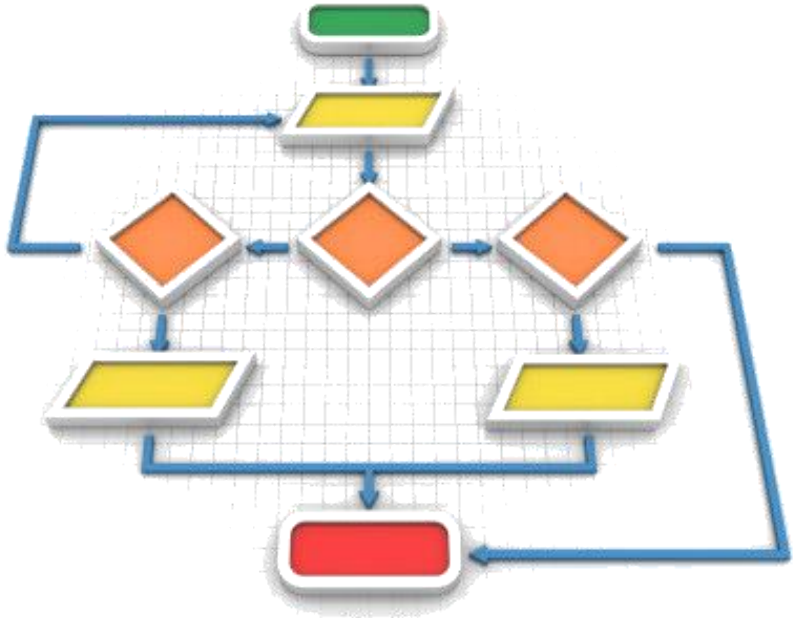
Previous class Review

- We have learnt about
  - Constants
  - Variable Names
  - Declarations
  - Data Types and Sizes

Decision Making, Branching & Switch



Objectives

- To learn and appreciate the following concepts

  - The if Statement
  - The if-else Statement
  - Nested if Statements
  - The else if Ladder
  - The switch Statement

Outcome

- At the end of session student will be able to learn and understand
  - The if Statement
  - The if-else Statement
  - Nested if Statements
  - The else if Ladder
  - The switch Statement

Control Structures

- **A control structure refers to the order of executing the program statements.**
- The following three approaches can be chosen depending on the problem statement:

✓ **Sequential (Serial)**

- In a **Sequential approach**, all the statements are executed in the same order as it is written.

✓ **Selectional (Decision Making and Branching)**

- In a **Selectional approach**, based on some conditions, different set of statements are executed.

✓ **Iterational (Repetition)**

- In an **Iterational approach** certain statements are executed repeatedly.

DECISION MAKING AND BRANCHING

C decision making and branching statements are:

1. if statement
2. switch statement

Different forms of if statement

1. Simple **if** statement.

2. **if…else** statement.

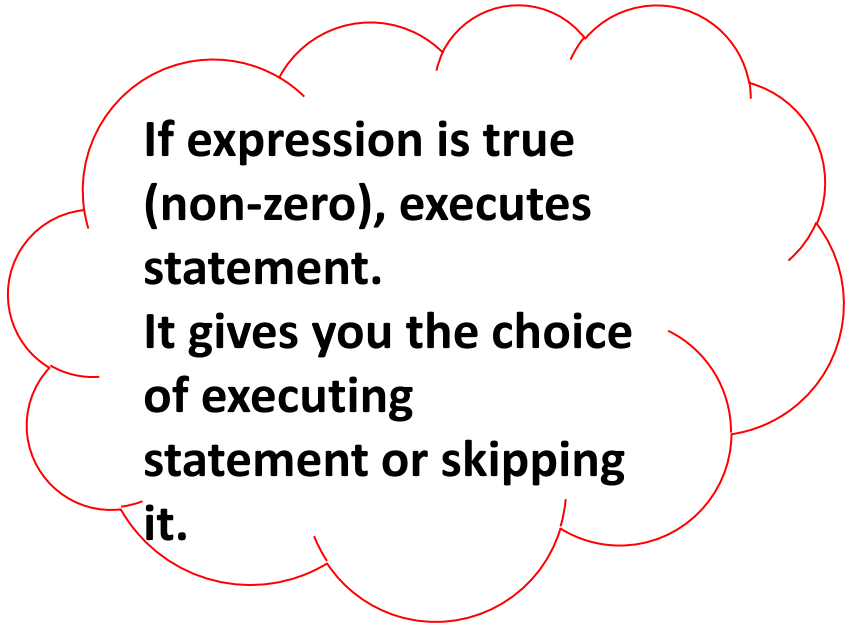
3. Nested **if…else** statement.

4. **else if** ladder.

Simple `if` Statement

General form of the simplest if statement:

```
if (test Expression)
    {
     statement-block;
     }
next_statement;
```



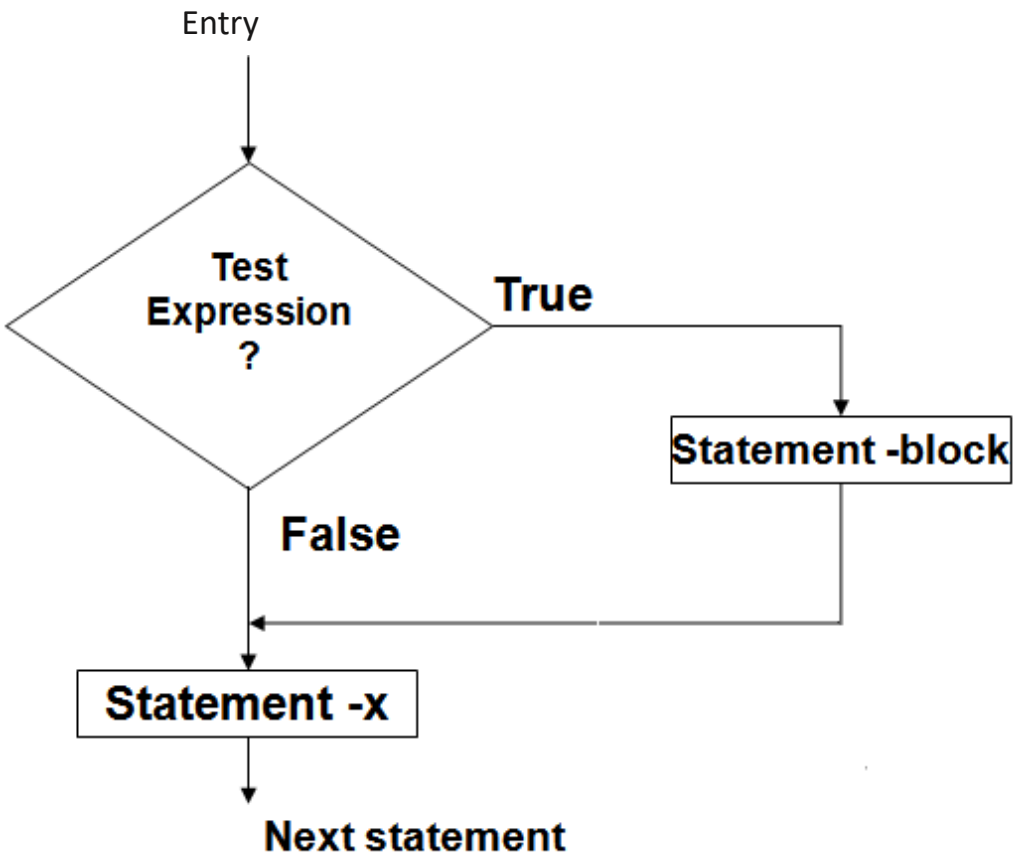
`if` Statement- explanation

- (*test Expression*) is first evaluated.
- If TRUE (non-zero), the 'if' statement block is executed.
- If FALSE (zero) the next statement following the if statement block is executed.
- So, during the execution, based on some condition, some code will not be executed (skipped).

For example:

```
bonus = 0;
if (hours > 70)
bonus = 10000;
salary= salary + bonus;
```

Flow chart of simple `if`

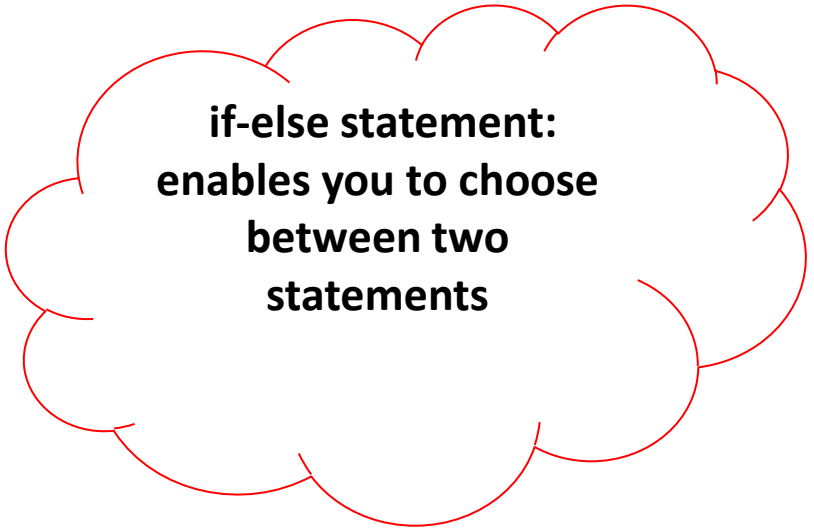


Find out whether a number is even or odd.

```
#include <stdio.h>
int main()
{
  int  x;
  printf("input an integer\n");
  scanf("%d",&x);
  if ((x % 2) == 0)
  {
        printf("It is an even number\n");
  }
  if ((x%2) == 1)
  {
        printf("It is an odd number\n");
  }
   return 0;
}
```

The `if-else` statement

```
if (test expression )
{
    statement _block1
}
else
{
    statement _block2
}
Next_statement
```



if-else statement

Explanation:

1.First ,the (test expression) is evaluated.

2.If it evaluates to non-zero (TRUE), statement_1 is executed, otherwise, if it evaluates to zero (FALSE), statement_2 is executed.

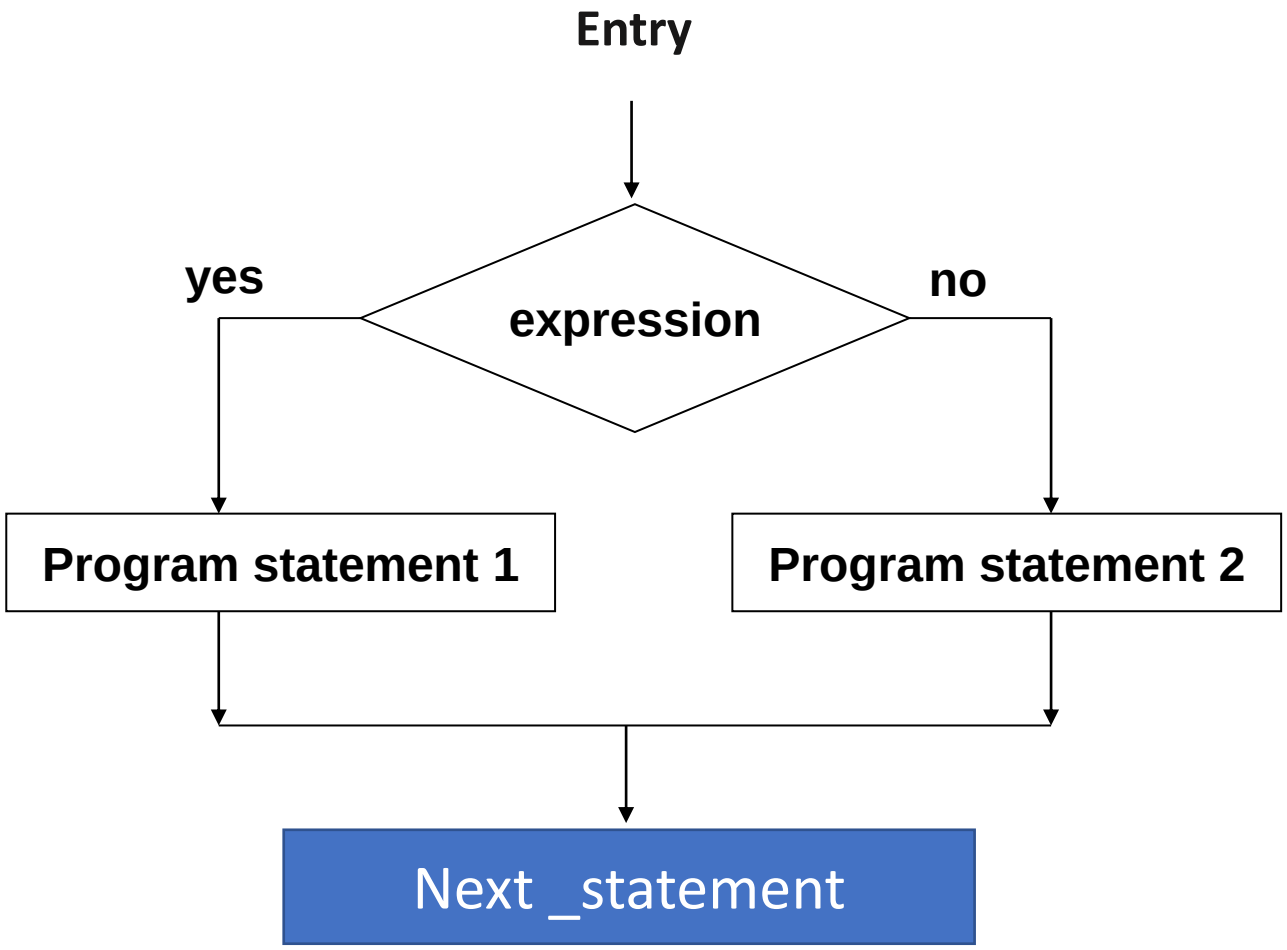
3.They are mutually exclusive, meaning, either statement_1 is executed or statement_2, but not both.

4.If the statements_ 1 and statements_ 2 take the form of block , they must be put in curly braces.

Example:

```
if(job_code ==  1)
        rate = 7.00;
else
        rate = 10.00;
prinf("%d",rate);
```

The `if-else` statement



Find out whether a number is even or odd.

```
#include <stdio.h>
int main() {
   int  x;
   printf("Input an integer\n");
   scanf("%d",&x);
   if ((x % 2) == 0)
   {
         printf("It is an even number\n");
   }
   else
   {
         printf("It is an odd number\n");
}
    return 0;
}
```

WAP to find largest of 2 numbers

```
#include<stdio.h>
int main()
{
    int a, b;
    printf("Enter 2 numbers\n");
    scanf("%d %d",&a,&b);
     if(a > b)
            printf("Large is %d\t", a);
     else
            printf("Large is %d\t", b);

    return 0;
 }
```

Attention on `if-else` syntax !

```
if ( expression )
        program
statement 1
else
        program
statement 2
```

```
if ( remainder == 0 )
        prinf("The number is even.\n“);
else
        printf(“The number is odd.\n“);
```

In C, the ; is part (end) of a statement !

```
if ( x == 0 );
        printf(“The number is zero.\n“);
```

Syntactically OK but a semantic error !

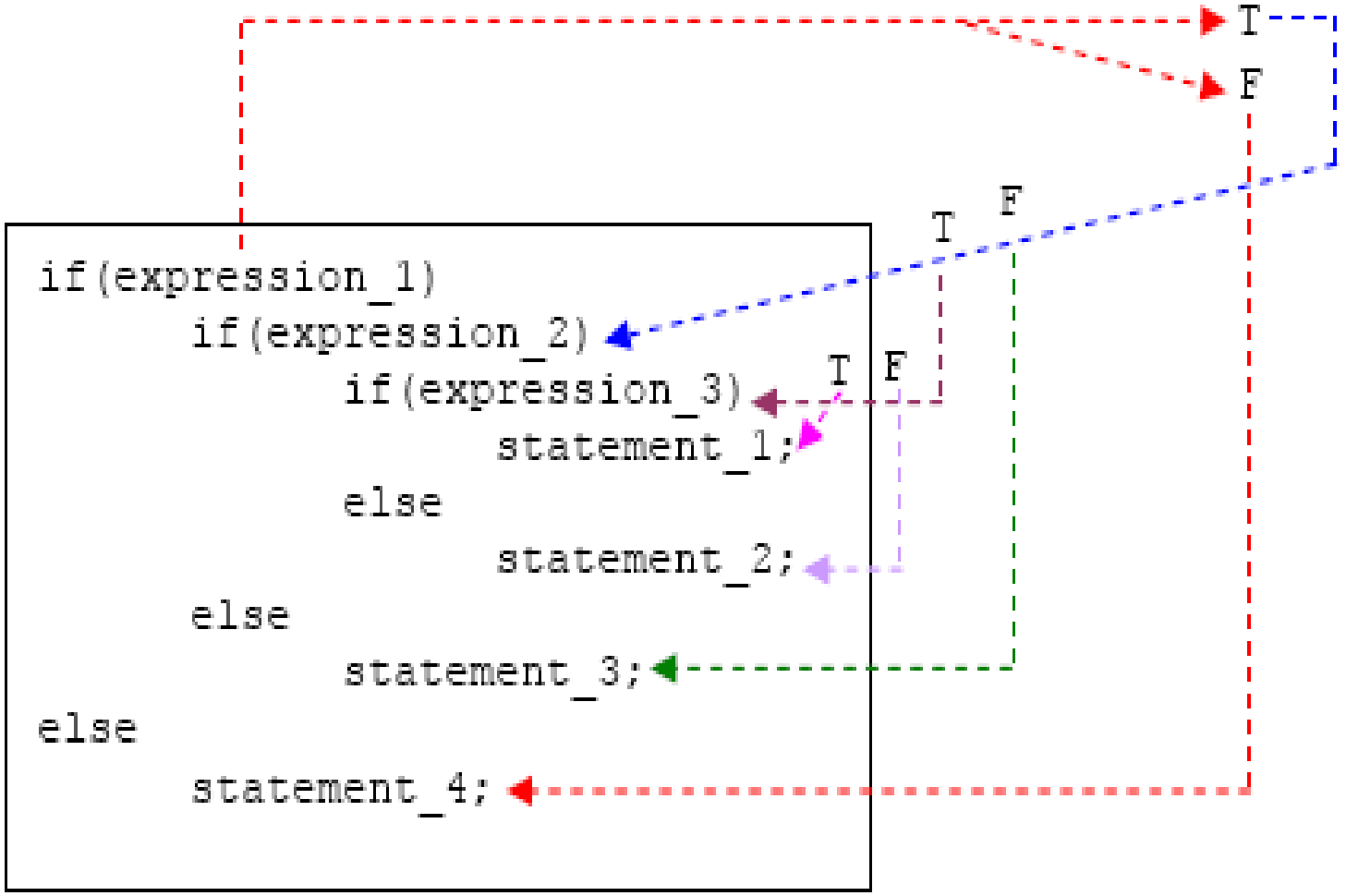
Testing for character ranges

```
#include<stdio.h>
int main()
{
   char ch;
   printf(“enter a character\n”);
    scanf(“%c”,&ch);
   if (ch >= 'a' && ch <= 'z')
                printf(“lowercase char\n“);
   if (ch >= ‘A' && ch <= ‘Z')
                printf(“uppercase char\n“);
   if (ch >= ‘0' && ch <= ‘9')
                printf(“digit char\n“);
   else
                printf(“ special char\n”);
return 0;
}
```

Output:

```
enter a
character:
 C
uppercase char
special char

enter a
character:
 j
lowercase char
special char

enter a
character:
 5
digit char
```

Nested `if-else` Statement



`if-else` nesting -Explanation

1. The if-else constructs can be nested (placed one within another) to any depth.
2. In this nested form, expression_1 is evaluated.
   - If it is zero (FALSE-F), statement_4 is executed and the entire nested if statement is terminated;
   - If not (TRUE-T), control goes to the second if (within the first if) and expression_2 is evaluated. If it is zero, statement_3 is executed;
   - If not, control goes to the third if (within the second if) and expression_3 is evaluated.
   - If it is zero, statement_2 is executed;
   - If not, statement_1 is executed. The statement_1 (inner most) will only be executed if all the if statement is true.

Smallest among three numbers

```
#include <stdio.h>
int main()
{
int  a, b, c, smallest;

printf("Enter a, b & c\n");
scanf("%d %d %d, &a, &b, &c);
```

```
if (a < b)
 {       if (a < c)
          { smallest = a;   }
          else
           {       smallest = c;  }
 }
else
 {       if (b < c)
          {       smallest = b;  }
else
          {         smallest = c;   }
  }
printf("Smallest is %d", smallest);
return 0;
}
```

Nested `if` statements

```
if (number > 5)
    if (number < 10)
        printf("1111\n");
    else printf("2222\n");
```

```
if (number > 5) {
    if (number < 10)
        printf("1111\n");
}
else printf("2222\n");
```

Rule: an else goes with the most recent if, unless braces indicate otherwise

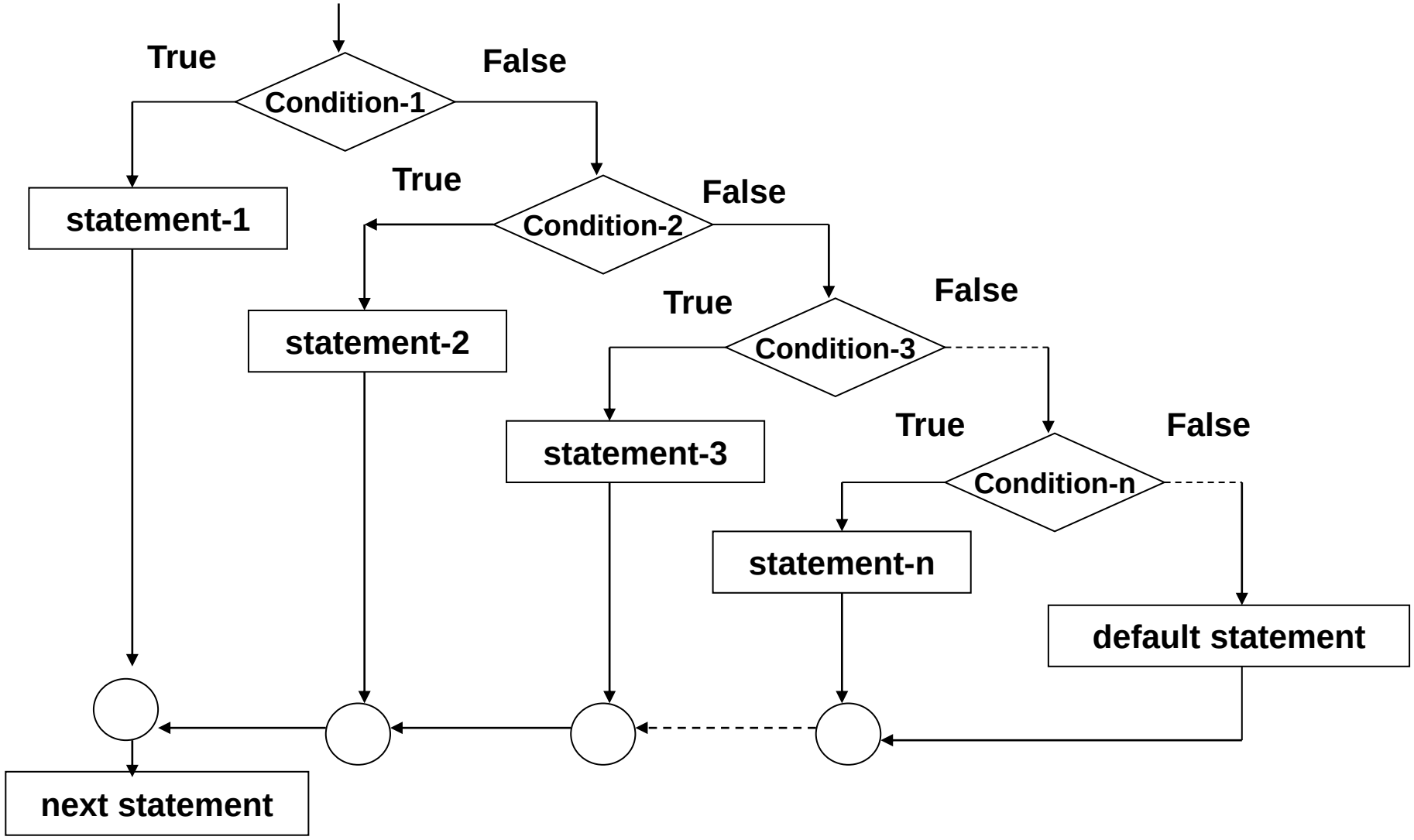
The `else-if` ladder

```
if (Expression_1 )
{
    statement _block_1
}
else if (Expression_2)
{
    statement _block_2
}
…….
else if (Expression_n)
{
    statement _block_n
}
else
{
    last_statement
}

Next_statement
```

`else-if` ladder - Explanation

- expression_1 is first evaluated. If it is TRUE, statement_1 is executed and the whole statement terminated and the next_statement is executed.
- On the other hand, if expression_1 is FALSE, control passes to the else if part and expression_2 is evaluated.
- If it is TRUE, statement_2 is executed and the whole system is terminated.
- If it is False, other else if parts (if any) are tested in a similar way.
- Finally, if expression_n is True, statement_n is executed; if not, last_statement is executed.
- Note that only one of the statements will be executed others will be skipped.
- The statement_n's could also be a block of statement and must be put in curly braces.

`else-if` ladder Flow of control



Testing for character ranges

```
#include<stdio.h>
int main()
{
   char ch;
   printf("enter a character\n");
    scanf("%c",&ch);
   if (ch >= 'a' && ch <= 'z')
            printf("lowercase char\n");
   else if (ch >= 'A' && ch <= 'Z')
            printf("uppercase char\n");
   else if (ch >= '0' && ch <= '9')
            printf("digit char\n");
   else
            printf(" special char\n");
return 0;
}
```

WAP using else-if ladder to calculate grade for the marks entered

```
int main() {
        char cgrade;
        int imarks;
        printf("enter marks“);
        scanf(“%d”,&imarks);
        if(imarks>79)
                cgrade = 'A';
        else if (imarks>59)
                cgrade = 'B';
        else if (imarks>49)
                cgrade = 'C';
        else if (imarks>39)
                cgrade = 'D';
        else
                cgrade = 'F';
        printf(“Grade :%c\n”,cgrade);
    return 0;
}
```

For inputs

imarks= 46

grade = D

imarks= 64

grade = B

Example: else-if

```
// Program to implement the sign function

#include <stdio.h>
int main ( )
{
        int number, sign;
        printf("Please type in a number: “);
        scanf(“%d”,&number);
        if ( number < 0 )
                sign = -1;
        else if ( number == 0 )
                sign = 0;
        else // Must be positive
                sign = 1;
        printf(“Sign = %d“,sign);
        return 0;
}
```

Example – multiple choices

```
/* Program to evaluate simple expressions of the form number operator number */
int main ( ) {
    float value1, value2,result;
    char operator;
    printf("Type in your expression.\n“);
    scanf(“%f %c %f”, &value1,&operator,&value2);
    if ( operator == '+' ){
        result=value1+value2;
        printf(“%f”,result);
    }
    else if ( operator == '-' ) {
        result=value1-value2;
        printf(“%f”,result);
    }
    else if ( operator == '*' ) {
        result=value1*value2;
        printf(“%f”,result);
    }
    else if ( operator == '/' ) {
        result=value1/value2;
        printf(“%f”,result);
    }
    else
        printf("Unknown operator.\n“);
    return 0;
}
```

Problem…

- Find the roots of a quadratic equation ax^2+bx+c using if elsecontrol statements.
- Roots of a quadratic equation

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- 3 cases
  - Discriminant< 0 ; roots are imaginary → $1 + i\ 2.45$
  - Discriminant = 0 ; roots are real and equal → $-\ b/2a$
  - Discriminant> 0 ; roots are real and unequal → $r1 = (-b + \sqrt{}disc)/(2a)$

    $r2 = (-b - \sqrt{}disc)/(2a)$

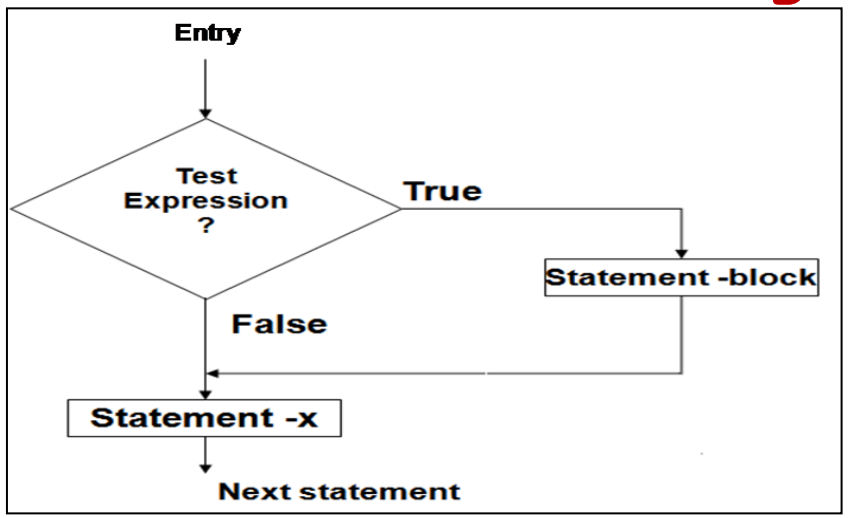
Find the roots of Quadratic equation using `if-else` statement

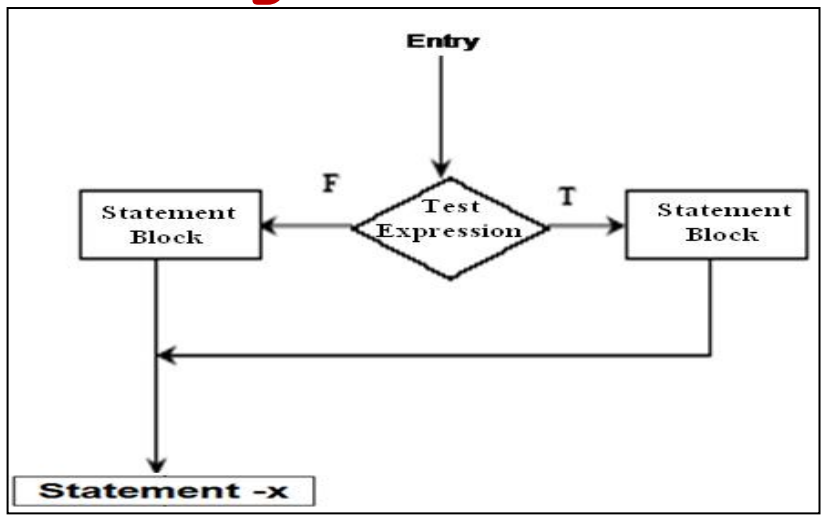
```
#include<stdio.h>
int main()
{
float a,b,c,root1,root2,re,im, disc;
printf(" Enter the coefficients:");
scanf("%f %f %f",&a,&b,&c);

disc=b*b-4*a*c; // discriminant computation

   if  (disc<0) {
      printf("imaginary roots\n");
      re= - b / (2*a);
      im = pow(fabs(disc),0.5)/(2*a);
      printf("root1=%.2f+%.2fi and
       root2 =%.2f-%.2fi", re,im,re,im);
}
```

```
Enter the coefficients:1 2  4
imaginary roots
root1=-1.00+1.73i and root2 =-1.00-1.73i
```

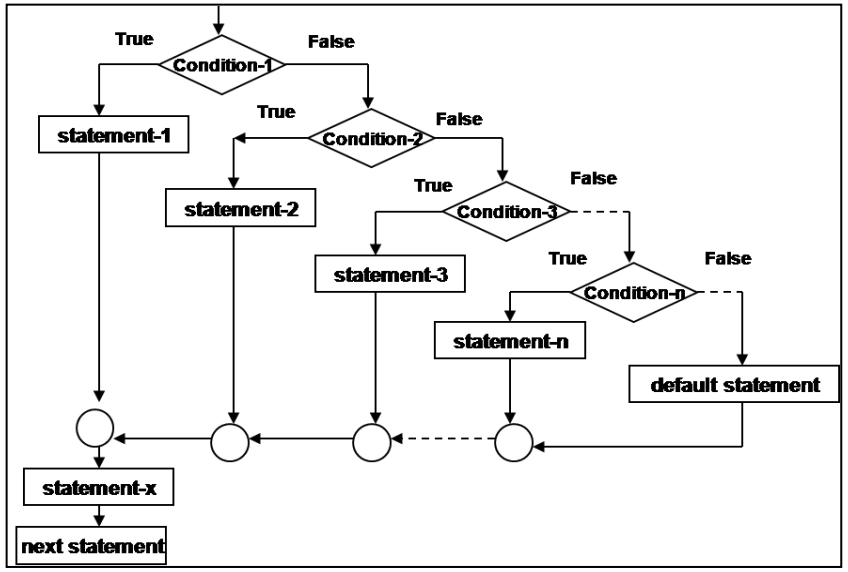
```
else if (disc==0)
    {
     printf("Real & equal roots");
     re=-b / (2*a);
     printf("Root1 and root2 are %.2f", re);
     }

   else   /*disc > 0 */
     {
      printf("Real & distinct roots");
      printf("Roots are");
      root1=(-b + sqrt(disc))/(2*a);
      root2=(-b - sqrt(disc))/(2*a);
      printf("Root1 = %.2f and root2
                    =%.2f",root1,root2);
     }
return 0;
}
```

Review on decision making & branching





- if
- if-else
- Nested if
- else if Ladder



The `switch` Statement

- Switch is `multiple-branching` statement where based on a condition, the control is transferred to one of the many possible points.

- Enables the program to execute different statements based on an expression that can have more than two values. Also called multiple choice statements.

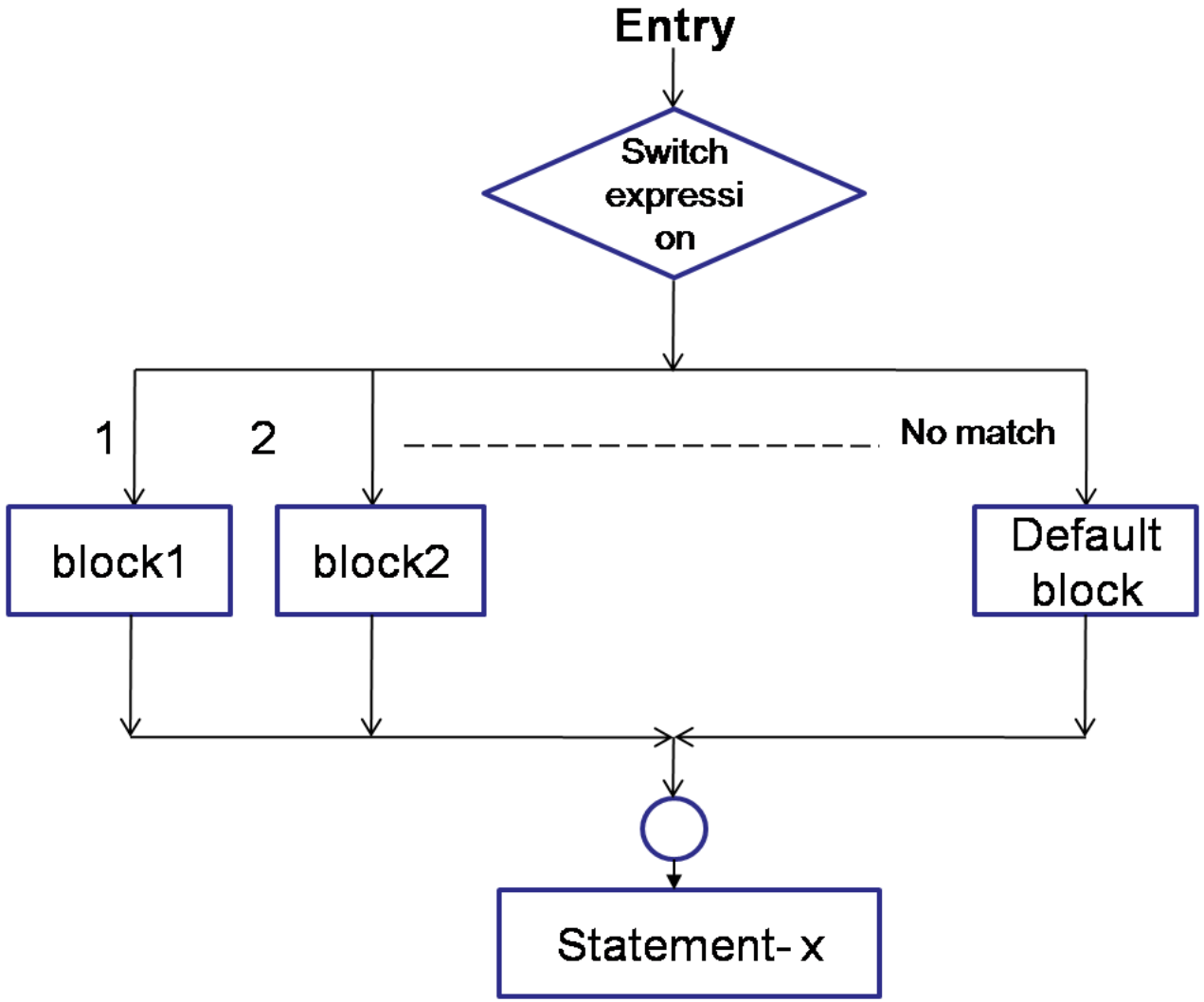
The `switch` statement

```
switch ( expression )
{
    case value1:
        program statement(s)
        program statement(s)
        break;
    case value2:
        program statement(s)
        ...
        break;
    case value n:
        program statement(s)
        program statement(s)
        break;
    default:
        program statement(s)
        program statement(s)
}
```

The *expression* is successively compared against the values *value1*, *value2*, ..., *valuen*. If a case is found whose value is equal to the value of *expression*, the program statements that follow the case are executed.

- ✓ The `switch` test expression must be one with an integer value (including type char) (No float !).
- ✓ The case values must be integer-type constants or integer constant expressions (You can't use a variable for a case label !)

switch- control flow



switch- example 1

```
#include<stdio.h>
int main()
{
   int choice;
   printf("Enter your choice: 1-yes, 2-no\n");
   scanf("%d",&choice);
   switch(choice)
   {
          case 1:   printf("YESSSSSSS……");
                    break;
          case 2:   printf("NOOOOOO……");
                    break;
          default:  printf("DEFAULT CASE…….");
   }      printf("The choice is %d",choice);
return 0;
}
```

switch- example

```
scanf("%d",&mark);

switch (mark)
{
case 100:
case  90:
case  80:   grade='A';
            break;

case  70:
case  60:
          grade='B';
          break;

case  50:
          grade='C';
          break;
case 40:
          grade='D';
          break;

default:   grade='F';
          break;
   }

printf("%c", grade);
```

switch- example

```
char ch;
scanf("%c",&ch);

switch(ch)
{
  case 'a' : printf("Vowel");          break;
  case 'e' : printf("Vowel");          break;
  case 'i' : printf("Vowel");          break;
  case 'o' : printf("Vowel");          break;
  case 'u' : printf("Vowel");          break;
  default: printf("Not a Vowel");
}
```

switch- example

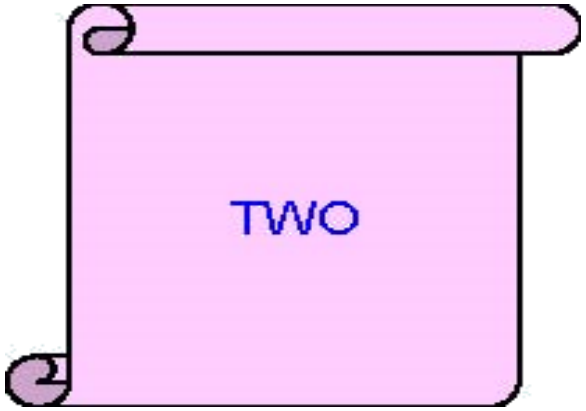
```
char ch;
scanf("%c",&ch);

switch(ch) {
  case 'a' :
  case 'e' :
  case 'i' :
  case 'o' :
  case 'u' : printf("Vowel"); break;

  default: printf("Not a Vowel"); }
```

Example - switch

```
/* Program to evaluate simple expressions of the form
value operator value */
#include <stdio.h>
int main (void){
        float value1, value2;
        char operator;
        int result;
        printf("Type in your expression.\n“);
        scanf(“%f %c %f”, &value1,&operator,&value2);
        switch (operator) {
        case '+':
                result=value1+value2;
                printf(“%f”,result);
                break;
         case '-':
                result=value1-value2;
                printf(“%f”,result);
                break;
        case ‘*':
                result=value1*value2;
                printf(“%f”,result);
                break;
        case ‘/':
                if ( value2 == 0 )
                        printf(“Division by zero.\n“);
                else result=value1 / value2;
                        printf(“%f”,result);
                break;
        default:
        printf(“Unknown Operator”);
        }
        return 0;
}
```

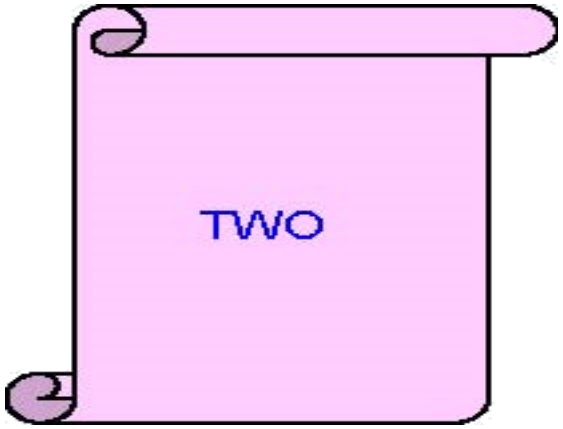
What is the output of the following code snippet?

```
int iNum = 2;
switch(iNum) {
  case 1:
          printf("ONE");
          break;
  case 2:
          printf("TWO");
           break;
  case 3:
          printf("THREE");
            break;
  default:
          printf("INVALID");
              break;
}
```



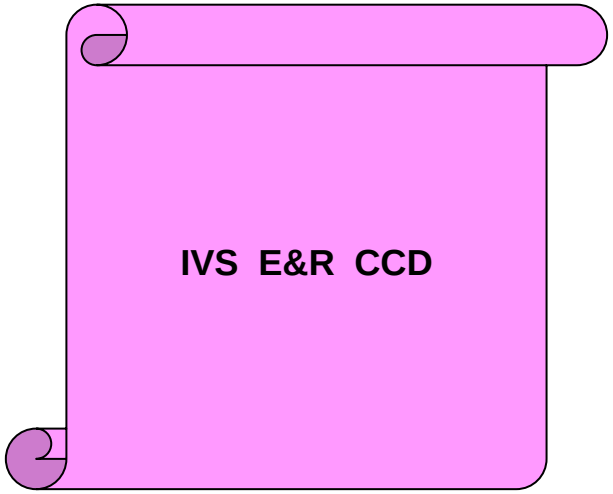
What is the output of the following code snippet?

```
iNum = 2;
switch(iNum)  {
   default:
        printf(“INVALID”);
    case 1:
        printf(“ONE”);
    case 2:
        printf(“TWO”);
        break;
     case 3:
        printf(“THREE”;)
}
```



What is the output of the following code snippet?

```
switch (iDepartmentCode)
{
  case 110 : printf("HRD ");
  case 115 : printf("IVS ");
  case 125 : printf("E&R ");
  case 135 : printf("CCD ");
}
```

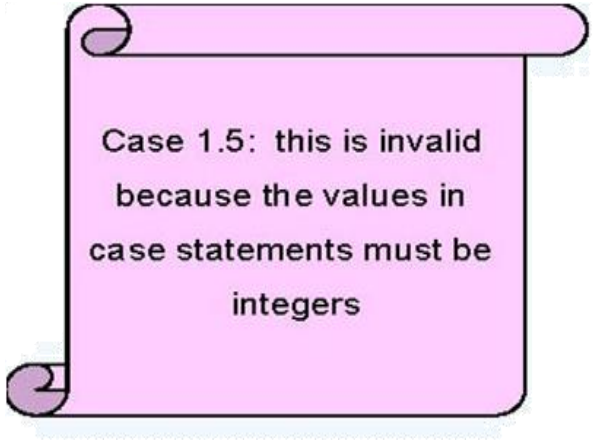


Assume iDepartmentCode is 115

find the output ?

What is the output of the following code snippet?

```
int iNum = 2;
switch(iNum)
{
   case 1.5:
      printf("ONE AND HALF");
      break;
case 2:
      printf("TWO");
case 'A' :
       printf("A character");
}
```



Problem: Find the roots of Quadratic equation using `switch` statement

```
#include<stdio.h>
int main()
{
Int d;
float a,b,c,root1,root2,re,im, disc;
printf("Enter the values of a, b & c:");
scanf("%f %f %f",&a,&b,&c);
disc=b*b-4*a*c;
printf("\nDiscriminant= %f",disc);
        if(disc<0) d=1;
        if(disc==0) d=2;
        if(disc>0)  d=3;
switch(d) {
   case 1:
        printf("imaginary roots\n");
        re= - b / (2*a);
        im = pow(fabs(disc),0.5)/(2*a);
        printf("root1=%.2f+%.2fi and root2 =%.2f-%.2fi", re,im,re,im);
        break;
```

Problem: Find the roots of Quadratic equation using switch statement

```
case 2:
        printf("Real & equal roots");
        re=-b / (2*a);
        printf("Root1 and root2 are %.2f",re);
        break;
case 3:
        printf("Real & distinct roots");
        printf("Roots are");
        root1=(-b + sqrt(disc))/(2*a);
        root2=(-b - sqrt(disc))/(2*a);
        printf("Root1 = %.2f and root2 =%.2f",root1,root2);
        break;
   } // end of switch
return 0;
} //End of Program
```

Some guidelines for writing switch case statements

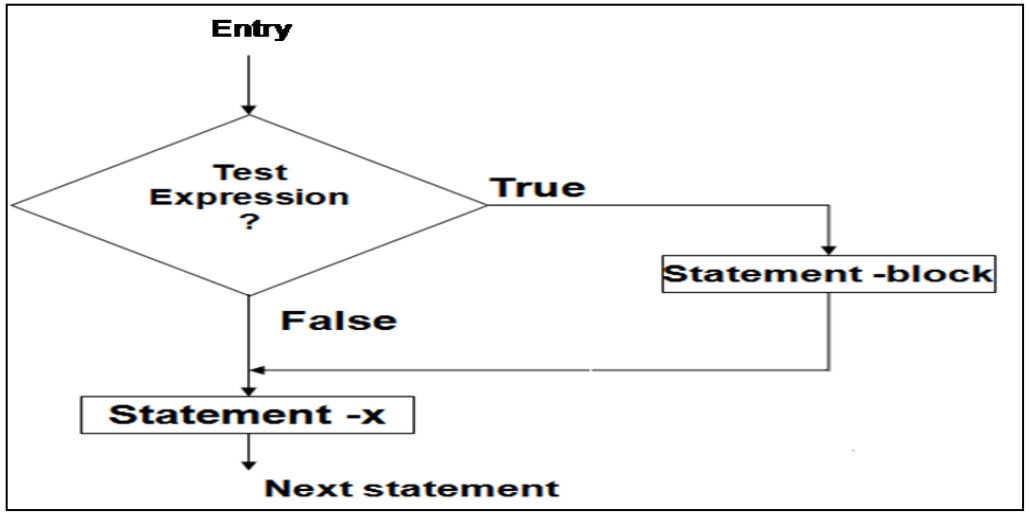
(1) Order the cases alphabetically or numerically – improves readability.

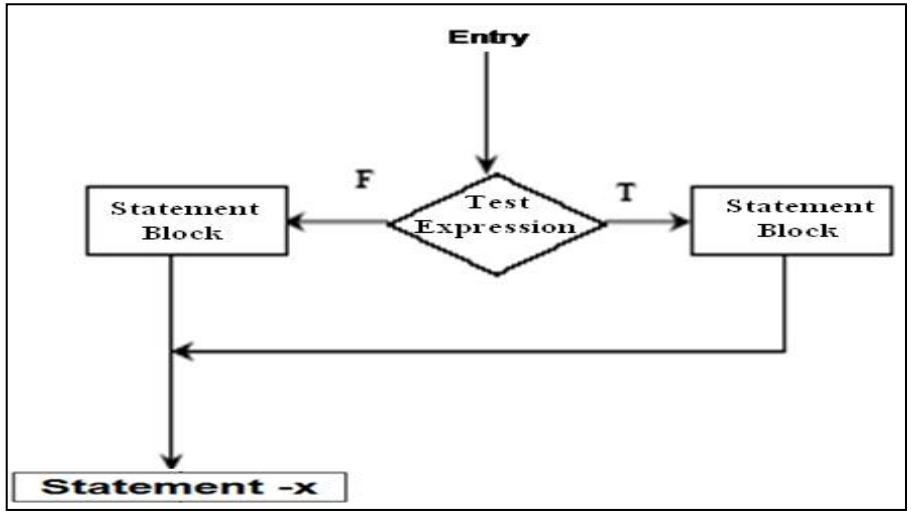
(2) Put the normal cases first ; put the exceptional cases later.

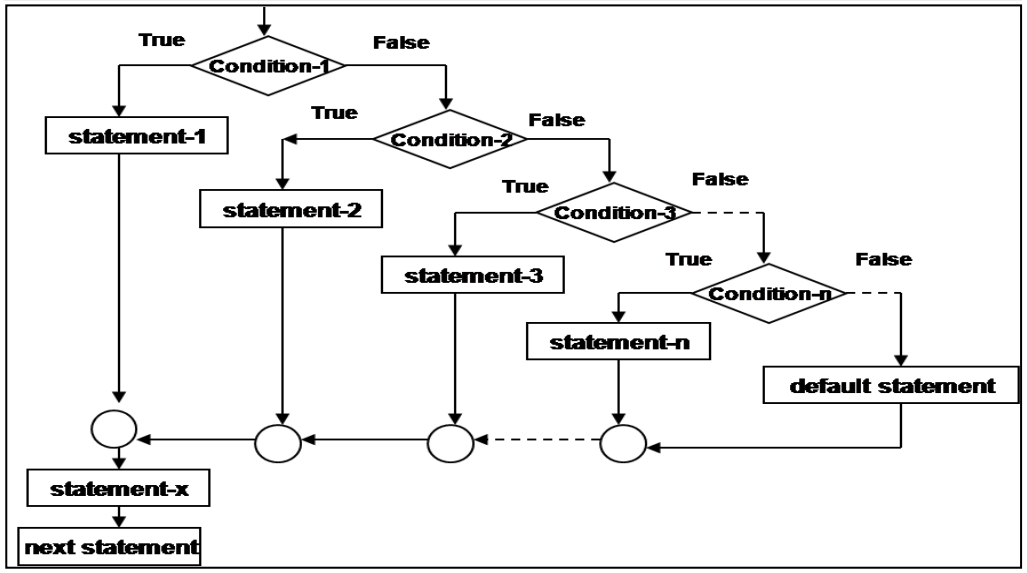
(3) Order cases by frequency:-put the most frequently executed cases first and the least frequently used cases later.

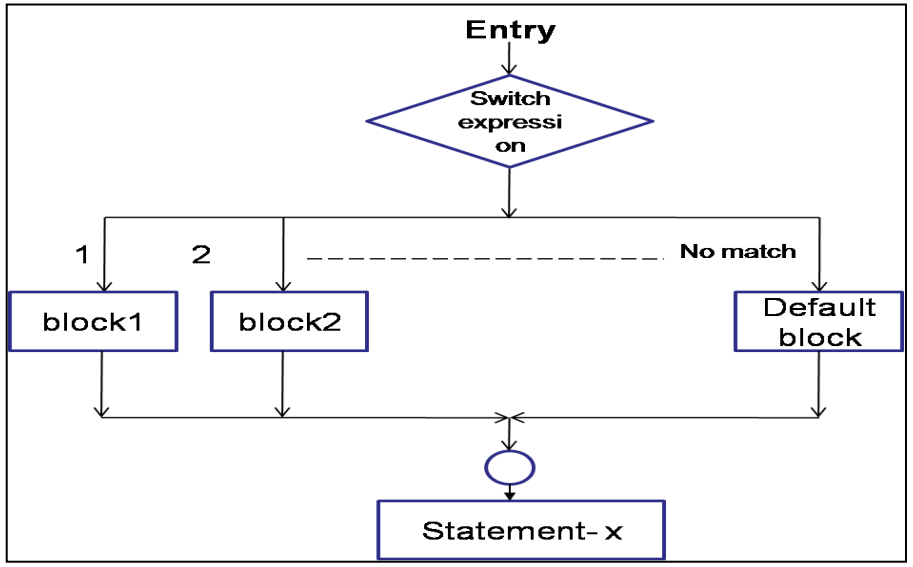
(4) Use default case to detect errors and unexpected cases [user friendly messages].

Flow of control in various control structures







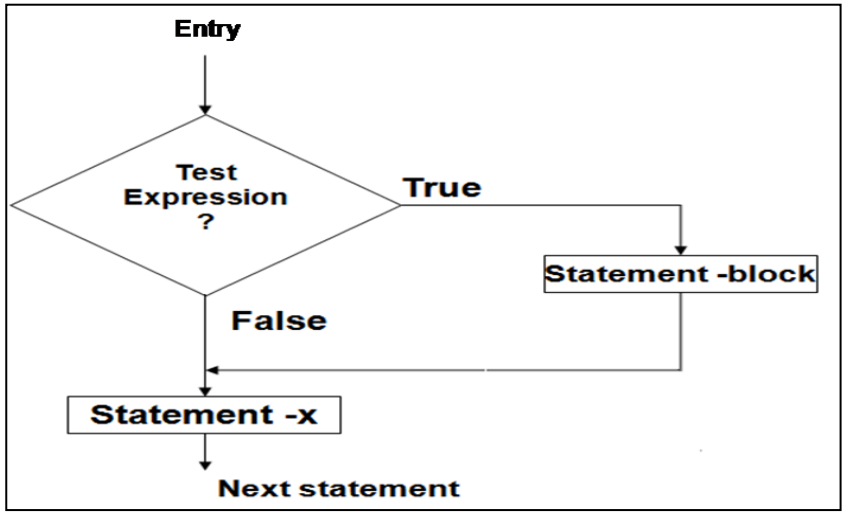


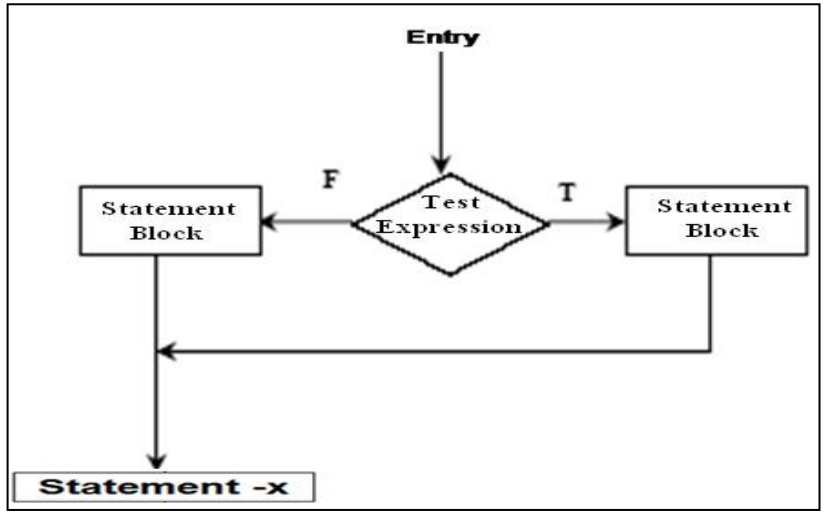
Summary

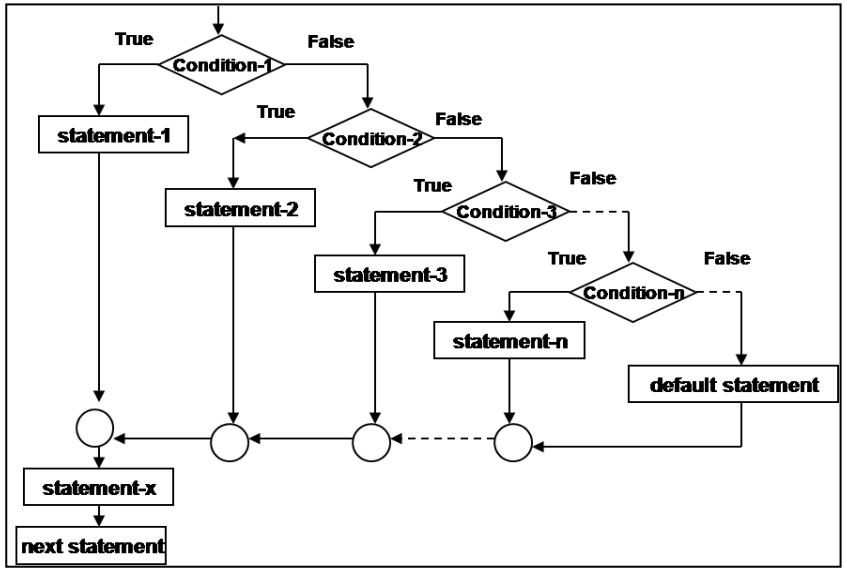
- The if Statement
- The if-else Statement
- Nested if Statements
- The else if Ladder
- The switch Statement

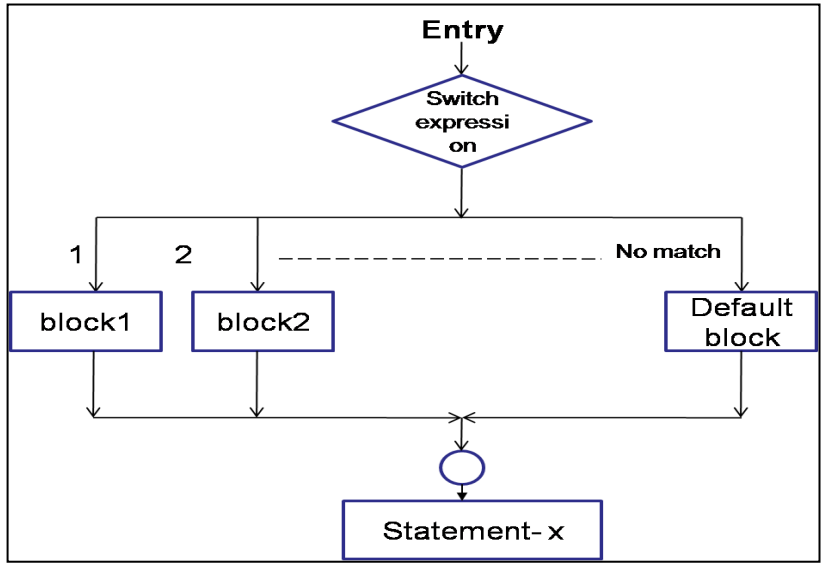
Review on decision making & branching control Structures

- if
- if-else
- Nested if
- else if Ladder
- switch



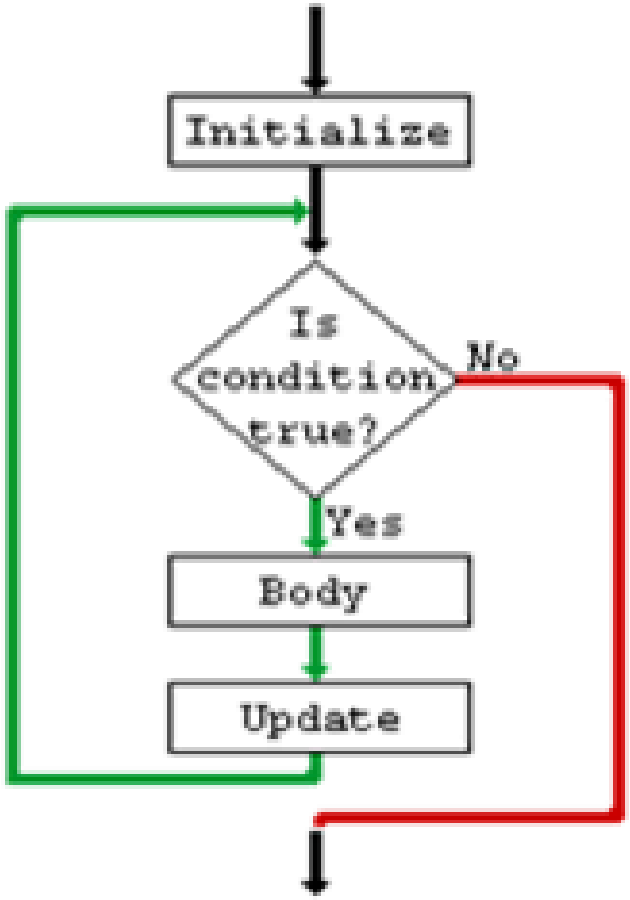






Loop Control Structures

L12-L13



Objectives

- To learn and appreciate the following concepts
  - The `for` Statement
  - Nested `for` Loops
  - `for` Loop Variants
  - The `while` Statement
  - The `do` Statement
  - The `break` Statement
  - The `continue` Statement
  - Typedef and Enum

At the end of session student will be able to learn and understand

- The for Statement
- Nested for Loops
- for Loop Variants
- The while Statement
- The do Statement
- The break Statement
- The continue Statement
- Typedef and Enum

Controlling the program flow

- Forms of controlling the program flow:
  - –Executing a sequence of statements (**Sequential**)
  - –Using a test to decide between alternative sequences (**branching**)
  - –Repeating a sequence of statements (until some condition is met) (**looping**)

Sequential

```
Statement1
Statement2
Statement3
Statement4
Statement5
Statement6
Statement7
Statement8
```

Program Looping

- **A set of statements that executes repetitively for a number of times.**
- Simple example: displaying a message 100 times:

```
printf(hello !\n”);
printf(hello !\n”)
printf(hello !\n”)
…
printf(hello !\n”)
printf(hello !\n”)
```

```
Repeat 100 times
        printf(hello !\n”)
```

***Program loop**:* enables you to develop concise programs containing repetitive processes that could otherwise require many lines of code !

Iterative (loop) control structures

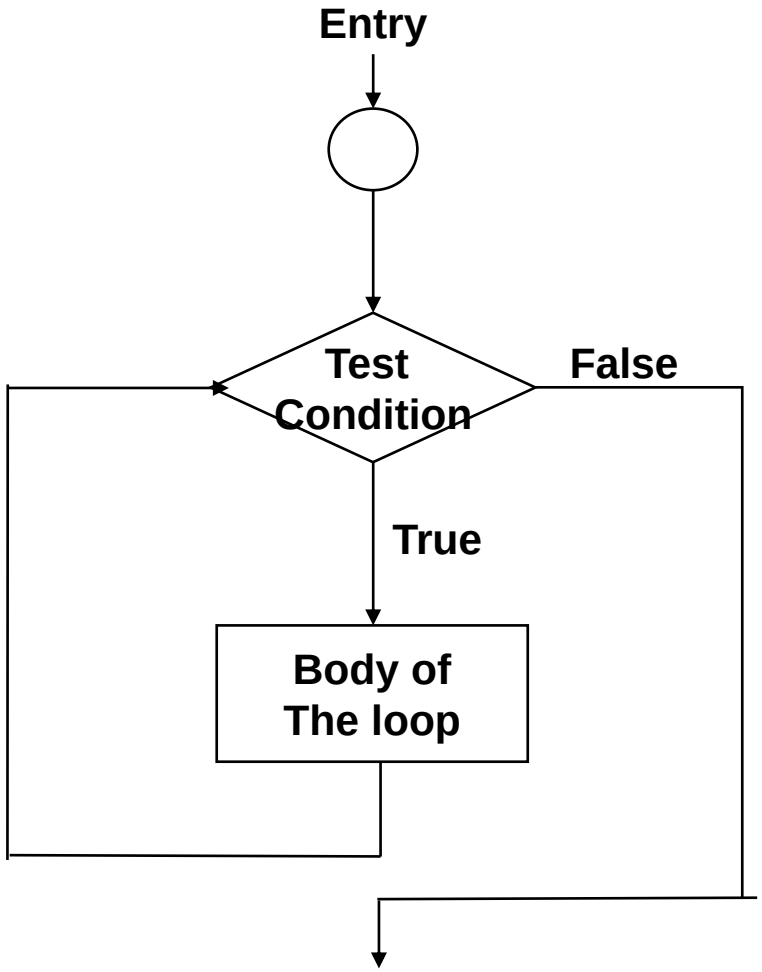
- Each loop control structure will have
  - ✓ **Program loop:** body of loop.
  - ✓ **control statement** → tests certain conditions & then directs repeated execution of statements within the body of loop.

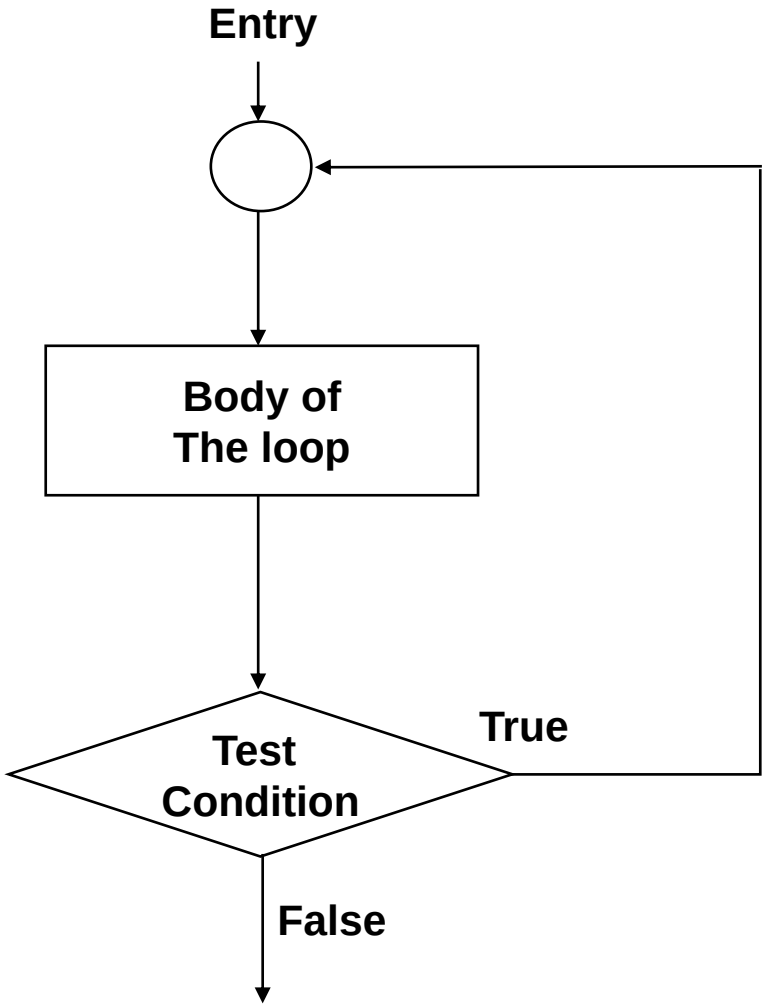
- **Two types:** Based on position of control statement.

1. **Entry controlled loop**: control is tested before the start of the loop. If false, body will not be executed.

2. **Exit controlled loop**: test is performed at the end of the body. i.e. body of loop executed at least once.

Entry Controlled & Exit controlled loops





Iterative (loop) control structures

- Three kinds of loop control structures:
  - ✓while
  - ✓do-while
  - ✓for

`while` statement

General form:

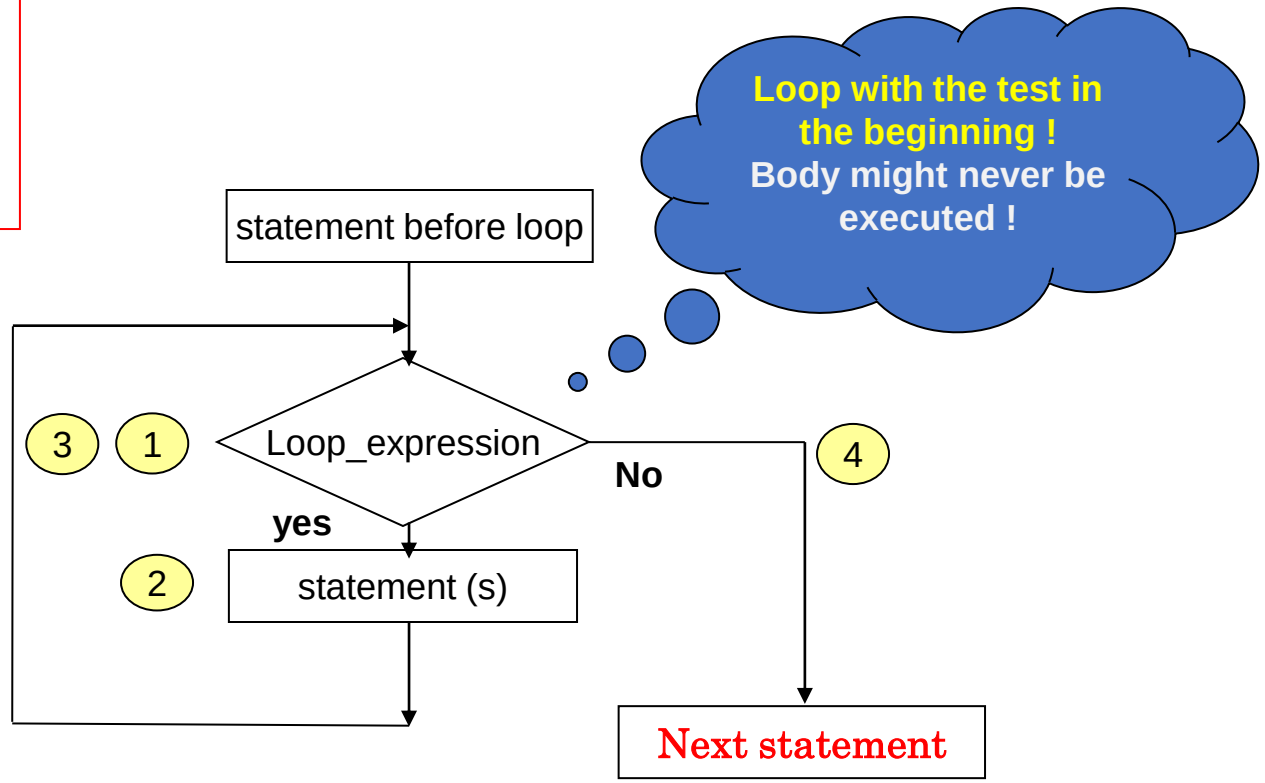
```
while (test expression)
    {
       body of the loop
     }
```

Note: braces optional if only one statement.

- ✓**Entry controlled** loop statement.
- ✓**Test condition** is evaluated & if it is true, then body of the loop is executed.
- ✓This is **repeated until the test condition becomes false**, & control transferred out of the loop.
- ✓**Body of loop is not executed if the condition is false at the very first attempt.**
- ✓**While loop can be nested.**

The `while` statement

```
while ( expression )
 {
        program statement(s)
 }
```



Finding sum of natural numbers up to 100

```
#include <stdio.h>
int main()
{
   int n;
   int sum;
   sum=0; //initialize sum
   n=1;
   while (n < 100)
   {
       sum= sum + n;
       n = n + 1;
   }
   printf("%d",sum);
   return 0;
}
```

Program to reverse the digits of a number

Algorithm

Name of the algorithm: reverse the digits of a number **n**

Step 1: Start

step 2: Input n

Step 3: rev←0

Step 4: WHILE n!=0

begin

rd← n mod 10

rev← rev *10 + rd

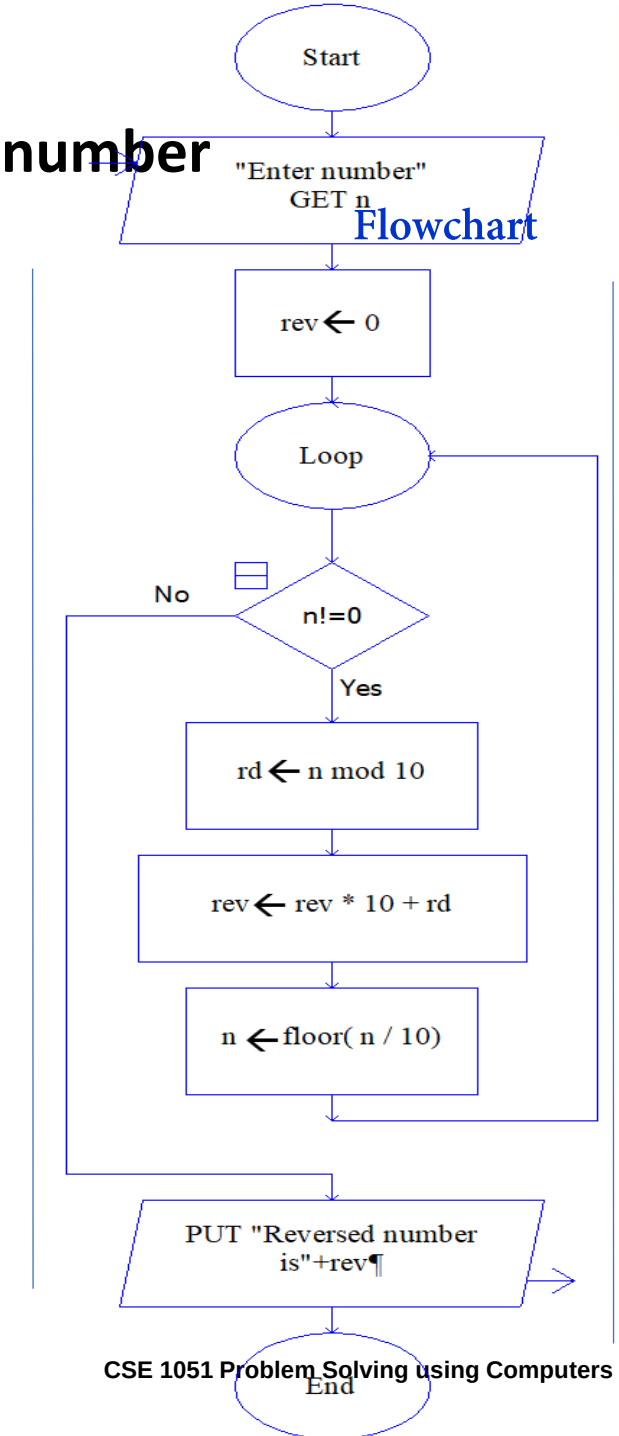
n←n/10 (integer)

end

Step 5: Print 'The reversed no is =', rev

Step 6: Stop

Flowchart



Program

```
#include <stdio.h>
int main()
{
    int n, rev=0, rd;

    printf("Enter your number.\n");
    scanf("%d",&n);

    while ( n != 0 )
    {
        rd = n % 10;
        rev=rev*10 + rd;
        n = n / 10;
    }
    printf("The reversed no is %d", rev);
    return 0;
}
```

The `do-while` statement

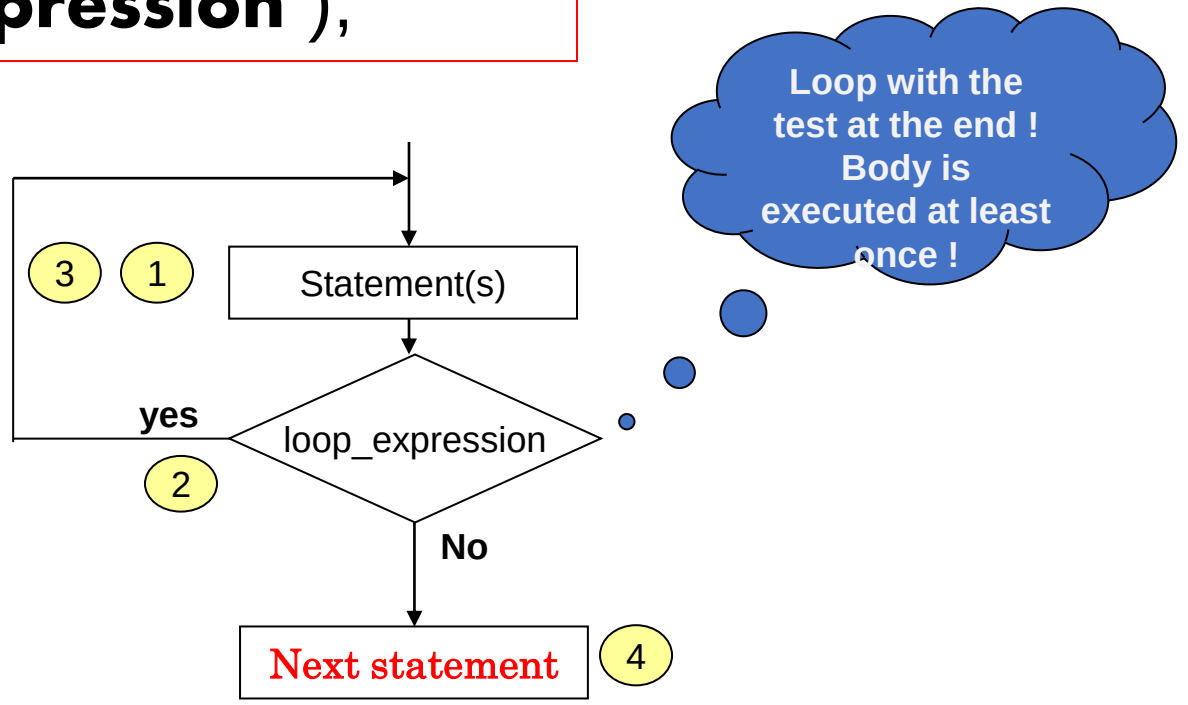
General form:

```
do
{
    body of the loop
}
while(test condition);
```

- ✓**Exit controlled loop**. At the end of the loop, the test condition is evaluated.
- ✓After do statement, program executes the body of the Loop.
- ✓Then, the condition is tested, if it is true, body of the loop is executed once again & this process continues as long as the condition is true.
- ✓**Body of the loop is executed at least once.**
- ✓**do-while loop can be nested.**

The `do-while` statement

```
do {
        program statement (s) }
while ( loop_expression );
```



Example: Finding sum of natural numbers up to 100

```
#include <stdio.h>
int main()
{
   int n;
   int sum =0;

   n=1;
   while (n<=100)
   {
       sum=sum+n;
       n = n +1;
   }
   printf("%d", sum);
   return 0;
}
```

```
#include <stdio.h>
int main()
{
   int n;
   int sum=0;
    n=1;
   do
     {
        sum = sum + n;
         n = n +1;
     }  while (n <=100);

   printf("%d", sum);
   return 0;
}
```

Program to add numbers until user enters zero

```
#include <stdio.h>
int main()
{
   int number, sum = 0;

         // loop body is executed at least once
  do {
      printf("Enter a no:\n(enter zero to exit and display sum)\n");
      scanf("%d", &number);
      sum += number;
      } while(number != 0);

   printf("Sum = %d",sum);

   return 0;
}
```

`for` statement

General form:

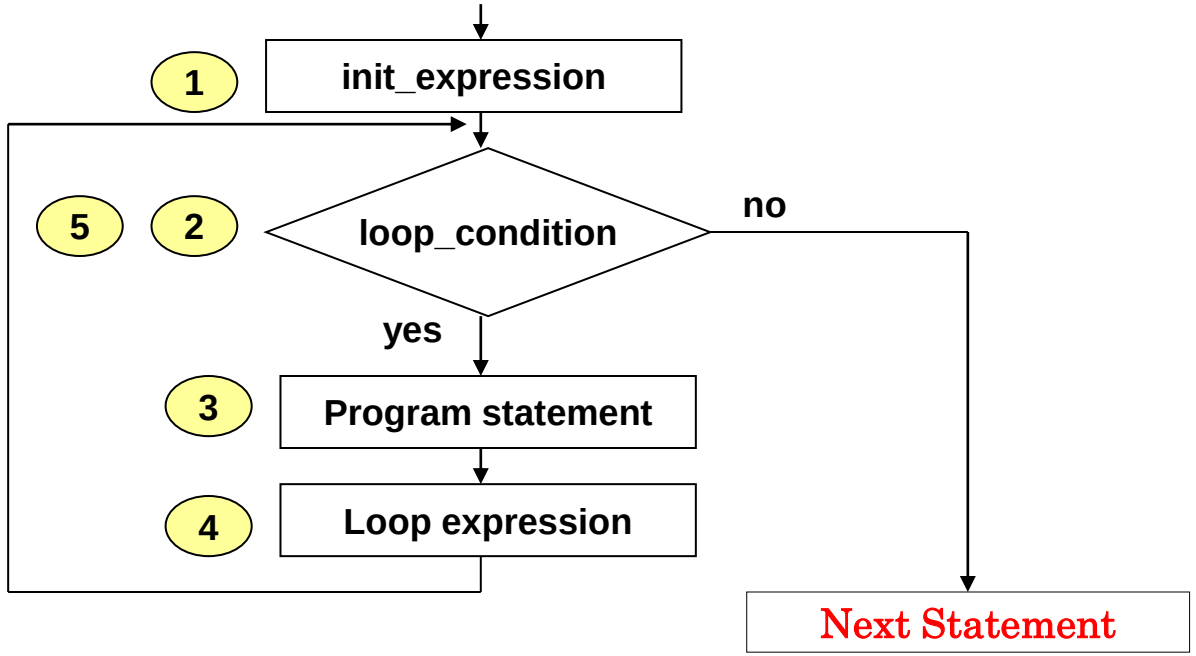
```
for (initialization; loop_condition; loop_expression)
    {
       body of the loop
    }
```

Note: braces optional if only one statement.

- ✓Entry controlled loop statement.
- ✓Test condition is evaluated & if it is true, then body of the loop is executed.
- ✓This is repeated until the test condition becomes false, & control transferred out of the loop.
- ✓Body of loop is not executed if the condition is false at the very first attempt.
- ✓for loop can be nested.

The `for` statement

```
for ( init_expression; loop_condition;
loop_expression )
 {
        program statement(s)
 }
```

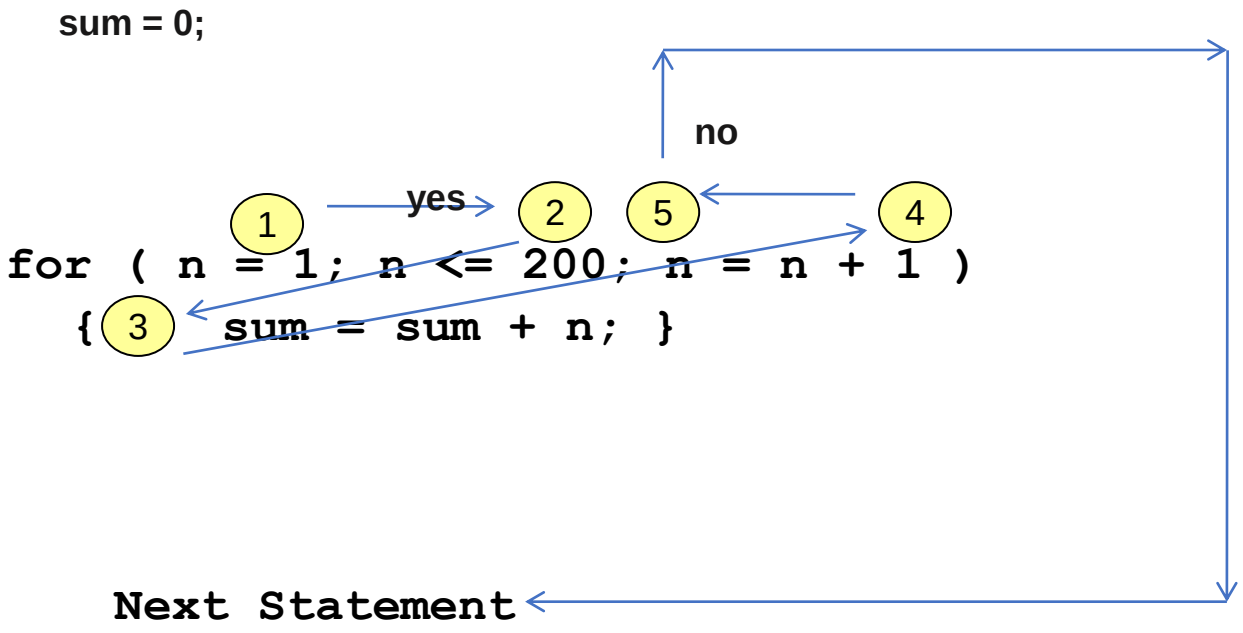


How `for` works

- The execution of a for statement proceeds as follows:

1. The initial expression is evaluated first. This expression usually sets a variable that will be used inside the loop, generally referred to as an *index* variable, to some initial value.
2. The looping condition is evaluated. If the condition is not satisfied (the expression is false – has value 0), the loop is immediately terminated. Execution continues with the program statement that immediately follows the loop.
3. The program statement that constitutes the body of the loop is executed.
4. The looping expression is evaluated. This expression is generally used to change the value of the index variable
5. Return to step 2.

The `for` statement



for (**init_expression**; **loop_condition**; **loop_expression**)
{ *program statement(s)*
}

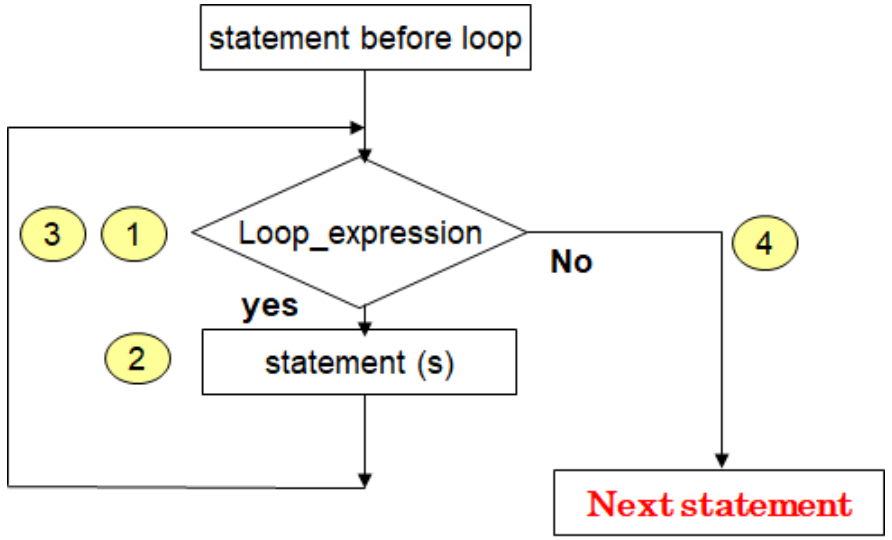
Finding sum of natural numbers up to 100

```
#include <stdio.h>
int main() {
    int n;
    int sum;
    sum=0; //initialize sum
    n=1;
    while (n < 100)
    {
        sum= sum + n;
        n = n + 1;
    }
    printf("%d",sum);
    return 0;
}
```

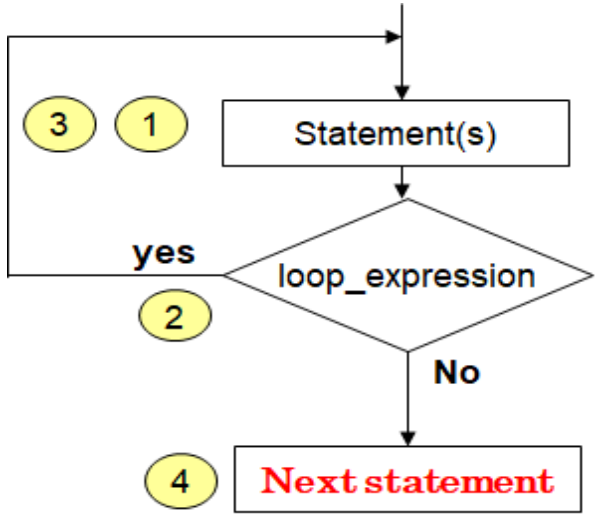
```
#include <stdio.h>
int main(){
    int n;
    int sum;
    sum=0; //initialize sum
    for(n = 1; n < 100; n=n + 1)
        {
            sum=sum + n;
        }

    printf("%d",sum);
return 0;
}
```

Review on decision making & looping

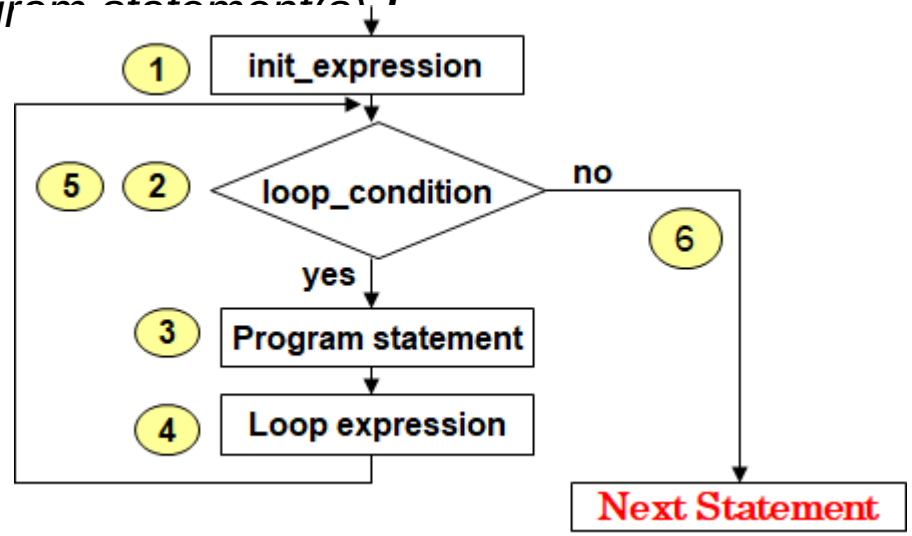
while (**expression**)
{ *program statement(s)* **}**



do *{ program statement (s) }*
while (**loop_expression**);



for (**init_expression**; **loop_condition**; **loop_expression**)
{ *program statement(s)* **}**



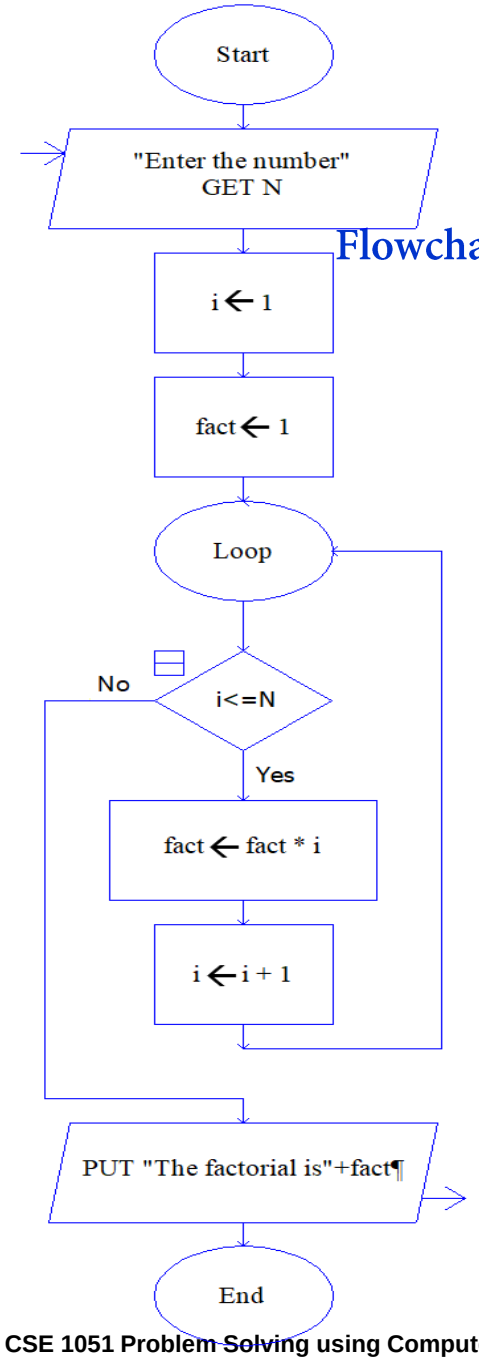
Compute the factorial of a number

Algorithm

Name of the algorithm: Compute the factorial of a number

Step1: Start
Step 2: Input N
Step 3: fact ←1
Step 4: **for i=1 to N in step of 1 do**
begin
fact←fact*i
end
Step 5: Print 'fact of N=', fact
Step 6: [End of algorithm]
Stop

Flowchart



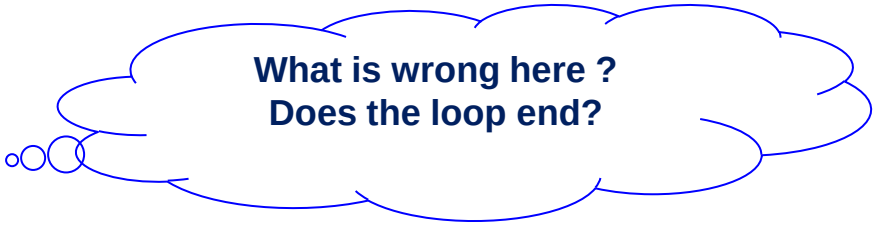
Program

```
#include <stdio.h>
int main()
{
  int N, i, fact=1;
  printf("Enter the number");
  scanf("%d", &N);

   for(i=1; i<=N; i++)
        fact=fact * i;

  printf("The factorial is %d", fact);
  return 0;
}
```

Infinite loops

It's the task of the programmer to design correctly the algorithms so that loops end at some moment !

```
// Program to count 1+2+3+4+5
#include <stdio.h>
int main() {
    int  i, n = 5, sum =0;
    for ( i = 1; i <= n; n = n + 1 ) {
      sum = sum + i;
      printf("%d",sum);
    }
    return 0;
}
```



Nesting of `for` loop

- One for statement can be nested within another for statement.

```
for (i=0; i< m; ++i)
    {       .....
            ....
            for (j=0; j < n;++j)
            {
                    Statement S;
            } // end of  inner 'for' statement
     }// end of  outer 'for' statement
```

Multiplication table for ‘n’ tables up to ‘k’ terms

```
scanf(“%d %d”,&n,&k);

for (i=1; i<=k; i++)
    {
     for (j=1; j<=n; j++)
      {
        prod = i * j;
         printf(“%d * %d = %d\t”, j, i, prod);
   }
     printf(“\n”);
    }
```

```
Enter n & k values: 3  5
The table for 3 X 5 is
1 * 1= 1      2 * 1= 2      3 * 1= 3
1 * 2= 2      2 * 2= 4      3 * 2= 6
1 * 3= 3      2 * 3= 6      3 * 3= 9
1 * 4= 4      2 * 4= 8      3 * 4= 12
1 * 5= 5      2 * 5= 10     3 * 5= 15
```

`for` loop variants

- **Multiple expressions *(comma between…)***

```
for(i=0 , j=10 ; i<j ; i++ , j--)
```

- **Omitting fields *(semicolon have to be still…)***

```
i=0;
for( ;  i<10 ; i++ )
```

- **Declaring variables**

```
for(int i=0 ; i=10  ; i++)
```

Which loop to choose ?

- *Criteria*: category of looping
  - Entry-controlled loop -> for, while
  - Exit-controlled loop -> do

- *Criteria*: Number of repetitions:
  - Indefinite loops -> while
  - Counting loops -> for

- You can actually rewrite any *while* as a *for* and vice versa !

The `break` Statement

- Used in order to immediately exit from a loop
- After a break, following statements in the loop body are skipped and execution continues with the first statement after the loop
- If a break is executed from within nested loops, only the innermost loop is terminated

Exiting a loop with `break` statement

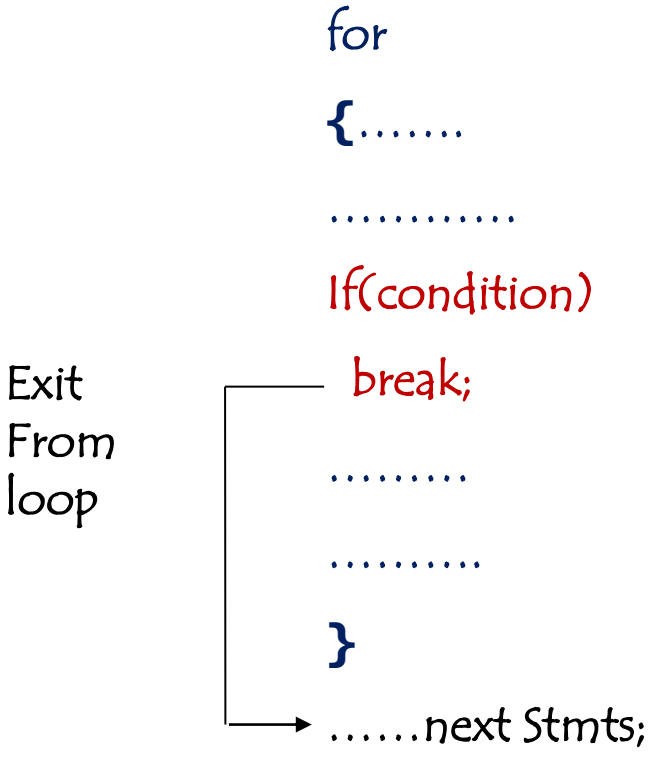
```
while (…….)
{…….
…….
If(condition)
   break;
……..
……..
} // end of while
…….. //next
          statement
```

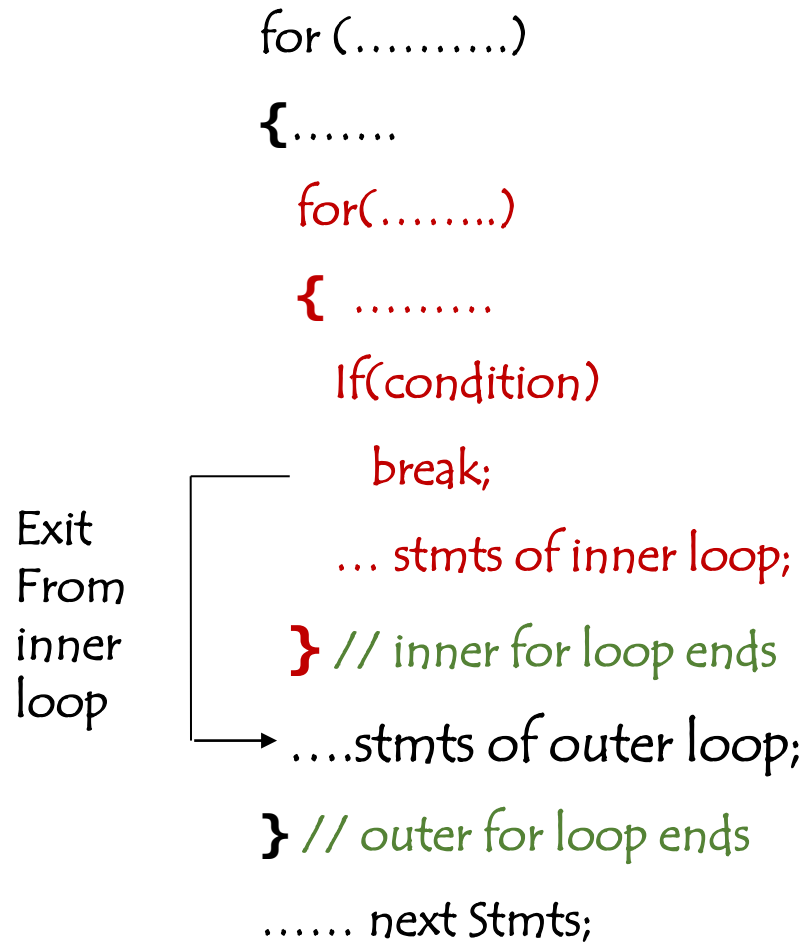
Exit From loop

```
do
{…….
…….
If(condition)
   break;
……..
……..
} while(…);
…….. // next
          statement
```

Exit From loop

Exiting a loop with `break` statement





Check whether given number is prime or not

```
int   j,  prime=1;
 scanf("%d",&N);
for( int j=2; j<N; j++ )
 {
    if( (N % j) == 0)
    {
    prime=0;
    break; /* break out of for loop */
    }
 }
 if (prime == 1)
    printf("%d is a prime no",N);
else
    printf("%d is a not a prime no",N);
```

Program to generate prime numbers between given 2 limits

```
scanf("%d %d",&m,&n);

for( int i=m; i<=n; i++) {
    int prime=1;
    for( int j=2; j<i; j++ ) {
        if( i % j == 0)
          {
                prime=0;
                break; /* break out of inner loop */
          }
     }
    if (prime == 1) printf("%d\t",i);
}
```

Skipping a part of loop – `continue` statement

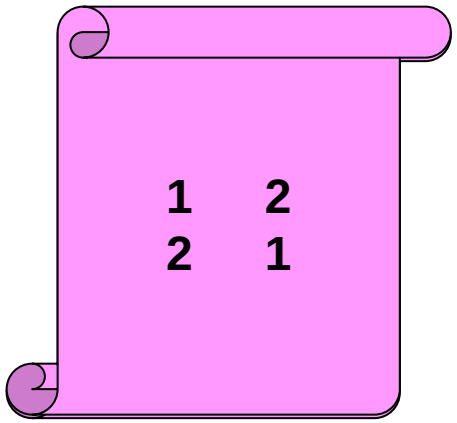
- Skip a part of the body of the loop under certain conditions is done using **continue** statement.
- As the name implies, **continue** causes the loop to be continued with next iteration, after skipping rest of the body of the loop.

```
while (…….) {
    Statement-1;
    Statement-2;

    If(condition)
        continue;

    Statement-3;
    Statement-4;
}

Next_statement
```

```
do {
    Statement-1;
    Statement-2;

    If(condition)
        continue;

    Statement-3;
    Statement-4;
} while(…);

Next_statement
```

Skipping a part of loop

```
for ( i = 1 ; i <= 2 ; i++ )
{
   for ( j = 1 ; j <= 2 ; j++ )
   {
      if ( i == j )
         continue ;
      printf("\n %d\t %d\n",i, j);
   }
}
```



User defined Type declarations

- *typedef*
  - Type definition – lets you define your own identifiers.
- *enum*
  - Enumerated data type - a type with restricted set of values.

User defined Type Declaration

- *typedef* **type identifier;**

  The "type" refers to an existing data type and "identifier" refers to the new name given to the data type.

- After the declaration as follows:

  typedef **int marks;**

  typedef **float units;**

  we can use these to declare variables as shown

  marks *m1,m2* **;** *//m1 & m2*are declared as integer variables

  units *u1, u2;* *//u1 & u2* are declared as floating point variables

  The main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

User defined Type Declaration - enum

enum identifier { value1, value2,..,value$_n$ };

- Here, ***identifier*** is the name of enumerated data type or tag.
  And ***value1, value2,....,valueN*** are values of type identifier.

- By default, *value1* will be equal to 0, *value2* will be 1 and so on but, the programmer can change the default value.

```
enum card {club, diamonds, hearts, spades};
enum card {club=0, diamonds, hearts=20, spades};

enum card shape, s1;

shape = hearts;

s1 = spades;
```

Mainly used to assign names to integral constants, the names make a program easy to read and maintain.

User defined Type Declaration - enum

```
#include<stdio.h>
int main()
{
enum year{Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec};
    int i;
    for (i=Jan; i<=Dec; i++)
        printf("%d ", i);
    return 0;
}
```

Output

```
0  1  2  3  4  5  6  7  8  9  10  11
```

```
enum week { Monday =1, Tuesday, Wednesday, Thursday, Friday,  Saturday, Sunday};
```

```
printf( " Enter  n >1 & <7 ");
 scanf("%d",&n);

switch(n) {

case Monday:
        printf("Monday");
        break;

case Tuesday:
        printf("Tuesday" );
        break;

case Wednesday:
        printf("Wednesday");
        break;

case Thursday:
        printf(" Thursday");
        break;

case Friday:
        printf(" Friday");
        break;

case Saturday:
        printf("Saturday");
        break;

case Sunday:
        printf("Sunday");
        break;

default:
        printf("\nInvalid Entry. Enter  1 to 7 ");

}
```

Check a given number for palindrome

```
rev=0;
n = num;
  while(num>0)
          {
            dig = num % 10;
            rev = rev * 10 + dig;
            num = num / 10;
          }
 if (n == rev)
  printf("\n\t GIVEN NO IS A PALINDROME");
 else
  printf("\n\t GIVEN NO NOT A PALINDROME");
```

Palindrome (num)
e.g.- 121

Convert binary to decimal

$$dec = bd*2^n + bd*2^{n-1} + \ldots + bd*2^1 + bd*2^0$$

e.g.- given
n=101 → $1*2^2 + 0*2^1 + 1*2^0 = 5$

```
int  n, p=0, sum=0, k;
printf("Enter a binary number : ");
scanf("%d",&n);

do {
    k=n%10; // binary number  in n; extracts the digit
    sum= sum + k * pow(2,p);//decimal number in sum; multiplication and summing
    p++; //incrementing p value for next iteration
    n= n/10; //remaining digits for next iteration
    } while (n!=0);

printf("Decimal Equivalent = %d", sum);
```

Sine series for a given ‘n’ terms & angle ‘x’

```
# define PI 3.141592
scanf("%d %f",&n,&x);
 no=x;
 x=x*PI/180.0; // degrees to radians conversion
 term=x; // first term value
 sum=x; //term stored in sum
  for (i=1;i<=n;i++) // computation & summation for second term onwards
   {
    term= term*(((-1)*x*x)/(2*i*(2*i+1)));
    sum+=term;
   }
printf("Library value of Sin(%.2f) = %.2f ", no, sin(x));
printf("\nSin (%.2f) = %.2f", no, sum);
```

Sine series

$\sin(x) = x - x^3/3! + x^5/5! - \ldots x^n/n!$

Armstrong nos for a given limit ‘n’

```
scanf(“%d”,&lim);
for(n=1;n<lim;n++){
  sum = 0;
  num = n;
  while(num>0) {
   dig = num%10;
   sum = sum+pow(dig,3);
   num = num/10;
  }
if(sum == n)
   printf(“%d\n\t”,n);
}
```

Armstrong Number
e.g. - 371
Σ (cubes of digits)= num
$3^3 + 7^3 + 1^3 = 371$

Count the even and odd digits in a given ‘n’ digit number

```
scanf(“%d”,&num);
while(num > 0)
 {
   rem=num%10;
   num =num/10;
   if(rem%2==0)
     ecnt++;
   else
     ocnt++;
  }
```

e.g.- num = 31467
OUTPUT
2 even & 3 odd digits

```
printf(“%d even & %d odd digits”, ecnt, ocnt);
```

Tutorial Problems

1. Write a C program to count number of digits in any number
2. Write a C program to find last and first digit of any number
3. Write a C program to enter any number and print all its factors
4. Write a C program to find LCM of two numbers
5. Write a C program to convert Binary to Octal number

- Summary
  - The `for` Statement
  - Nested `for` Loops
  - `for` Loop Variants
  - The `while` Statement
  - The `do` Statement
  - The `break` Statement
  - The `continue` Statement
  - Typedef and Enum

Error or not !...

```
int x=3;
 if (x=2)
   x=0;
 if (x==3)
   x+=1;
 else
   x+=2;
printf("%d", x);
```

Warning!
Possibly incorrect assignment

Output:
2

Error or not !...

```
int x=3, y=15;
  if (x = y%3)
    x=0;
  else
    x+=2;

printf("%d", x);
```

Warning!
Possibly incorrect assignment

Output:
2

Why output =2??
x=1; Condition is always TRUE for any value other than 0!

Error or not !...

```
int i, n=10;
  for (i=0; i<n; i++);
    i+=i;
printf("%d", i);
```

No Error!

Output:

150