Tutorial Link https://codequotient.com/tutorials/Iteration - for loop variants/5a22a545c66cfe38f2962289

TUTORIAL

Iteration - for loop variants

Chapter

1. Iteration - for loop variants

The part of for construct need not to be defined in for loop body itself, we can define them above and below also. If the program found them in desired sequence, the program will still be fine and execute smoothly. For example, following are some variations of valid for loops: -

```c
#include<stdio.h>

int main()
{
  int c=0;
  printf("Before the for loop. c = %d\n",c);
  for(c = 1; c <= 5; )
    // increment is omitted but semicolon needs to be there.
  {
    printf("%d \n",c);
    c++;    // as it again comes in same sequence of execution so it works.
  }
  printf("After the for loop. c = %d",c);
  return 0;
}
```

```
16
```

Even we can change the initialization as below:

```c
#include<stdio.h>

int main()
{
  int c = 1;
  printf("Before the for loop. c = %d\n",c);

  for( ; c <= 5; c++)
    // initialization is not done here as c is already initialized to 1 above.
    // increment is omitted but semicolon needs to be there.
  {
    printf("%d \n",c);
  }

  printf("After the for loop. c = %d",c);
  return 0;
}

```

Beware about the semicolons, as if you forgot to place the semicolons, it will result in syntax error.

```c
#include<stdio.h>

int main()
{
  int c = 1;
  printf("Before the for loop. c = %d\n",c);

  for( ; c <= 5 )
```

```
    // initialization is not done here as c is already initialized to 1 above.
    // increment is omitted but semicolon needs to be there.
  {
    printf("%d \n",c);
    //        c++;    // as it again comes in same sequence of execution so it works.
  }

  printf("After the for loop. c = %d",c);
  return 0;
}

```

Also you can place both initialization and increment out of for loop as below: -

```c
#include<stdio.h>

int main()
{
  int c = 1;
  printf("Before the for loop. c = %d\n",c);

  for( ; c <= 5; )
    // initialization is not done here as c is already initialized to 1 above.
    // increment is omitted but semicolon needs to be there.
  {
    printf("%d \n",c);
    c++;    // as it again comes in same sequence of execution so it works.
  }

  printf("After the for loop. c = %d",c);
  return 0;
```

```
}

```

If we put the condition outside also, then the control will go for infinite times in the loop body, as it will come out of the for loop only when condition evaluates to false. And in absence of condition, it is treated as true. For example,

```c
#include<stdio.h>

int main()
{
  int c = 0;
  printf("Before the for loop. c = %d\n",c);
  for(c = 1 ; ; c++)
  {
    printf("%d \n",c);
  }
  printf("After the for loop. c = %d",c);
  return 0;
}

```

This program will go to infinite loop as there is no reason to come out of loop. Also the following is a common way to execute a infinite loop in C: -

```c
#include<stdio.h>

int main()
{
  int c = 0;
  printf("Before the for loop. c = %d\n",c);
```

```
  for(; ;)
  {
    printf("%d \n",c);
    c++;
  }
  printf("After the for loop. c = %d",c);
  return 0;
}

```

We can come out of infinite for loop also with the help of other constructs like break statements etc. We will study them in a while.

In for loop initialization and increment, we can write more than one expressions. But there should be only one condition. However we can combine multiple conditions with logical operators, but ultimately there must be only one condition to be evaluated. For example, see the below examples

```c
#include <stdio.h>

int main()
{
  int var1, var2;
  // for loop will contain two initializations and two increments
  for(var1 = 1, var2 = 3; var1 <= 3; var1++, var2--)
  {
    printf("var1 = %d \t var2 = %d\n",var1, var2);
  }

  return 0;
}

```