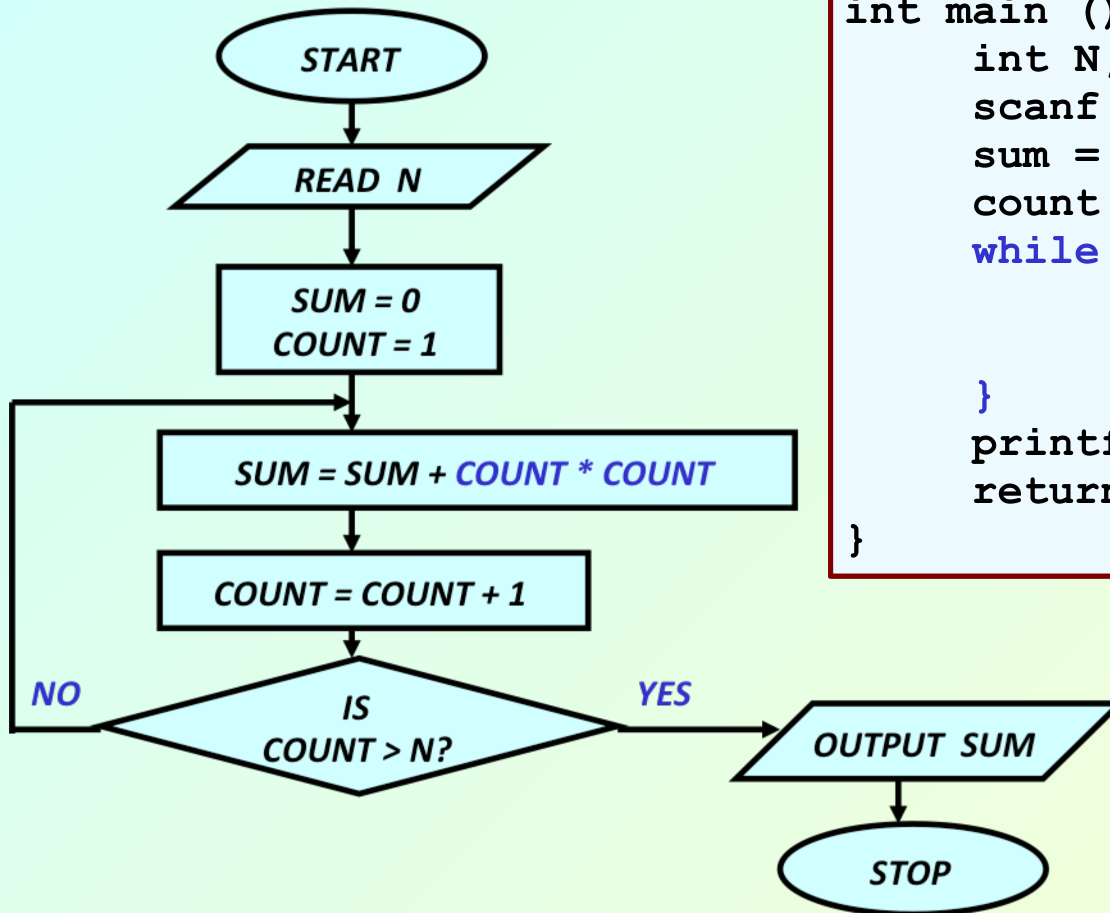
- Skips the remaining statements in the body of a *while, for* or *do/while* structure.
  - Proceeds with the next iteration of the loop.
- while and do/while
  - Loop-continuation test is evaluated immediately after the continue statement is executed.
- for structure
  - *expression3* is evaluated,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

An example with “*break*” and “*continue*”

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

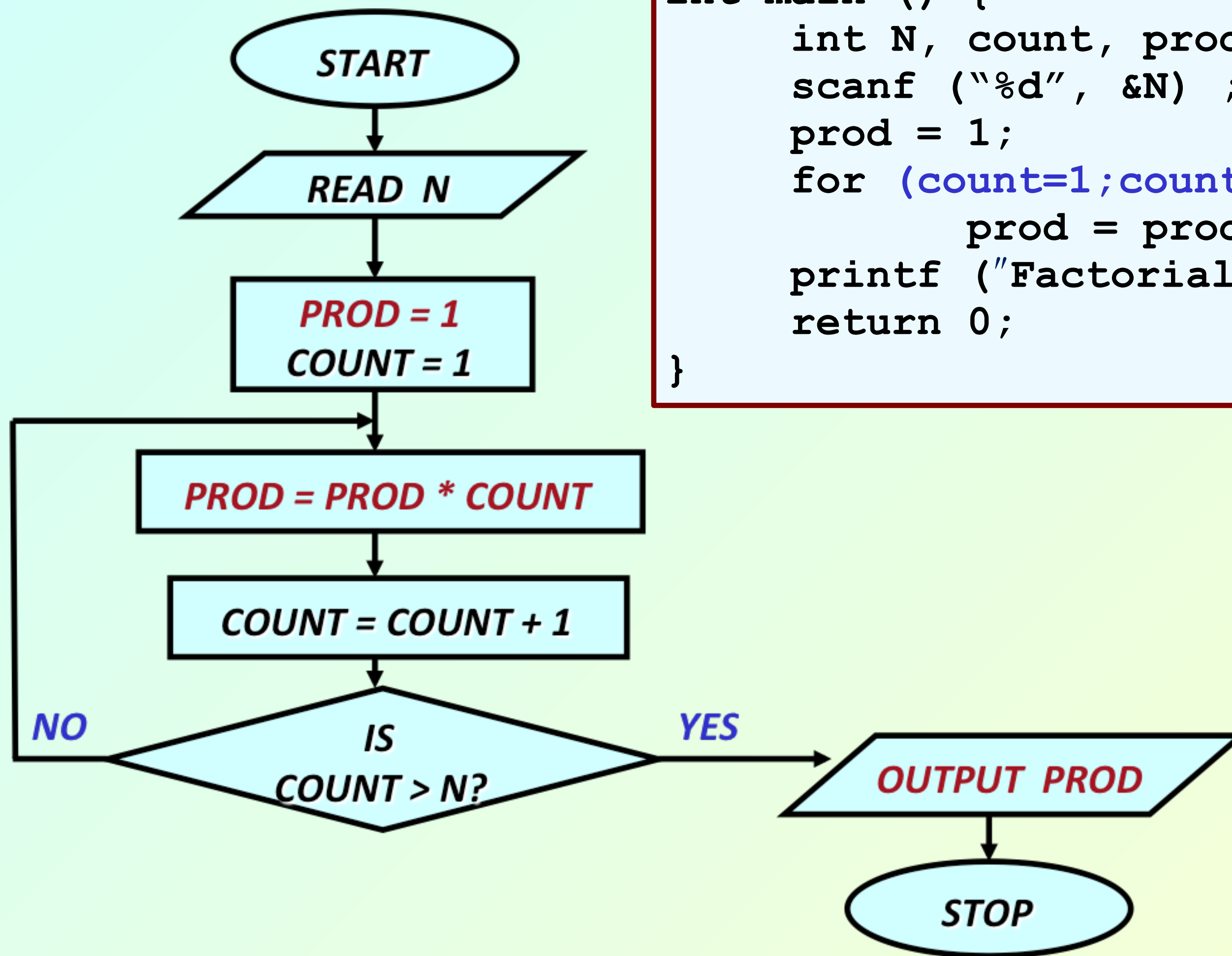
Some Examples

Example: $SUM = 1^2 + 2^2 + 3^2 + \ldots + N^2$



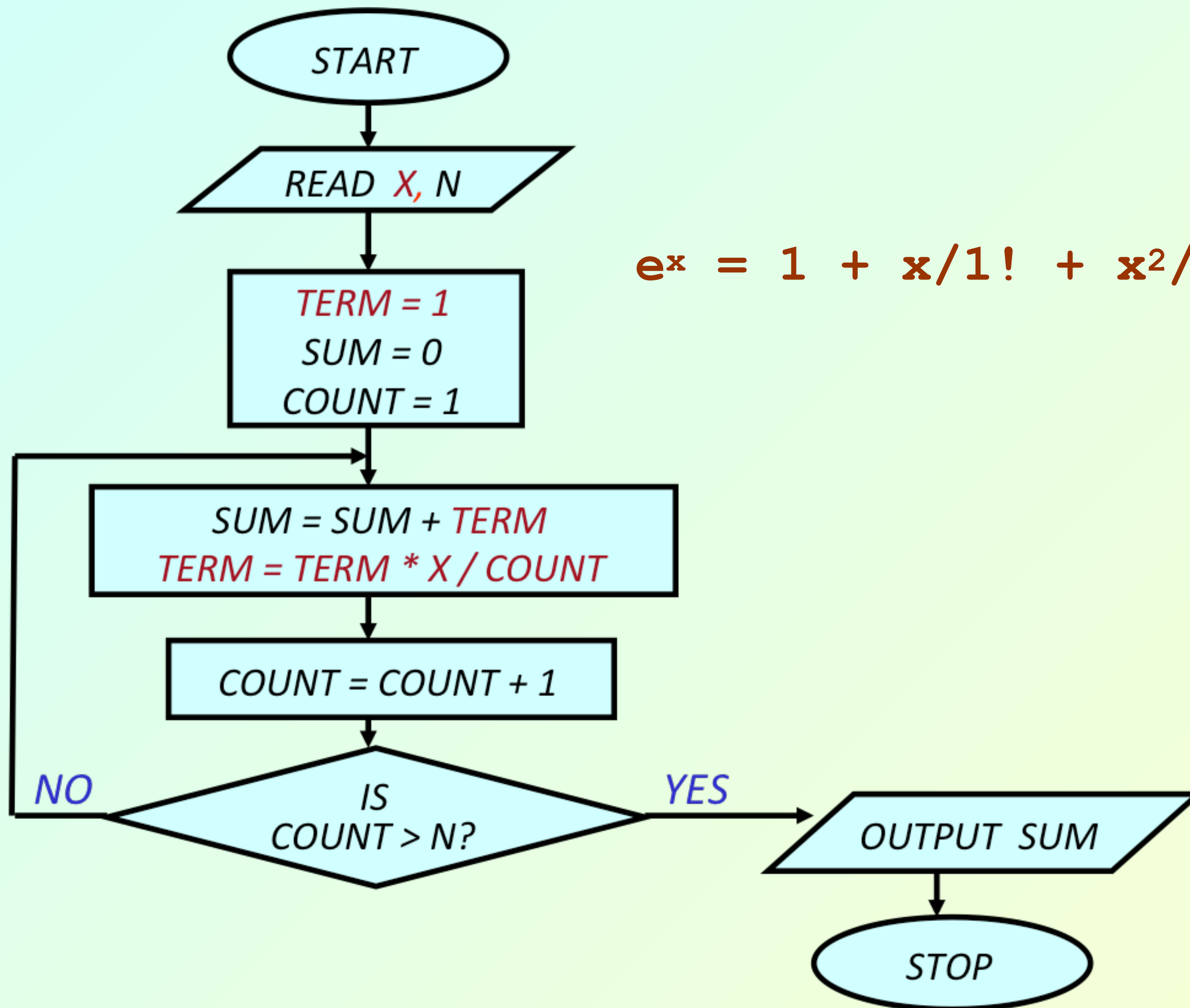

```c
int main () {
      int N, count, sum;
      scanf ("%d", &N) ;
      sum = 0;
      count = 1;
      while (count <= N)   {
              sum = sum + count*count;
              count = count + 1;
      }
      printf ("Sum = %d\n", sum) ;
      return 0;
}
```

Example: *Computing Factorial*



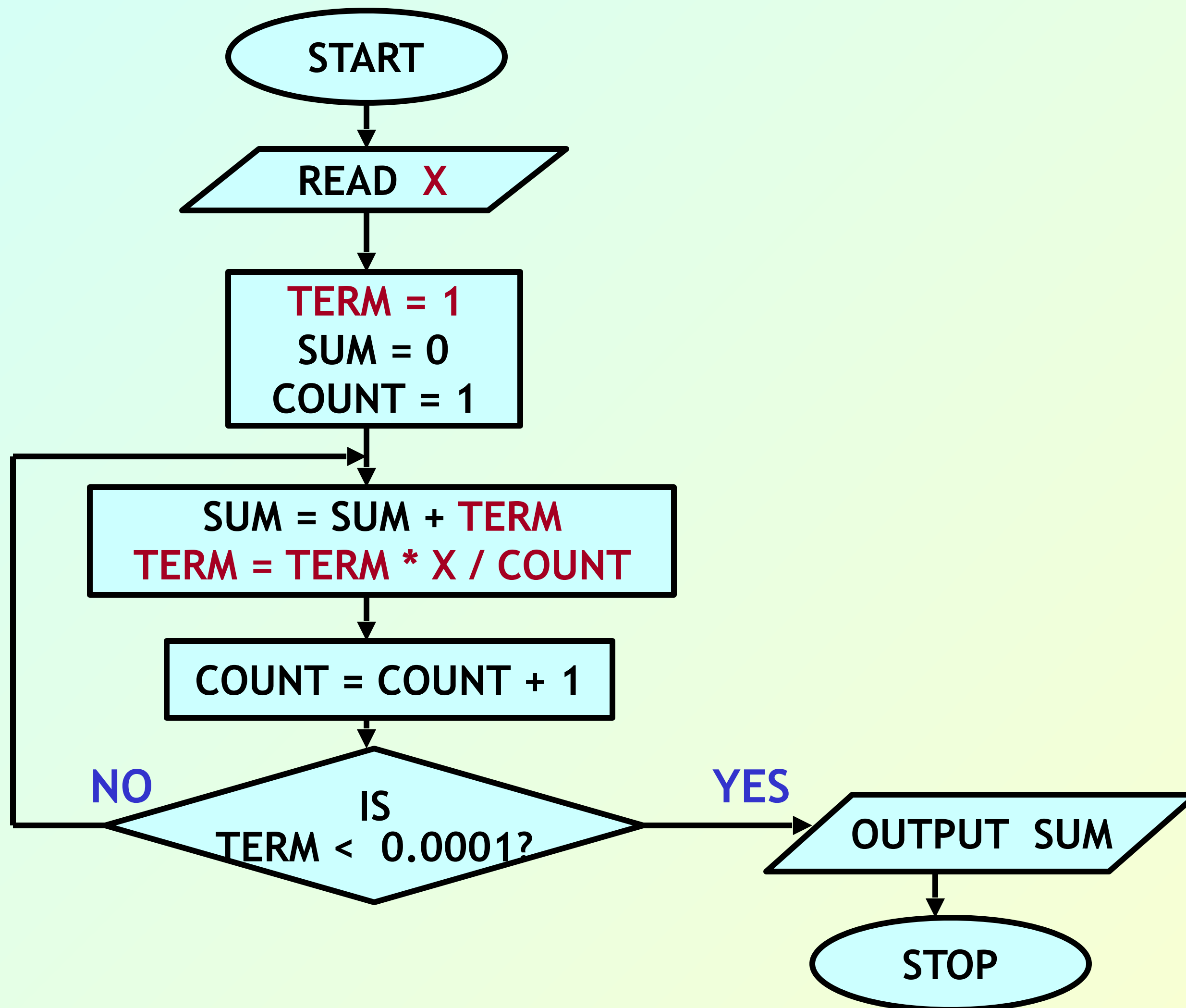

```
int main () {
      int N, count, prod;
      scanf ("%d", &N) ;
      prod = 1;
      for (count=1;count <= N; count++)  {
             prod = prod*count;
      printf ("Factorial = %d\n", prod) ;
      return 0;
}
```

Example: Computing e^x series up to N terms



$e^x = 1 + x/1! + x^2/2! + x^3/3! + \ldots$

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



Example: *Test if a number is prime or not*

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

Example: *Find the sum of digits of a number*

```
#include  <stdio.h>
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: *Decimal to binary conversion*

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

Example: *Compute GCD of two numbers*

```
#include  <stdio.h>
main()
{
   int  A, B, temp;
   scanf ("%d %d", &A, &B);
   if  (A > B)
     {temp = A;  A = B;  B = temp;}
   while ((B % A) != 0)  {
        temp = B % A;
        B = A;
        A = temp;
   }
   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

Shortcuts in Assignments

- Additional assignment operators:

  + =, – =, * =, / =, % =

a += b is equivalent to a = a + b

a *= (b+10) is equivalent to a = a * (b + 10)

and so on.

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 same as in scanf.
- Can specify the width of the data fields.
  - %5d, %7.2f, etc.

- 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.

An example

```
#include  <stdio.h>
main()
{
   int fahr;

    for (fahr=0; fahr<=100; fahr+=20)
      printf ("%3d %6.3f\n",
           fahr, (5.0/9.0)*(fahr-32));
}
```

```
  0 -17.778
 20  -6.667
 40   4.444
 60  15.556
 80  26.667
100  37.778
```

Print with leading zeros

```
#include  <stdio.h>
main()
{
   int fahr;

   for (fahr=0; fahr<=100; fahr+=20)
     printf ("%03d %6.3f\n",
         fahr, (5.0/9.0)*(fahr-32));
}
```

```
000 -17.778
020  -6.667
040   4.444
060  15.556
080  26.667
100  37.778
```