UNIT 2

Operators and Expressions: Arithmetic operators, Unary operators, Relational and Logical Operators, Assignment Operators, Conditional Operators. Expressions, Precedence and Associativity, Evaluating Expressions, Type Conversion.

Selection-Making Decision: Two Way Selection, Multiway Selection, Standard Character Functions. Repetition: Concepts of a loop, Pretest and Post-test Loops, Initialization and Updating,

Loops in C: The while loop, the do While loop, the for loop, the break and continue statement, the goto statement.

OPERATORS AND EXPRESSIONS

- C supports a rich set of operators. Operators are used in programs to manipulate data and variables.
- They usually form a part of the mathematical of logical expressions.

CATEGORIES OF OPERATORS

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and Decrement operators
- Conditional operators
- Bitwise operators
- Special operators

ARITHMETIC OPERATORS

- The operators are
- + (Addition)
- - (Subtraction)
- * (Multiplication)
- / (Division)
- % (Modulo division)
- The Division operator produces the quotient.
- The modulo division produces the remainder of an integer division.
- The modulo division operator cannot be used on floating point data.
- Note: C does not have any operator for exponentiation

INTEGER ARITHMETIC

- When both the operands in a single arithmetic expression are integers, the expression is called an integer expression, and the operation is called integer arithmetic.
- If a=14 and b=4, then
- a-b=10
- a+b=18
- a*b=56
- a/b=3
- a%b=2

INTEGER ARITHMETIC

- ■ What are the results of the following?
- ■ 6/7=?
- ■ 3/7=?
- ■ 21/3=?
- ■ During modulo division the sign of the result is always the sign of the first operand.
- ■ -14 % 3 = -2
- ■ -14 % -3 = -2
- ■ 14 % -3 = 2

REAL ARITHMETIC

An arithmetic operation involving only real operands is called real arithmetic.

- If x and y are floating then we will have
- 1) x = 6.0 / 7.0 = 0.857143
- 2) y = 1.0 / 3.0 = 0.333333
- The operator % cannot be used with real

operands.

MIXED-MODE ARITHMETIC

- ■ When one of the operands is real and the other is integer, the expression is called a mixed mode arithmetic expression and its result is always a real number.
- ■ Eg: 1) 15 / 10.0 = 1.5

RELATIONAL OPERATORS

- Comparisons can be done with the help of relational operators.
- The expression containing a relational operator is termed as a relational expression.
- The value of a relational expression is either one or zero

RELATIONAL OPERATORS

- 1) < (is less than)
- 2) <= (is less than or equal to)
- 3) > (is greater than)
- 4) >= (is greater than or equal to)
- 5) = = (is equal to)
- 6) != (is not equal to)

LOGICAL OPERATORS

- Following are three logical operators.
- && (logical AND)
- || (logical OR)
- ! (logical NOT)
- Eg:
- 1) if(age>55 && sal<1000)
- 2) if(number<0 || number>100)

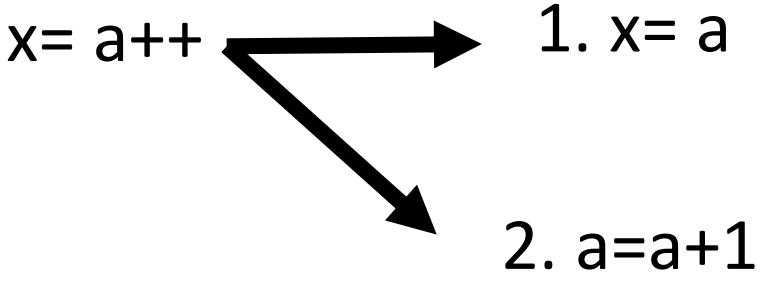
ASSIGNMENT OPERATORS

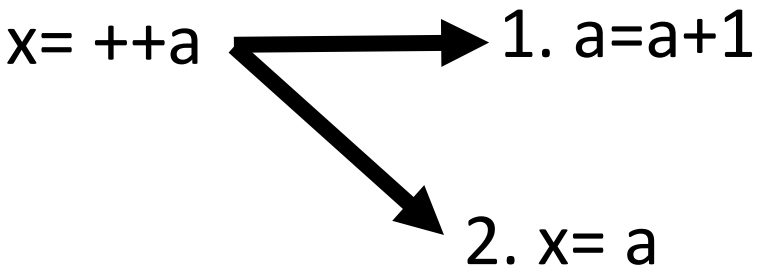
- The usual assignment operator is ‘=’.
- In addition, C has a set of ‘shorthand’ assignment operators.
- Eg: x += y+1;
- This is same as the statement x=x+(y+1);
- a + =1 → a = a + 1
- a - = 1 →a = a – 1
- a *= n + 1 → a = a * (n+1)
- a /= n + 1 → a = a / (n+1)
- a %= b → a = a % b

INCREMENT AND DECREMENT OPERATORS

- These are the increment and decrement operator:
- ++ and --
- The operator ++ adds 1 to the operands while -- subtracts 1.
- It takes the following form:
- ++m; or m++
- --m; or m—
- x= 4++; // gives error, because 4 is constant

INCREMENT AND DECREMENT OPERATORS





INCREMENT AND DECREMENT OPERATORS-EXAMPLE

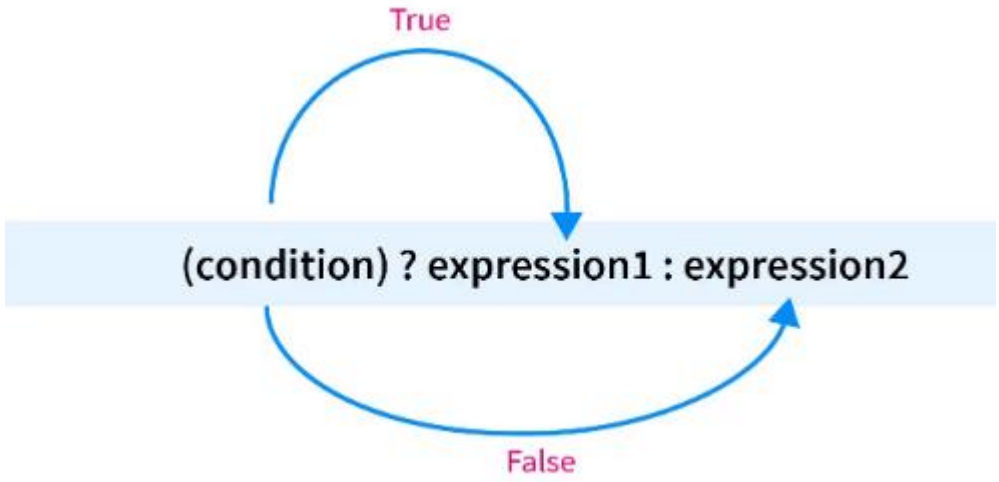
```
void main()
{
        int x,i; i=10;
        x=i++;
        printf("x: %d",x);
        printf("i: %d",i);
}
```

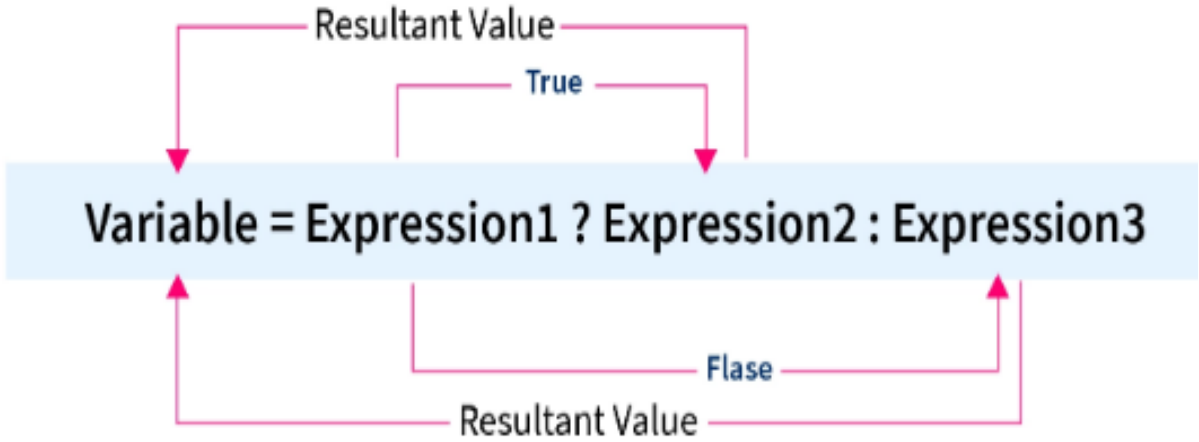
```
void main()
{
int x,i; i=10;
x=++i;
printf("x: %d",x);
printf("i: %d",i);
}
```

```
void main()
{
int x,i; i=10;
x=++i;
printf("x: %d",x);
printf("i: %d",i);
}
```

CONDITIONAL OPERATORS

- The conditional operator takes three operands, so it is a ternary operator. The conditional operator is the only ternary operator available in the C programming language, so the names ternary operator and conditional operator are used alternatively to mean the conditional operator.





BITWISE OPERATORS

Symbol	Operator
&	bitwise AND
|	bitwise inclusive OR
^	bitwise XOR (exclusive OR)
<<	left shift
>>	right shift
~	bitwise NOT (one's complement) (unary)

bit a	bit b	a & b (a AND b)
0	0	0
0	1	0
1	0	0
1	1	1

```
  11001000
& 10111000
  --------
= 10001000
```

bit a	bit b	a | b (a OR b)
0	0	0
0	1	1
1	0	1
1	1	1

```
  11001000
| 10111000
  --------
= 11111000
```

bit a	bit b	a ^ b (a XOR b)
0	0	0
0	1	1
1	0	1
1	1	0

```
  11001000
^ 10111000
  --------
= 01110000
```

UNARY OPERATORS

1. sizeof :

   sizeof(int)

2. Unary Plus/Minus:

   +3, -3

3. Cast Operator:

   float(x)

```
#include <stdio.h>
int main()
{
        int a;
        float b;
        char c;
        double d;
        short e;
        long int f;
        long long int g;
        long double h;
        printf("%d\n",sizeof(a));
        printf("%d\n",sizeof(b));
        printf("%d\n",sizeof(c));
        printf("%d\n",sizeof(d));
        printf("%d\n",sizeof(e));
        printf("%d\n",sizeof(f));
        printf("%d\n",sizeof(g));
        printf("%d",sizeof(h));
        return 0;
}
```

EXPRESSIONS

An **expression** is a sequence of operands and operators that reduces to a single value. It can be simple or compound.

Simple Expression contains only one operator. Ex: 2+3

Complex Expression contains more than one operator

ex: 2+3+6*5

The order in which the operators in complex expressions are evaluated is determined by a set of priorities known as **precedence**.

If two operators with the same precedence occur in a complex expression, another attribute that takes control is **associativity.**

EXPRESSIONS (CONTI.)

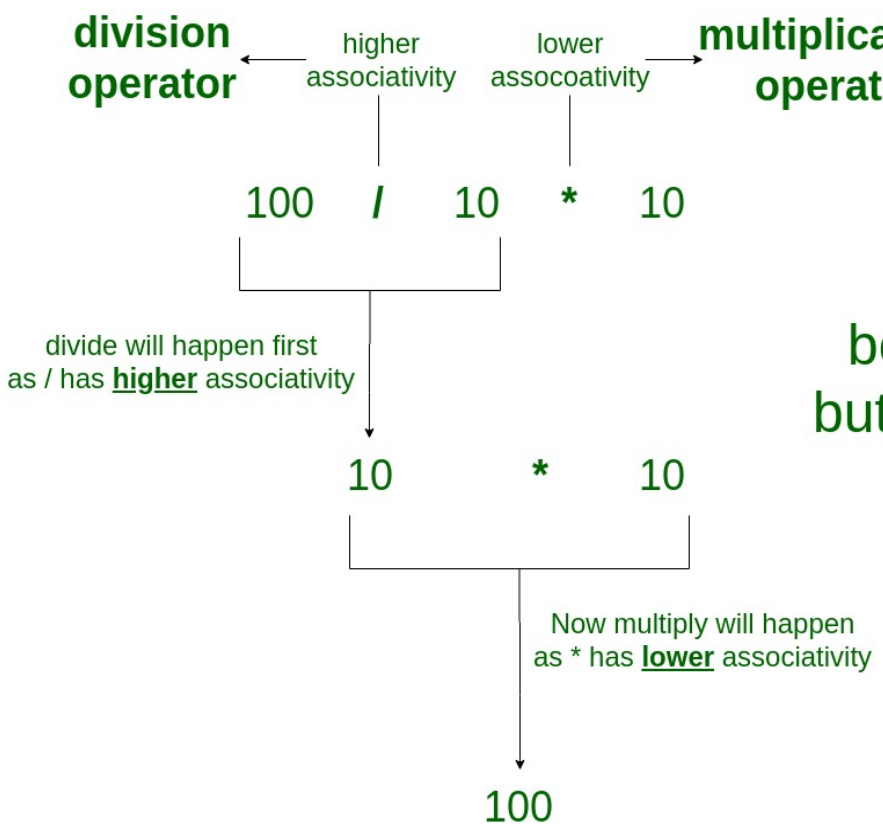
- An expression always reduces to a single value.
- **Simple expressions** are divided into six categories:

  - Primary
  - Postfix
  - Prefix
  - Unary
  - Binary
  - Ternary

Precedence and Associativity

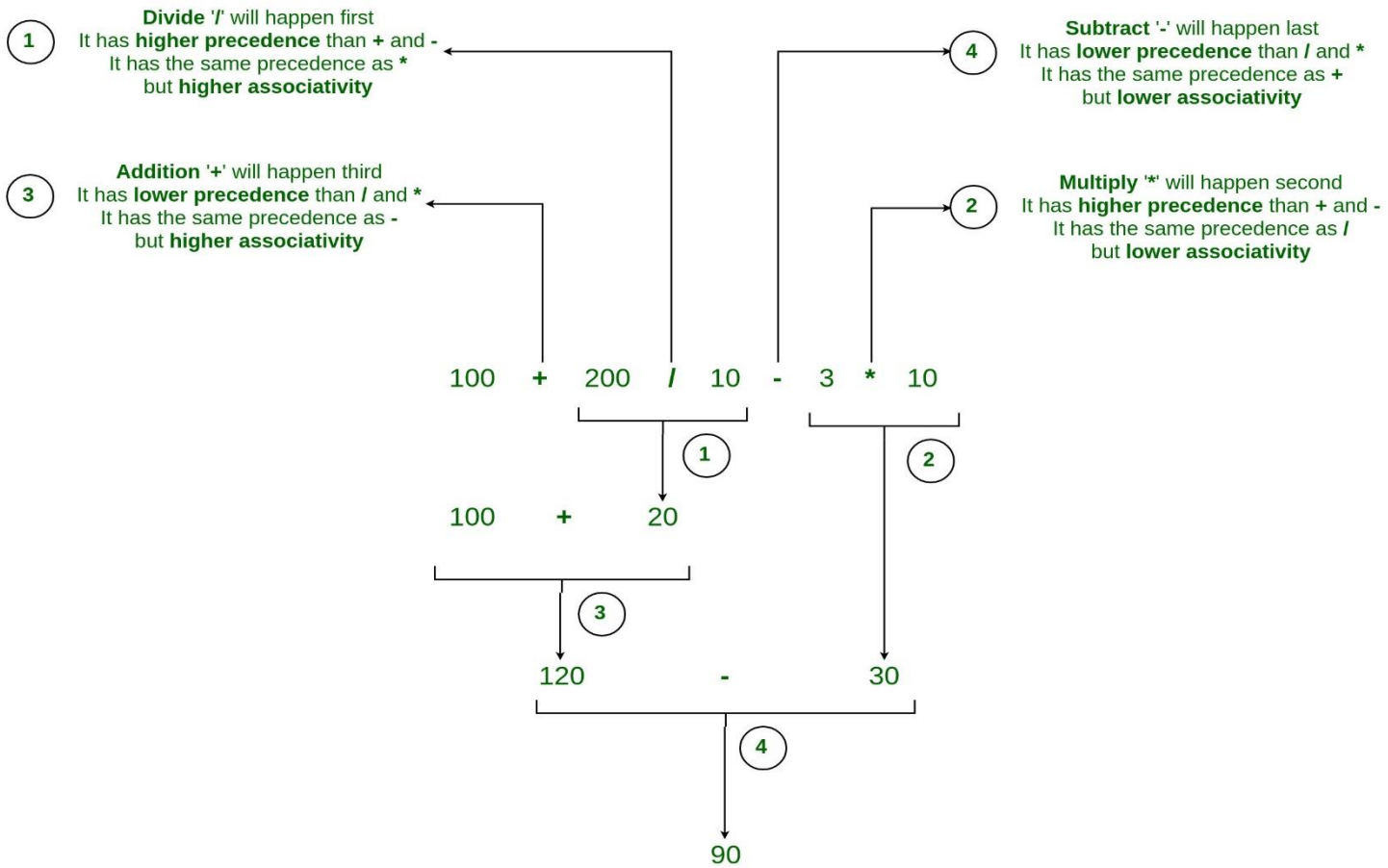
OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

Operator Associativity



/ and ***
both have the same precedence
but Left to Right (**LTR**) associativity

Operator Precedence and Associativity



/ and *****
both have the same precedence
but Left to Right (**LTR**) associativity

+ and **-**
both have the same precedence
but Left to Right (**LTR**) associativity

/ and *****
have the higher precedence
than **+** and **-**

Evaluate an Expression

Let a=3 , b=4, c=5 , then Solve:

x= a* 4 + b / 2 – c * b

y = --a * (3+b) / 2 – c++ * b

EVALUATE AN EXPRESSION

Let a=3 , b=4, c=5 , then Solve:

x= a* 4 + b / 2 – c * b

y = --a * (3+b) / 2 – c++ * b

```
#include<stdio.h>
int main()
{
//local declarations
        int a=3;
        int b=4;
        int c=5;
        int x;
        int y;

//statements
        printf("initial values of variables:\n");
        printf("a=%d\tb=%d\tc=%d\n\n",a,b,c);
        x=a*4+b/2-c*b;
        printf("value of the expression a*4+b/2-c*b :%d\n",x);
        y=--a*(3+b)/2-c++*b;
        printf("value of the expression --a*(3+b)/2-c++*b :%d\n",y);
        printf("values of variables are now:\n");
        printf("a=%d\tb=%d\tc=%d\n\n",a,b,c);
        return 0;
}
```

PROBLEMS : EVALUATION OF EXPRESSIONS

- If x=2, y=3 and z=1 ; evaluate the following expressions
  - x+2/6+y
  - y-3*z+2
  - z-(x+z)%2+4
  - x-2*(3+z)+y
  - y++ + z-- + x++

TYPE CONVERSION

- Expressions with different data types?

  To evaluate, one of the types must be converted

  1. **Implicit Type Conversion.**
  2. **Explicit Type Conversion.**

IMPLICIT TYPE CONVERSION

When the types of the two operands in a binary expression are different, C automatically converts one type to another based on:

- Conversion Rank
- Conversion in Assignment Expressions
- Promotion
- Demotion

- Implicit type casting means conversion of data types without losing its original meaning. This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.
- Implicit type conversion in C happens automatically when a value is copied to its compatible data type. During conversion, strict rules for type conversion are applied. If the operands are of two different data types, then an operand having lower data type is automatically converted into a higher data type.

```
#include<stdio.h>
int main(){
        short a=10; //initializing variable of short data type
        int b; //declaring int variable
        b=a; //implicit type casting
        printf("%d\n",a);
        printf("%d\n",b);
}
```

Output:

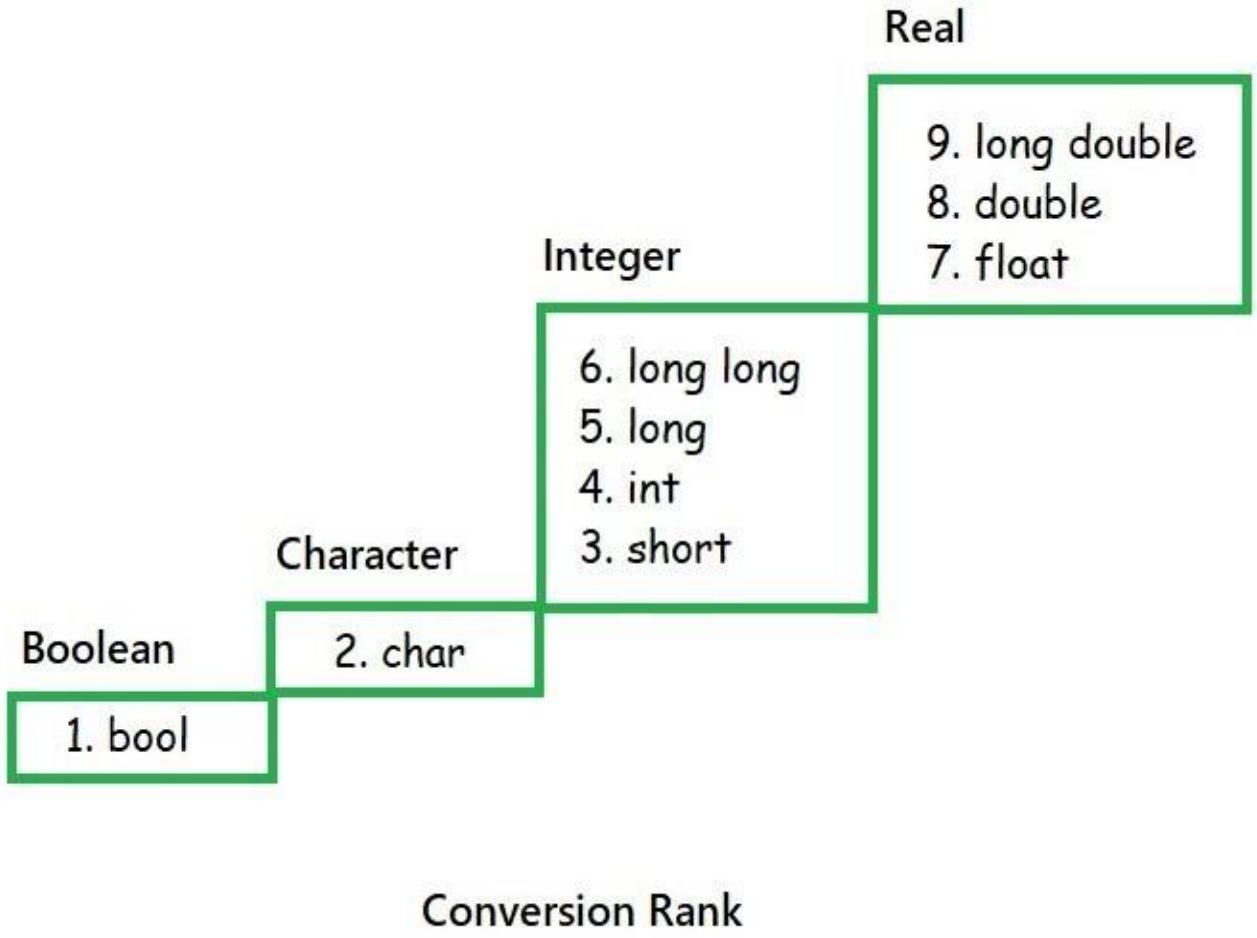
10
10

```
#include <stdio.h>
main() {
   int  number = 1;
   char character = 'k'; /*ASCII value is 107 */
   int sum;
   sum = number + character;
   printf("Value of sum : %d\n", sum );
}
```

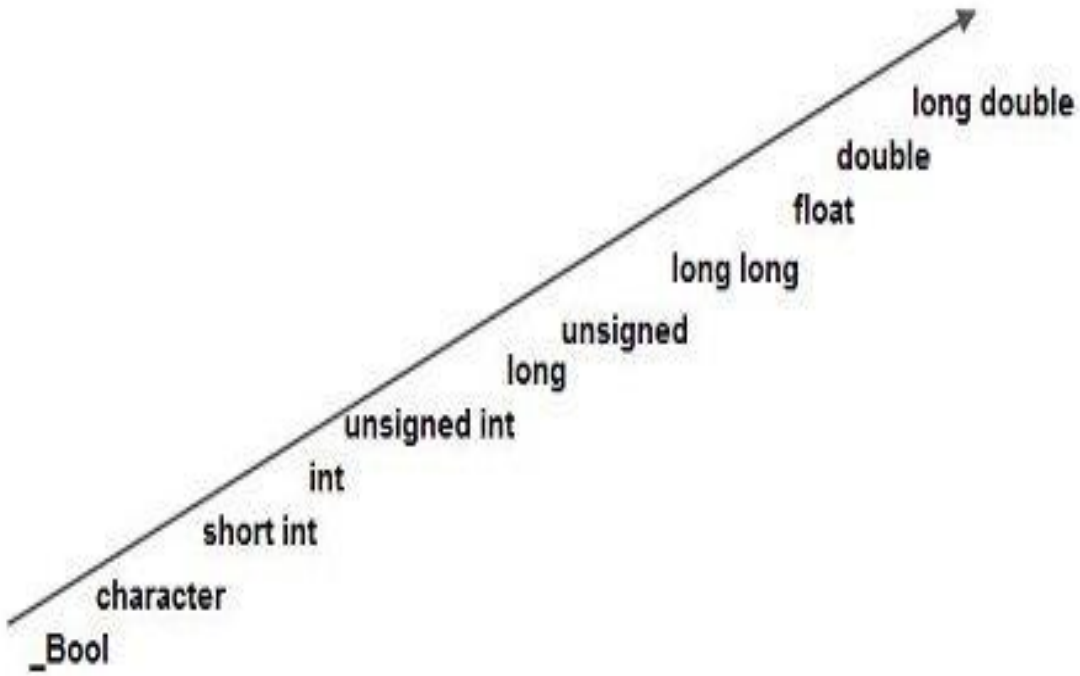
Output:

Value of sum : 108

CONVERSION RANK



Conversion Rank



Ranks allocated for implicit conversion (the arrow points to the higher rank)

```
#include <stdio.h>
main() {
   int  num = 13;
   char c = 'k'; /* ASCII value is 107 */
   float sum;
   sum = num + c;
   printf("sum = %f\n", sum );
}
```

Output:

sum = 120.000000

Explicit Type Conversions

- Uses unary cast operator:

```
int a;
(float) a;
```

```
#include<stdio.h>
int main()
{
        float a = 1.2;
        int b = (int)a + 1;
        printf("Value of a is %f\n", a);
        printf("Value of b is %d\n", b);
        return 0;
}
```

Output:

Value of a is 1.200000
Value of b is 2

```
#include <stdio.h>
int main()
{
        int a=5;
        float x=5.63;
        float y=a+x;          //implicit promotion
        float c= x+y;
        int b=  x + (int) y;    // explicit conversion
        int z=  x+ y;
        printf("a= %d\n", a);
        printf("x= %f\n", x);
        printf("y= %f\n", y);
        printf("c=%f\n", c);
        printf("b=%d\n", b);
        printf("z=%d\n", z) ;
        return 0;
}
```

```c
#include <stdio.h>
int main()
{
        int a=5;
        float x=5.63;
        float y=a+x;          //implicit promotion
        float c= x+y;
        int b=  x + (int) y;    // explicit conversion
        int z=  x+ y;
        printf("a= %d\n", a);
        printf("x= %f\n", x);
        printf("y= %f\n", y);
        printf("c=%f\n", c);
        printf("b=%d\n", b);
        printf("z=%d\n", z) ;
        return 0;
}
```

Output:

a= 5
x= 5.630000
y= 10.630000
c=16.260000
b=15
z=16

// Demonstrate Automatic Promotion of Numeric Types:

```
#include<stdio.h>
#include<stdbool.h>
int main()
{
        //local declarations
        bool b=true;
        char c='A';
        float d=245.3;
        int i=3650;
        short s=78;

        //statements
                printf("bool + char is char:%c\n", b+c);
                printf("int * short is int: %d\n",i*s);
                printf("float * char is float: %f\n",d*c);

                c= c + b;  //bool promoted to char
                d= d + c;  // char promoted to float
                b= false;
                b= - d;     //Float demoted to bool

                printf("\n After execution...\n");
                printf("char  + true:  %c\n",c);
                printf("float + char :%f\n",d);
                printf("bool = - float:%f\n",b);

                return 0;
}
```

```
#include<stdio.h>
int main()
{
        //local declarations
        char achar='\0‘;
        int intnum1=100;
        int intnum2=45;
        double fltnum1=100.0;
        double fltnum2=45.0;
        double fltnum3;

        //statements
        printf("achar numeric : %3d\n",achar);
        printf("intnum1 contains: %3d\n",intnum1);
        printf("intnum2 contains: %3d\n",intnum2);
        printf("fltnum1 contains: %6.2f\n",fltnum1);
        printf("fltnum2 contains: %6.2f\n",fltnum2);
        fltnum3=(double)(intnum1/intnum2);
        printf("\n(double)(intnum1 /intnum2):%6.2f\n",fltnum3);

        fltnum3=(double)intnum1/intnum2;
        printf("\n(double) intnum1 /intnum2:%6.2f\n",fltnum3);
        achar=(char)(fltnum1 - fltnum2);
        printf("(char)(fltnum1 - fltnum2) :%c\n",achar);
        return 0;
        }
```

Programs

1. C-Program to calculate quotient and reminder of two numbers.
2. C-Program to calculate the sum of three numbers / C-Program to demonstrate a Simple Calculator.
3. C-Program to calculate the area and circumference of a circle using PI as a defined constant.

$$c=2*pi*r$$

$$area= pi*r*r$$

4. C-Program to convert temperature given in Celsius to Fahrenheit and Fahrenheit to Celsius

$$C = 5/9(°F – 32)$$

$$F = 9/5°C + 32$$

Selection-Making Decision

Two-Way Selection

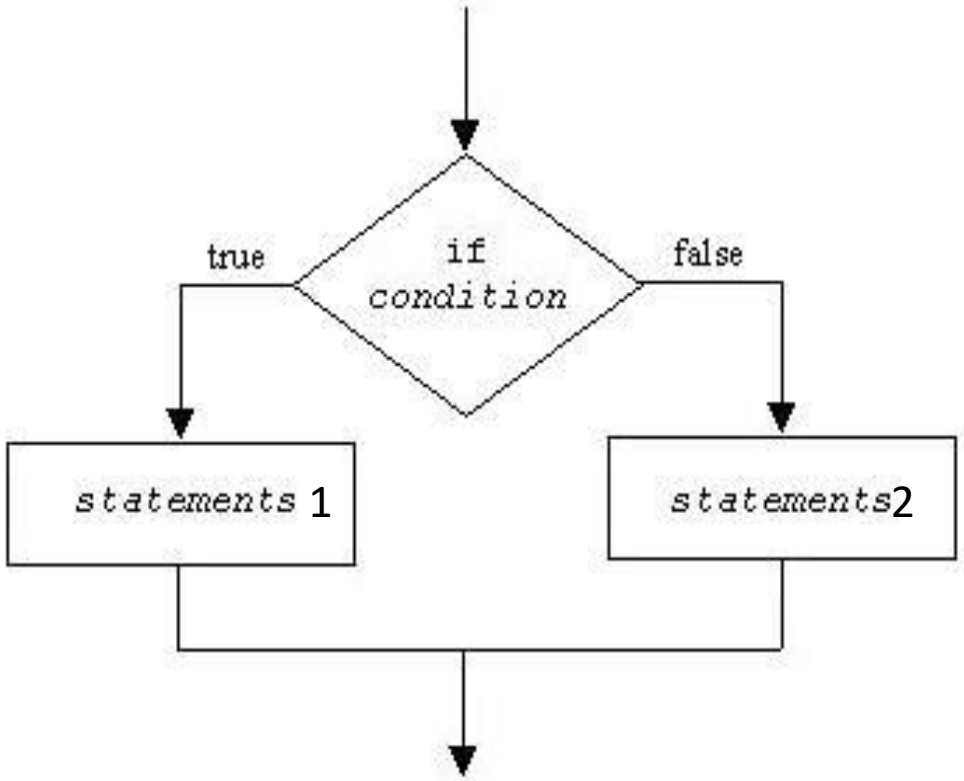
- If… else
- Null else
- Nested if statements

IF…ELSE

- An if…else statement is a composite statement used to make decisions between two alternatives.

Ex:

```
If (a > 1)
        a++;
else
        a--;
```



IF…..ELSE WITH COMPOUND STATEMENTS:

```
if (j !=3)
{
        b++;
        printf(“%d”, b);
}
else
{
        c++;
        printf(“%d”,c);
}
```

NULL ELSE STATEMENTS

```
If (expression)
{
        ------------
}
else
;
```

```
If (expression)
{
        ------------
}  //endif
```

1. Program to find the entered number is odd or even?

Using Simple if(Null if)

if …. Else

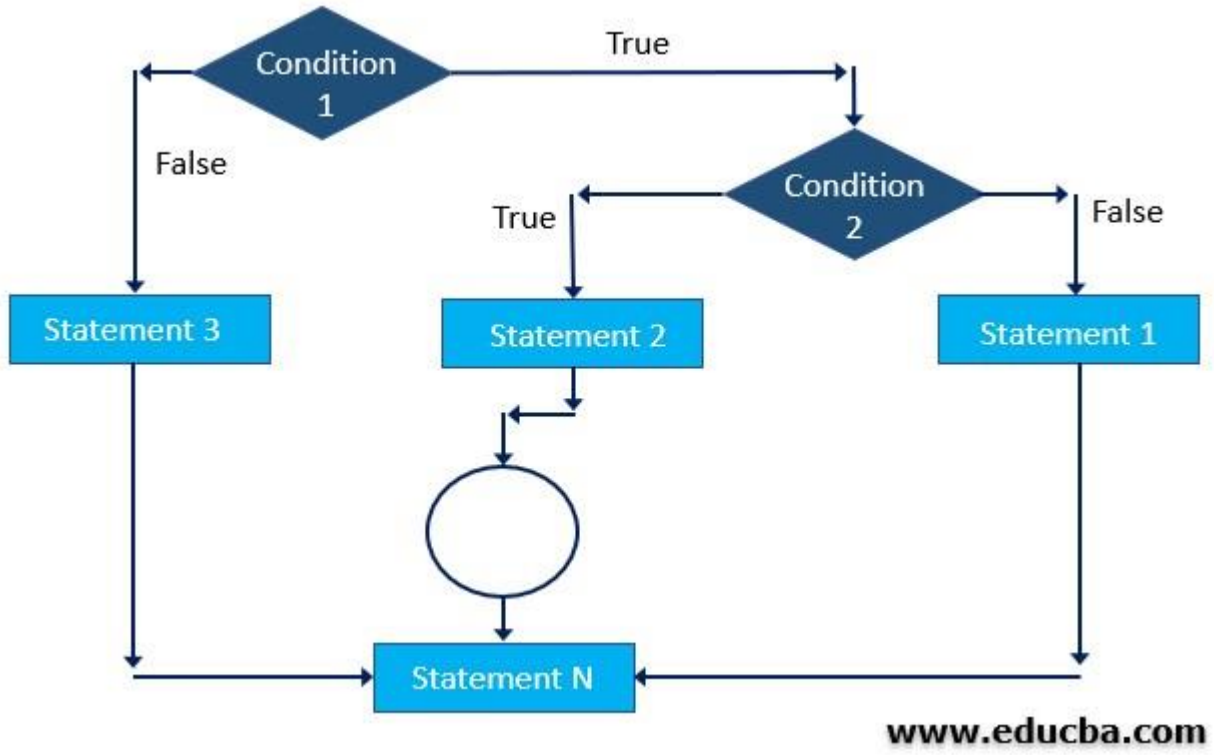
2. Program to check whether a person is eligible to vote or not.

```
#include<stdio.h>
int main()
{
        int number=0;
        printf("Enter a number:");
        scanf("%d",&number);
        if(number%2==0)
        {
       printf("%d is even number",number);
        }
        return 0;
}
```

```
#include<stdio.h>
int main()
{
        int number=0;
        printf("enter a number:");
        scanf("%d",&number);
        if(number%2==0)
        {

printf("%d is even number",number);
        }
        else
        {

        printf("%d is odd number",number);
        }
        return 0;
}
```

NESTED ... IF STATEMENTS

```
if ( Expression 1)
{
        if( Expression 2)
                statement2;
        else
                statement 1;
}
else
        statement 3;

statement n;
```



EXAMPLE PROGRAMS

- Write a program to find largest of 3 numbers.

```
#include<stdio.h>
int main()
{
   int a,b,c;
   printf("Enter the value of a,b,c\n");
   scanf("%d %d %d",&a, &b, &c);
   if((b>a && c>a))
    {
                if(b>c)
                            printf("b is largest");
                else
                            printf("c is largest");
    }
    else
                printf("a is largest");
    return 0;
}
```

DANGLING ELSE PROBLEM

The dangling else problem

```
if (Expression1)
  if (Expression2)
    Statement1
else
  Statement2
```

This indentation is misleading.

DANGLING ELSE PROBLEM

The dangling else problem

```
if (Expression1)
  if (Expression2)
    Statement1
else
  Statement2
```

This indentation is misleading.

```
if (exp)
{
        if(exp)
                st1
}
else
{
        st2
}
```

Return Statement

- A return statement terminates a function. All functions including the main must have a return statement. When there is no return statement at the end of the function, the system inserts one with a void return value.

```
return expression;
```

A return value of 0 tells the OS that the program executed successfully

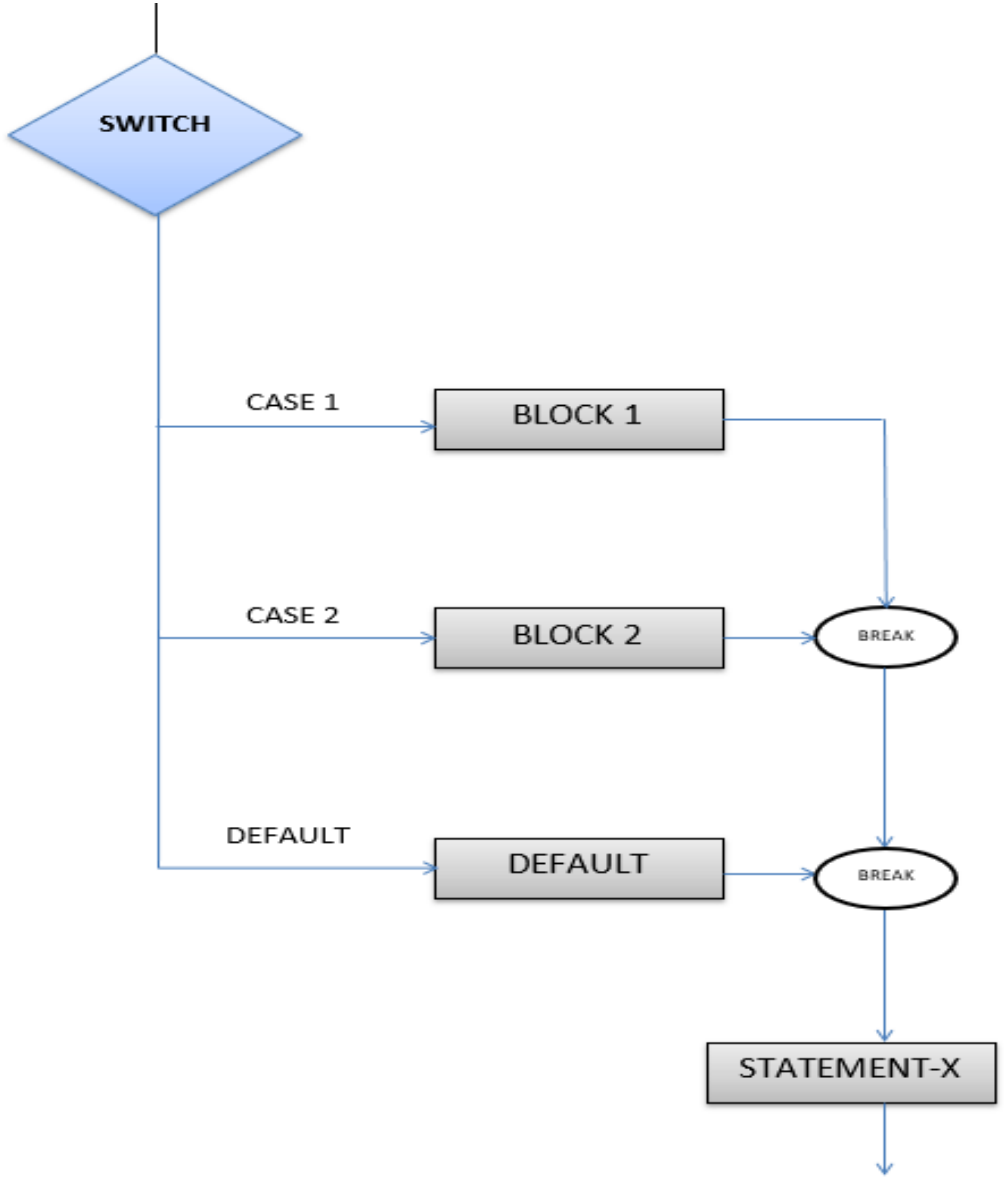
MULTIWAY SELECTION

In addition to two-way selection, most programming languages provide another selection concept known as multiway selection. Multiway selection chooses among several alternatives. C has two different ways to implement multiway selection: the switch statement and else-if construct.

Topics discussed in this section:

The *switch* Statement
The *else-if*

```
switch( expression )
{
        case value-1:Block-1;
                        Break;
        case value-2:Block-2;
                        Break;
        case value-n:Block-n;
                        Break;
        default: Block-1;
                Break;
}
Statement-x;
```



PROGRAM 5-6 Demonstrate the *switch* Statement

```
1  // Program fragment to demonstrate switch
2     switch (printFlag)
3         {
4          case 1:  printf("This is case 1\n");
5
6          case 2:  printf("This is case 2\n");
7
8          default: printf("This is default\n");
9         } // switch
```

```c
#include <stdio.h>
int main()
 {
char operation;
double n1, n2;
 printf("Enter an operator (+, -, *, /): ");
 scanf("%c", &operation);
printf("Enter two operands: ");
scanf("%lf %lf",&n1, &n2);
switch(operation)
 {
            case '+': printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
                      break;
            case '-': printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
                      break;
            case '*': printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
                      break;
            case '/': printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
                      break;
            // operator doesn't match any case constant +, -, *, /
            default: printf("Error! operator is not correct"); }
            return 0;
}
```

```
Enter an operator (+, -, *, /): -
Enter two operands: 32.5
12.4
 32.5 - 12.4 = 20.1
```

What will be the output?

```
#include <stdio.h>
 int main()
{
        int language = 10;
         switch (language)
        {
                case 1: printf("C#\n");
                        break;
                case 2: printf("C\n");
                        break;
                case 3: printf("C++\n");
                        break;
                default: printf("Other programming language\n");
        }
}
```

What will be the output

```
#include <stdio.h>
int main()
{
int number=5;
switch (number)
{
  case 1:
  case 2:
  case 3:
    printf("One, Two, or Three.\n");
    break;
  case 4:
  case 5:
  case 6:
    printf("Four, Five, or Six.\n");
    break;
  default:
    printf("Greater than Six.\n");
}
}
```

Output?

```
#include <stdio.h>
int main()
{
        int ID = 500;
        int password = 000;
        printf("Plese Enter Your ID:\n ");
        scanf("%d", & ID);
        switch (ID)
        {
               case 500:      printf("Enter your password:\n ");
                              scanf("%d", & password);
                              switch (password)
                              {
                                     case 000: printf("Welcome Dear Programmer\n");
                                               break;
                                     default:  printf("incorrect password");
                                               break;
                              }
                              break;
               default: printf("incorrect ID");
                     break;
        }
}
```

PROGRAM 5-7 Multivalued *case* Statements

```
1  /* Program fragment that demonstrates multiple
2     cases for one set of statements
3  */
4  switch (printFlag)
5     {
6      case 1:
7      case 3:  printf("Good Day\n");
8               printf("Odds have it!\n");
9               break;
10     case 2:
11     case 4:  printf("Good Day\n");
12              printf("Evens have it!\n");
13              break;
14     default: printf("Good Day, I'm confused!\n");
15              printf("Bye!\n");
16              break;
17    } // switch
```

Rules and Summary

- **Rules for switch statement**
- An expression must always execute to a result.
- Case labels must be constants and unique.
- Case labels must end with a colon (:).
- A break keyword must be present in each case.
- There can be only one default label.
- We can nest multiple switch statements.
- **Summary**
- A switch is a decision making construct in ‘C.’
- A switch is used in a program where multiple decisions are involved.
- A switch must contain an executable test-expression.
- Each case must include a break keyword.
- Case label must be constants and unique.
- The default is optional.
- Multiple switch statements can be nested within one another.

C-PROGRAM TO READ A TEST SCORE, CALCULATE THE GRADE FOR THE SCORE AND PRINT THE GRADE

```
#include<stdio.h>
int main()
{
	int marks,temp;
	printf("\n--------------------------------");
	printf("\nEnter The Marks Between 0 To 100:");
	printf("\nEnter The Mark: ");
	scanf("%d", &marks);
	if(marks>100)
	{
		/* Marks greater than 100 */
		printf("\nDon't Be Smart Enter your Marks Between Limit\n");
	}
	else
```

```
{
   temp=marks/10;
   switch(temp)
   {
      case 10 :
      case 9 :
         printf("\n Your Grade is: A");              /* Marks between 90-100 */
         break;
      case 8 :
         printf("\n Your Grade is: B" );     /* Marks between 80-89 */
         break;
```

```
 case 7 :
        printf("\n Your Grade is: C" );     /* Marks between 70-79 */
         break;
case 6 :
        printf("\n Your Grade is: D" );     /* Marks between 60-69 */
        break;
default :
        printf("\n You Grade is: F or Fail\n");    /* Marks less than 40 */
  }
 }
```

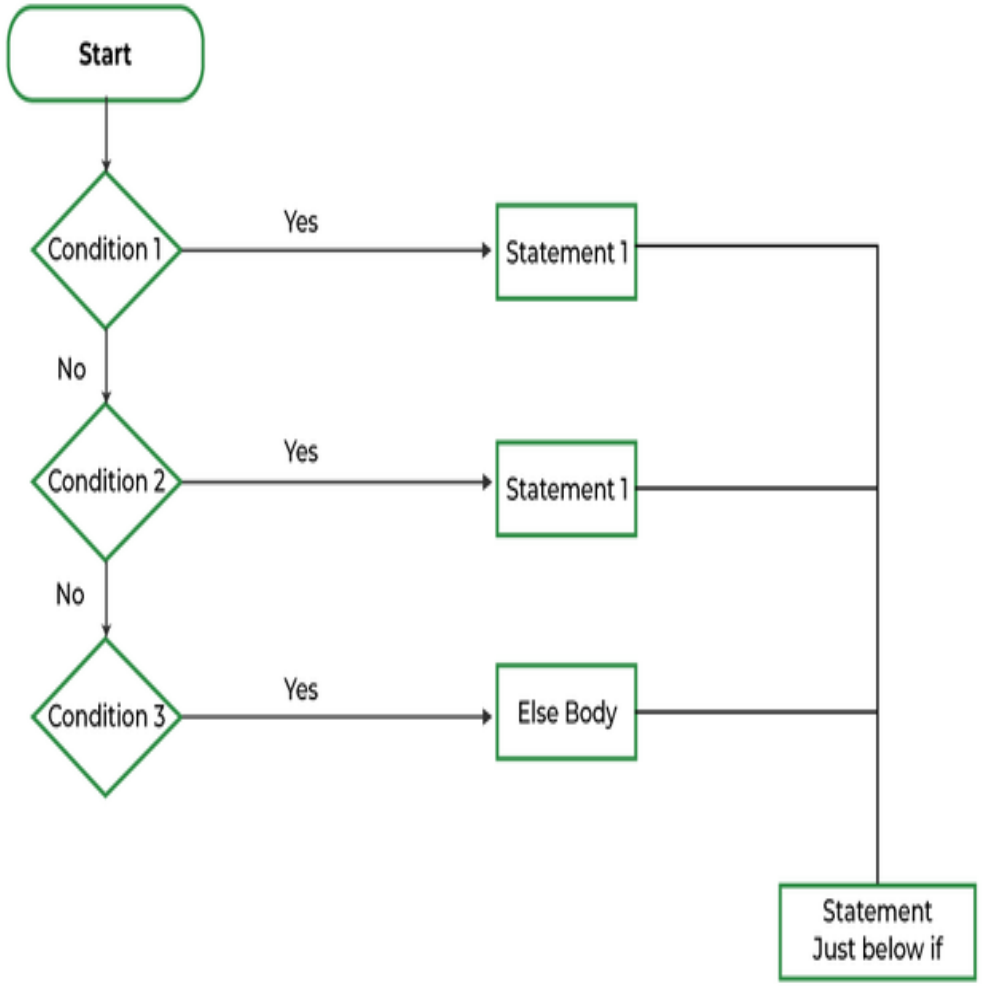
if else if *ladder*

- ***if else if*** *ladder in C programming* is used to test a series of conditions sequentially. Furthermore, if a condition is tested only when all previous if conditions in the if-else ladder are false. If any of the conditional expressions evaluate to be true, the appropriate code block will be executed, and the entire if-else ladder will be terminated.

Else if ladder

Syntax:

```
// any if-else ladder starts
   with an if statement only
if(condition)
{        }
else if(condition)
{      // this else if will be executed
          when condition in if is
           false and
       // the condition of this else if
              is true
}....
       // once if-else ladder can
          have multiple else if
else { // at the end we put else }
```



Write a program to display wether a number is +ve, -ve or zero

```c
int main()
{
    int n = 0;

    // all Positive numbers will make this
    // condition true
    if (n > 0) {
        printf("Positive");
    }

    // all Negative numbers will make this
    // condition true
    else if (n < 0) {
        printf("Negative");
    }

    // if a number is neither Positive nor Negative
    else {
        printf("Zero");
    }
    return 0;
}
```

C Program to Calculate Grade According to marks

```c
#include <stdio.h>
int main()
{
    int marks = 91;
    if (marks <= 100 && marks >= 90)
        printf("A+ Grade");
    else if (marks < 90 && marks >= 80)
        printf("A Grade");
    else if (marks < 80 && marks >= 70)
        printf("B Grade");
    else if (marks < 70 && marks >= 60)
        printf("C Grade");
    else if (marks < 60 && marks > 50)
        printf("D Grade");
    else if (marks < 50)
        printf("F Failed");
    return 0;
}
```

An electricity board charges the following rates for the use of electricity:
for the first 200 units 80 paise per unit:
for the next 100 units 90 paise per unit:
beyond 300 units rupees 1 per unit.

All users are charged a minimum of rupees 100 as a meter charge. If the total amount is more than Rs 400, then an additional surcharge of 15% of the total amount is charged.
Write a program to read the name of the user, the number of units consumed, and print out the charges.

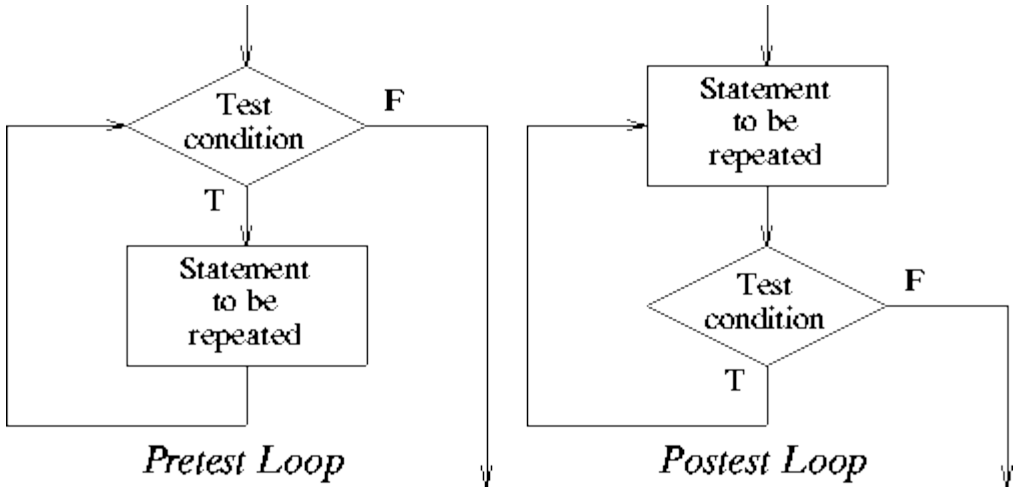
```
#include<stdio.h>
#include<string.h>
void main()
{
int cust_no, unit_con;
float charge,surcharge=0, amt, total_amt;
char nm[25];
printf("Enter the customer IDNO :\t");
scanf("%d",&cust_no);
printf("Enter the customer Name :\t");
scanf("%s",nm);
printf("Enter the unit consumed by customer :\t");
scanf("%d",&unit_con);
if (unit_con <200 )
        charge = 0.80;
else if (unit_con>=200 && unit_con<300)
        charge = 0.90;
else
        charge = 1.00;
amt = unit_con*charge;
if (amt>400)
        surcharge = amt*15/100.0;

total_amt = amt+surcharge;

printf("\t\t\t\nElectricity Bill\n\n");
printf("Customer IDNO :\t%d",cust_no);
printf("\nCustomer Name :\t%s",nm);
printf("\nunit Consumed :\t%d",unit_con);
printf("\nAmount Charges @Rs. %4.2f per
unit:\t%0.2f",charge,amt);
printf("\nSurchage Amount :\t%.2f",surcharge);
printf("\nMinimum meter charge Rs :\t%d",100);
printf("\nNet Amount Paid By the Customer
 :\t%.2f",total_amt+100);
}
```

Repetition Statements (loops)

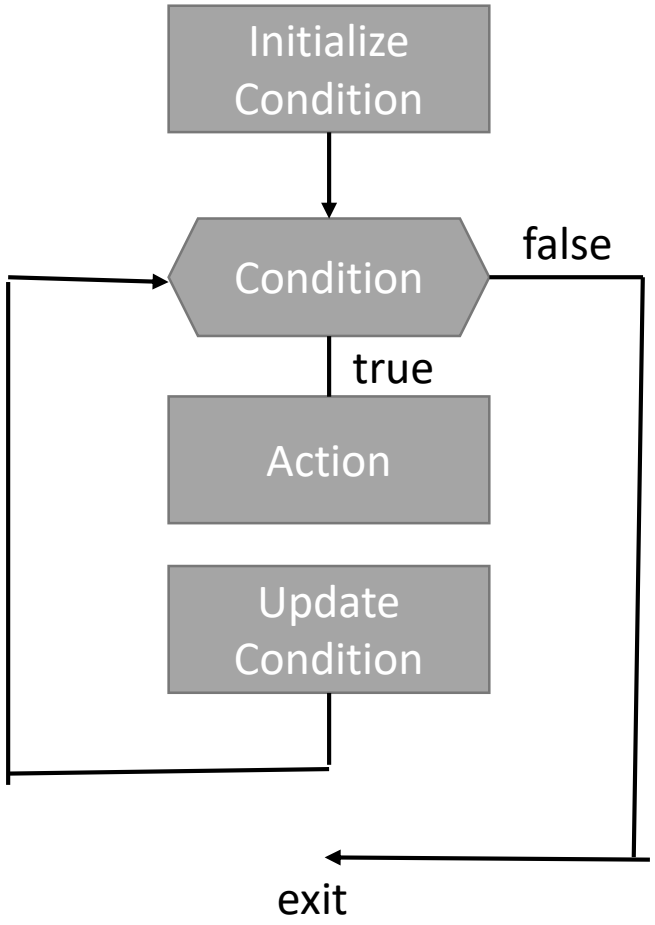
CONCEPT OF LOOP

We must design a loop so that before and after the iteration, it checks to see if the task is done. If it is not done the loop repeats one more time, or else it terminates.

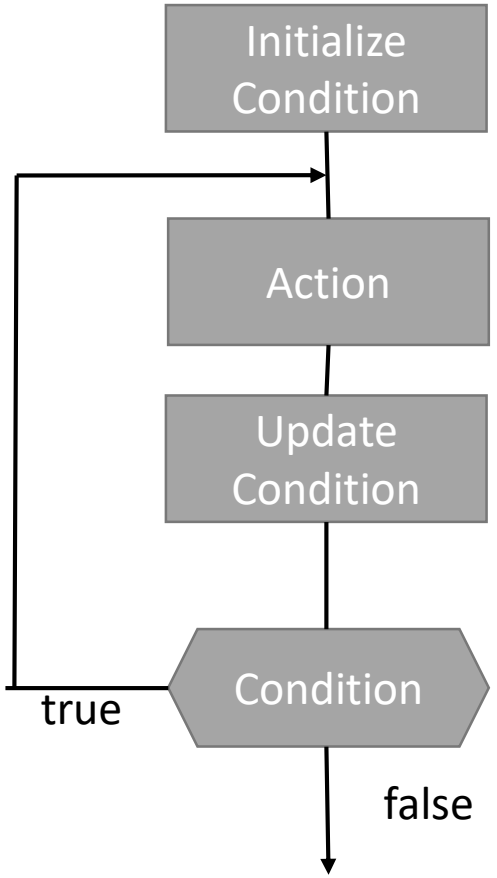
- Pretest Loop
- Post-Test Loop



EVENT CONTROLLED LOOPS

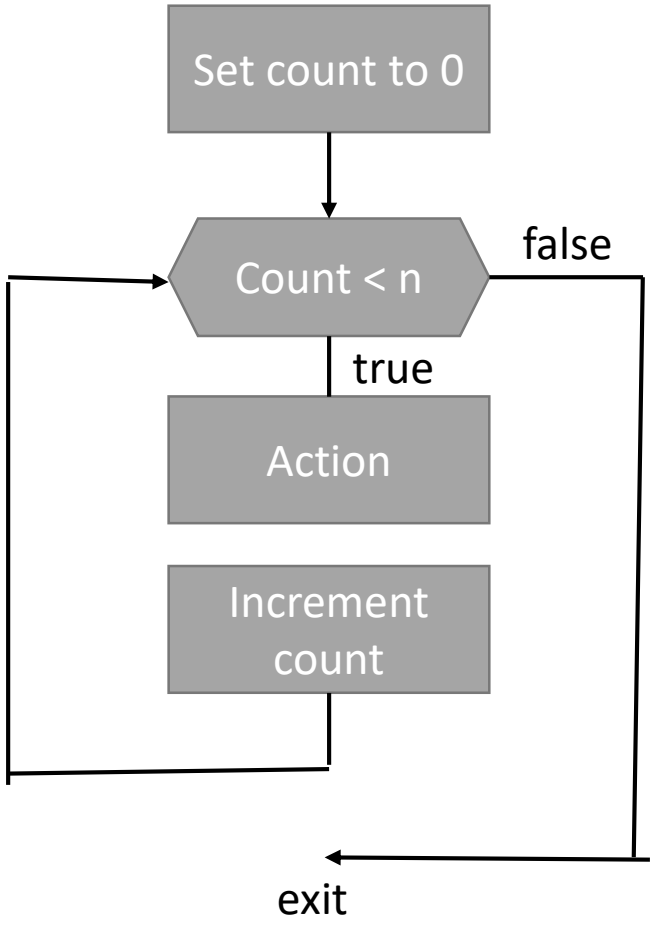


Pre-test loop

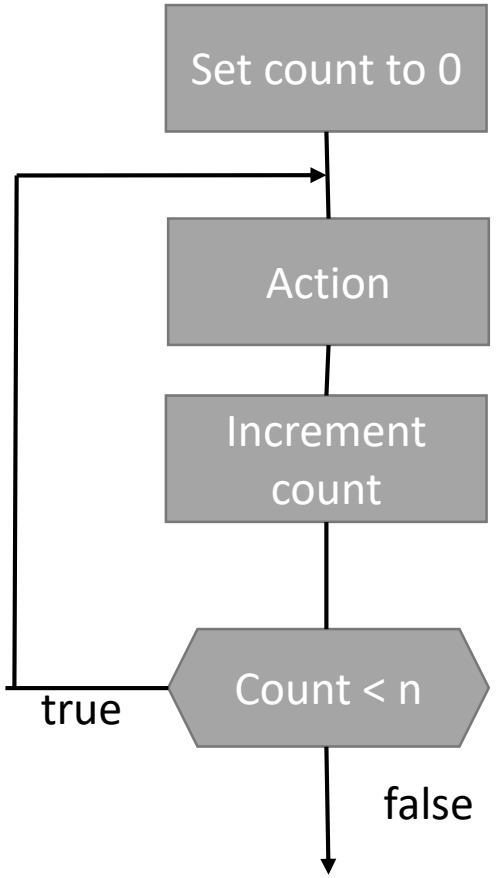


Post-test loop

COUNTER CONTROLLED LOOPS



Pre-test loop



Post-test loop

LOOP COMPARISON

Pretest Loop		Post-test Loop	
Initialization	1	Initialization	1
Number of tests	n+1	Number of tests	n
Action executed	n	Action executed	n
Updating executed	n	Updating executed	n
Minimum Iteration	0	Minimum Iteration	1

Loops – While, Do, For

- Repetition Statements
  - While
  - Do
  - For

Repetition Statements

- *Repetition statements* allow us to execute a statement or a block of statements multiple times
- Often they are referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements:

  while
  do
  for

- The programmer should choose the right kind of loop statement for the situation

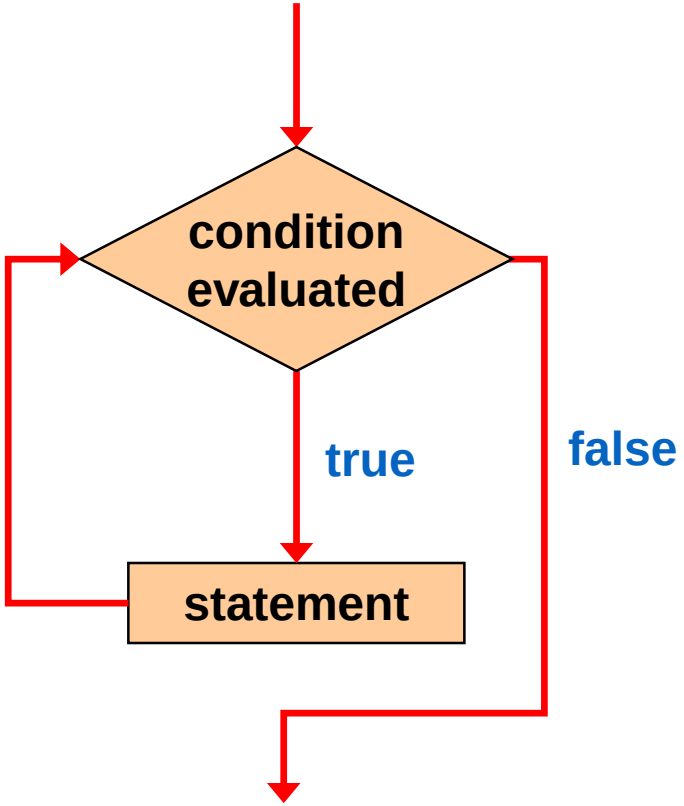
The While Statement

- A *while statement* has the following syntax:

```
while ( condition )
    statement;
```

- If the `condition` is true, the `statement` is executed
- Then the condition is evaluated again, and if it is still true, the statement is executed again
- The statement is executed repeatedly until the condition becomes false

Logic of a while loop



The While Statement

- An example of a while statement:

```
int count = 0;
while (count < 2)
{
    printf("welcome to java!");
    count++;
}
```

- If the condition of a `while` loop is false initially, the statement is never executed
- Therefore, the body of a `while` loop will execute zero or more times

animation

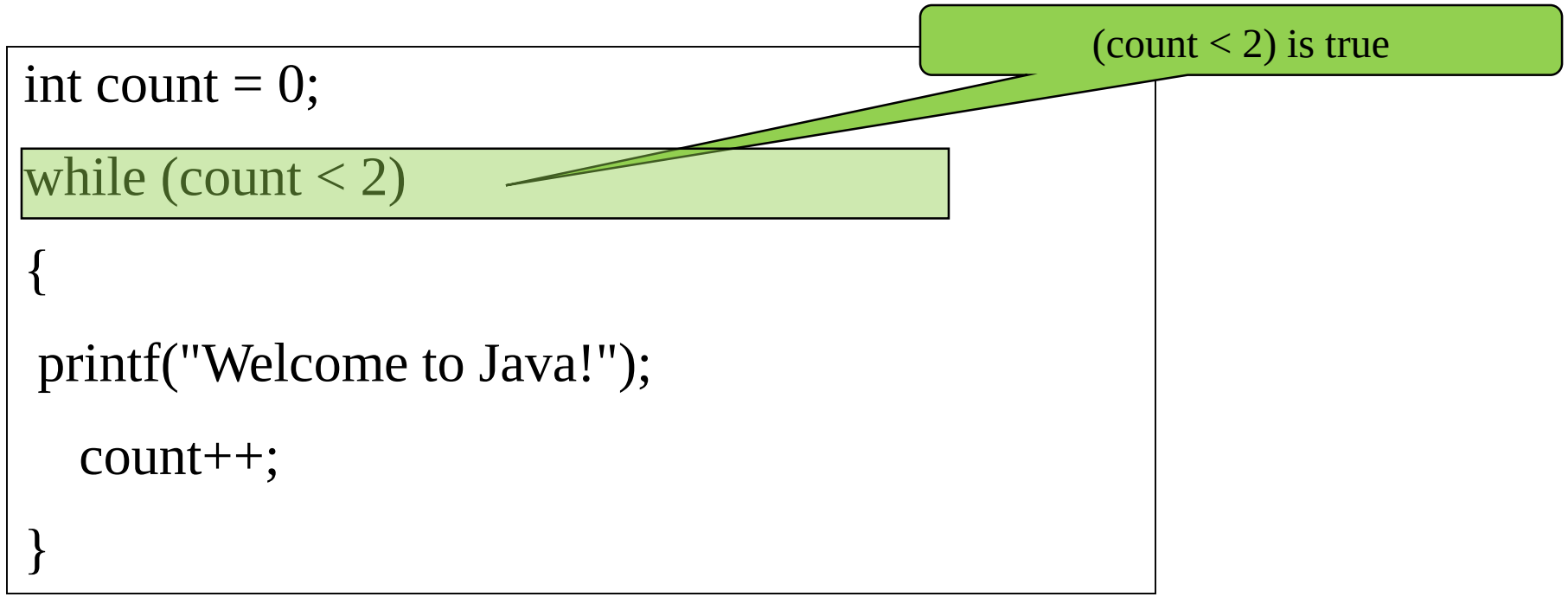
Trace while Loop

```
int count = 0;
while (count < 2)
{
 printf("Welcome to Java!");
    count++;
}
```

Initialize count

animation

Trace while Loop, cont.



animation

Trace while Loop, cont.

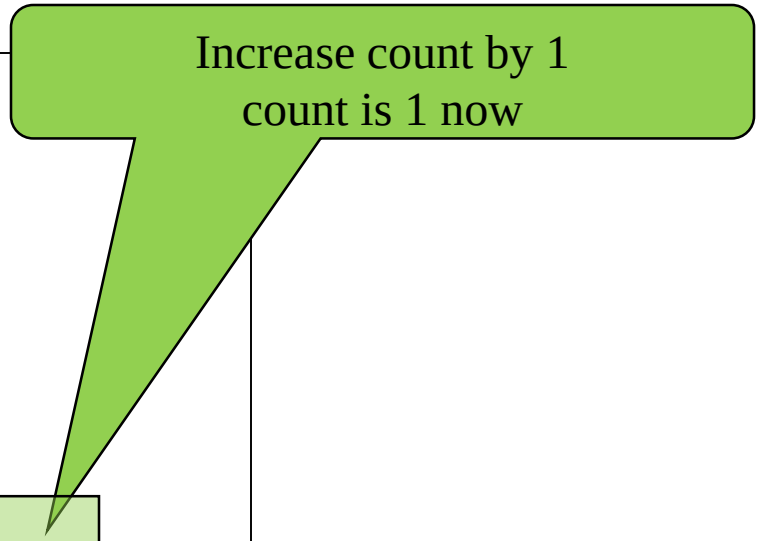
```
int count = 0;
while (count < 2)
{
    printf("Welcome to Java!");
    count++;
}
```

Print Welcome to Java

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
 printf("Welcome to Java!");
   count++;
}
```



animation

Trace while Loop, cont.

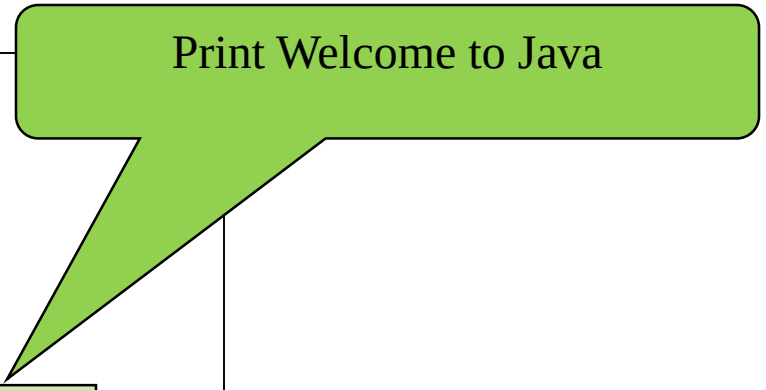
```
int count = 0;
while (count < 2)
{
    printf("Welcome to Java!");
    count++;
}
```

(count < 2) is still true since count is 1

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
  printf("Welcome to Java!");
    count++;
}
```



animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
printf("Welcome to Java!");
  count++;
}
```

Increase count by 1
count is 2 now

animation

Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
    printf("Welcome to Java!");
    count++;
}
```

(count < 2) is false since count is 2 now

animation

Trace while Loop

```
int count = 0;
while (count < 2)
{
   printf("Welcome to Java!");
    count++;
}
```

The loop exits. Execute the next statement after the loop.

EXAMPLE (AVERAGE)

```
#include<stdio.h>
int main()
{
        int sum=0, count=0;
        float avg;

        while( count <10)
        {
                count++;
                sum+= count;
        }
        avg= (float ) sum/count;
        printf("%f",avg);
}
```

Infinite Loops

- Executing the statements in the body of a `while` loop must eventually make the condition false
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate

Infinite Loops

- An example of an infinite loop:

```
int count = 1;
while (count <= 25)
{
    printf("%d",count);
    count = count - 1;
}
```

- This loop will continue executing until the user externally interrupts the program.

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely

Nested Loops

- How many times will the string "Here" be printed?

```
count1 = 1;
while (count1 <= 10)
{
   count2 = 1;
   while (count2 <= 20)
   {
      System.out.println ("Here");
      count2++;
   }
   count1++;
}
```

10 * 20 = 200

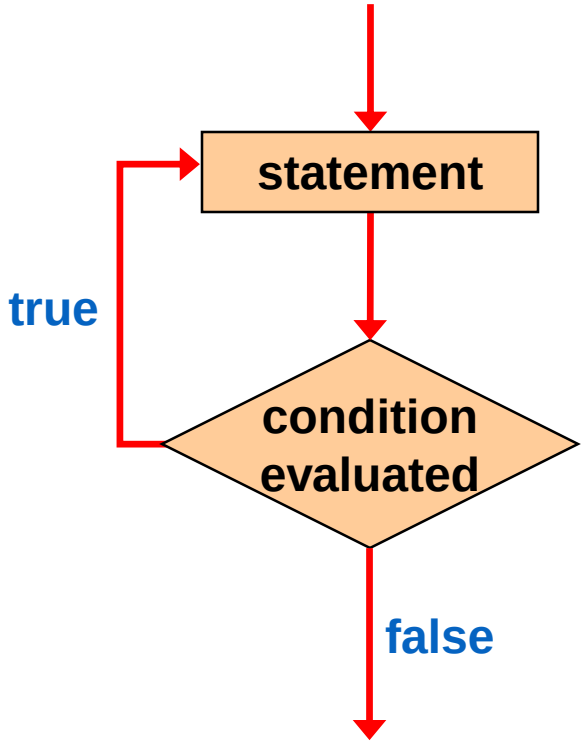
The do Statement

- A *do statement* has the following syntax:

```
do
{
    statement;
}
while ( condition );
```

- **The `statement` is executed once initially, and then the `condition` is evaluated**
- **The statement is executed repeatedly until the condition becomes false**

Flowchart of a do Loop



The do Statement

- An example of a `do` loop:

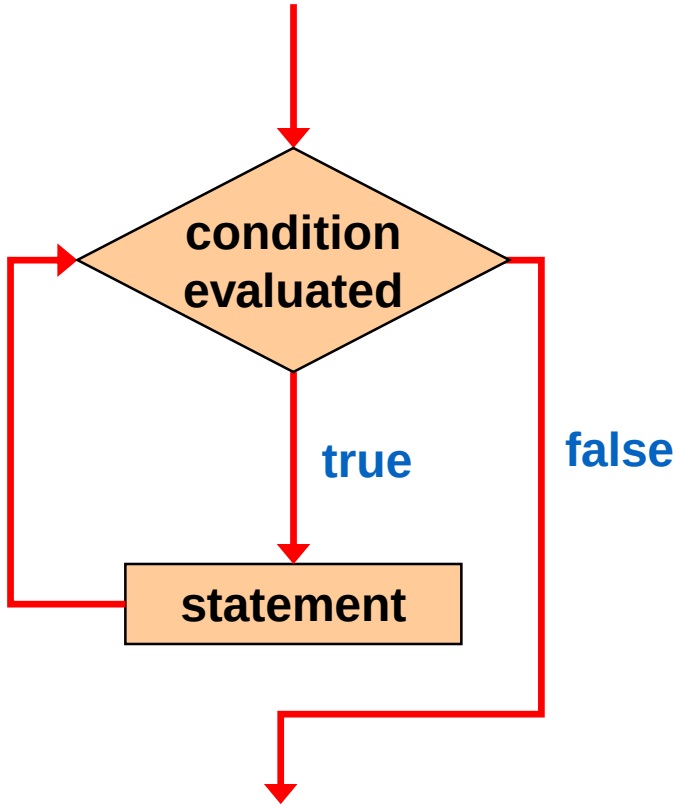
```
int count = 0;
do
{
    printf("Welcome to Java!");
    count++;
}
while (count < 2);
```

Don't forget this semicolon!

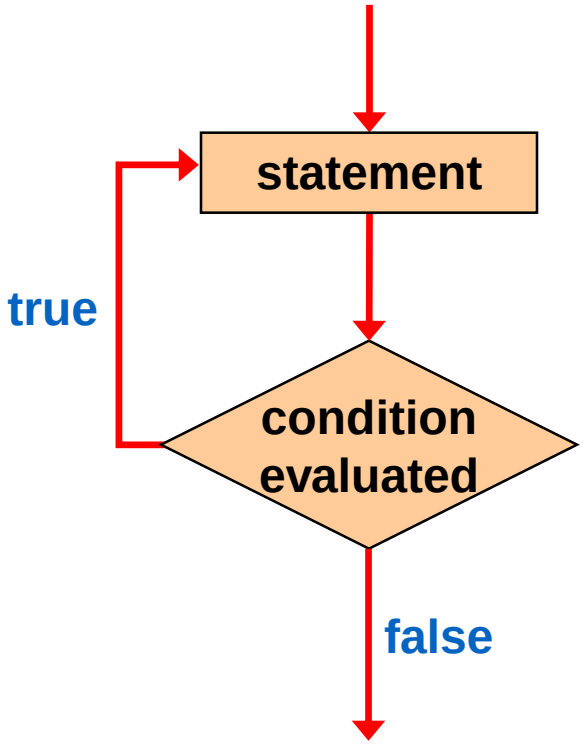
- **The body of a `do` loop executes at least once**

Comparing while and do

The while Loop



The do Loop



animation

Trace do Loop

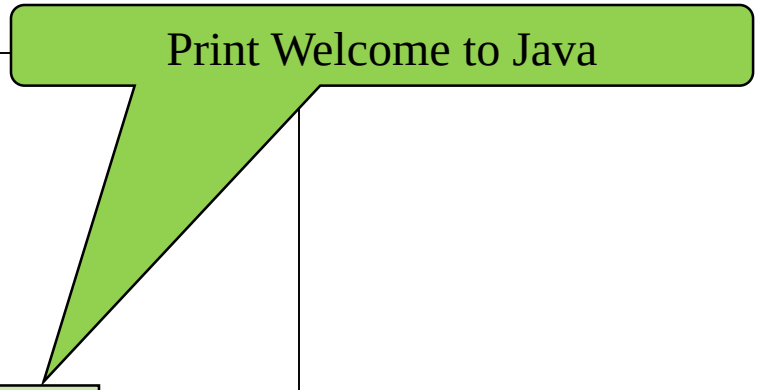
```
int count = 0;
do
{
    printf("Welcome to Java!");
    count++;
} while (count < 2);
```

Initialize count

animation

Trace do Loop, cont.

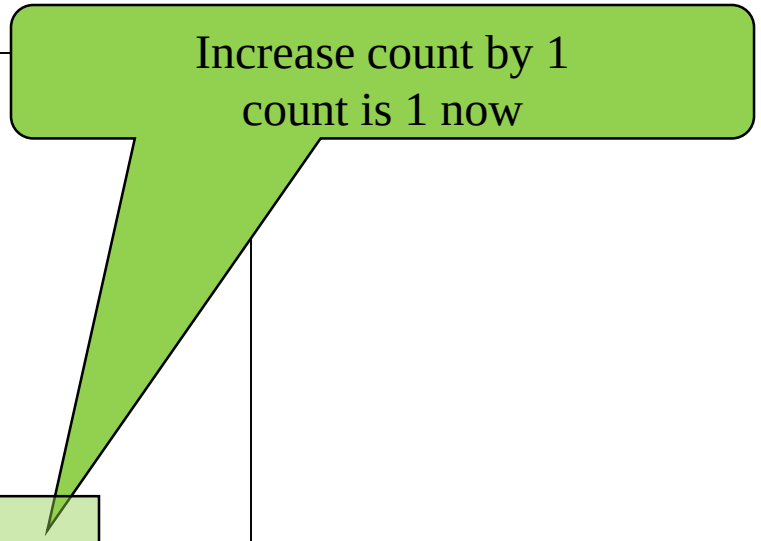
```
int count = 0;
do
{
   printf("Welcome to Java!");
    count++;
} while (count < 2);
```



animation

Trace do Loop, cont.

```
int count = 0;
do
{
    printf("Welcome to Java!");
    count++;
} while (count < 2);
```



animation

Trace do Loop, cont.

```
int count = 0;
do
{
 printf("Welcome to Java!");
    count++;
} while (count < 2);
```

(count < 2) is true

animation

Trace do Loop, cont.

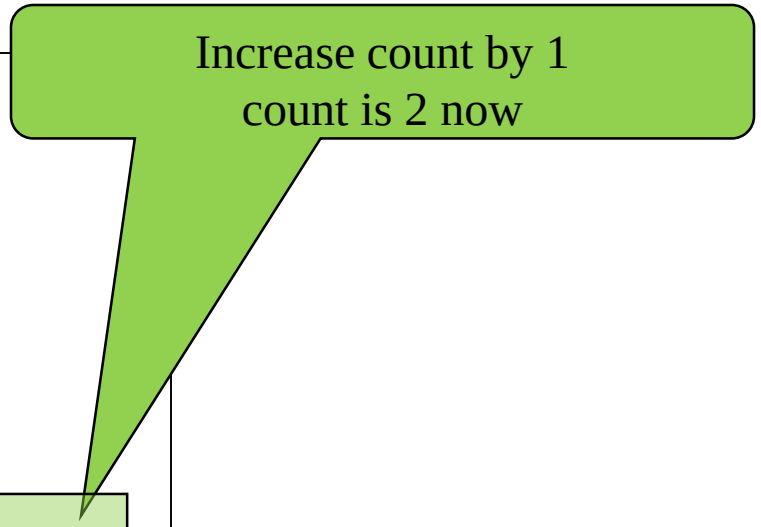
```
int count = 0;
do
{
printf("Welcome to Java!");
    count++;
} while (count < 2);
```

Print Welcome to Java

animation

Trace do Loop, cont.

```
int count = 0;
do
{
   printf("Welcome to Java!");
   count++;
} while (count < 2);
```



animation

Trace do Loop, cont.

```
int count = 0;
do
{
 printf("Welcome to Java!");
    count++;
} while (count < 2);
```

(count < 2) is false since count is 2 now

animation

Trace do Loop

```
int count = 0;
do
{
 printf("Welcome to Java!");
    count++;
} while (count < 2);
```

The loop exits. Execute the next statement after the loop.

The for Statement

- A *for statement* has the following syntax:

The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

The *increment* portion is executed at the end of each iteration

The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

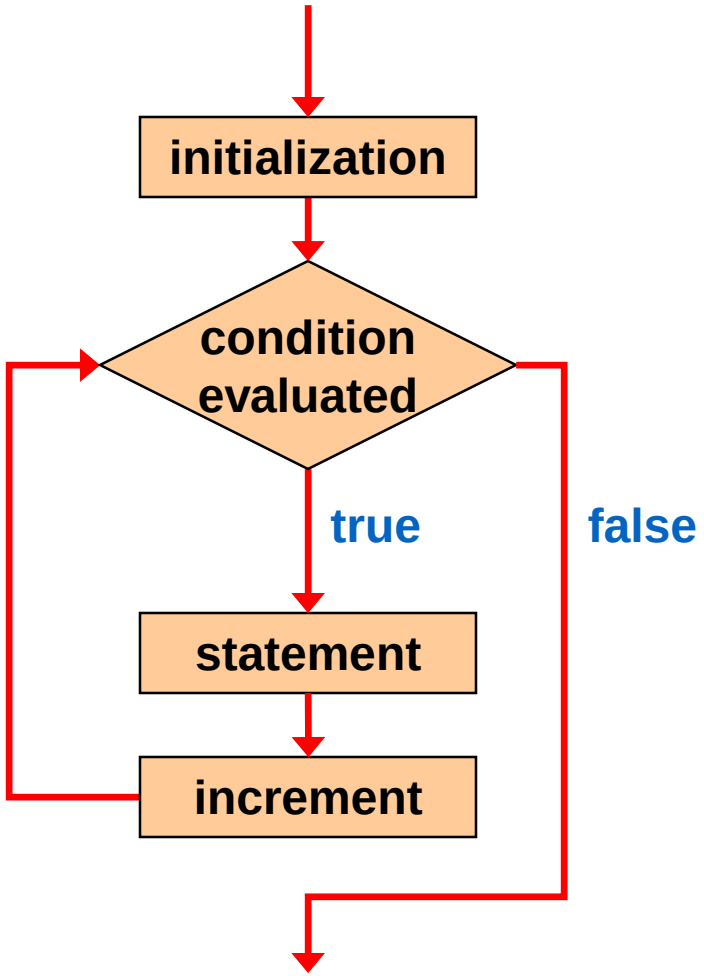
```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

- Example:

```
for(int i = 0; i < 10; i++)
{
    //some code
}
```

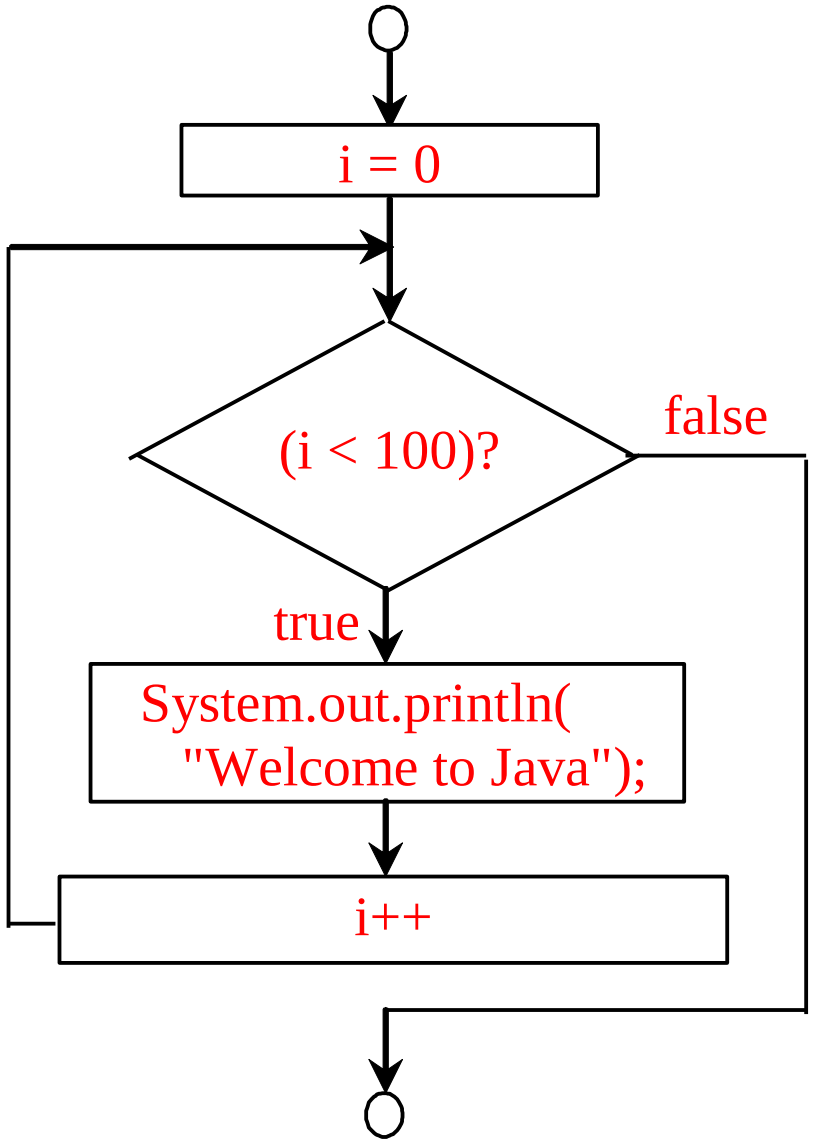
```
int i = 0;
while(i < 10)
{
    //some code
    i++;
}
```

Flowchart of a for loop



Flowchart - *example*

```
for (int i = 0; i < 100; i++)
{
    printf("Welcome to Java!");
}
```



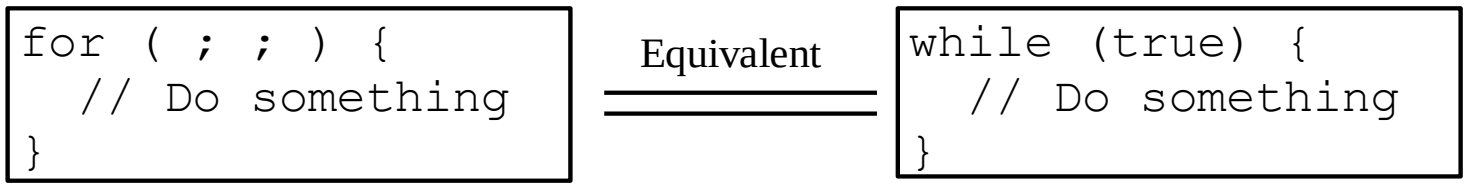
The for Statement

- The initialization section can be used to declare a variable
- Like a while loop, the condition of a for loop is tested *prior* to executing the loop body. Therefore, the body of a for loop will execute zero or more times
- The increment section can be used to increment or decrement a variable, for example:

```
for (int num=100; num > 0; num -= 5)
   printf("%d",num);
```

Infinite for-loop

- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop



(a) (b)

animation

Trace for Loop, cont.

```
for (int i = 0; i < 2; i++)
{
 printf("Welcome to Java!");
}
```

Execute initializer
i is now 0

animation

Trace for Loop, cont.

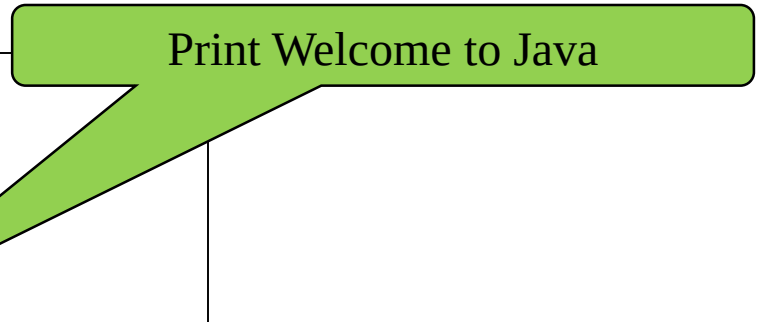
```
for (int i = 0; i < 2; i++)
{
 printf("Welcome to Java!");
}
```

(i < 2) is true
since i is 0

animation

Trace for Loop, cont.

```
for (int i = 0; i < 2; i++)
{
  printf("Welcome to Java!");
}
```



animation

Trace for Loop, cont.

```
for (int i = 0; i < 2; i++)
{
 printf("Welcome to Java!");
}
```

Execute increment statement
i now is 1

animation

Trace for Loop, cont.

```
for (int i = 0; i < 2; i++)
{
 printf("Welcome to Java!");
}
```

(i < 2) is still true
since i is 1

animation

Trace for Loop, cont.

```
for (int i = 0; i < 2; i++)
{
  printf("Welcome to Java!");
}
```

Print Welcome to Java

animation

Trace for Loop, cont.

```
for (int i = 0; i < 2; i++)
{
  printf("Welcome to Java!");
}
```

Execute increment statement
i now is 2

animation

Trace for Loop, cont.

```
for (int i = 0; i < 2; i++)
{
  printf("Welcome to Java!");
}
```

(i < 2) is false
since i is 2

animation

Trace for Loop, cont.

```
for (int i = 0; i < 2; i++)
{
  printf("Welcome to Java!");
}
```

Exit the loop. Execute the next statement after the loop

Which Loop to Use?

Use the one that is most intuitive and comfortable for you. In general:

- A for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.
- A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.
- A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

```
#include <stdio.h>
int main() {
  int i = 1;

  while (i <= 5)
{
        printf("%d\n", i);
        ++i;
  }

  return 0;
}
```

```
#include <stdio.h>
int main() {
  int i = 1;

  while (i <= 5)
{
        printf("%d\n", i);
        ++i;
  }

  return 0;
}
```

Output:

```
1
2
3
4
5
```

```
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
  return 0;
}
```

```c
#include <stdio.h>


int main() {
  int i;


  for (i = 1; i < 11; ++i)
  {
    printf("%d\n ", i);
  }
  return 0;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

// Program to calculate the sum of first n natural numbers

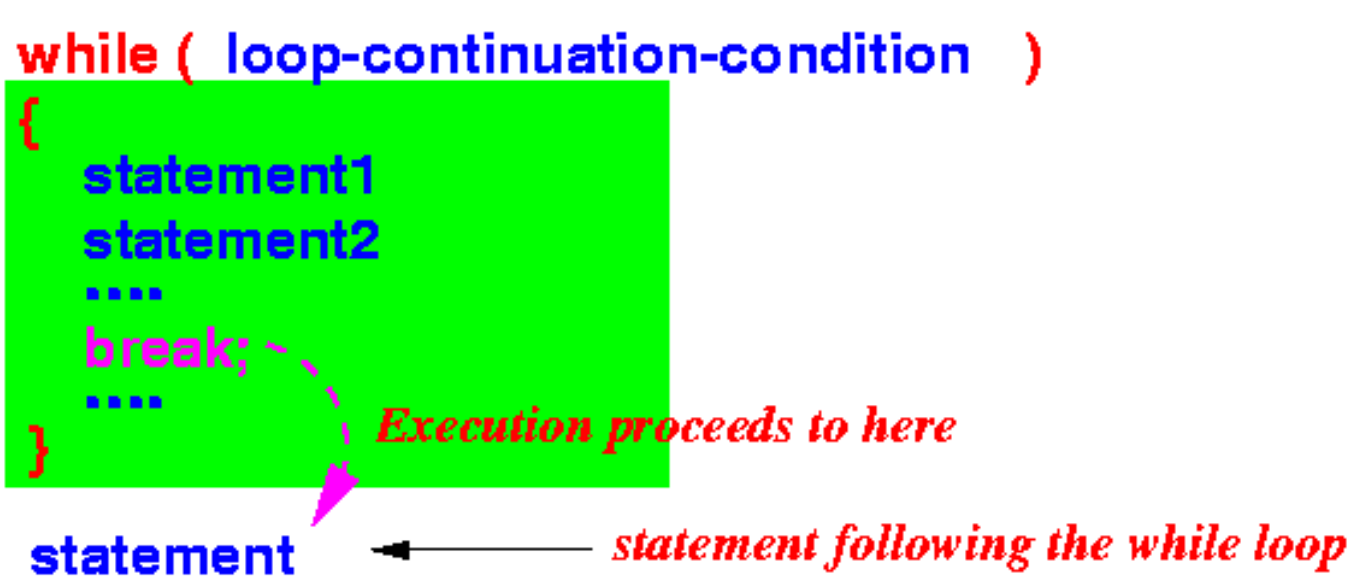
// Positive integers 1,2,3...n are known as natural numbers

```c
#include <stdio.h>
int main()
{
    int num, count, sum = 0;

    printf("Enter a positive integer: ");
    scanf("%d", &num);

// for loop terminates when num is less than count
    for(count = 1; count <= num; ++count)
    {
        sum += count;
    }

    printf("Sum = %d", sum);

    return 0;
}
```

Output:
Enter a positive integer: 10
Sum = 55

The break statement

- The `break` statement causes execution to “break” out of the repetitive loop execution (goes to just outside the loop’s closing “}”)



Example:

```
int i;

for (i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  printf("%d\n", i);
}
```

Example:

```
int i;

for (i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  printf("%d\n", i);
}
```

Output:

```
0
1
2
3
```

Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

```
int i;

for (i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  printf("%d\n", i);
}
```

Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

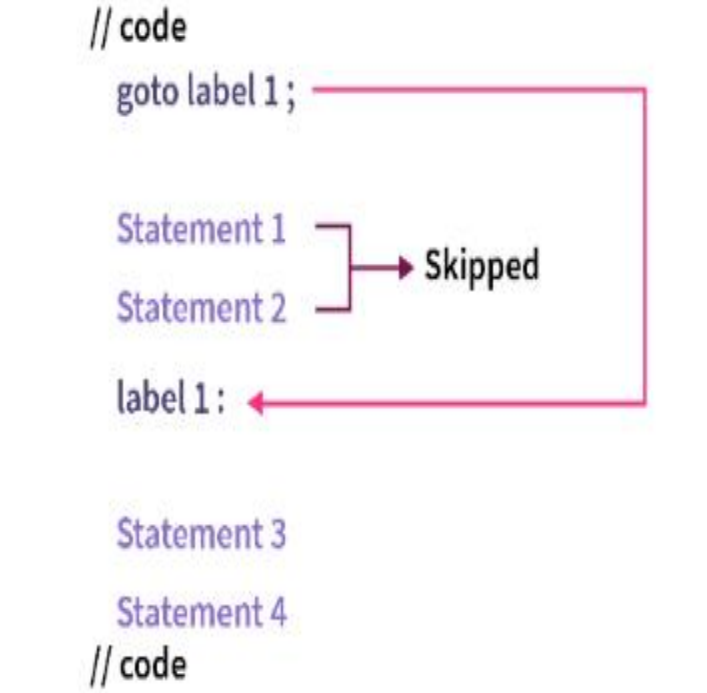
```
int i;

for (i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  printf("%d\n", i);
}
```

Output:

```
0
1
2
3
5
6
7
8
9
```

goto

- Goto statement in C is a jump statement that is used to jump from one part of the code to any other part of the code in C.
- Goto statement helps in altering the normal flow of the program according to our needs.
- This is achieved by using labels, which means defining a block of code with a name so that we can use the goto statement to jump to that label.



Jumped to where label 1 has been declared

Example:

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

int main()
{
   // initialize variable whose absolute value is to be printed
   int n = -11;

   // if the number is already positive, print it directly
   if(n>=0)
   {
      // goto statement
      goto positive;
   }
// to make the number positive multiply by -1
   n = n*(-1);

   // declare positive label

   positive :

      printf("The absolute value is %d", n);

   return 0;

}
```

Example:

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

int main()
{
   // initialize variable whose absolute value is to be printed
   int n = -11;

   // if the number is already positive, print it directly
   if(n>=0)
   {
      // goto statement
      goto positive;
   }
// to make the number positive multiply by -1
   n = n*(-1);

   // declare positive label

   positive :

      printf("The absolute value is %d", n);

   return 0;

}
```

Output:

The absolute value is 11

Example programs:

1. Write a C program to generate Fibonacci series of n numbers.
2. Write a C program to find the factorial of a given number
3. Write a C program to find the sum of digits of a given number.
4. Write a C program to find whether a given number is palindrome or not

```c
#include <stdio.h>
int main()
{
        int i, n;  // initialize first and second terms
        int t1 = 0, t2 = 1;  // initialize the next term (3rd term)
        int nextTerm;  // get no. of terms from user
        printf("Enter the number of terms: ");
        scanf("%d", &n);  // print the first two terms t1 and t2
        printf("Fibonacci Series: %d, %d, ", t1, t2);  // print 3rd to nth terms
        for (i = 2; i < n; ++i)
        {
                nextTerm = t1 + t2;
                printf("%d\n ", nextTerm);
                t1 = t2;
                t2 = nextTerm;
        }
        return 0;
}
```

```c
#include <stdio.h>
int main()
{
	int n, i;
	int fact = 1;
	printf("Enter an integer: ");
	scanf("%d", &n);    // shows error if the user enters a negative integer
	if (n < 0)
		printf("Error! Factorial of a negative number doesn't exist.");
	else
	{
		for (i = 1; i <= n; ++i)
		{
			fact *= i;
		}
		printf("Factorial of %d = %d", n, fact);
	}
	return 0;
}
```

Sum of digits

To get sum of each digits by c program, use the following algorithm:

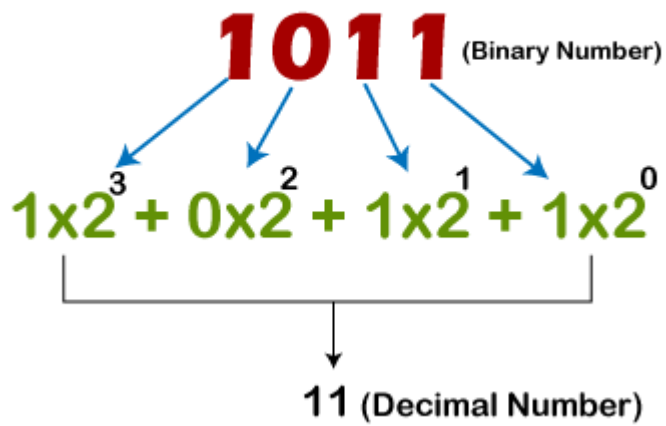
- Step 1: Get number by user
- Step 2: Get the modulus/remainder of the number
- Step 3: sum the remainder of the number
- Step 4: Divide the number by 10
- Step 5: Repeat the step 2 while number is greater than 0.

```
#include<stdio.h>
 int main()
{
        int n,sum=0,temp;
        printf("Enter a number:");
        scanf("%d",&n);
        while(n>0)
        {
                temp=n%10;
                sum=sum+m;
                n=n/10;
        }
        printf("Sum is=%d",sum);
        return 0;
}
```

```
#include <stdio.h>
int main()
{
        int n, reversed = 0, remainder, original;
        printf("Enter an integer: ");
        scanf("%d", &n);
        original = n;
// reversed integer is stored in reversed variable
        while (n != 0)
        {
                remainder = n % 10;
                reversed = reversed * 10 + remainder;
                n /= 10;
        }
}
```

```
// palindrome if orignal and reversed
//are equal
   if (original == reversed)
      printf("%d is a palindrome.",
original);
   else
      printf("%d is not a palindrome.",
original);

   return 0;
}
```

Convert Binary to Decimal number in C



```
#include <stdio.h>
#include <conio.h>
void main()
{
   // declaration of variables
            int num, binary_num, decimal_num = 0, base = 1, rem;
            printf (" Enter a binary number with the combination of 0s and 1s \n");
            scanf (" %d", &num); // accept the binary number (0s and 1s)
            binary_num = num; // assign the binary number to the binary_num variable
             while ( num > 0)
            {
                        rem = num % 10; /* divide the binary number by 10 and store the remainder in rem variable. */
                         decimal_num = decimal_num + rem * base;
                         num = num / 10; // divide the number with quotient
                        base = base * 2;
            }
            printf ( " The binary number is %d \t", binary_num); // print the binary number
            printf (" \n The decimal number is %d \t", decimal_num); // print the decimal
}
```