# CS11001/CS11002
# Programming and Data Structures
# (PDS) (Theory: 3-0-0)

Class Teacher: Pralay Mitra
Jayanta Mukhopadhyay
Soumya K Ghosh

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

## 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 |

## Relational Operators

```
int x = 20;
int y=3;
float a=20.3;

if  ( x > y )                        /* 20 > 3 ➔ True */
   printf (“%d is larger\n”, x);

If ( x + x > y * 6 )          /* 20+20 > 3*6 ➔ (20+20)>(3*6) ➔ True */
    printf(“Double of %d is larger than 6 times %d”,x,y);

If ( x > a )                         /* Type cast??? */
    printf(“%d is larger than %f”,x, a);
else
    printf(“%d is smaller than %f”,x, a);
```

## Logical Operators

- Unary and Binary  Operators

  ! ➔ Logical NOT, logical negation (True if the operand is False.)

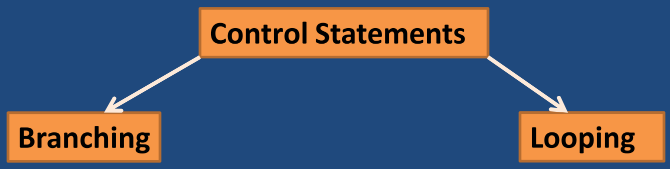
  && ➔ Logical AND  (True if both the operands are True.)

  | | ➔ Logical OR (True if either one of the operands is True.)

| X | !X |
|---|---|
| FALSE | TRUE |
| TRUE | FALSE |

| X | Y | X && Y | X \|\| Y |
|---|---|---|---|
| FALSE | FALSE | FALSE | FALSE |
| FALSE | TRUE | FALSE | TRUE |
| TRUE | FALSE | FALSE | TRUE |
| TRUE | TRUE | TRUE | TRUE |

```
int x = 20;
int y=3;
float a=20.3;

if((x>y) && (x>a))     /* FALSE */
        printf(“X is largest.”);

if((x>y) || (x>a))      /* TRUE */
        printf(“X is not smallest.”);

if(!(x==y))              /* TRUE */
        printf(“X is not same as Y.”);

if(x!=y)                 /* TRUE */
        printf(“X is not same as Y.”);
```



Statement takes more than one branches based upon a **condition test** comprising of relational and/or logical (may be arithmetic) operators.

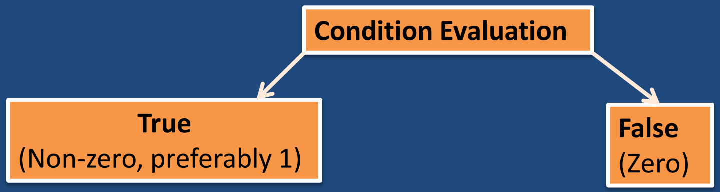
Some set of statements are being executed **iteratively** until a **condition test** comprising of relational and/or logical (may be arithmetic) operators are not being satisfied.

## Conditions

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

# Condition Tests

```
if(count <= 100)                    /* Relational */
if((math+phys+chem)/3 >= 60)        /* Arithmetic, Relational */
if((sex==‘M’) && (age>=21))         /* Relational, Logical */
if((marks>=80) && (marks<90) )      /* Relational, Logical */
if((balance>5000) | | (no_of_trans>25))     /* Relational, Logical */
if(! (grade==‘A’))                          /* Relational, Logical */
```



**Condition Evaluation**

**True**
(Non-zero, preferably 1)

**False**
(Zero)

# Operator confusion

**Equality (==) and Assignment (=) Operators**

- What is expected in condition?
  - Nonzero values are true, zero values are false
  - Any expression that produces a value can be used in control structures

```
int age=20;
         if ( age > 18 )   /* Logical Operator; Evaluated as TRUE */
           printf( "You are not a minor!\n" );

        if ( age >= 18 ) /* Logical Operator; Evaluated as TRUE */
           printf( "You are not a minor!\n" );

        if ( age == 20 ) /* Logical Operator; Evaluated as TRUE */
           printf( "You are not a minor!\n" );

        if ( age = 18 )  /* Arithmetic Operator; Evaluated as TRUE */
           printf( "You are not a minor!\n" );

         if ( age = 17 ) /* Arithmetic Operator; Evaluated as TRUE */
           printf( "You are not a minor!\n" );
```

# Operator confusion

## Equality (==) and Assignment (=) Operators

int age=20;

Better is avoid.

```
 if ( age > 18 )    /*Logical Operator; Evaluated TRUE*/
   printf( "You are not a minor!\n" );

if ( age >= 18 )   /*Logical Operator; Evaluated TRUE*/
   printf( "You are not a minor!\n" );

if ( age == 20 ) /*Logical Operator; Evaluated TRUE*/
   printf( "You are not a minor!\n" );

if ( age = 18 )  /*Arithmetic Operator; Evaluated TRUE*/
   printf( "You are not a minor!\n" );

 if ( age = 17 )  /*Arithmetic Operator; Evaluated TRUE*/
   printf( "You are not a minor!\n" );
```

Value of age will be 18

Value of age will be 17

These statements are not logically correct!!!

There will be no syntax error.

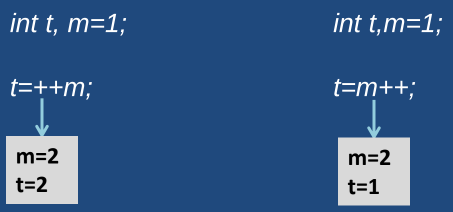
# Operator confusion

## Equality (==) and Assignment (=) Operators

```
#include <stdio.h>
int main()
{
        int x,y;
        scanf("%d",&x);
        y=x%2;      /* y will be 1 or zero based on value entered and stored as x */
        if(y=1) {   /* y will be assigned with 1, condition will be evaluated as TRUE */
                printf("Entered number is odd.");
        } else {
                printf("Entered number is even.");
        }

        return 0;
}
```

# Unary Operator

- Increment (++) Operation means i=i+1;
  - Prefix operation (++i) or Postfix operation (i++)
- Decrement (--) Operation means i=i-1;
  - Prefix operation (--i) or Postfix operation (i--)
- Precedence
  - Prefix operation : First increment / decrement and then used in evaluation
  - Postfix operation : Increment / decrement operation after being used in evaluation
- Example



```
int t, m=1;          int t,m=1;

t=++m;               t=m++;
```

**m=2**
**t=2**

**m=2**
**t=1**

# More Examples on Unary Operator

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

x = 50 + ++a;

a = 11, x = 61

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

x = 50 + a++;

x = 60, a = 11

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

x = a++ + --b;

b = 19, x = 29, a = 11

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

x = a++ – ++a; Undefined value (implementation dependent)

## Shortcuts in Assignment Statements

- A+=C → A=A+C
- A-=B → A=A-B
- A*=D → A=A*D
- A/=E → A=A/E

## Input

```
scanf (“control string”,arg1,arg2, …, argn);
```

– Performs input from the standard input device, which is the keyboard by default.

– It requires a control string refers to a string typically containing data types of the arguments to be read in.

– And the (arguments) address or pointers of the list of variables into which the value received from the input device will be stored.

– The address of the variables in memory are required to mention (& before the variable name) to store the data.

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

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

# Input

| Conversion Character | Data Item meaning |
|---|---|
| c | Single chacater |
| d | Decimal integer |
| e | Floating point value |
| f | Floating point value |
| g | Floating point value |
| h | Short int |
| i | Decimal/hexadecimal/octal integer |
| o | Octal integer |
| s | String |
| u | Unsigned decimal integer |
| 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);`

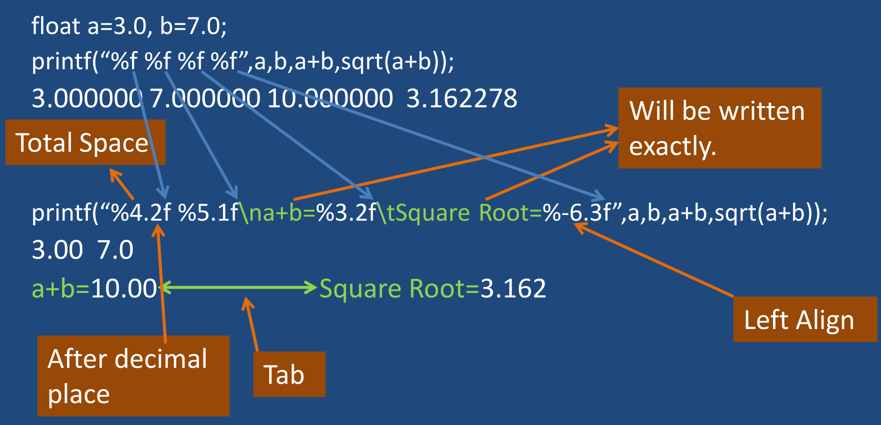
# Output

```
printf (“control string”,arg1,arg2, …, argn);
```

- Performs output to the standard output device (typically defined to be the screen).
- 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.

```
int size,a,b;
float length;
scanf ("%d", &size) ;            printf(“%d”,size);
scanf ("%f", &length) ;          printf(“%f”,length);
scanf (“%d  %d”, &a, &b);        printf(“%d %d”,a,b);
```

# Formatted Output



```
float a=3.0, b=7.0;
printf("%f %f %f %f",a,b,a+b,sqrt(a+b));
3.000000 7.000000 10.000000  3.162278
```

```
printf("%4.2f %5.1f\na+b=%3.2f\tSquare Root=%-6.3f",a,b,a+b,sqrt(a+b));
3.00  7.0
a+b=10.00	Square Root=3.162
```

For integer, character and string decimal point will not be there. Rest is same.

# Character I/O

```
char ch1;
scanf("%c",&ch1);           /* Reads a character */
printf("%c",ch1);           /* Prints a character */
ch1=getchar();          /* Reads a character */
putchar(ch1);           /* Prints a character */
```

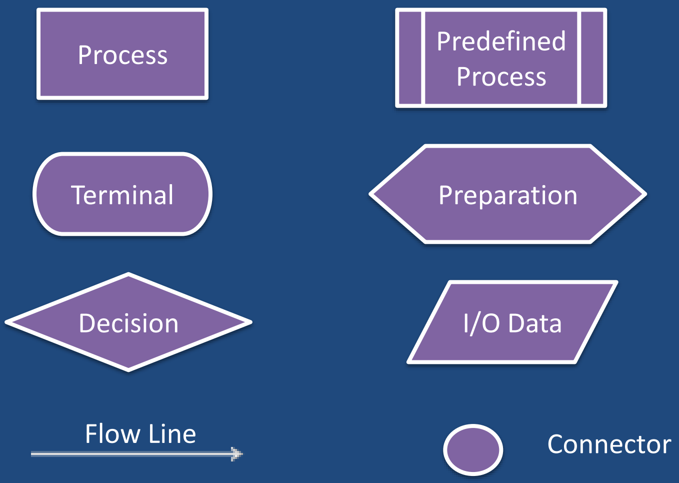
```
char name[20];
scanf("%s",name);           /* Reads a string */
printf("%s",name);          /* Prints a string */
gets(name);                 /* Reads a string */
puts(name);                 /* Prints a string */
```

Help for any command:
**$ man gets**

## Problem solving

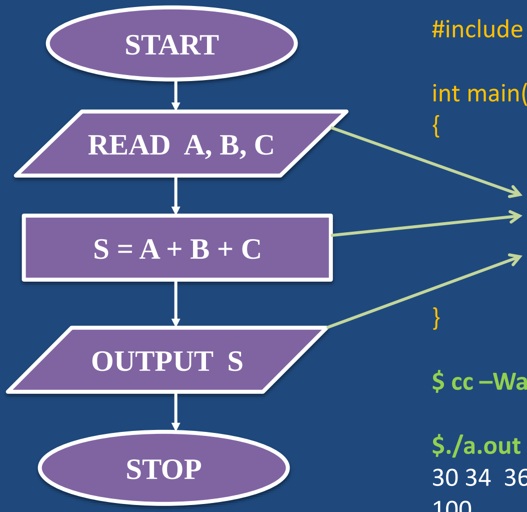
- Step 1:
  - Clearly specify the problem to be solved.
- Step 2:
  - Draw flowchart or write algorithm.
- Step 3:
  - Convert flowchart (algorithm) into program code.
- Step 4:
  - Compile the program into executable file.
- Step 5:
  - For any compilation error, go back to step 3 for debugging.
- Step 6:
  - Execute the executable file (program).

## Flowchart: basic symbols



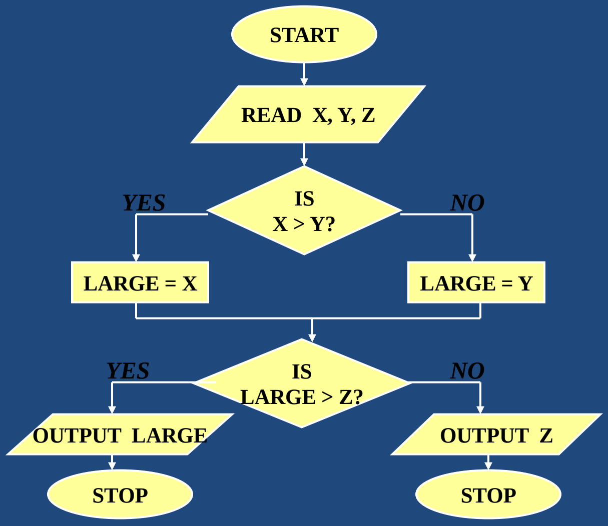
# Example 1

Problem: Add three numbers.



```
#include <stdio.h>    /* example1.c */

int main()
{
          int A,B,C,S;
          scanf(“%d %d %d”,&A,&B,&C);
          S=A+B+C;
          printf(“The sum is: %d”,S);
          return 0;
}

$ cc –Wall example1.c

$./a.out
30 34  36
100
```

# Example 2: find the largest among three numbers

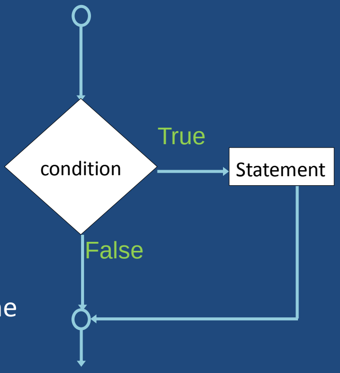


# Branching: *if* Statement

- General syntax:

  **if (condition) { …….. }**

- Test the condition, and follow appropriate path.
- Contains an expression that can be TRUE or FALSE.
- Single-entry / single-exit structure.
- If there is a single statement in the block, the braces can be omitted.


condition
True
Statement
False

```
if (basicPay<18000)
        printf("Bonus Applicable");
```

```
if (basicPay<18000)
{
        int bonus;
        bonus=basicPay*0.30;
        printf("Bonus is %d",bonus);
}
```

# Branching: *if-else* Statement

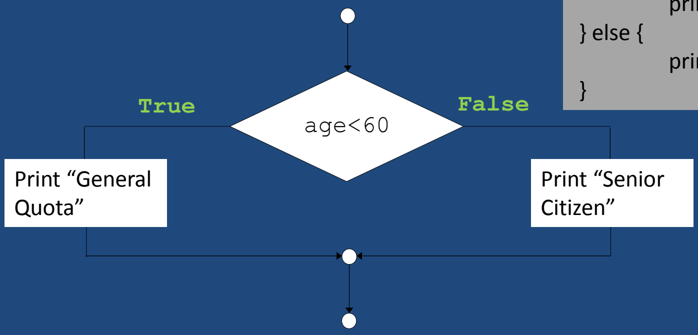
- General syntax:

  **if (condition) { …… block 1 ……. }**

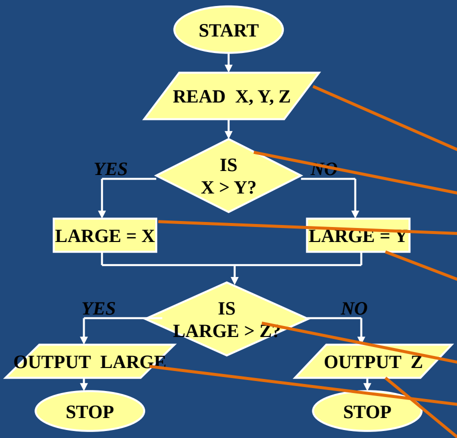
  **else { …….. block 2 …….. }**

- 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.
- If a block contains a single statement, the braces can be omitted.

```
if(age<60) {
        printf("General Quota");
} else {
        printf("Senior Citizen");
}
```


True
age<60
False
Print "General Quota"
Print "Senior Citizen"

## Example 2: find the largest among three numbers



```
#include <stdio.h>
int main()
{
   int  X,Y,Z,Large;
   scanf (“%d %d %d”, &X, &Y, &Z);
   if (X>Y)    {
      Large=X;
   }    else {
      Large=Y;
    }
    if (Large>Z)
      printf (“\nThe largest number is: %d”,Large);
    else
      printf (“\n The largest number is: %d”,Z);
    return 0;
}
```

## Nested branching

- 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”.

- Example:

```
if(age<60) {
        if(age<5) {
                printf(“Kid Quota”);
        } else if (age<10) {
                printf(“Child Quota”);
        } else {
                printf(“General Quota”);
        }
} else {
        printf(“Senior Citizen”);
}
```

## Proper Indentation

```
if c1 s1                       if  c1  s1
else if c2 s2                  else  if  c2  s2

if c1 s1                       if  c1  s1
else if c2 s2                  else  if  c2  s2
else s3                              else s3

if c1 if c2 s1                  if  c1  if  c2  s1
else s2                                 else  s2
else s3                        else  s3

if c1 if c2 s1                 if  c1  if  c2  s1
else s2                                 else s2
```

## Desirable Programming Style

- Clarity
  - The program should be clearly written.
  - It should be easy to follow the program logic.
- Meaningful variable names
  - Make variable/constant names meaningful to enhance program clarity.
    - ‘area’ instead of ‘a’
    - ‘radius’ instead of ‘r’
- Program documentation
  - Insert comments in the program to make it easy to understand.
  - Never use too many comments.
- Program indentation
  - Use proper indentation.
  - Structure of the program should be immediately visible.

# Indentation Example :: Good Style

```
/* A program to check the age based quota in Indian Railway ticketing system */

#include <stdio.h>
#define SENIOR      60                      /* Declare the age of Senior Citizen */

int main()
{
        int age;
        scanf(“%d”,&age);
        if(age< SENIOR) {
                if(age<5) {
                        printf(“Kid Quota”);
                } else if (age<10) {
                        printf(“Child Quota”);
                } else {
                        printf(“General Quota”);
                }
        } else {
                printf(“Senior Citizen”);
        }
        return 0;
}
```

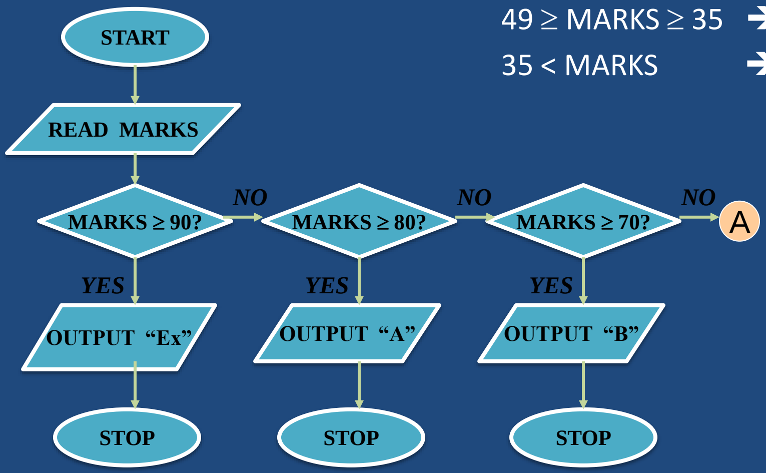
# Indentation Example :: Bad Style

```
#include <stdio.h>
#define SENIOR      60

int main()
{
int age;
scanf(“%d”,&age);
if(age< SENIOR) {
 if(age<5) {
 printf(“Kid Quota”);
 } else if (age<10) {
 printf(“Child Quota”);
 } else {
 printf(“General Quota”);
 }
 } else {
 printf(“Senior Citizen”);
 }
return 0;
}
```

```
#include <stdio.h>
#define SENIOR      60
int main()
{
int age;
scanf(“%d”,&age);
if(age< SENIOR) {
 if(age<5) {  printf(“Kid Quota”); }
 else if (age<10) { printf(“Child Quota”);
 } else {  printf(“General Quota”); }
 } else {  printf(“Senior Citizen”);  }
return 0;
}
```

## Example 3:
## *Grade computation*

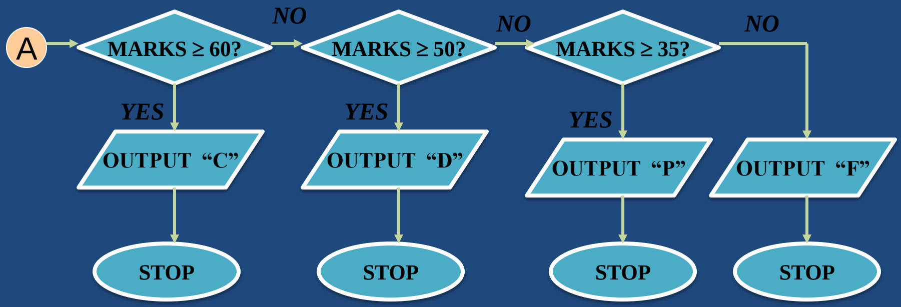
| | | |
|---|---|---|
| MARKS ≥ 90 | → | Ex |
| 89 ≥ MARKS ≥ 80 | → | A |
| 79 ≥ MARKS ≥ 70 | → | B |
| 69 ≥ MARKS ≥ 60 | → | C |
| 59 ≥ MARKS ≥ 50 | → | D |
| 49 ≥ MARKS ≥ 35 | → | P |
| 35 < MARKS | → | F |



## Example 3:
## *Grade computation*

| | | |
|---|---|---|
| MARKS ≥ 90 | → | Ex |
| 89 ≥ MARKS ≥ 80 | → | A |
| 79 ≥ MARKS ≥ 70 | → | B |
| 69 ≥ MARKS ≥ 60 | → | C |
| 59 ≥ MARKS ≥ 50 | → | D |
| 49 ≥ MARKS ≥ 35 | → | P |
| 35 < MARKS | → | F |

**Homework:**
**Convert to a C program**



## Example 4: find the largest among three numbers

**Homework:**
**Convert to a flowchart**

```
#include <stdio.h>

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);
     return 0;
}
```

## Ternary conditional operator (?:)

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

*(condition1)? (expr1): (expr2);*

```
age >= 60 ? printf(“Senior Citizen\n”) : printf(“General Quota\n”);
```

Example:

```
    bonus = (basicPay<18000) ? basicPay*0.30 : basicPay*0.05;
```

*Returns a value*

## *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 expression1: { …….. }
   case expression2: { …….. }

   case expressionm: { …….. }
   default: { ……… }
}
```

## *switch* example

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

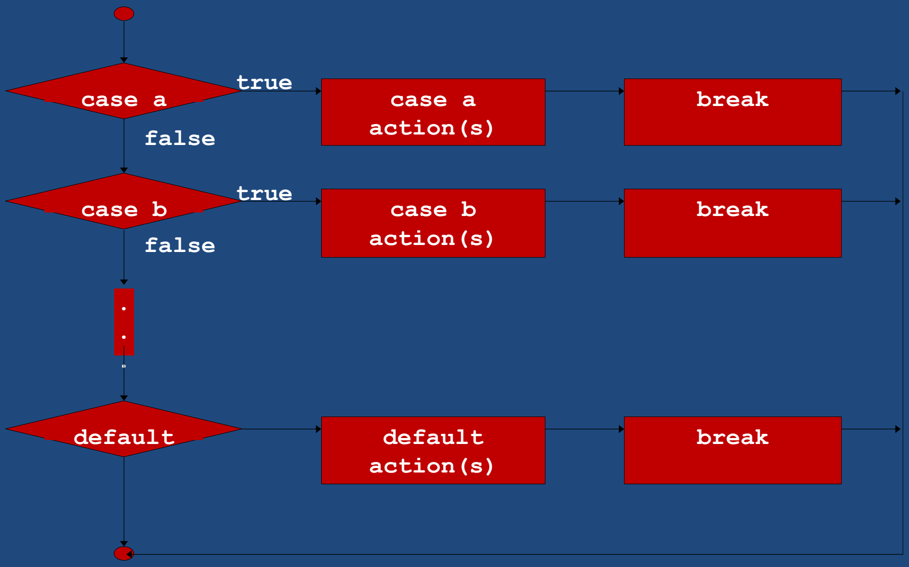
“break” statement is used to break the order of execution.

## The *break* Statement

- Used to exit from a switch or terminate from a loop.

- 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.

## Flowchart for switch statement



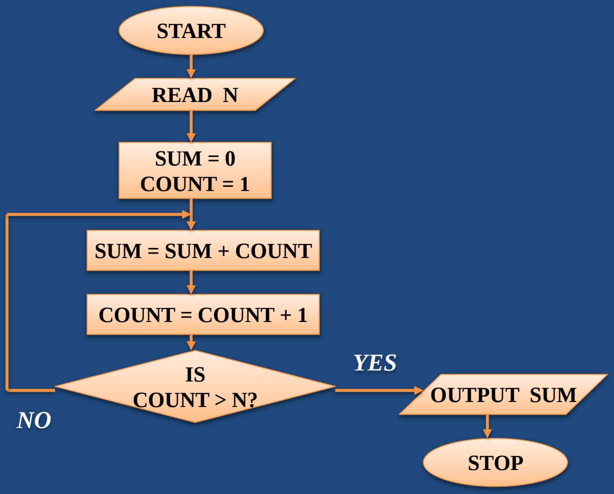
# Example: switch break

```
switch  (primaryColor = getchar())  {


    case ‘R’:        printf (“RED \n”);
                     break;

    case ‘G’:        printf (“GREEN \n”);
                     break;

    case ‘B’:        printf (“BLUE \n”);
                     break;
    default:         printf (“Invalid Color \n”);
                     break;            /* break is not mandatory here */
}
```

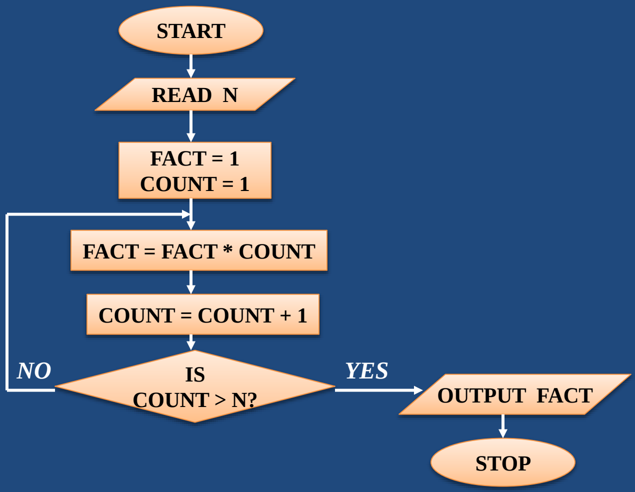
# Example: switch break

```
switch  (primaryColor = 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 Color \n”);

}
```

## Example 5: *Sum of first N natural numbers*



## Example 6: *Computing Factorial*



## Exercise 1: Find the *Roots of a quadratic equation*

$$ax^2 + bx + c = 0$$

Coefficients (a,b,c) are your input.

## The Essentials of Repetition

- Loop
  - Group of instructions computer executes repeatedly while some condition remains true
- Counter-controlled repetition
  - Definite repetition - know how many times loop will execute
  - Control variable used to count repetitions
- Sentinel-controlled repetition
  - Indefinite repetition
  - Used when number of repetitions not known
  - Sentinel value indicates "end of data"

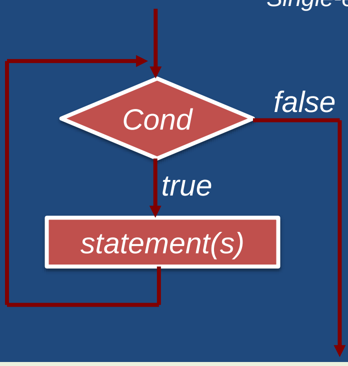
# Counter-Controlled Repetition

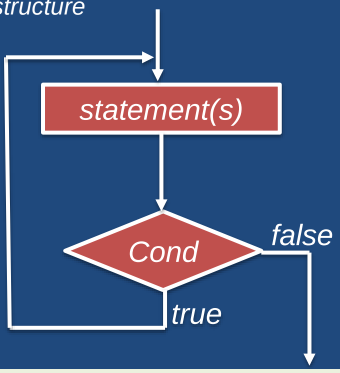
- Counter-controlled repetition requires
  - *name* of a control variable (or loop counter).
  - *initial value* of the control variable.
  - condition that tests for the *final value* of the control variable (i.e., whether looping should continue).
  - *increment* (or *decrement*) by which the control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

```
 int counter =1;                                    /* initialization */
 while (counter <= 10) {                            /* repetition condition */
             printf( "%d\n", counter );
             ++counter;                             //increment
}
```

# Repetition: Flowchart

*Single-entry / single-exit structure*





```
 int counter =1;

 while (counter <= 10) {
             printf( "%d\n", counter );
             ++counter;
}
```

**May not execute at all based on condition.**

```
 int counter =1;

 do {
             printf( "%d\n", counter );
             ++counter;
} while (counter <= 10) ;
```

**Will be executed at least once, whatever be the condition.**

# *while, do-while* Statement

```
while (condition)
      statement_to_repeat;

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

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

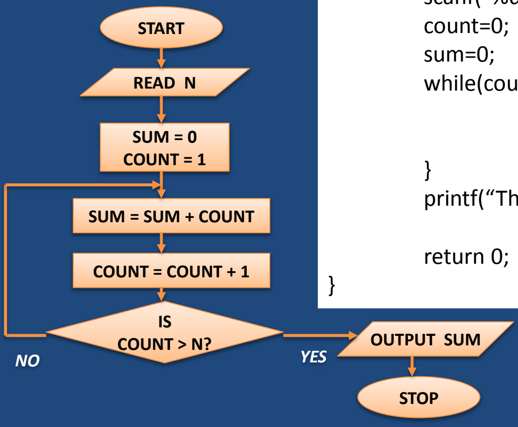
```
int weight=75;
while ( weight > 65 ) {
      printf("Go, exercise, ");
      printf("then come back. \n");
      printf("Enter your weight: ");
      scanf("%d", &weight);
   }
```

```
weight=75;
do {
      printf("Go, exercise, ");
      printf("then come back. \n");
      printf("Enter your weight: ");
      scanf("%d", &weight);
 } while ( weight > 65 ) ;
```

```
do {
            statement-1;
            statement-2;
            ……..
             statement-n;
} while ( condition );
```

**At least one round of exercise is ensured.**

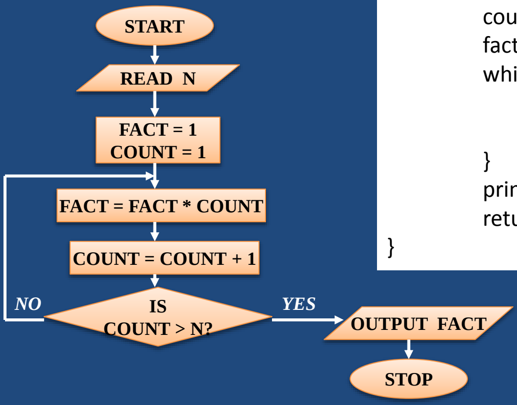
# Example 5: *Sum of first N natural numbers*



```
#include <stdio.h>

int main()
{
            int n, sum, count;
            printf(“Enter a natural number: ”);
            scanf(“%d”,&n);
            count=0;
            sum=0;
            while(count<=n) {
                        sum+=count;         // sum=sum+count
                        count++;
            }
            printf(“The sum of first %d natural numbers is: %d”,
                                          n,sum);
            return 0;
}
```

Line break in a statement is allowed after a comma.

## Example 6: *Computing Factorial*



```
#include <stdio.h>

int main()
{
            int n, fact, count;
            printf("Enter a number: ");
            scanf("%d",&n);
            count=1;
            fact=1;
            while(count<=n) {
                        fact=fact*count;
                        count++;
            }
            printf("The factorial of %d is: %d",n,fact);
            return 0;
}
```

Considering large factorial value, you may declare *fact* as float.

## Example 7: *Computing Factorial*

```
#include <stdio.h>

int main()
{
            int n, fact, count;
            printf("Enter a number: ");
            scanf("%d",&n);
            count=1;
            fact=1;
            while(count<=n) {
                        fact=fact*count;
                        count++;
            }
            printf("%d",fact);
            return 0;
}
```

**count may increment.**

```
#include <stdio.h>

int main()
{
            int n, fact, count;
            printf("Enter a number: ");
            scanf("%d",&n);
            count=n;
            fact=1;
            while(count>=1) {
                        fact=fact*count;
                        count--;
            }
            printf("%d",fact);
            return 0;
}
```

**count may decrement.**

**Loop variable may decrement**

# 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.

```
for (initial; condition; iteration)
          statement_to_repeat;


for (initial; condition; iteration)  {
          statement_to_repeat;
          ……
          statement_to_repeat;
}
```

*All are expressions.*
*initial → expr1*
*condition → expr2*
*iteration → expr3*

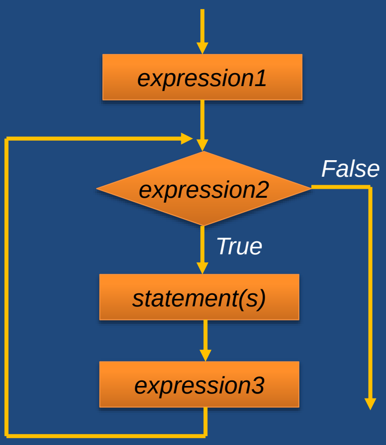
```
fact = 1;    /* Calculate 10! */
for ( i = 1; i < =10; i++)
          fact = fact * i;
```

No semicolon after last expression

## *for* loop

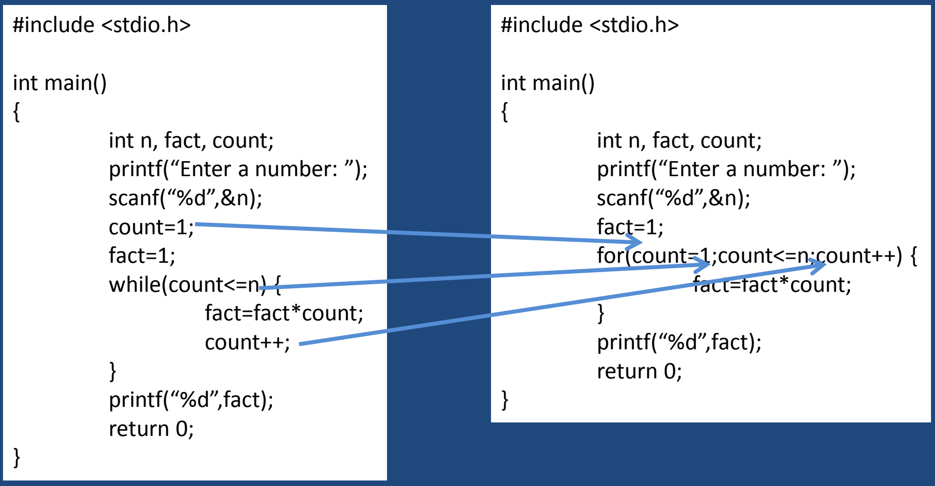
*Single-entry / single-exit structure*

```
for (initial; condition; iteration)  {
          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".

# Example 8: *Computing Factorial*



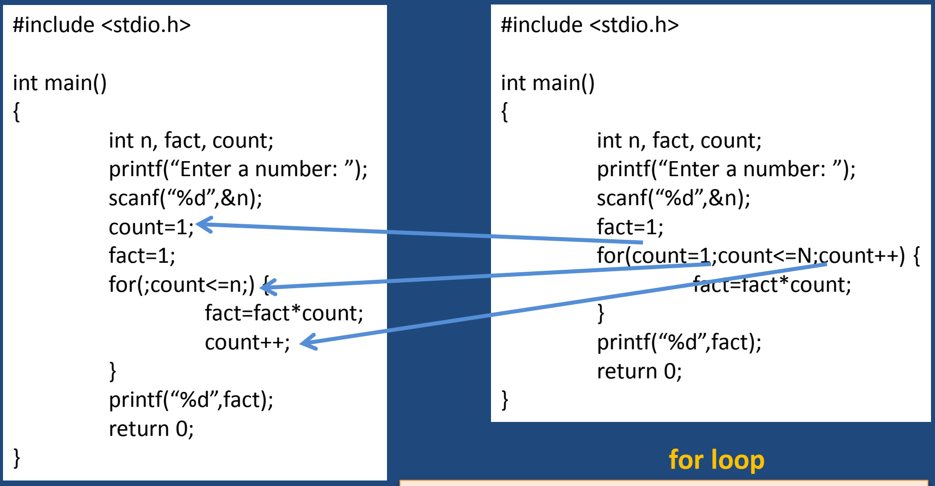
```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        count=1;
        fact=1;
        while(count<=n) {
                fact=fact*count;
                count++;
        }
        printf("%d",fact);
        return 0;
}
```

while loop

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        fact=1;
        for(count=1;count<=n;count++) {
                fact=fact*count;
        }
        printf("%d",fact);
        return 0;
}
```

for loop

# Example 9: *Computing Factorial*



```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        count=1;
        fact=1;
        for(;count<=n;) {
                fact=fact*count;
                count++;
        }
        printf("%d",fact);
        return 0;
}
```

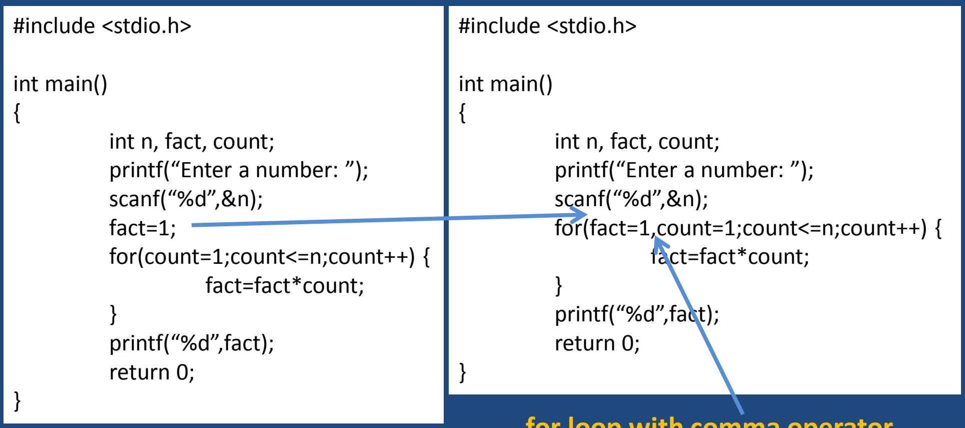
for loop working as while

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        fact=1;
        for(count=1;count<=N;count++) {
                fact=fact*count;
        }
        printf("%d",fact);
        return 0;
}
```

for loop

**Homework:**
Rewrite the factorial using *for* loop and by decrementing count.

# Example 10: *Computing Factorial*



```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        fact=1;
        for(count=1;count<=n;count++) {
                fact=fact*count;
        }
        printf("%d",fact);
        return 0;
}
```

for loop

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        for(fact=1,count=1;count<=n;count++) {
                fact=fact*count;
        }
        printf("%d",fact);
        return 0;
}
```

for loop with comma operator

**The comma operator:**
We can give several statements separated by commas in place of "expression1", "expression2", and "expression3".

# Advanced expression in *for* structure

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.
  - e.g. Let `x = 2` and `y = 10`

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

is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```

"Increment" may be negative (decrement)

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

# Specifying “Infinite Loop”

```
count=1;
while(1) {
            printf(“Count=%d”,count);
            count++;
}
```

```
count=1;
do {
            printf(“Count=%d”,count);
            count++;
} while(1);
```

```
count=1;
for(;;) {
            printf(“Count=%d”,count);
            count++;
}
```

```
for(count=1;;count++) {
            printf(“Count=%d”,count);
}
```

# *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, for, do/while or switch structure
- Program execution continues with the first statement after the structure
- Common uses of the break statement
  - Escape early from a loop
  - Skip the remainder of a switch structure

## Break from “Infinite Loop”

```
count=1;
while(1) {
          printf(“Count=%d”,count);
          count++;
          if(count>100)
                    break;
}
```

```
count=1;
do {
          printf(“Count=%d”,count);
          count++;
          if(count>100)
                    break;
} while(1);
```

```
count=1;
for(;;) {
          printf(“Count=%d”,count);
          count++;
          if(count>100)
                    break;
}
```

```
for(count=1;;count++) {
          printf(“Count=%d”,count);
          if(count>100)
                    break;
}
```

## *continue* Statement

- continue
  - 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
    - Increment expression is executed, then the loop-continuation test is evaluated.
    - *expression3* is evaluated, then *expression2* is evaluated.

## An Example with *break* and *continue*

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

## Example 11: Primality testing

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

## Example 12: Compute GCD of two numbers

```
#include  <stdio.h>
int 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);
     return 0;
}
```

```
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*

## Example 13: 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);
     return 0;
}
```

```
N=56342;

56342 % 10=2;
                          56342 / 10 = 5634;
5634 % 10 = 4;
                          5634 / 10 = 563;
563 % 10 = 3;
                          563 / 10 = 56;
56 % 10 = 6;
                          56 / 10 = 5;
5 % 10 = 5;
                          5 / 10 = 0;
N=0;
```

## Exercise 2:

Write a C program that will read a decimal integer and will convert to equivalent to binary number.