**Introduction to Control Statement**

In a 'C' program are executed sequentially. This happens when there is no condition around the statements. If you put some conditions for a block of statements the flow of execution might change based on the result evaluated by the condition. This process is referred to as decision making in 'C.' The decision-making statements are also called as control statements.

There are three types of control statements in C:

1. Selection statements
   1. if statement
   2. if else statement
   3. else if statement
   4. Nested if else statement
   5. Switch Case statement
2. Iteration statements
   1. while
   2. for
   3. do while
3. Jump statements
   1. break
   2. continue
   3. return

## 

## 

## 

## Decision making with if statement

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are,

1. Simple if statement
2. if....else statement
3. Nested if....else statement
4. Using else if statement

### **Simple if statement**

The general form of a simple if statement is,



If the *expression* returns true, then the **statement-inside** will be executed, otherwise **statement-inside** is skipped and only the **statement-outside** is executed.

**Example:**



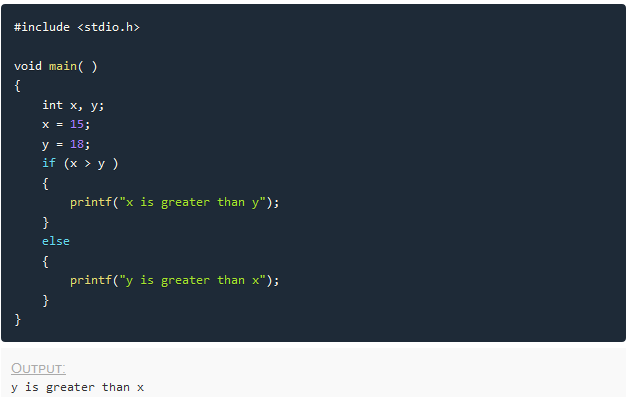
### **if...else statement**

The general form of a simple if...else statement is,



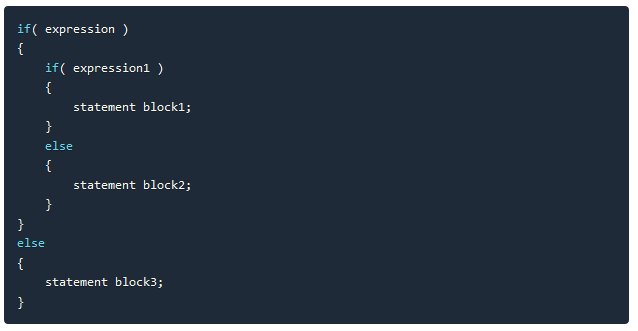
If the *expression* is true, the **statement-block1** is executed, else **statement-block1** is skipped and **statement-block2** is executed.

**Example:**

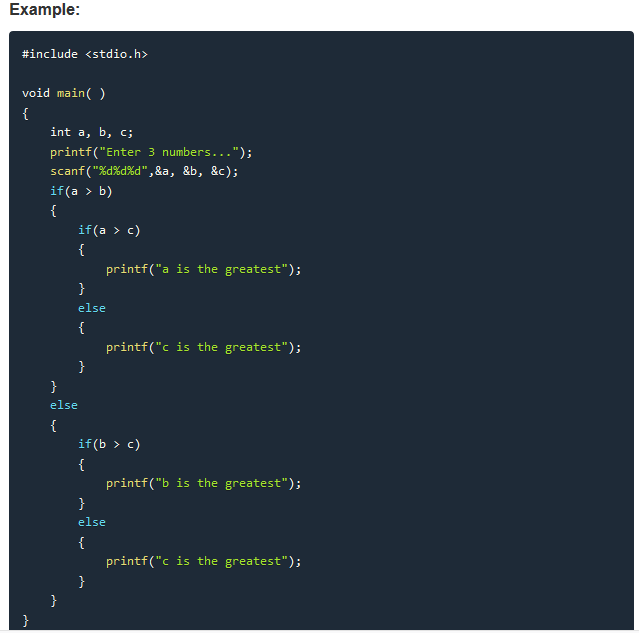


### **Nested if....else statement**

The general form of a nested if...else statement is,



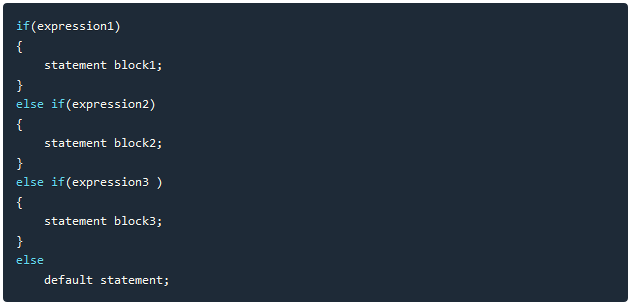
if *expression* is false then **statement-block3** will be executed, otherwise the execution continues and enters inside the first if to perform the check for the next if block, where if *expression 1* is true the **statement-block1** is executed otherwise **statement-block2** is executed.



### 

### **else if ladder**

The general form of else-if ladder is,

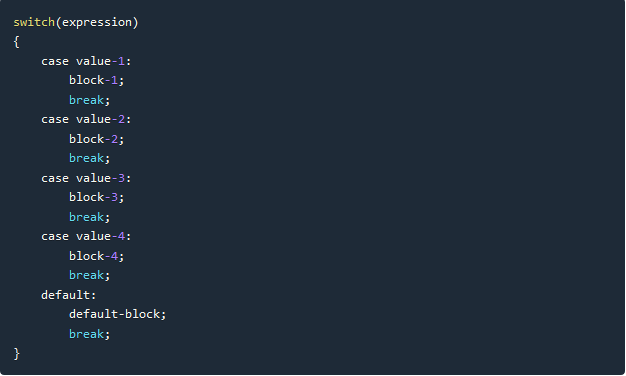


The expression is tested from the top(of the ladder) downwards. As soon as a **true** condition is found, the statement associated with it is executed.



**Switch Statement:**

Switch statement is a control statement that allows us to choose only one choice among the many given choices. The expression in switch evaluates to return an integral value, which is then compared to the values present in different cases. It executes that block of code which matches the case value. If there is no match, then **default** block is executed(if present). The general form of switch statement is,



## **Rules for using switch statement**

1. The expression (after switch keyword) must yield an **integer** value i.e the expression should be an integer or a variable or an expression that evaluates to an integer.
2. The case **label** values must be unique.
3. The case label must end with a colon(:)
4. The next line, after the **case** statement, can be any valid C statement.

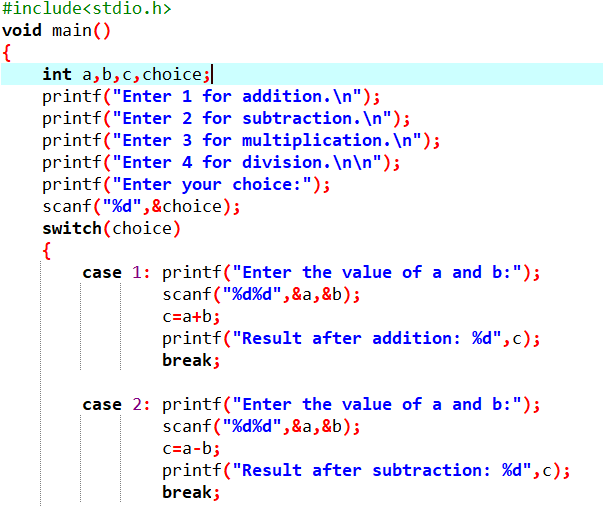
### 

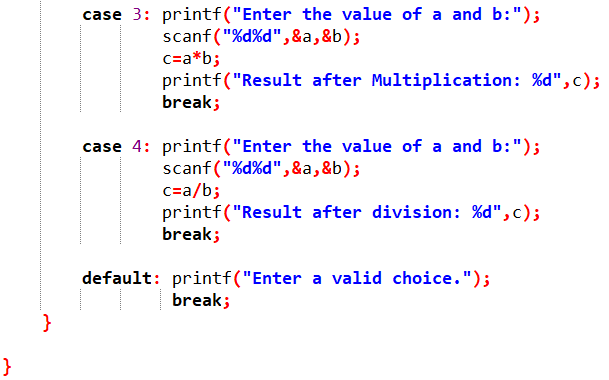
### 

### 

### 

### **Example of switch statement**





**Output:**

Write on your own...

### **Difference between switch and if**

* if statements can evaluate float conditions. switch statements cannot evaluate float conditions.
* if statement can evaluate relational operators. switch statement cannot evaluate relational operators i.e they are not allowed in switch statement.

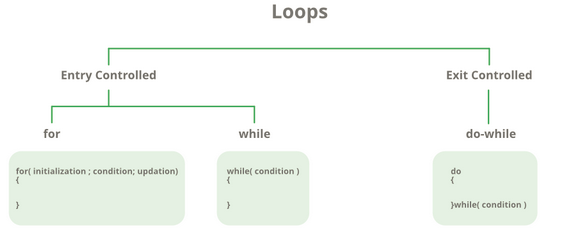
**Iteration Statement / Loop Statement**

Loops in programming come into use when we need to repeatedly execute a block of statements. For example: Suppose we want to print “Hello World” 10 times. This can be done using Loop statement.

In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached.

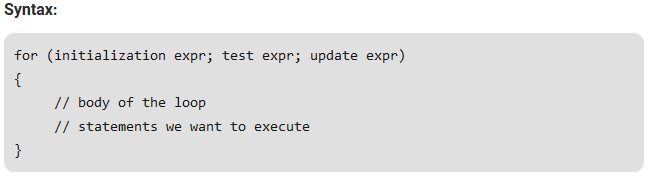
There are mainly two types of loops:

1. **Entry Controlled loops**: In this type of loops the test condition is tested before entering the loop body. **For Loop** and **While Loop** are entry controlled loops.
2. **Exit Controlled Loops**: In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute atleast once, irrespective of whether the test condition is true or false. **do – while loop** is exit controlled loop.



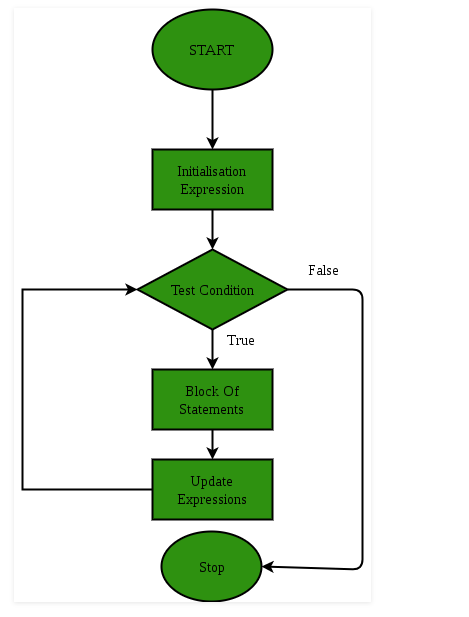
For Loop:

A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.



* **Initialization Expression**: In this expression we have to initialize the loop counter to some value. for example: int i=1;
* **Test Expression**: In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of loop and go to update expression otherwise we will exit from the for loop. For example: i <= 10;
* **Update Expression**: After executing loop body this expression increments/decrements the loop variable by some value. for example: i++;

Equivalent Flow chart of For loop is:



Example:

// C program to illustrate for loop

#include <stdio.h>

int main()

{

int i=0;

for (i = 1; i <= 10; i++)

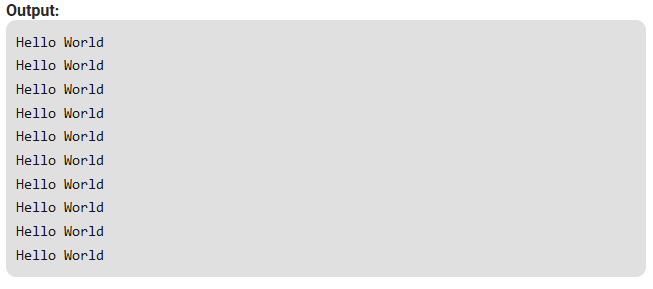
{

printf( "Hello World\n");

}

return 0;

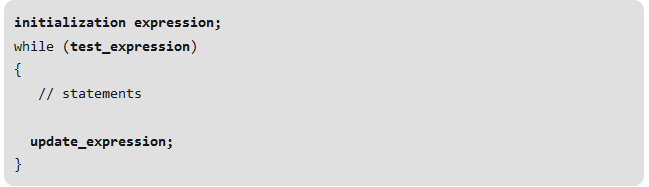
}



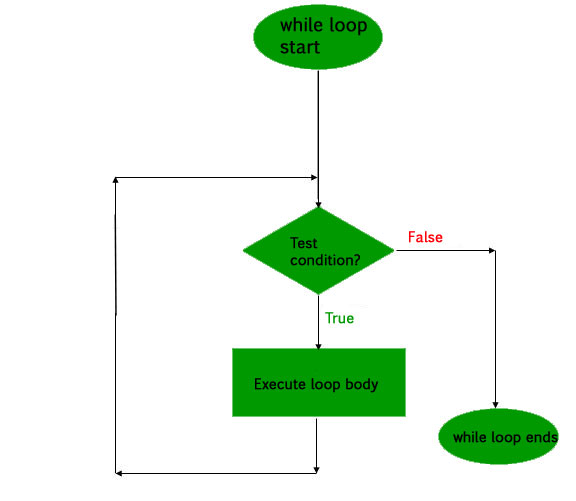
While Loop:

While studying **for loop** we have seen that the number of iterations is known beforehand, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where we do not know the exact number of iterations of loop beforehand. The loop execution is terminated on the basis of test condition.

Syntax:



Flow Diagram:



Example:

// C program to illustrate while loop

#include <stdio.h>

int main()

{

// initialization expression

int i = 1;

// test expression

while (i < 6)

{

printf( "Hello World\n");

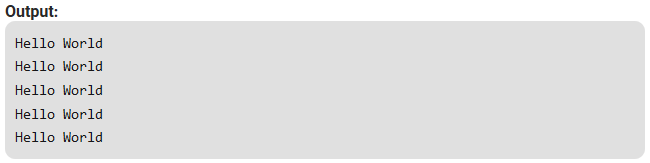
// update expression

i++;

}

return 0;

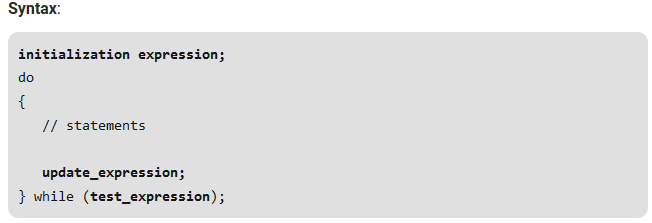
}

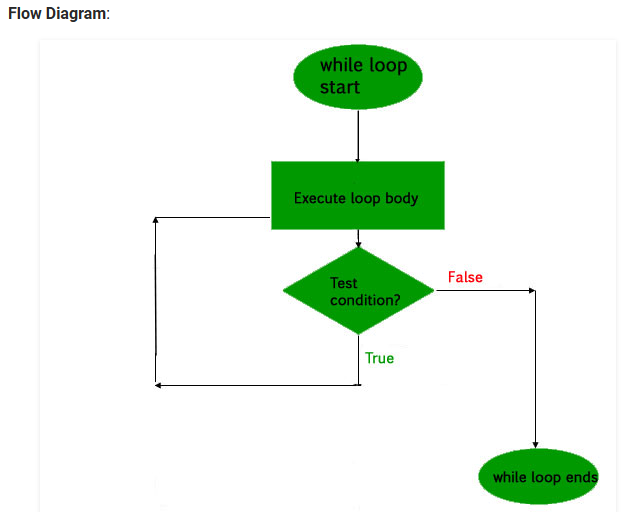


**do while Loop**

In do while loops also the loop execution is terminated on the basis of test condition. The main difference between do while loop and while loop is in do while loop the condition is tested at the end of loop body, i.e do while loop is exit controlled whereas the other two loops are entry controlled loops.

**Note**: In do while loop the loop body will execute at least once irrespective of test condition.





**Example:**

// C program to illustrate do-while loop

#include <stdio.h>

int main()

{

int i = 2;

do

{

printf( "Hello World\n");

i++;

} while (i < 1);

return 0;

}

**Output:**

Hello World

What about an Infinite Loop?

An infinite loop (sometimes called an endless loop ) is a piece of coding that lacks a functional exit so that it repeats indefinitely. An infinite loop occurs when a condition always evaluates to true. Usually, this is an error.

**Exercise: 1**

**C program to generate multiplication table :**

1. #include <stdio.h>
2. int main() {
3. int n, i;
4. printf("Enter an integer: ");
5. scanf("%d", &n);
6. for (i = 1; i <= 10; ++i) {
7. printf("%d \* %d = %d \n", n, i, n \* i);
8. }
9. return 0;
10. }

**Output**

Enter an integer: 9

9 \* 1 = 9

9 \* 2 = 18

9 \* 3 = 27

9 \* 4 = 36

9 \* 5 = 45

9 \* 6 = 54

9 \* 7 = 63

9 \* 8 = 72

9 \* 9 = 81

9 \* 10 = 90

# Nested Loops in C with Examples

**Nested loop** means a [loop statement](https://www.geeksforgeeks.org/loops-in-c-and-cpp/) inside another loop statement. That is why nested loops are also called as “**loop inside loop**“.

**Syntax for Nested For loop:**

for ( initialization; condition; increment ) {

for ( initialization; condition; increment ) {

// statement of inside loop

}

// statement of outer loop

}

**Syntax for Nested While loop:**

while(condition) {

while(condition) {

// statement of inside loop

}

// statement of outer loop

}

**Syntax for Nested Do-While loop:**

do{

do{

// statement of inside loop

}while(condition);

// statement of outer loop

}while(condition);

***Note:*** *There is no rule that a loop must be nested inside its own type. In fact, there can be any type of loop nested inside any type and to any level.*

**Syntax:**

do{

while(condition) {

for ( initialization; condition; increment ) {

// statement of inside for loop

}

// statement of inside while loop

}

// statement of outer do-while loop

}while(condition);

### **Program to print half pyramid using \***

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

Source Code

1. #include<stdio.h>
2. int main() {
3. int i, j, rows;
4. printf("Enter number of rows: ");
5. scanf("%d", &rows);
6. for (i=1; i<=rows; ++i) {
7. for (j=1; j<=i; ++j)
8. { printf("\* "); }
9. printf("\n");
10. }
11. return 0;
12. }

### **Program to print half pyramid a using numbers**

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

Source Code

1. #include<stdio.h>
2. int main() {
3. int i,j,rows;
4. printf("Enter number of rows: ");
5. scanf("%d", &rows);
6. for (i=1; i<=rows; ++i) {
7. for (j=1; j<=i; ++j)
8. { printf("%d ",j); }
9. printf("\n");
10. }
11. return 0;
12. }

### **Inverted half pyramid using \***

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

Source Code

1. #include<stdio.h>
2. int main() {
3. int i, j, rows;
4. printf("Enter number of rows: ");
5. scanf("%d", &rows);
6. for (i=rows; i>=1; --i) {
7. for (j=1; j<=i; ++j)
8. { printf("\* "); }
9. printf("\n");
10. }
11. return 0;
12. }

### **Inverted half pyramid using numbers**

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

Source Code

1. #include<stdio.h>
2. int main() {
3. int i ,j, rows;
4. printf("Enter number of rows: ");
5. scanf("%d", &rows);
6. for (i=rows; i>=1; --i) {
7. for (j=1; j<=i; ++j)
8. { printf("%d ",j); }
9. printf("\n");
10. }
11. return 0;
12. }

### **Print Floyd's Triangle.**

1

2 3

4 5 6

7 8 9 10

Source Code

1. #include<stdio.h>
2. int main() {
3. int rows, i, j, number= 1;
4. printf("Enter number of rows: ");
5. scanf("%d", &rows);
6. for (i=1; i<=rows; i++) {
7. for (j=1; j<=i; ++j)
8. { printf("%d ", number);
9. ++number;
10. }
11. printf("\n");
12. }
13. return 0;
14. }

**C program to generate pyramid:**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

#include <stdio.h>

int main()

{

int i,j,k,rows,space=4;

printf("Enter the number of rows:");

scanf("%d",&rows);

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

{

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

{

printf(" ");

}

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

{

printf("\* ");

}

printf("\n");

space--;

}

}

**C program to check whether a number is prime or not:**

1. **#include <stdio.h>**
2. **int main() {**
3. **int n, i, flag = 0;**
4. **printf("Enter a positive integer: ");**
5. **scanf("%d", &n);**
6. **for (i = 2; i <= n / 2; ++i) {**
7. **// condition for non-prime**
8. **if (n % i == 0) {**
9. **flag = 1;**
10. **break;**
11. **}**
12. **}**
13. **if (n == 1) {**
14. **printf("1 is neither prime nor composite.");**
15. **}**
16. **else {**
17. **if (flag == 0)**
18. **printf("%d is a prime number.", n);**
19. **else**
20. **printf("%d is not a prime number.", n);**
21. **}**
22. **return 0;**
23. **}**

## **Program to Check Palindrome**

1. **#include <stdio.h>**
2. **int main() {**
3. **int n, reversedN = 0, remainder, originalN;**
4. **printf("Enter an integer: ");**
5. **scanf("%d", &n);**
6. **originalN = n;**
7. **// reversed integer is stored in reversedN**
8. **while (n != 0) {**
9. **remainder = n % 10;**
10. **reversedN = reversedN \* 10 + remainder;**
11. **n /= 10;**
12. **}**
13. **// palindrome if orignalN and reversedN are equal**
14. **if (originalN == reversedN)**
15. **printf("%d is a palindrome.", originalN);**
16. **else**
17. **printf("%d is not a palindrome.", originalN);**
18. **return 0;**
19. **}**

**Output**

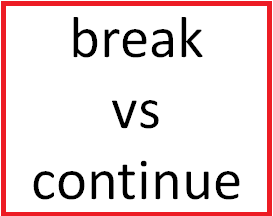
**Enter an integer: 1001**

**1001 is a palindrome.**

**break and continue statement**

The break statement is used in switch or loops and continue statement is used only in loops. When break statement is encountered it immediately stops the switch or loop execution. When continue statement is encountered, all the statements next to it are skipped and the loop control goes to next iteration. Generally we use break and continue with some condition.

Below I have shared difference between break and continue statements along with an example in C.



## **Difference Between break a5knd continue in C**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **break** | **continue** |
| 1. | break statement is used in switch and loops. | continue statement is used in loops only. |
| 2. | When break is encountered the switch or loop execution is immediately stopped. | When continue is encountered, the statements after it are skipped and the loop control jump to next iteration. |
| 3. | **Example:**  #include<stdio.h>  int main(){  int i;  for(i=0;i<5;++i){  if(i==3)  break;  printf(“%d “,i);  }  return 0;  }    **Output:**  0 1 2 | **Example:**  #include<stdio.h>  int main(){  int i;  for(i=0;i<5;++i){  if(i==3)  continue;  printf(“%d “,i);  }  return 0;  }    **Output:**  0 1 2 4 |

**Goto Statement in C**

goto is a jumping statement in c language, which transfer the program’s control from one statement to another statement (where label is defined).

goto can transfer the program’s within the same block and there must a label, where you want to transfer program’s control.

## Syntax

The syntax for a goto statement in C is as follows −

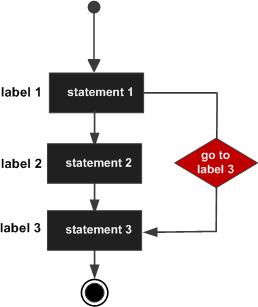
goto label;

….

label: statement;

Here label can be any plain text except C keyword and it can be set anywhere in the C program above or below to goto statement.

## Flow Diagram



## Example

/\*To print numbers from 1 to 10 using goto statement\*/

#include <stdio.h>

**int** main()

{

**int** number;

number=1;

repeat:

printf("%d\n",number);

number++;

**if**(number<=10)

goto repeat;

**return** 0;

}

**Output:**

Write on your own...