# Clarifications : last class

- printf(“Price=Rs. %8.2f\n”,price);
- switch(ab+cd)

# Examples for float format specifier

```
float a = 1.12 ;
printf(“ a = %f ”, a );
```

# Examples for float format specifier

```
float a = 1.12 ;
printf(“ a = %f ”, a );
```

Output :- a = 1.12

# Examples for float format specifier

**a = 1;**
printf(“ a = %f ”, a );

# Examples for float format specifier

**a = 1;**

printf(“ a = %f ”, a );

**Output :-** a = 1.000000

# Examples for float format specifier

```
float a = 3.122222223;
printf("a = %.2f ", a);
```

# Examples for float format specifier

```
float a = 3.122222223;
printf("a = %.2f ", a);
```

Output :- a = 3.12

# Examples for float format specifier

```
float a = 1.289999;
printf(“ a = %6.2f ”, a);
```

# Examples for float format specifier

float a = 1.289999;
printf(“ a = %6.2f ”, a);
*Output:- _ _1.29 where _ are spaces*

**%6.2f :-** means output will be in 6 columns
Here**,** a = 1.28999;
So after applying **%6.2f** it will print two spaces before **1** and **.2f** means two digits after dot. Include dot also in count, as  it also requires space to get stored.

# Examples for float format specifier

```
float a = 1111.289999;
printf(“ a = %5.3f ”, a);
```

# Examples for float format specifier

float a = 1111.289999;
printf(“ a = %5.3f ”, a);

Output: a = 1111.290

Before dot it contains enough digits than 5 so it will store 1111 in one block and rest in another.
Space will only be added when there are not enough digits.

# Invalid switch expressions

- switch(ab+cd)
- switch(ab+cd) is invalid if ab +cd does not evaluate to either integer or character or enumeration

# CS2002D PROGRAM DESIGN
# Lecture 3

# Recall Control Structures

- **Control structures** control the flow of execution in a program or function.
- There are three kinds of execution flow:
  - **Sequence**:
    - the execution of the program is sequential. (add/sub/mul/div of two numbers)
  - **Selection**:
    - A control structure which chooses alternative to execute.
  - **Repetition**:
    - A control structure which repeats a group of statements.
- We will today focus on the **repetition** control structure.

# ITERATIVE CONTROL STRUCTURES
# REPETITION

# Objectives

- Concept of loop
- Loop invariant
- Pretest and post-test loops
- Initialization and updating
- Counter controlled loops
- Event controlled loops

# In detail

- Introduction to iterative construct
- Counter controlled loops
  - While loop
  - Do-while loop
  - For loop
- Nesting of loops
- Control of loop execution
- Infinite loops
- Event controlled loops
  - Sentinel controlled
  - Flag controlled

# Loops

- **A loop is a sequence of statements that will be executed repeatedly zero or more times.**

- **A loop can be executed a set number of times, or as long as some condition is met.**

- **Each single repetition of the loop is known as an iteration of the loop.**

✓ *Looping until one condition is met is the same as looping as long as the opposite of the condition is met.*

✓ *For instance, if you loop* ***until x equals 5****, that is the same as looping* ***as long as x does not equal 5.***

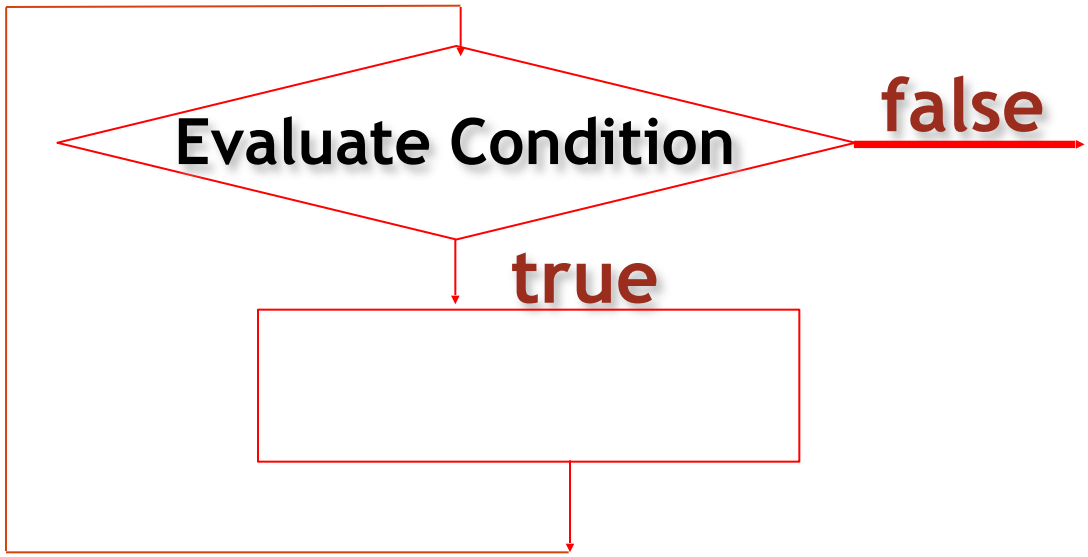
# Iterative construct: while loop

**while(expression)**
**statement s;**

✓ *The statement s will be executed as long as the expression remains true, or until a special command is encountered to end the loop.*

✓ *The statement s can be a compound statement*

# Looping in a while loop




✓ *To repeatedly execute a statement over and over while a given condition is true*

✓ *When the condition of the while loop is no longer logically true, the loop terminates and program execution resumes at the next statement following the loop*

# Example #1

```
int x = 3; while (x > 0)
{
        printf("Hello World!\n");  x = x –1;
}
```

✓ *The* ***loop condition*** *is written first,*
*followed by the* ***body of the loop***

✓ *The loop condition is evaluated first, and*
*if it is true, the loop body is executed*

✓ *After the execution of the loop body, the condition in the while is evaluated again.*

✓ *This repeats until the condition becomes false.*

```
int x =  3;
while (x > 0)
{
printf("Hello World!\n");
x=x-1;
}
```

- *This "while" loop will undergo three iterations.*
  - ✓ *During each, the phrase "Hello World!"*
  - ✓ *will be printed on a separate line.*
- *Why does it execute three times?*
  - ✓ *The variable "x" is initialized above the loop with the value of 3.*
  - ✓ *The loop will repeat as long as the expression "x > 0" is true.*
  - ✓ *At the end of each loop iteration, "x" is decreased by 1.*
  - ✓ *After three iterations, "x" will have the value of 0, and the expression will no longer be true, so the loop will end.*
  - ✓ *How many times the condition "x > 0" will be checked?*
- *Note that there is* ***no semicolon*** *after the right parenthesis ending the expression that "while" is checking.*
  - ✓ *If there were, it would mean that the program would* ***repeat the null statement*** *(statements that do nothing) until the condition were not true.*
  - ✓ *The condition starts off true, it will stay true, and will loop without stopping…*

# Example #2

```
int x, y;
   printf("Enter two numbers: ");
   scanf("%d %d", &x, &y);
   while (y != 0)
   {      printf("%d / %d = %d\n", x, y, x/y);
          printf("Enter two numbers: ");
          scanf("%d %d", &x, &y);
   }
```

✓ *This code repeatedly asks the user to enter two integers.*

✓ *As long as the second number is not zero, the program prints the result of dividing the first number by the second.*

✓ *If the second number is 0, the program ends.*

# Example #2: Try it yourself

```
int i = 0;
int loop_count = 5;

 printf("Case1:\n");
while (i<loop_count) {
        printf("%d\n",i); i++; }

printf("Case2:\n");
i=20;
while (0){
   printf("%d\n",i); i++; }
```

# Example #3 *Contd…*

```
printf("Case3:\n");
i=0;
while (i++<5) {
   printf("%d\n",i); }

printf("Case4:\n");
i=3;
while (i < 5 && i >=2) {
    printf("%d\n",i); i++; }
```

# Cases:1 and 2

- **Case1 (Normal)** : *Variable 'i' is initialized to 0 before 'while' loop; iteration is increment of counter variable 'i'; condition is execute loop till 'i' is lesser than value of 'loop_count' variable i.e. 5.*

- **Case2 (Always FALSE condition)** : *Variables 'i' is initialized before 'while' loop to '20'; iteration is increment of counter variable 'i'; condition is FALSE always as '0' is provided that causes NOT to execute loop statements and loop statement is NOT executed.*

✓ *Here, it is noted that as compared to 'do-while' loop, statements in 'while' loop are NOT even executed once which executed at least once in 'do- while' loop because 'while' loop only executes loop statements only if condition succeeds.*

# Cases:3 and 4

- **Case3 (Iteration in condition check expression)** :*Variable 'i' is initialized to 0 before 'while' loop; here note that iteration and condition is provided in same expression. Here, observe the condition is execute loop till 'i' is lesser than '5' and loop iterates 5 times.*

✓ *Unlike 'do-while' loop, here condition is checked first then 'while' loop executes statements.*

- **Case4 (Using logical AND condition)** :*Variable 'i' is initialized before 'while' loop to '3'; iteration is increment of counter variable 'i'; condition is execute loop when 'i' is lesser than '5' AND 'i' is greater or equal to '2'.*

# ./a.out

- Case1: 0 1 2 3 4
- Case2:
- Case3: 1 2 3 4 5
- Case4: 3 4 #

# *Find any differences ?*

```
int x =  3;
while (x-- > 0)
  printf("Hello World!\n");
```

```
int x = 3;
   while (--x >=  0)
      printf("Hello World!\n");
```

# *Find any differences ?*

```
int x = 3;
while (x-- > 0)
  printf("Hello World!\n");
```

- ✓ *Here, since the decrement operator is placed after the variable, the old value of the variable is used to compare against 0.*
- ✓ *The first time through, this value is 3, then 2, then 1.*
- ✓ *The fourth time, it is 0 so we exit the loop.*

```
int x = 3;
  while (--x >= 0)
    printf("Hello World!\n");
```

- ✓ *Now, the decrement operator is placed before the variable, so the value of "x" is decreased and then its value is used.*
- ✓ *The first time through, this value is 2, then 1, then 0.*
- ✓ *Then, it is –1 so we exit the loop.*

# Write a program….

- Ask the user to enter a number, and if it is positive, sum the digits.
- The user enters a number, and the value is stored in "x".
- Store sum in the variable "sum_digits", which is initialized to zero.

❖ *Hint:*

✓ *As long as "x" is greater than zero, we mod it by 10, which gets the right-most digit of the number, and we add this digit to "sum_digits".*

✓ *Then we divide "x" by 10. Remember, when we do integer division, the fractional part is cut off, so in effect, we are removing the right-most digit of "x" (which we have already added to the sum).*

# Solution

```
int x, digit, sum_digits;
printf("Enter a positive integer: ");
scanf("%d", &x);
sum_digits = 0;
while (x > 0)
{
        digit =  x %10;
        sum_digits =  sum_digits + digit;
        x =  x / 10;
}
printf("The sum of the digits is %d!\n",
sum_digits);
```

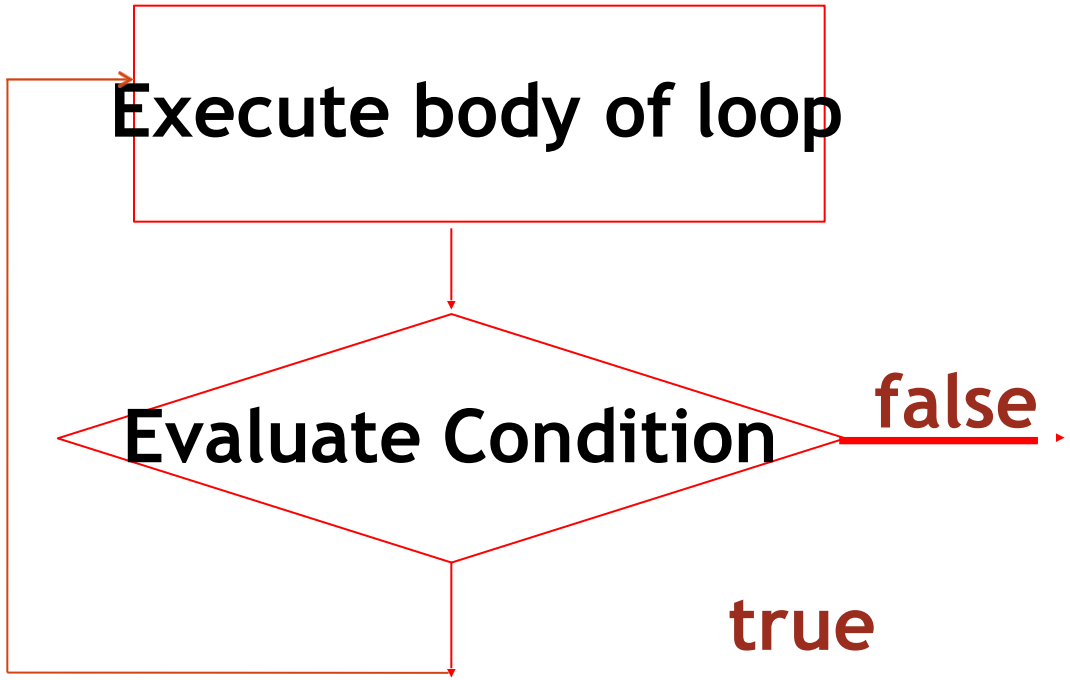
# Iterative construct: do - while loop

```
do
    statement

while(condition);
```

✓ *It is similar to the "while" statement, but the condition is checked at the end.*

# Looping in a do-while loop




✓ *Statements inside the statement block are executed once, and then expression is evaluated , in order to determine whether the looping is to continue*

# Example #1

```
int  i = 0;
do{
    printf("The value of i  %d \n",i);
        i = i + 1;
} while (i < 5);
```

✓ *The body of the loop comes first,*
*followed by the loop condition at the end*

✓ *The loop is entered into straightaway,*
*and after the first execution of the loop body,*
*the loop condition is evaluated*

✓ ***The body of the loop is guaranteed to execute at least once***

✓ *Further executions of loop body would be subject to loop condition evaluating to true*

# Example #2

```
int x = 3;
do
{
      printf("Hello World!\n");
      x =  x–1;
 } while (x > 0);
```

✓ *Notice that there is a semicolon after the right parenthesis ending the expression while is checking.*

✓ *If you forget it, you will get a compiler error when you try to compile the program.*

# Example #3

```
do
{
   printf("Enter a number from 1 to 100: ");
   scanf("%d", &x);
} while ((x < 1) || (x > 100));
```

✓ *If the user does not enter a number in the correct range, the "while" condition will be met, and the loop will undergo another iteration, prompting the user again.*

✓ *Only after the user enters a number from 1 to 100 will the expression be false and the loop end.*

# Example #4: Try it yourself

```
int i = 0;
int loop_count = 5;
printf("Case1:\n");
do {
        printf("%d\n",i); i++;
    } while (i<loop_count);

printf("Case2:\n");
i=20;
do {
        printf("%d\n",i); i++;
    } while (0);
```

# Example #4 *Contd…*

```
printf("Case3:\n");
i=0;
do {
        printf("%d\n",i);
    } while (i++<5);

printf("Case4:\n");
i=3;
do {
        printf("%d\n",i); i++;
    } while (i < 5 && i >=2);
```

# Cases: 1 and 2

- **Case1 (Normal)** :*Variable 'i' is initialized to 0 before 'do-while' loop; iteration is increment of counter variable 'i'; condition is to execute loop till 'i' is lesser than value of 'loop_count' variable i.e. 5.*

- **Case2 (Always FALSE condition)** : *Variables 'i' is initialized before 'do-while' loop to '20'; iteration is increment of counter variable 'i'; condition is FALSE always as '0' is provided that causes NOT to execute loop statements.*

✓ *But, it is noted here in output that loop statement is executed once because do-while loop always executes its loop statements at least once even if condition fails at first iteration.*

# Cases: 3 and 4

- **Case3 (Iteration in condition check expression)** : *Variable 'i' is initialized to 0 before 'do-while' loop; here note that iteration and condition is provided in same expression.*

✓ *Here, observe the condition is to execute loop till 'i' is lesser than '5', but in output 5 is also printed that is because, here iteration is being done at condition check expression, hence on each iteration 'do-while' loop executes statements ahead of condition check.*

- **Case4 (Using logical AND condition)** : *Variable 'i' is initialized before 'do-while' loop to '3'; iteration is increment of counter variable 'i'; condition is execute loop when 'i' is lesser than '5' AND 'i' is greater or equal to '2'.*

# # ./a.out

- Case1: 0 1 2 3 4
- Case2: 20
- Case3: 0 1 2 3 4 5
- Case4: 3 4 #

# Example #5: Programs with Menus

A)dd part to catalog
R)emove part from catalog
F)ind part in catalog
Q)uit

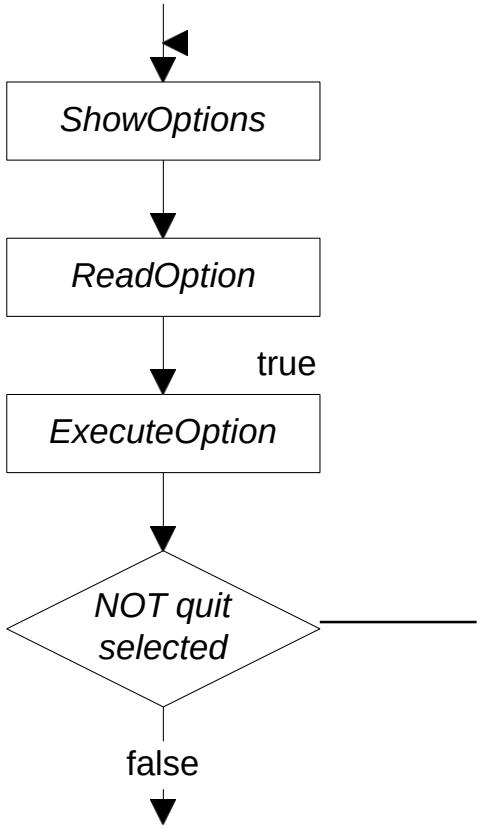
Select option: A
*<interaction to add a part>*

A)dd part to catalog
R)emove part from catalog
F)ind part in catalog
Q)uit

Select option: *<next option>*

# Menu Loop

```
do {
  showOptions();
  printf("Select option:");
  scanf(" %c",&optn);
  execOption(optn);
  }while (!((optn ==  'Q') || (optn ==  'q')));
```



# Menu Options

```
void showOptions() {
  printf("A)dd part to catalog\n");
  printf("R)emove part from catalog\n");
  printf("F)ind part in catalog\n");
  printf("Q)uit\n");
}
```

# Executing Options

```
void execOption( char option ){
  switch (option) {
    case ‘A’: case ‘a’: addPart(); break;
    case ‘R’: case ‘r’: delPart(); break;
    case ‘F’: case ‘f’: fndPart(); break;
    case ‘Q’: case ‘q’: break;
    default: printf(“Unknown option
      %c\n”,option); break;
  }
}
```

# Iterative construct: for loop

```
for (expr1; expr2;expr3)
{
        statement1;
        statement2; . . .
}
```

✓ *The for loop construct is by far the* ***most powerful and compact*** *of all the loop constructs provided by* ***C.***

✓ *This loop keeps* ***all loop control statements on top of the loop****, thus making it visible to the programmer.*

✓ *This loop works well where* ***the number of iterations of the loop is known before*** *the loop is entered into.*

# for (initialization; condition; update)

✓ *The* ***initialization(expression1)*** *is usually an assignment of a variable to some starting value, and the* ***update (expression3)*** *is often an assignment which changes this variable.*

✓ *The statement will be executed as long as the* ***condition(expression2)*****,** *which is an expression, is true.*

✓ *All three fields are optional.*

✓ ***If the initialization or update are left out****, they are considered null statements (statements that do nothing).*

✓ ***If the condition is left out****, it is considered to be always true, and the loop will continue until a statement is reached to break out of the loop.*

**The first part :**

- Expression 1: is executed before the loop is entered

✓ This is usually the initialization of the loop variable

**The second part :**

- Expression 2: is a test, is evaluated immediately after expression1, and then later is evaluated again after each successful looping

✓ The loop is terminated when this test returns a false

**The third part :**

- Expression 3: is a statement to be run every time the loop body is completed

✓ It is not evaluated when the for statement is first encountered. However, expression3 is evaluated after each looping and before the statement goes back to test expression2 again.

✓ This is usually an increment of the loop counter

# for loop….

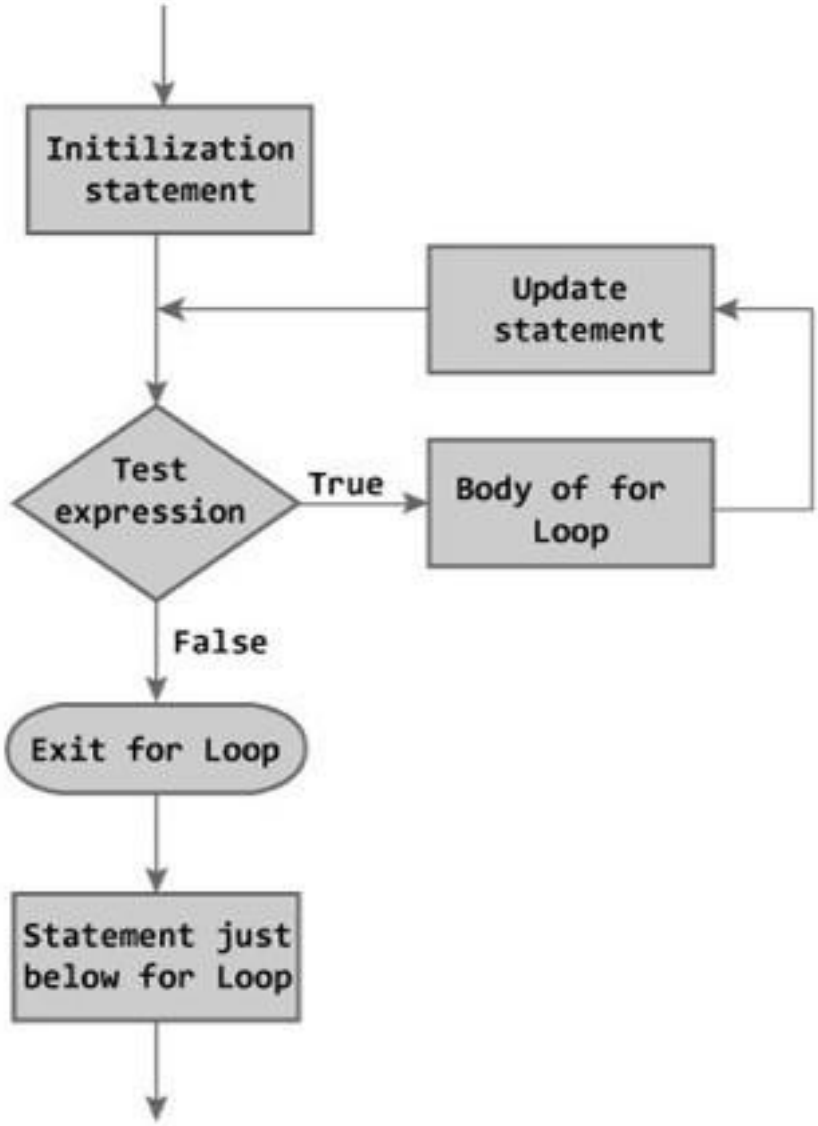



Figure: Flowchart of for Loop

# Example #1

```
int n, count, sum=0;
printf("Enter the value of  n.\n");
 scanf("%d",&n);
for(count=1;count<=n;++count) //for loop terminates if count>n
{
    sum+=count; // this statement  is  equivalent to sum=sum+count
 }
 printf("Sum=%d",sum);
 return 0;
```

- ✓ *In this program, the user is asked to enter the value of n.*
- ✓ *Suppose you entered 19 then, count is initialized to 1 at first.*
- ✓ *Then, the test expression in the for loop, i.e., (count<= n) becomes true.*
- ✓ *So, the code in the body of for loop is executed which makes sum to 1.*

- ✓ *Then, the expression ++count is executed and again the test expression is checked, which becomes true.*
- ✓ *Again, the body of for loop is executed which makes sum to 3 and this process continues.*
- ✓ *When count is 20, the test condition becomes false and the for loop is terminated.*
- **Note:** *Initial, test and update expressions are separated by semicolon(;).*

```
scanf("%d",&n);
for(count=1;count<=n;++count) //for loop terminates if count>n
{sum+=count;}
 printf("Sum=%d",sum);
```

# Example #2 Try it yourself

```
int i =  0, k =  0; float j = 0;
int loop_count =  5;
printf("Case1:\n");
for (i=0; i < loop_count; i++) {
   printf("%d\n",i); }

printf("Case2:\n");
for (j=5.5; j > 0; j--) {
   printf("%f\n",j); }

printf("Case3:\n");
for (i=2; (i < 5 && i >=2); i++) {
    printf("%d\n",i); }
```

# Example #2 *Contd…*

```
printf("Case4:\n");
for (i=0; (i != 5); i++) {
    printf("%d\n",i); }

printf("Case5:\n");
/* Blank loop */ for (i=0; i < loop_count; i++) ;

printf("Case6:\n");
for (i=0, k=0; (i < 5 && k < 3); i++, k++) {
printf("%d\n",i); }

printf("Case7:\n");
i=5;
for (; 0; i++) { printf("%d\n",i); }
```

# Cases: 1,2 and 3

- **Case1 (Normal)** : *Variable 'i' is initialized to 0; condition is to execute loop till 'i' is lesser than value of 'loop_count' variable; iteration is increment of counter variable 'i'*

- **Case2 (Using float variable)** : *Variable 'j' is float and initialized to 5.5; condition is to execute loop till 'j' is greater than '0'; iteration is decrement of counter variable 'j'.*

- **Case3 (Taking logical AND condition)** : *Variable 'i' is initialized to 2; condition is to execute loop when 'i' is greater or equal to '2' and lesser than '5'; iteration is increment of counter variable 'i'.*

# Case : 4,5,6 and 7

- **Case4 (Using logical NOT EQUAL condition)** : *Variable 'i' is initialized to 0; condition is to execute loop till 'i' is NOT equal to '5'; iteration is increment of counter variable 'i'.*

- **Case5 (Blank Loop)** : *This example shows that loop can execute even if there is no statement in the block for execution on each iteration.*

- **Case6 (Multiple variables and conditions)** : *Variables 'i' and 'k' are initialized to 0; condition is to execute loop when 'i' is lesser than '5' and 'k' is lesser than '3'; iteration is increment of counter variables 'i' and 'k'.*

- **Case7 (No initialization in for loop and Always FALSE condition)** : *Variables 'i' is initialized before for loop to '5'; condition is FALSE always as '0' is provided that causes NOT to execute loop statement; iteration is increment of counter variable 'i'.*

# ./a.out

- Case1: 0 1 2 3 4
- Case2: 5.500000 4.500000 3.500000 2.500000 1.500000 0.500000
- Case3: 2 3 4
- Case4: 0 1 2 3 4
- Case5:
- Case6: 0 1 2
- Case7:

# *Predict the output*


1.

```
int i;
for (i=0; i<16; i++)
        printf("%X %x %d\n", i,  i,  i);
```

2.

```
for (i=0; i<8; i++)
sum += i;
```

3.

```
for (i=0; i<8; i++);
sum += i;
```

THANK YOU