

Control Statements

Chapter 4

Control Statements

C provides two types of Control Statements

- Branching Structure
- Looping Structure

Branching Structure

Branching is deciding what actions to take and looping is deciding how many times to take a certain action. Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

One of the common way to perform branching is using If.... else statements in C programming language.

C if else Statement

The syntax of the `if ... else` statement in C programming is:

```
if(test_expression)
{
    // run code if test expression is true
}
Else
{
    //run code if test expression if false
}
```

Nested if else statement

When an if else statement is present inside the body of another “if” or “else” then this is called nested if else.

Syntax of Nested if else statement:

```
if(condition) {  
    //Nested if else inside the body of "if"  
    if(condition2) {  
        //Statements inside the body of nested "if"  
    }  
    else {  
        //Statements inside the body of nested "else"  
    }  
}  
else {  
    //Statements inside the body of "else"  
}
```

C – else..if statement

The else..if statement is useful when you need to check multiple conditions within the program, nesting of if-else blocks can be avoided using else..if statement.

```
if (condition1)
{
    //These statements would execute if the condition1 is true
}

else if(condition2)
{
    //These statements would execute if the condition2 is true
}

.
.

else
{
    //These statements would execute if all the conditions return false.
```

Problems

- WAP to display the greater of two numbers.
- WAP to check whether a person is eligible to vote or not
- WAP to check whether a number is odd or even.
- WAP to display whether the two variables entered are equal or not. If not equal create a nested if to display if the variables are greater than or less than other variable.
- WAP to check whether two numbers are equal, greater or lesser than each other.

```
// Program to relate two integers using =, > or < symbol
```

```
#include <stdio.h>
```

```
int main() {
```

```
    int number1, number2;
```

```
    printf("Enter two integers: ");
```

```
    scanf("%d %d", &number1, &number2);
```

```
    //checks if the two integers are equal.
```

```
    if(number1 == number2) {
```

```
        printf("Result: %d = %d",number1,number2);    }
```

```
    //checks if number1 is greater than number2.
```

```
    else if (number1 > number2) {
```

```
        printf("Result: %d > %d", number1, number2);    }
```

```
    //checks if both test expressions are false
```

```
    else {
```

```
        printf("Result: %d < %d",number1, number2);    }
```

```
    return 0;
```

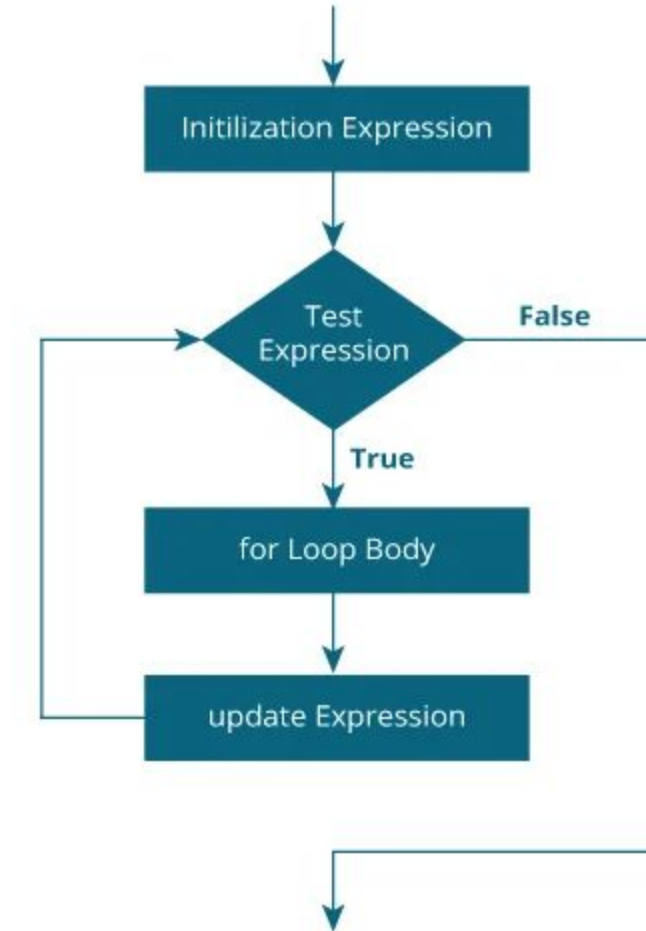
Loops

In programming, a loop is used to repeat a block of code until the specified condition is met.

C programming has three types of loops:

1. for loop
2. while loop
3. do...while loop

For Loop



For loop syntax

```
for (initializationStatement; testExpression; updateStatement)
{
    // statements inside the body of loop
}
```

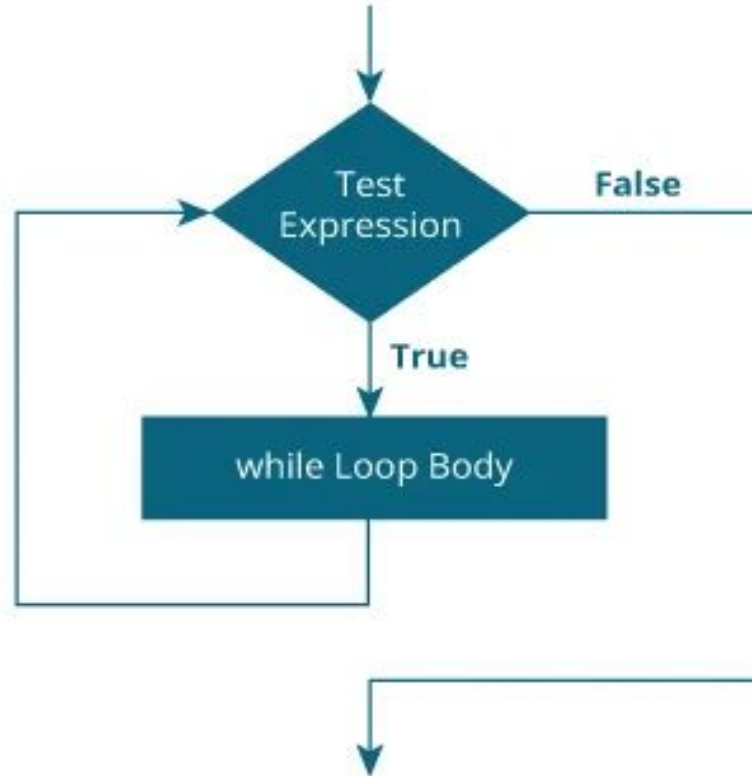
```
for (i=1;i<10;i++) // for syntax
```

```
{
    printf("%d ",i);           // for body      1 2 3 4 5 6 7 8 9
}
```

While loop

The syntax of the `while` loop is:

```
while (testExpression) {  
    // the body of the loop  
}
```



```
// Print numbers from 1 to 5
```

```
#include <stdio.h>
```

```
int main() {
```

```
    int i = 1; // Initial Statement
```

```
    while (i <= 5) {
```

```
        printf("%d\n", i);
```

```
        ++i;
```

```
    }
```

```
    return 0;
```

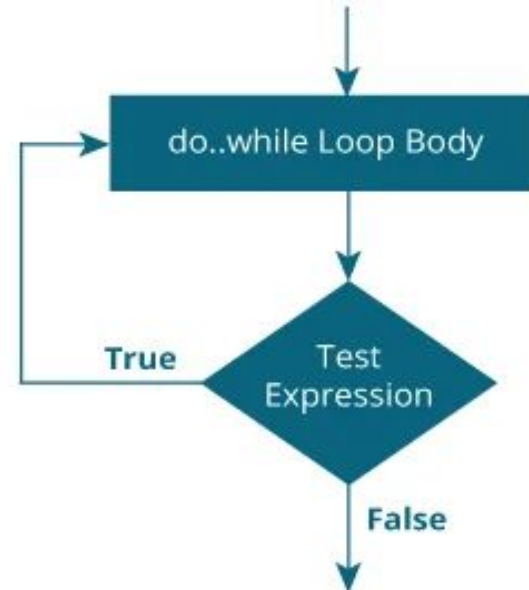
```
}
```

do while loop

The `do..while` loop is similar to the `while` loop with one important difference. The body of `do..while` loop is executed at least once. Only then, the test expression is evaluated.

The syntax of the `do...while` loop is:

```
do {  
    // the body of the loop  
}  
while (testExpression);
```



Problems

- WAP to display “Programming is easy 10 times”.
- WAP to display the first 10 natural numbers 1 2 3 4 5 6...10
- WAP to print n natural numbers.
- WAP to calculate sum of n natural numbers.
- WAP to display the numbers 1, 3, 5, 7, 9
- WAP to display the first 5 numbers of multiple of 5.
- WAP to display the series 1, 4,9,16,25,36,49,64,81.
- WAP to display the fibonacci series.

0 1 1 2 3 5 8 13 21 34.....

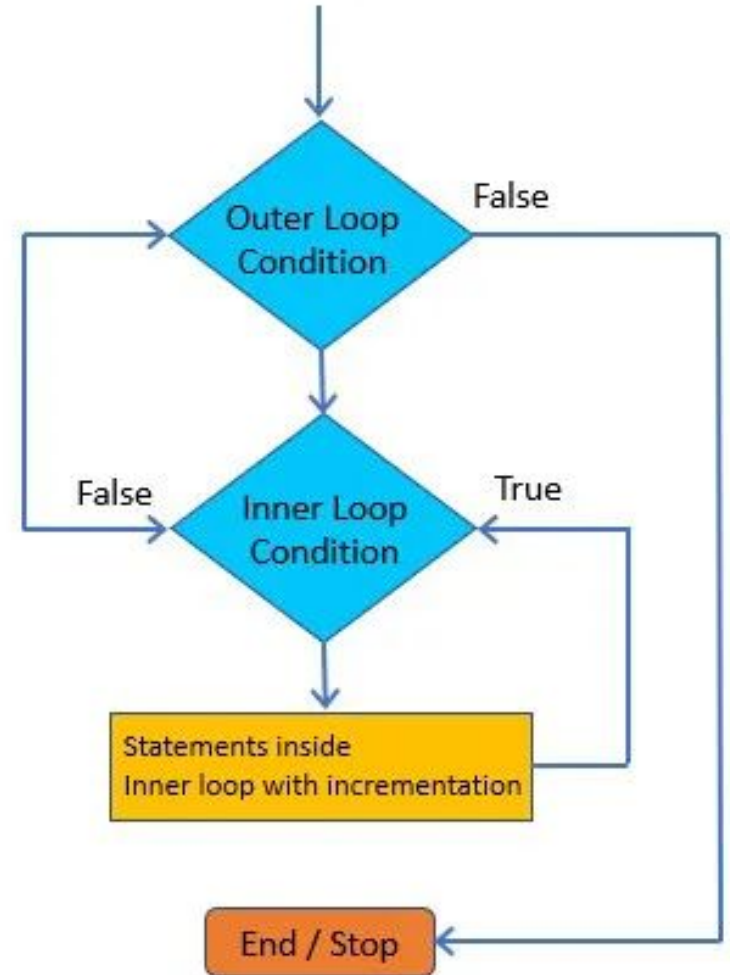
Problems

- WAP to find the reverse of a number.
- WAP to display whether the year is leap year or not.
- WAP to create a arithmetic series and sum with $a=2$ and $c.d=3$ upto 5 terms.
- WAP to create a geometric series and sum with $a=2$ and $c.r=3$ upto 5 terms.
- WAP to find HCF and LCM of two numbers.
- WAP to check a number is prime or not.
- WAP to check the palindrome number. 1001
- WAP to check the armstrong number. $153 = 1^3 + 5^3 + 3^3 = 153$

Nested Loop

A loop inside a loop is called Nested Loop.

```
Outside_loop
{
    //Outside Loop Statements
    Inside_loop
    {
        //Inside loop Statements
    }
}
```



Example

```
for(i=1;i<10;i++)  
{  
    for(j=1;j<10;j++)  
    {  
        printf("%d\t",j);  
    }  
    printf("\n");  
}
```

Problems

Create a pattern:

*	1					*	*	*	*	*	1	2	3	4	5
* *	1	2				*	*	*	*		1	2	3	4	
* * *	1	2	3			*	*	*			1	2	3		
* * * *	1	2	3	4		*	*				1	2			
* * * * *	1	2	3	4	5	*					1				

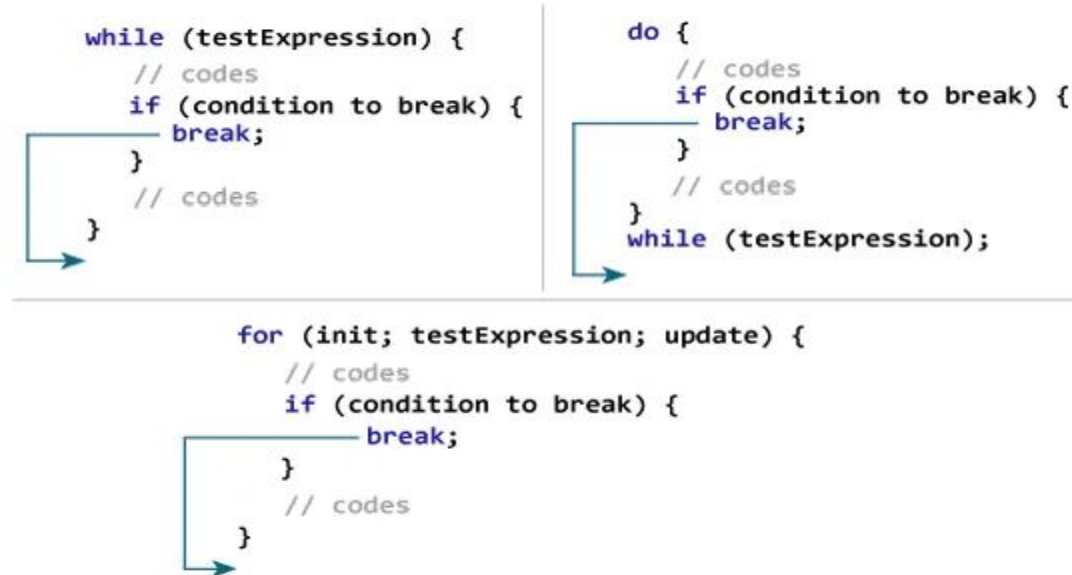
1				
2	3			
4	5	6		
7	8	9	10	

Break and continue statement

The break statement ends the loop immediately when it is encountered. Its syntax is:

```
break;
```

It can be used with either loops or switch statement.



// Program to calculate the sum of numbers (10 numbers max) // If the user enters a negative number, the loop terminates

```
#include <stdio.h>
```

```
int main() {
```

```
    int i;
```

```
    double number, sum = 0.0;
```

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

```
        printf("Enter n%d: ", i);
```

```
        scanf("%lf", &number);
```

```
        // if the user enters a negative number, break the loop
```

```
        if (number < 0.0) {
```

```
            break;    }
```

```
        sum += number; // sum = sum + number;    }
```

```
    printf("Sum = %.2lf", sum);
```

```
    return 0; }
```

C continue statement

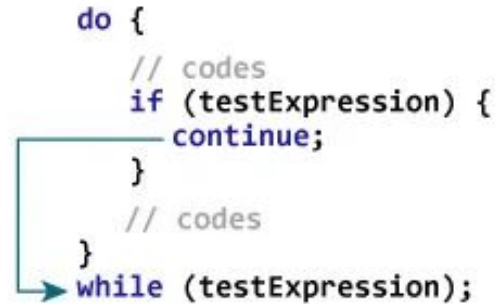
The `continue` statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

`continue;`



```
while (testExpression) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```

A diagram shows a green arrow starting from the `continue;` line, going up and then left to the opening curly brace of the `while` loop, indicating a jump to the start of the next iteration.



```
do {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
} while (testExpression);
```

A diagram shows a green arrow starting from the `continue;` line, going up and then left to the opening curly brace of the `do` block, indicating a jump to the start of the next iteration.



```
for (init; testExpression; update) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```

A diagram shows a green arrow starting from the `continue;` line, going up and then left to the opening curly brace of the `for` loop, indicating a jump to the start of the next iteration.

```
// Program to calculate the sum of numbers (10 numbers max)  
// If the user enters a negative number, it's not added to the result
```

```
#include <stdio.h>
```

```
int main() {
```

```
    int i;
```

```
    double number, sum = 0.0;
```

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

```
        printf("Enter a n%d: ", i);
```

```
        scanf("%lf", &number);
```

```
        if (number < 0.0) {
```

```
            continue;    }
```

```
        sum += number; // sum = sum + number;    }
```

```
    printf("Sum = %.2lf", sum);
```

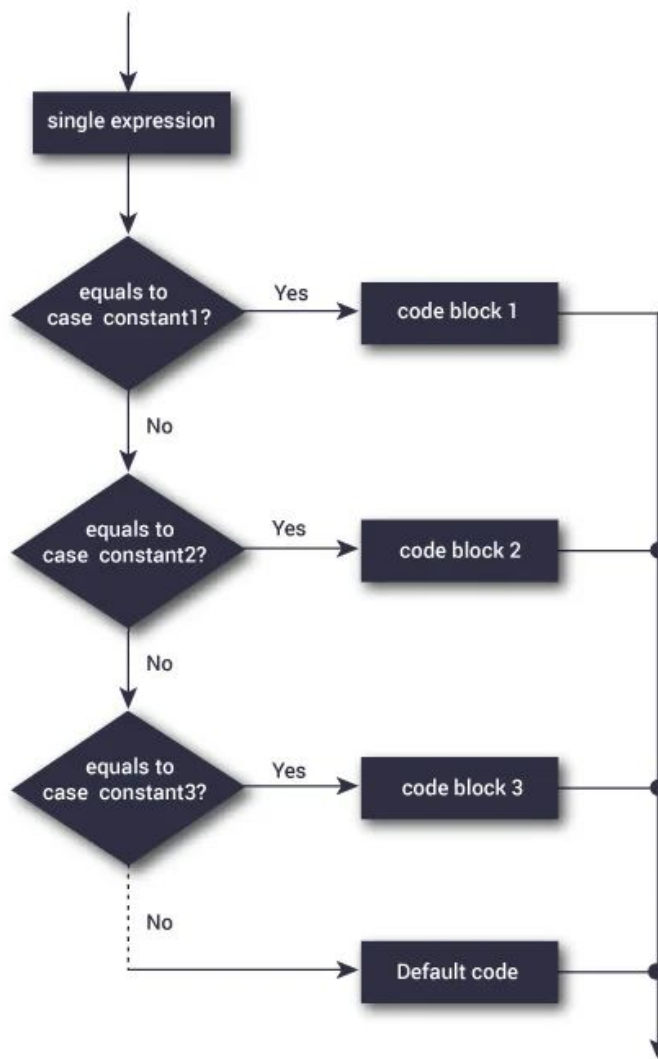
```
    return 0;    }
```

Switch case

The switch statement allows us to execute one code block among many alternatives.

You can do the same thing with the `if...else..if` ladder. However, the syntax of the `switch` statement is much easier to read and write.

```
switch (expression)
{
    case constant1:
        // statements
        break;
    case constant2:
        // statements
        break;
    .
    .
    .
    default:
        // default statements
}
```



goto statement

The `goto` statement allows us to transfer control of the program to the specified `label`.

```
goto label;
```

```
... ..
```

```
... ..
```

```
label:
```

```
    Statement;
```

The `label` is an identifier. When the `goto` statement is encountered, the control of the program jumps to `label` and starts executing the code.

The use of `goto` statement may lead to code that is buggy and hard to follow.

// Program to calculate the sum and average of positive numbers // If the user enters a negative number, the sum and average are displayed.

```
#include <stdio.h>
```

```
int main() {
```

```
    const int maxInput = 100;
```

```
    int i;
```

```
    double number, average, sum = 0.0;
```

```
    for (i = 1; i <= maxInput; ++i) {
```

```
        printf("%d. Enter a number: ", i);
```

```
        scanf("%lf", &number);
```

```
        // go to jump if the user enters a negative number
```

```
        if (number < 0.0) {
```

```
            goto jump;    }
```

```
        sum += number; }
```

```
jump:
```

```
    average = sum / (i - 1);
```

```
    printf("Sum = %.2f\n", sum);
```

```
    printf("Average = %.2f", average);
```

```
    return 0; }
```

Additional Information

The ? : Operator

Exp1? Exp2 : Exp3 is the general structure of the ? : operator.

The value of a ? expression is determined like this:

- Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.
- If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.