Control Structure/ Iterative /Repetition Structure

By: Atiya Jokhio

Break statement

- The break statement, when executed in a while, for, do…while or switch statement, causes an immediate exit from that statement. Program execution continues with the next statement.
- The break statement is typically used for two purposes:
  - To exit early from a loop
  - To skip the remainder of the switch structure

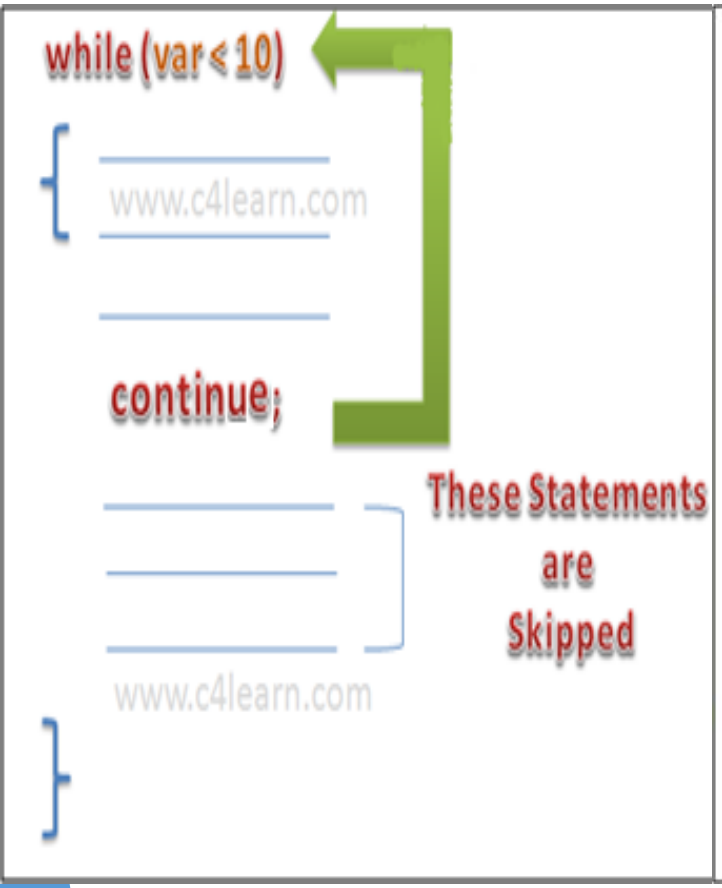
Break statement

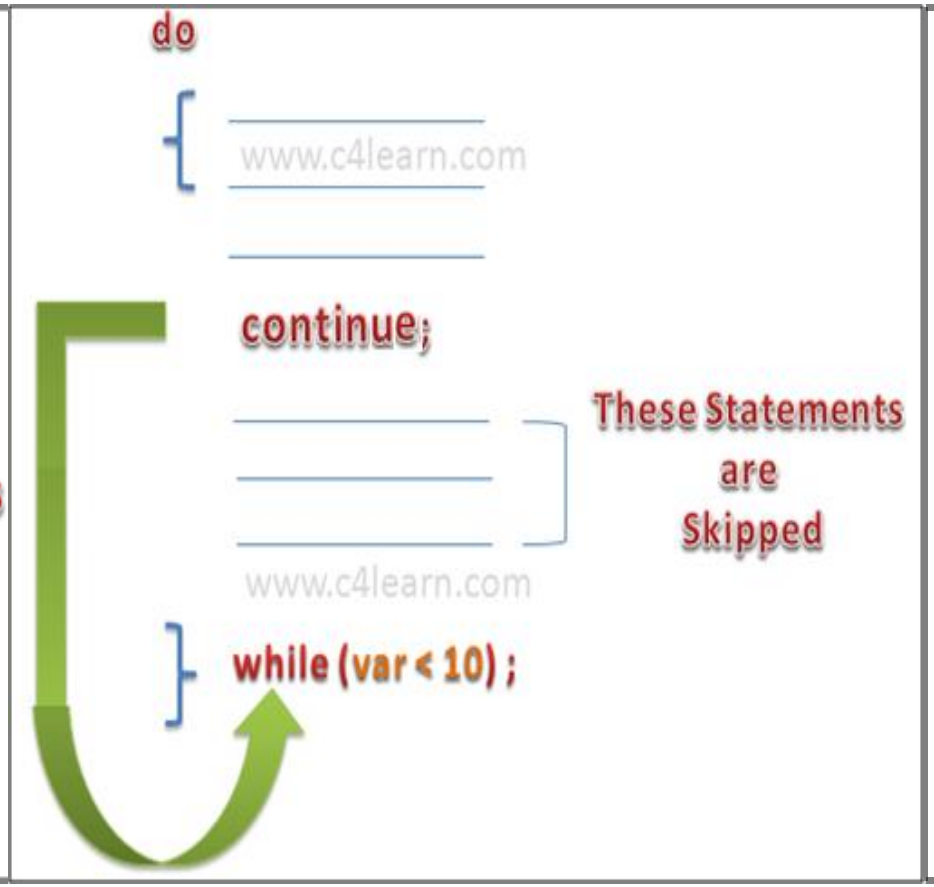
```c
// Using the break statement in a for statement.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   unsigned int x; // counter

   // loop 10 times
   for ( x = 1; x <= 10; ++x ) {

      // if x is 5, terminate loop
      if ( x == 5 ) {
         break; // break loop only if x is 5
      } // end if

      printf( "%u ", x ); // display value of x
   } // end for

   printf( "\nBroke out of loop at x == %u\n", x );
} // end function main
```

Continue statement

- The continue statement, when executed in a while, for or do…while statement, skips the remaining statements in the body of that control statement and performs the next iteration of the loop.
- In while and do…while statements, the loop-continuation test is evaluated immediately after the continue statement is executed. In the for statement, the increment expression is executed, then the loop-continuation test is evaluated.

Continue statement





Continue statement

```c
// Using the continue statement in a for statement.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   unsigned int x; // counter

   // loop 10 times
   for ( x = 1; x <= 10; ++x ) {

      // if x is 5, continue with next iteration of loop
      if ( x == 5 ) {
         continue; // skip remaining code in loop body
      } // end if

      printf( "%u ", x ); // display value of x
   } // end for

   puts( "\nUsed continue to skip printing the value 5" );
} // end function main
```

Continue statement

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

Continue statement

The output of the above program would be...

1 2
2 1

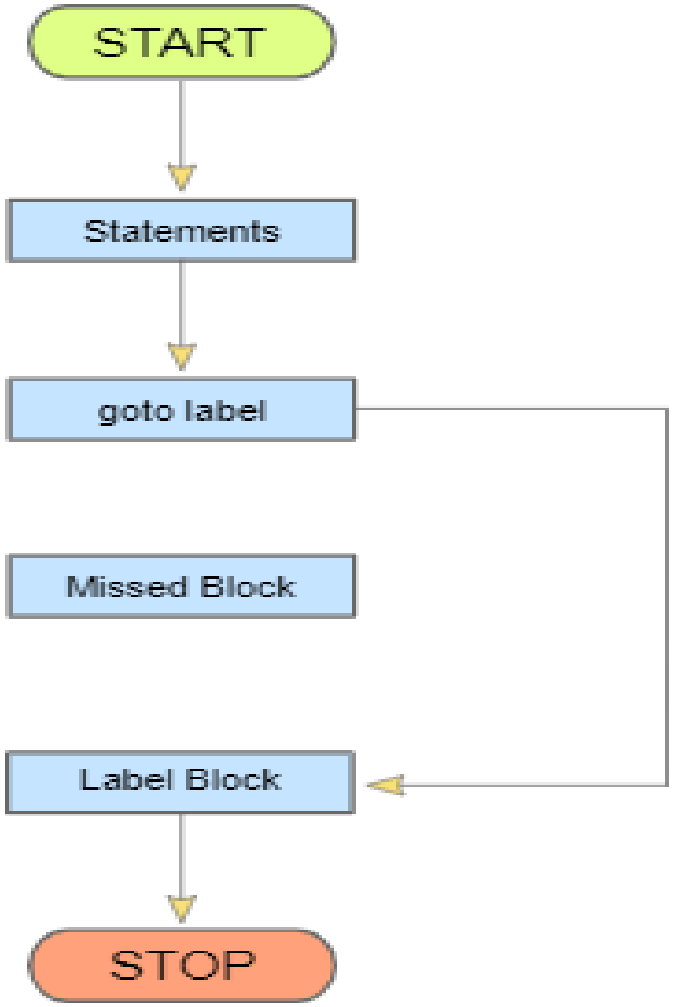
goto statement

- C programming introduces a goto statement by which you can jump directly to a line of code within the same file.
- The goto statement does not require any condition.
- The goto statement simply passes control anywhere in a program without testing any condition.

Syntax

```
goto Label;
.
.
Label:
{
statement n;
}
```

goto statement



goto statement

Example: In the following program we will print whether the number is positive or negative using goto statement.

```
#include <stdio.h>
int main()
{
int num;
printf("Enter a positive or negative integer number:\n ");
scanf("%d",&num);
if(num >= 0)
goto pos;
else
goto neg;
pos:
{
printf("%d is a positive number",num);
return 0;
}
neg:
{
printf("%d is a negative number",num);
return 0;
}
}
```

Nested Looping

- A loop (e.g., **while**, **do-while**, **for**) can be placed within the body of another loop.
- When one loop is nested within another, several iterations of the inner loop are performed for every single iteration of the outer loop.
- The inner and outer loops need not to be generated by the same type of control structures.

While-Nested Loop

```
while (expr1)

{
    :
    while (expr2)
    {
        :
        Update expr2;
    }
:
update expr1;
}
```

```
#include<stdio.h>
int main()
{
 int r,c,s;

 r=1;
 while(r<=5)  /*outer loop*/
 {
  c=1;
  while(c<=2)  /*inner loop*/
  {
   s=r+c;
   printf("r=%d c=%d sum=%d\n",r,c,s);
   c++;
  }
  printf("\n");
  r++; }}
```

```
r=1 c=1 sum=2
r=1 c=2 sum=3

r=2 c=1 sum=3
r=2 c=2 sum=4

r=3 c=1 sum=4
r=3 c=2 sum=5

r=4 c=1 sum=5
r=4 c=2 sum=6

r=5 c=1 sum=6
r=5 c=2 sum=7
```

Do-while-Nested Loop

```
do

{
  :
  do
  {
    :
    Update expr;
  }
    while(expr);
:
update expr;
}
while(expr);
```

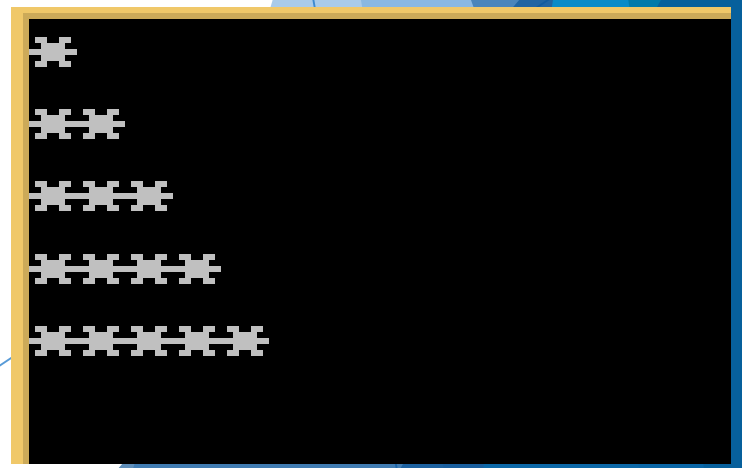
```
*

**

***

****

*****
```

```
#include <stdio.h>
int main()
{
    int i=1,j;
    do
    {
        j=1;
        do
        {
            printf("*");
            j++;
        }while(j <= i);
        i++;
        printf("\n");
    }while(i <= 5);
    return 0;
}
```

For-Nested Loop

```
for ( expr1a; expr2a; expr3a )
{
      :
      for ( expr1b; expr2b; expr3b )
       {
           :
        }
      :
}
```

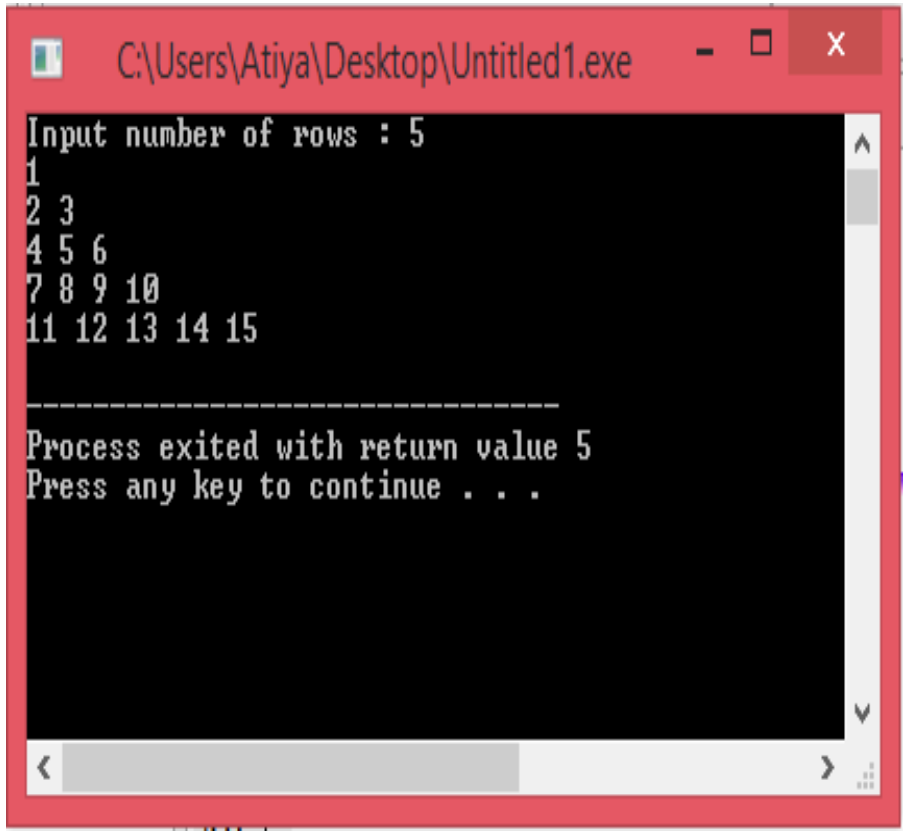
```
#include<stdio.h>
int main()
{
 int i,j;
for (i=1; i<=5;i++)                    //Line 1
{
  for (j=1;j<=i;j++)                   //Line 2
  {
       printf("*");                    //Line 3
   }
printf("\n");                          //Line 4
}
 }
```



For-Nested Loop

Example:

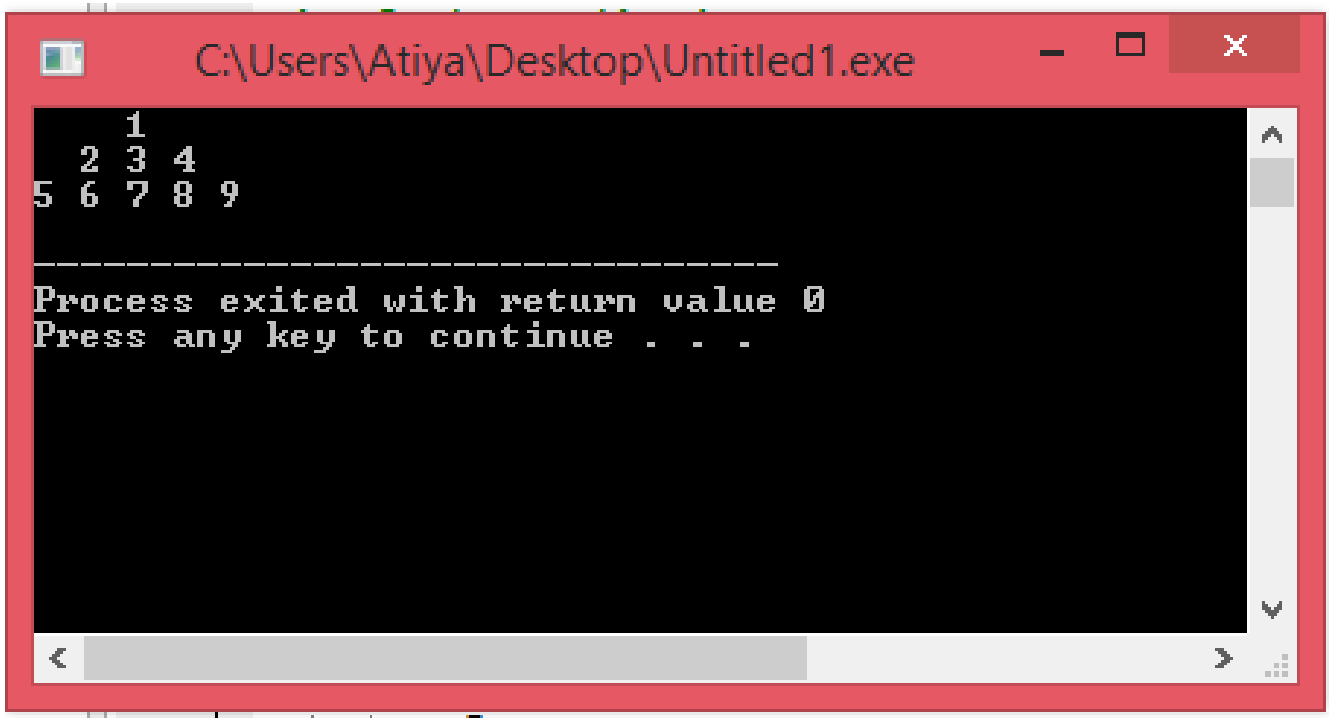
Generate the following pattern using nested loops




```c
#include <stdio.h>
void main()
{
    int i,j,rows,k=1;
    printf("Input number of rows : ");
    scanf("%d",&rows);

        for(i=1;i<=rows;i++)
        {

         for(j=1;j<=i;j++)

            printf("%d ",k++);

         printf("\n");
        }
}
```

TASK:

Generate the following pattern using nested loops



TASK:

```c
#include<stdio.h>
int main()
{
  int i,j,k;
  k=1;
  for(i=1;i<=5;i+=2)
  {
    for(j=5;j>=1;j--)
    {
      if(j>i)
        printf(" ");
      else
        printf("%d ",k++);
    }
    printf("\n");
  }
  return 0;
}
```

TASK:

1. Write a program in C to find the sum of the series 1 +11 + 111 + 1111 + .. n terms.

2. Write a c program to check whether a given number is a perfect number or not.