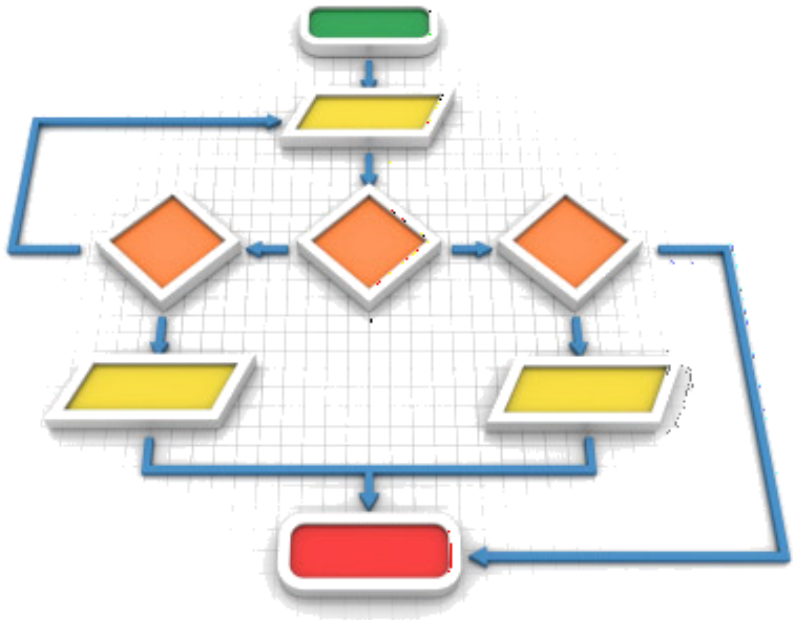
S9_1

Control Structures

Branching & Looping



Control Structures – Recap

- **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) [if & switch statements]**

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

✓ **Iterational (Repetition) [while, do-while & for statements]**

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

Session Objectives

- To learn and appreciate the following concepts
  - The `for` Statement
  - Nested `for` Loops
  - `for` Loop Variants

Session Outcomes

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

- The `for` Statement
- Nested `for` Loops
- `for` Loop Variants

`for` statement

General form:

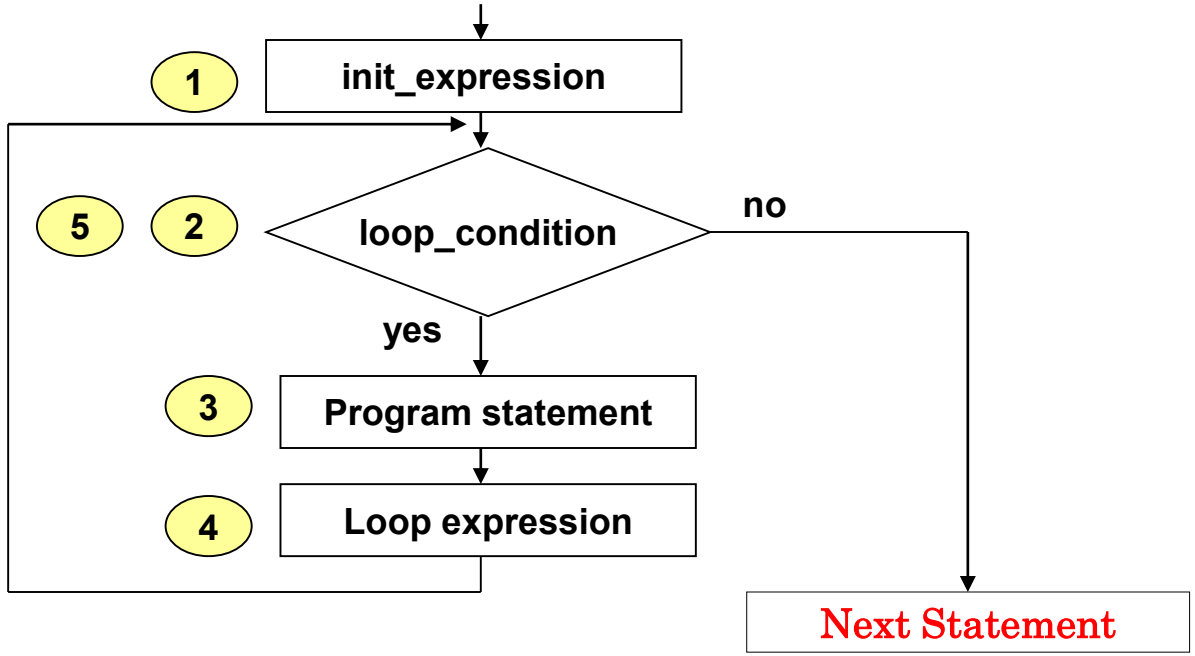
```
for (initialization; loop_condition; loop_expression)
    {
       body of the loop
    }
```

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)
 }
```

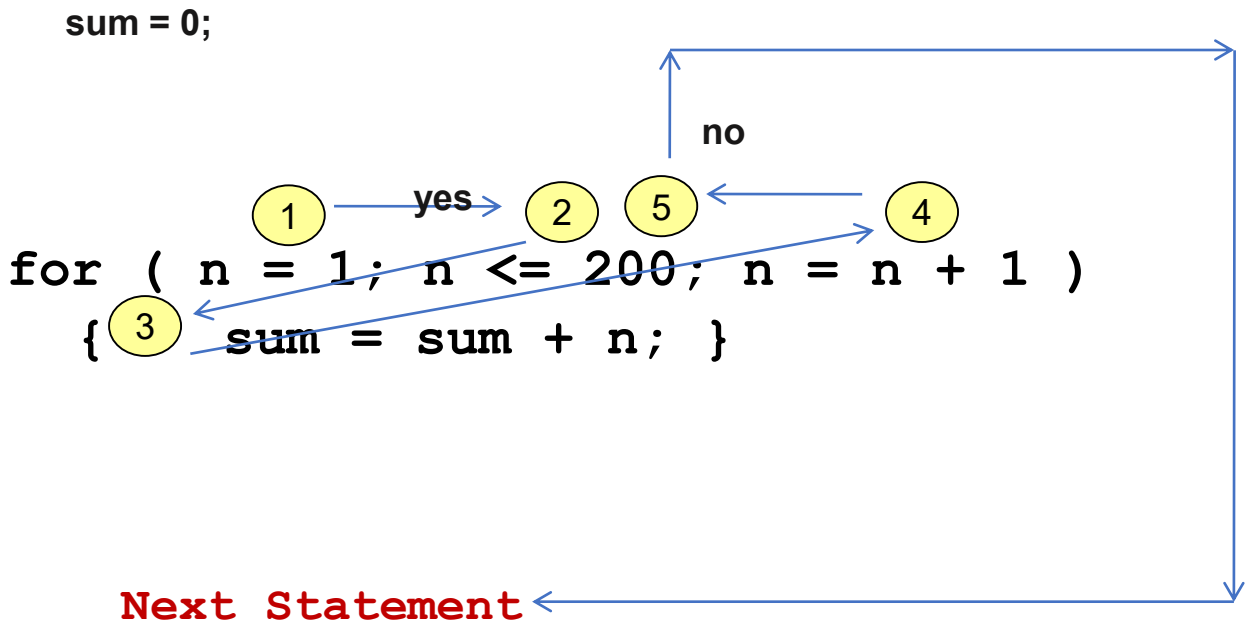


How `for` works

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)
}
```

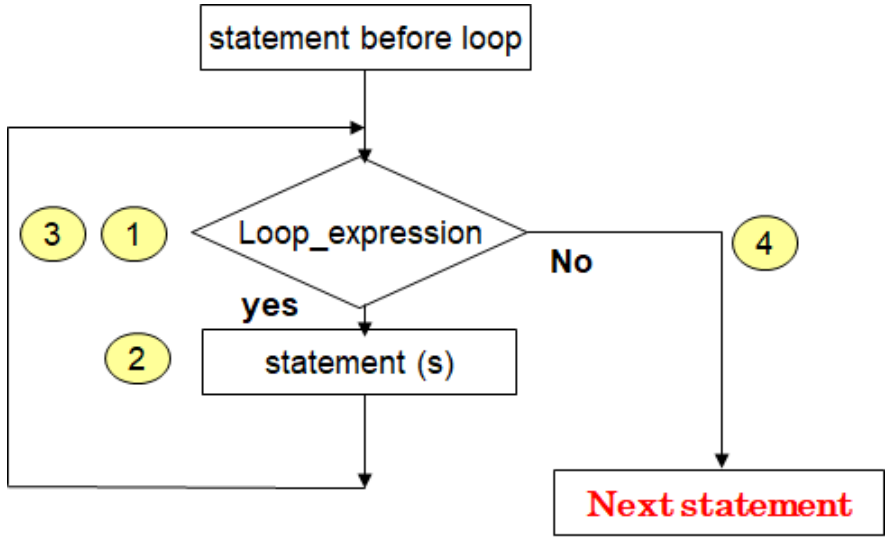
Finding sum of natural numbers up to 100

```
#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;
}
```

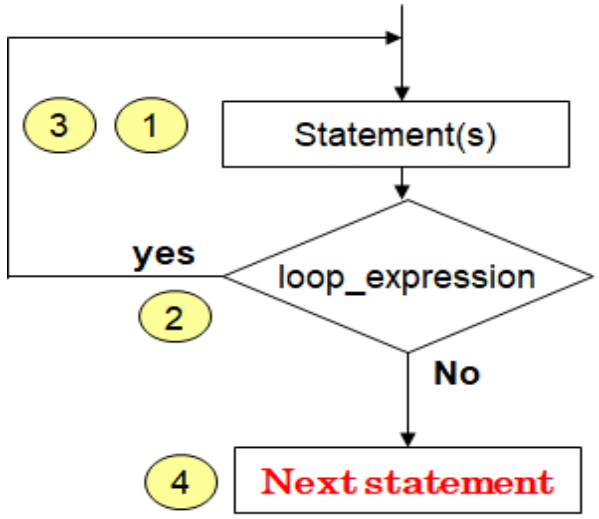
```
#include <stdio.h>
int main(){
    int n;
    int sum;
    sum=0; //initialize sum
    for(n = 1; n <= 100; n=n + 1)
        {
            sum=sum + n;
        }

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

Review on decision making & looping

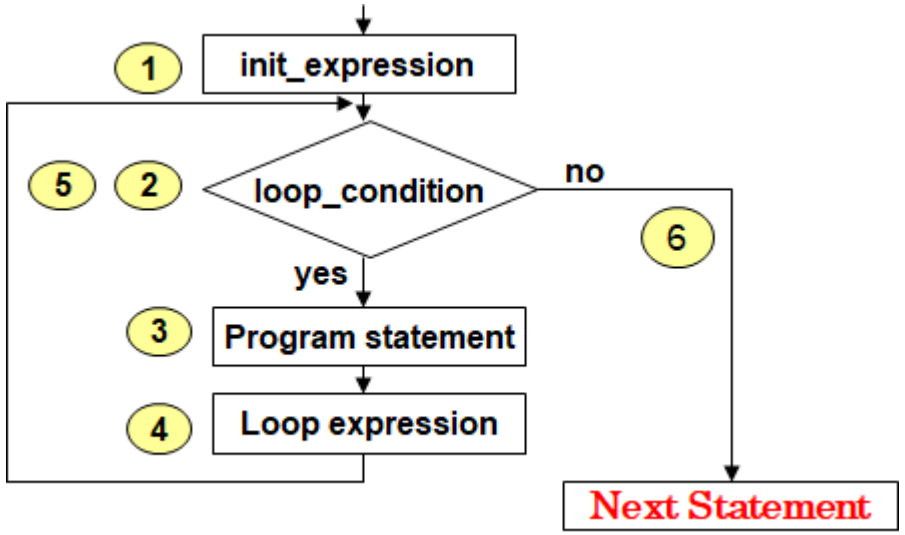
while (**expression**)
{ *program statement(s)* **}**



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



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

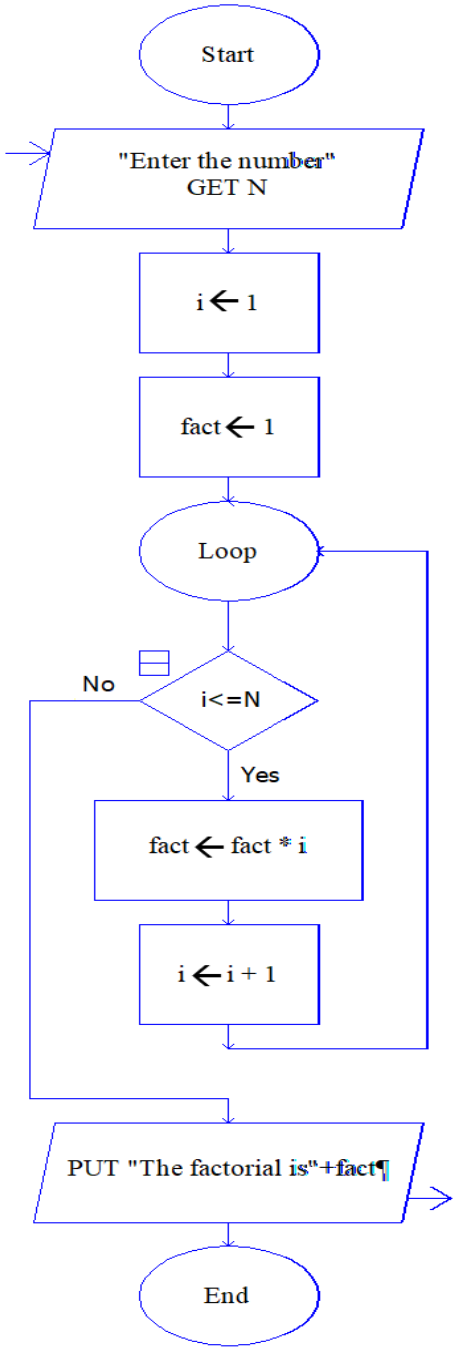


Compute the factorial of a number

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



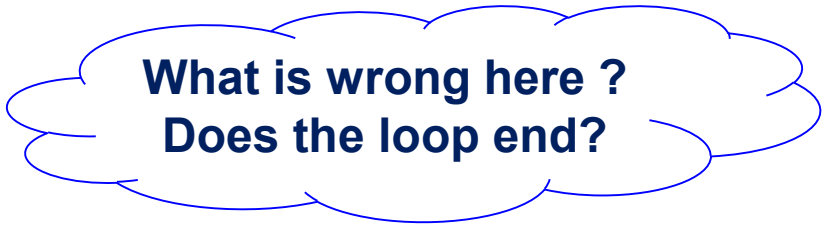
Program

```
#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…)***

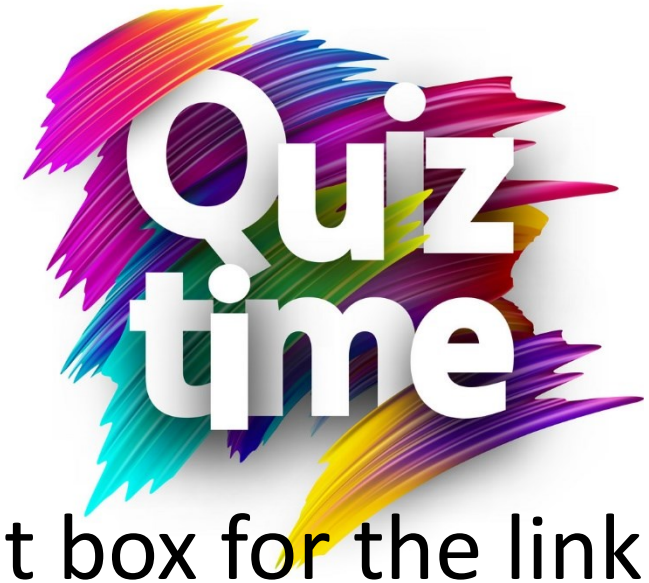
```
for(i=0 , j=10 ; i<j ; i++ , j--)
```

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

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

- **Declaring variables**

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



Go to posts/chat box for the link to the question

submit your solution in next 2 minutes

The session will resume in 3 minutes

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* using a *for* and vice versa !

Summary

- The `for` Statement
- Nested `for` Loops
- `for` Loop Variants