Operators in C, Type conversion and typecasting.
Decision control and Looping statements: Introduction to decision control, Conditional branching statements, iterative statements, nested loops, break and continue statements, goto statement.

Operator?

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- Operators used in programs to evaluate data and variables.

Operand?

- Operand is a variable or constant which returns an value.
- Example: a + b Here „a" and „b" are operands. + is operator.

Expression: A sequence of operands and operators that reduces to single value is an expression.

- C language supports a lot of operators to be used in expressions. These operators can be categorized into the following major groups:

- Arithmetic operators
- Relational Operators
- Equality Operators
- Logical Operators
- Unary Operators
- Conditional Operators
- Bitwise Operators
- Assignment operators
- Comma Operator
- Sizeof Operator

Arithmetic operators:

This operator used for numeric calculation. These are of either Unary arithmetic operator, Binary arithmetic operator. Where Unary arithmetic operator required only one operand such as +,-, ++, --,!, tiled. And these operators are addition, subtraction, multiplication, division. Binary arithmetic operator on other hand required two operand and its operators are +(addition), -(subtraction), *(multiplication), /(division), %(modulus). But modulus cannot applied with floating point operand as well as there are no exponent operator in c. Unary (+) and Unary (-) is different from addition and subtraction. When both the operand are integer then it is called integer arithmetic and the result is always integer. When both the operand are floating point then it is called floating arithmetic and when operand is of integer and floating point then it is called mix type or mixed mode arithmetic . And the result is in float type.

Arithmetic operators are given below; Let us assume a=9, b=3;

Note: Modulus operator is used to find the reminder of the integer division. This cannot be used on float and double operands.

OPERATION	OPERATOR	SYNTAX	COMMENT	RESULT
Multiply	*	a * b	result = a * b	27
Divide	/	a / b	result = a / b	3
Addition	+	a + b	result = a + b	12
Subtraction	-	a - b	result = a - b	6
Modulus	%	a % b	result = a % b	0

WAP to perform addition, subtraction, division, integer division, multiplication and modulo division of 2 numbers.

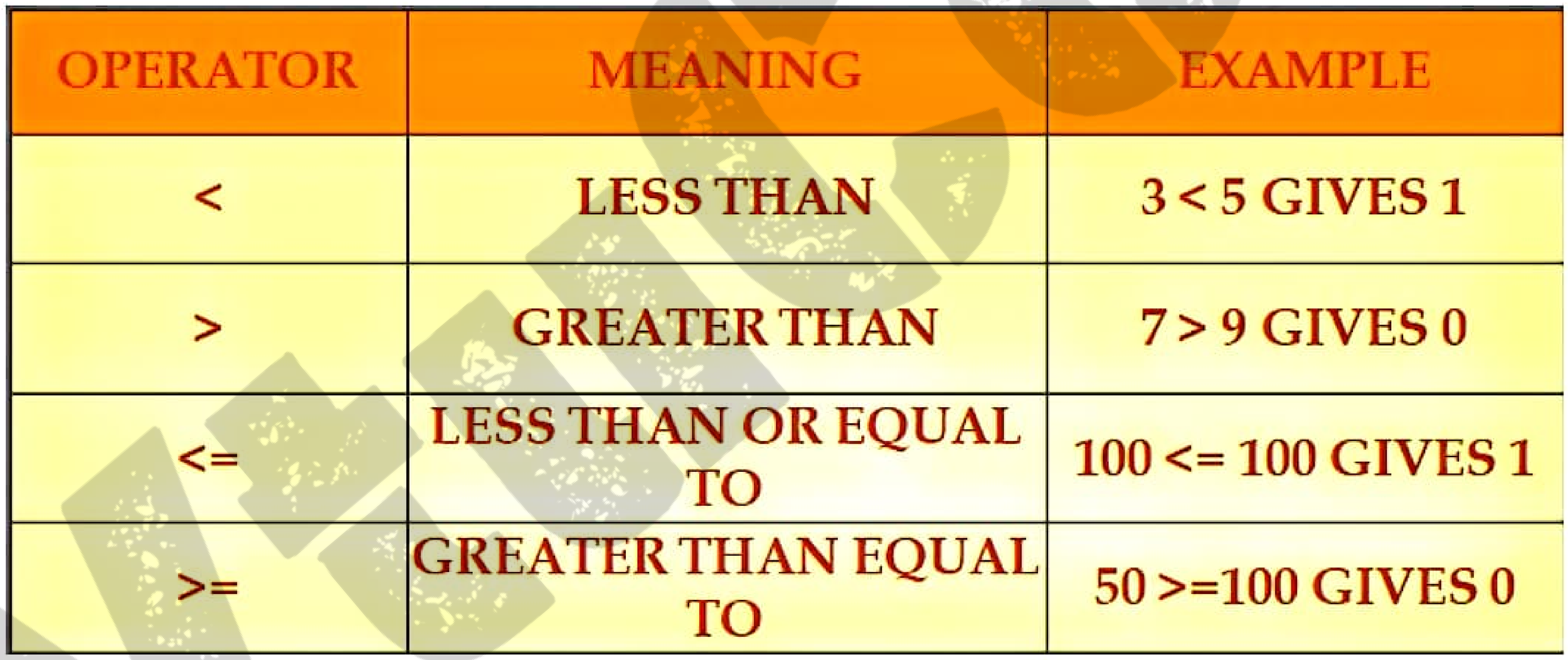
```
#include <stdio.h>

void main()

{

float a=5.0,b=2.0,A,S,M,D;

int ID,R;

A=a+b;//A=7.0

S=a-b;//S=3.0

M=a*b;//M=10.0

D=a/b;//D=2.5

printf("\n%f\n%f\n%f\n%f",A,S,M,D);
int c,d;

c=a;

d=b;

ID=c/d;//ID=2

R=c%d;//R=1

printf("\n%d\n%d",ID,R);

}
```

Relational operators:

• Also known as a comparison operator, it is an operator that compares two values.

• Expressions that contain relational operators are called relational expressions.

• Relational operators return true or false value, depending on whether the conditional relationship between the two operands holds or not.



OPERATOR	MEANING	EXAMPLE
<	LESS THAN	3 < 5 GIVES 1
>	GREATER THAN	7 > 9 GIVES 0
<=	LESS THAN OR EQUAL TO	100 <= 100 GIVES 1
>=	GREATER THAN EQUAL TO	50 >=100 GIVES 0

Equality operators

• C language supports two kinds of equality operators to compare their operands for strict equality or inequality. They are equal to (==) and not equal to (!=) operator.

• The equality operators have lower precedence than the relational operators.

OPERATOR	MEANING
==	RETURNS 1 IF BOTH OPERANDS ARE EQUAL, 0 OTHERWISE
!=	RETURNS 1 IF OPERANDS DO NOT HAVE THE SAME VALUE, 0 OTHERWISE

Example to showcase Relational and Equality operators

```
#include<stdio.h>

void main()

{

int a=10,b=20;

printf("\n %d < %d = %d",a,b,a<b);//1

printf("\n %d > %d = %d",a,b,a>b);//0

printf("\n %d <= %d = %d",a,b,a<=b);//1

printf("\n %d > %d = %d",a,b,a>=b);//0

printf("\n %d == %d = %d",a,b,a==b);//0

printf("\n %d != %d = %d",a,b,a!=b);//1

}
```

Logical operators

• Operator used with one or more operand and return either value zero (for false) or one (for true). The operand may be constant, variables or expressions. And the expression that combines two or more expressions is termed as logical expression.

• C language supports three logical operators. They are- Logical AND (&&), Logical OR (||) and Logical NOT (!).

• As in case of arithmetic expressions, the logical expressions are evaluated from left to right.

A	B	A &&B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

A	! A
0	1
1	0

Example:

```
#include <stdio.h>
void main()
{
int a, b;
printf("Enter the value of a (0 or 1): ");
scanf("%d", &a);//Input a=0
printf("Enter the value of b (0 or 1): ");
scanf("%d", &b);//Input b=1
printf("Logical AND (a && b): %d\n", a && b);//0
printf("Logical OR (a || b): %d\n", a || b);//1
printf("Logical NOT (!a): %d, !b: %d\n", !a, !b);//0
}
```

Example using equality and logical operators

```
#include <stdio.h>
void main()
{
int  a=5,b=5,c=10,R;
R=(a==b)&&(c>b);
printf("(a==b)&&(c>b) is %d\n",R);//1
R=(a==b)&&(c<b);
printf("(a==b)&&(c<b) is %d\n",R);//0
R=(a==b)||(c<b);

printf("(a==b)||(c<b) is %d\n",R);//1

R=!(a!=b);

printf("!(a!=b) is %d\n",R);//1

R=!(a==b);

printf("!(a==b)  is  %d\n",R);//0

}
```

Unary operator

• Unary operators act on single operands.

• C language supports three unary operators. They are unary minus(-), increment(++) and decrement operators(--).

• The increment operator is a unary operator that increases the value of its operand by 1.

• Similarly, the decrement operator decreases the value of its operand by 1.

Unary minus:

• When an operand is preceded by a minus sign, the unary operator negates its value.

Example:

```
#include <stdio.h>

void main()

{

int a,b=10;

a=-(b);

printf("%d",a);//-10

}
```

Postfix increment/decrement operator:

Example:

```
y=x++
```

Can also be written as

y=x;

x=x+1;

y=x-- can also be written as

y=x;

x=x-1;

Prefix increment/decrement operator:

Example:

y=++x

Can also be written as

x=x+1;

y=x;

y=--x can also be written as

x=x-1;

y=x;

Example:

```
#include <stdio.h>
void main()
{
int a=10;
printf("++a=%d\n",++a);//1
printf("--a=%d\n",--a);//10
printf("a++=%d\n",a++);//10
printf("a--=%d\n",a--);//11
}
```

Conditional operator:

- The conditional operator operator (?:) is just like an if .. else statement that can be written within expressions.
- The syntax of the conditional operator is

exp1 ? exp2 : exp3

Here, exp1 is evaluated first. If it is true then exp2 is evaluated and becomes the result of the expression, otherwise exp3 is evaluated and becomes the result of the expression.

Example: To check if eligible to vote or not:

```
#include <stdio.h>

void main()

{

int age=17;

(age>=18)?printf("Eligible to vote"):printf("Not Eligible to vote");

}
```

Example: To find largest of 2 numbers.

```
#include <stdio.h>

void main()

{

int a=10,b=5,c=16,large;

large=(a>b)?a:b;

printf("%d",large);

}
```

Example: To find largest of 3 numbers.

```
#include <stdio.h>

void main()

{

int a=10,b=5,c=16,large;
large=(a>b)?(a>c?a:c):(b>c?b:c);

printf("%d",large);

}
```

Bitwise operator

- Bitwise operators perform operations at bit level. These operators include: bitwise AND, bitwise OR, bitwise XOR and shift operators.

- **The bitwise AND operator** (&) is a small version of the boolean AND (&&) as it performs operation on bits instead of bytes, chars, integers, etc. Bit in the first operand is ANDed with corresponding bit in the second operand.

- **The bitwise OR operator (|)** is a small version of the boolean OR (||) as it performs operation on bits instead of bytes, chars, integers, etc. Bit in the first operand is ORed with corresponding bit in the second operand.

- **The bitwise NOT (~),** or complement, is a unary operation that performs logical negation on each bit of the operand. By performing negation of each bit, it actually produces the ones' complement of the given binary value.

- **The bitwise XOR operator (^)** performs operation on individual bits of the operands. The result of XOR operation is shown in the table

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

- **Bitwise left shift(<<):** When we apply left shift, every bit in the operand is shifted to left by one place.

Example: if x= 0001 1101

then x<<1=0011 1010.

if x= 0001 1101

then x<<4=1101 0000

- **Bitwise right shift(>>):** When we apply right shift, every bit in the operand is shifted to right by one place.

Example: if x= 0001 1101

then x>>1=0000 1110

if x= 0001 1101

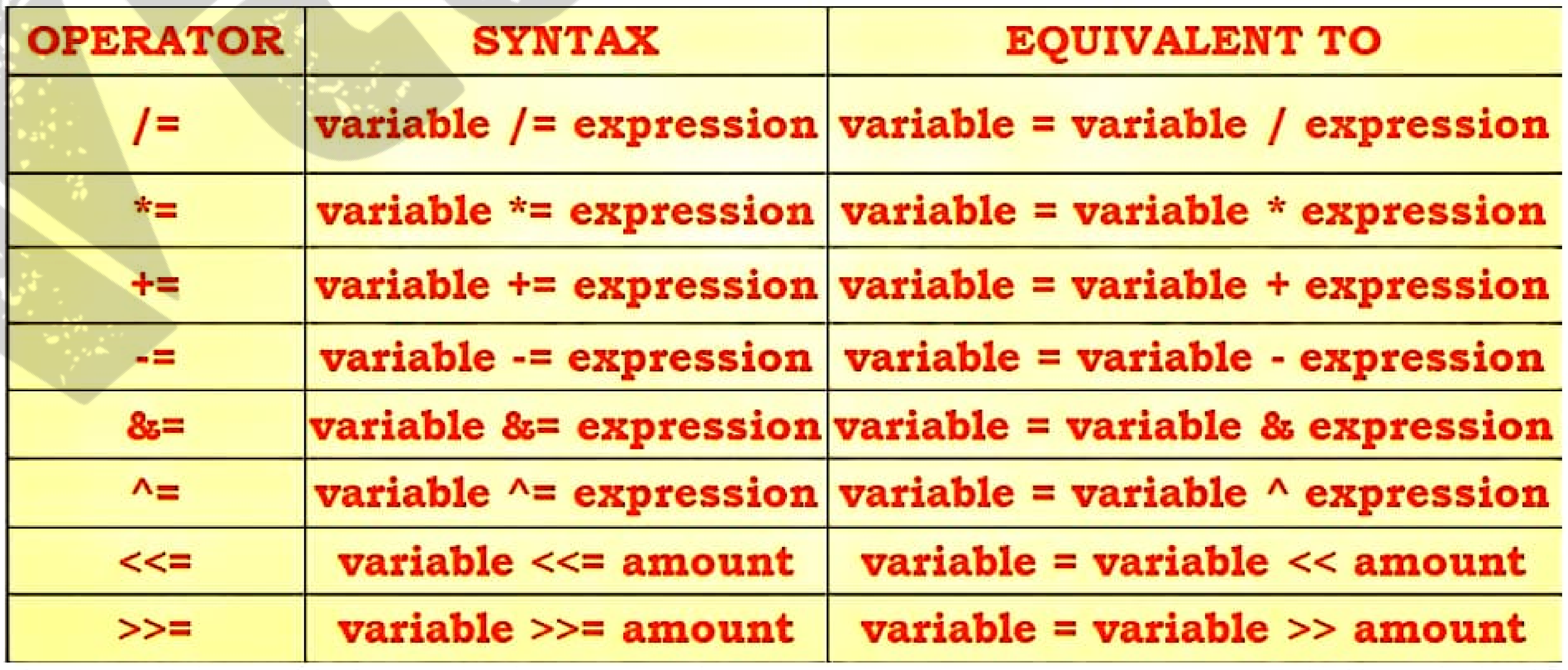
then x>>4=0000 0001

Example of Bitwise operator:

```
#include<stdio.h>
void main()
{
int a=27,b=39;
printf("a&b=%d\n",a&b);//3
printf("a|b=%d\n",a|b);//63
printf("~a=%d\n",~a);//-28
printf("a^b=%d\n",a^b);//60
printf("a<<1=%d\n",a<<1);//54
printf("b>>1=%d\n",b>>1);//19
}
```

Assignment operator

- The assignment operator is responsible for assigning values to the variables.
- While the equal sign (=) is the fundamental assignment operator, C also supports other assignment operators that provide shorthand ways to represent common variable assignments. They are shown in the table.



OPERATOR	SYNTAX	EQUIVALENT TO
/=	variable /= expression	variable = variable / expression
*=	variable *= expression	variable = variable * expression
+=	variable += expression	variable = variable + expression
-=	variable -= expression	variable = variable - expression
&=	variable &= expression	variable = variable & expression
^=	variable ^= expression	variable = variable ^ expression
<<=	variable <<= amount	variable = variable << amount
>>=	variable >>= amount	variable = variable >> amount

```
#include <stdio.h>
void main()
{   int a=5,c=10;
printf("c/=a=%d\n",c/=a); //c=c/a,c=2
printf("c*=a=%d\n",c*=a); //c=c*a,c=10
printf("c-=a=%d\n",c-=a); //c=c-a,c=5
printf("c+=a=%d\n",c+=a); //c=c+a,c=10
printf("c&=a=%d\n",c&=a); //c=c&a,c=0
printf("c^=a=%d\n",c^=a); //c=c^a,c=5
printf("c<<=a=%d\n",c<<=a); //c=c<<a,c=160
printf("c>>=a=%d\n",c>>=a); //c=c>>a,c=5}
```

Comma operator

- The comma operator in C takes two operands.
- It works by evaluating the first and discarding its value, and then evaluates the second and returns the value as the result of the expression.
- Comma separated operands when chained together are evaluated in left-to-right sequence with the right-most value yielding the result of the expression.
- Among all the operators, the comma operator has the lowest precedence. For example,

```
int a=2, b=3, x=0;
x = (++a, b+=a);
++a=> a=a+1
      a=3
b+=a=> b=b+a
        b=3+3=6
Now, the value of x = 6.
```

Size of operator

- The comma operator in C takes two operands.

- sizeof is a unary operator used to calculate the sizes of data types.
- It can be applied to all data types.
- The operator returns the size of the variable, data type or expression in bytes.
- 'sizeof' operator is used to determine the amount of memory space that the variable/expression/data type will take.

```
#include <stdio.h>

void main()

{

int a;

float b;

double c;

char d;

printf("size=%u\n",sizeof(a));

printf("size=%u\n",sizeof(b));

printf("size=%u\n",sizeof(c));

printf("size=%u\n",sizeof(d));}
```

Type Conversion

- Type conversion and type casting of variables refers to changing a variable of one data type into another.
- While type conversion is done implicitly, casting has to be done explicitly by the programmer. We will discuss both of them here.
- Type conversion is done when the expression has variables of different data types. So to evaluate the expression, the data type is promoted from lower to higher level where the hierarchy of data types can be given as: double, float, long, int, short and char.
- For example, type conversion is automatically done when we assign an integer value to a floating point variable.

For ex,

```
float x;

int y = 3;
```

x = y;

Now, x = 3.0,

Type Casting

- Type casting is also known as forced conversion. It is done when the value of a higher data type has to be converted in to the value of a lower data type. For example, we need to explicitly type cast an integer variable into a floating point variable.

float salary = 10000.00;

int sal;

sal = (int) salary;

- Typecasting can be done by placing the destination data type in parentheses followed by the variable name that has to be converted.

Decision control and Looping statements: Introduction to decision control,

Decision control statements:

- Decision control statements are used to alter the flow of a sequence of instructions.
- These statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not.
- Generally C program statement is executed in a order in which they appear in the program. But sometimes we use decision making condition for execution only a part of program that is called control statement.
- Control statement defined how the control is transferred from one part to the other part of the program.
- These decision control statements include

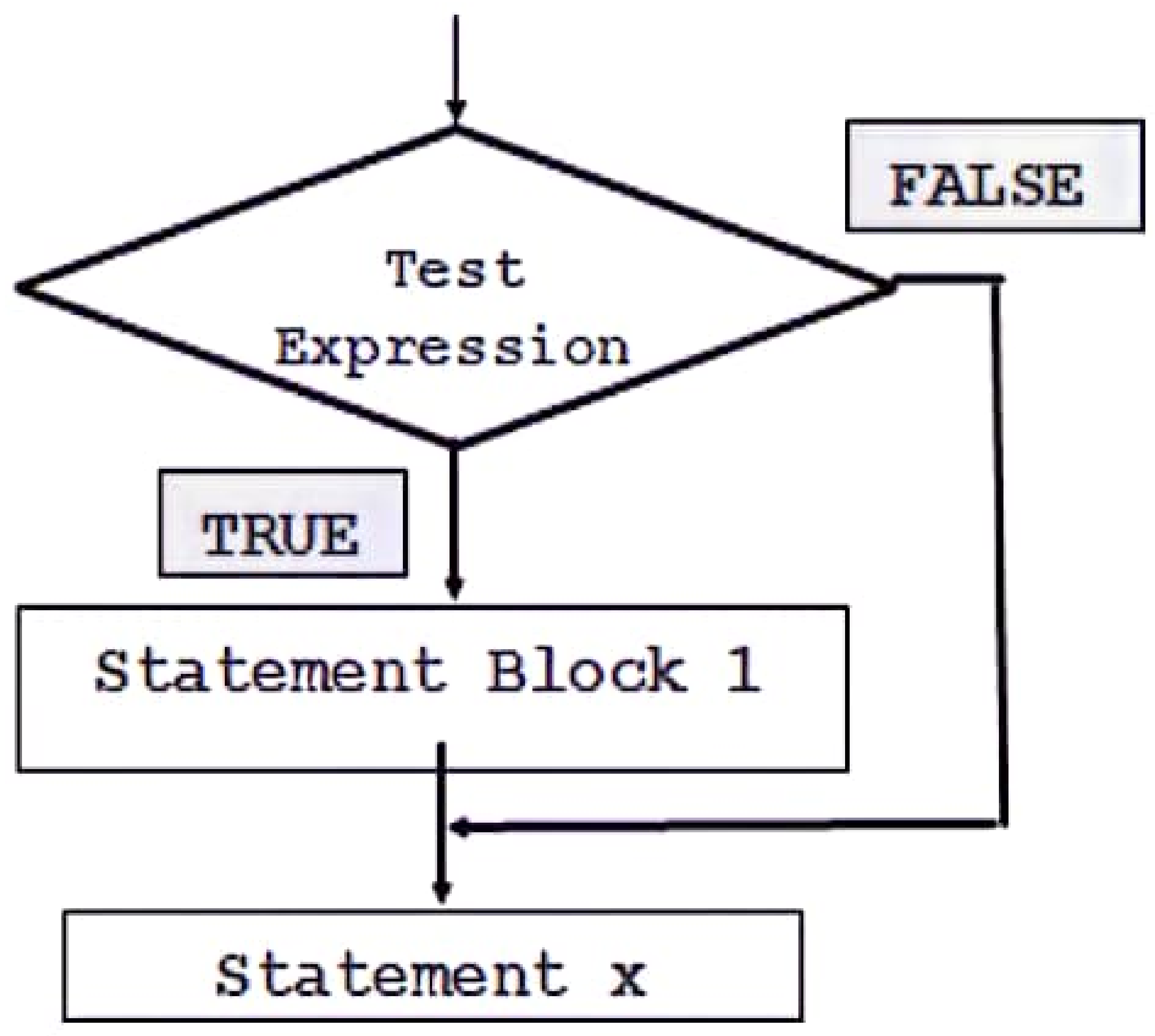
1. If statement
2. If else statement
3. If else if statement
4. Switch statement

if-Statement

- If statement is the simplest form of decision control statements that is frequently used in decision making.
- First the test expression is evaluated. If the test expression is true, the statement of if block (statement 1 to n) are executed otherwise these statements will be skipped and the execution will jump to statement x.

SYNTAX OF IF STATEMENT

```
if (test expression)
{
    statement 1;
    .............
    statement n;
}
statement x;
```



WAP to determine whether a person is eligible to vote or not.

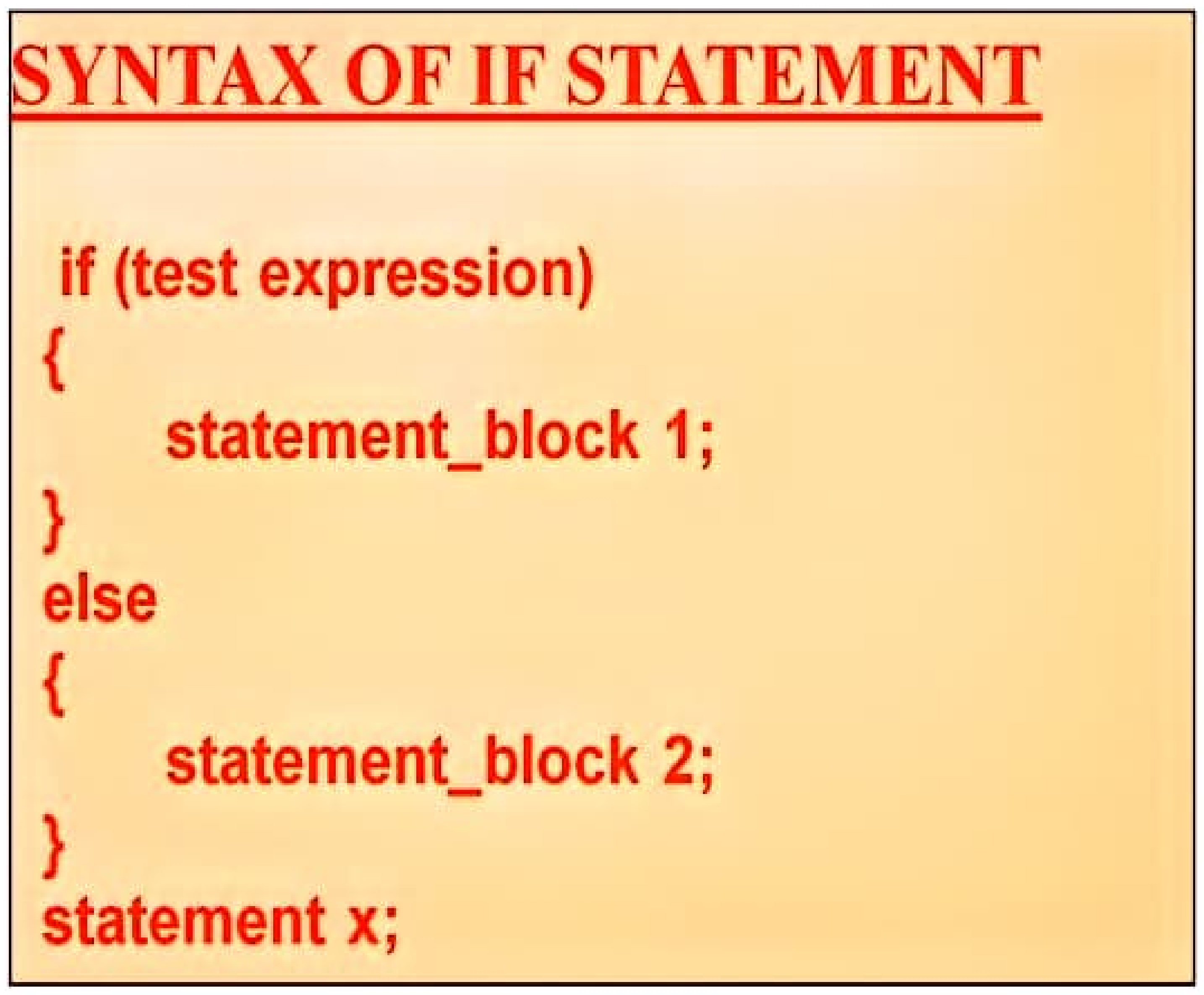
```
#include<stdio.h>
void main()
{
int age;
printf("Enter the age\n");
scanf("%d",&age);
if(age>=18)
{
printf("He/she is eligible to vote");
}
}
```

WAP to determine the character entered by the user.

```
#include<stdio.h>
void main()
{
char ch;
printf("Enter the character\n");
scanf("%c",&ch);
if(ch>0)
{
printf("The character has been entered");
} }
```

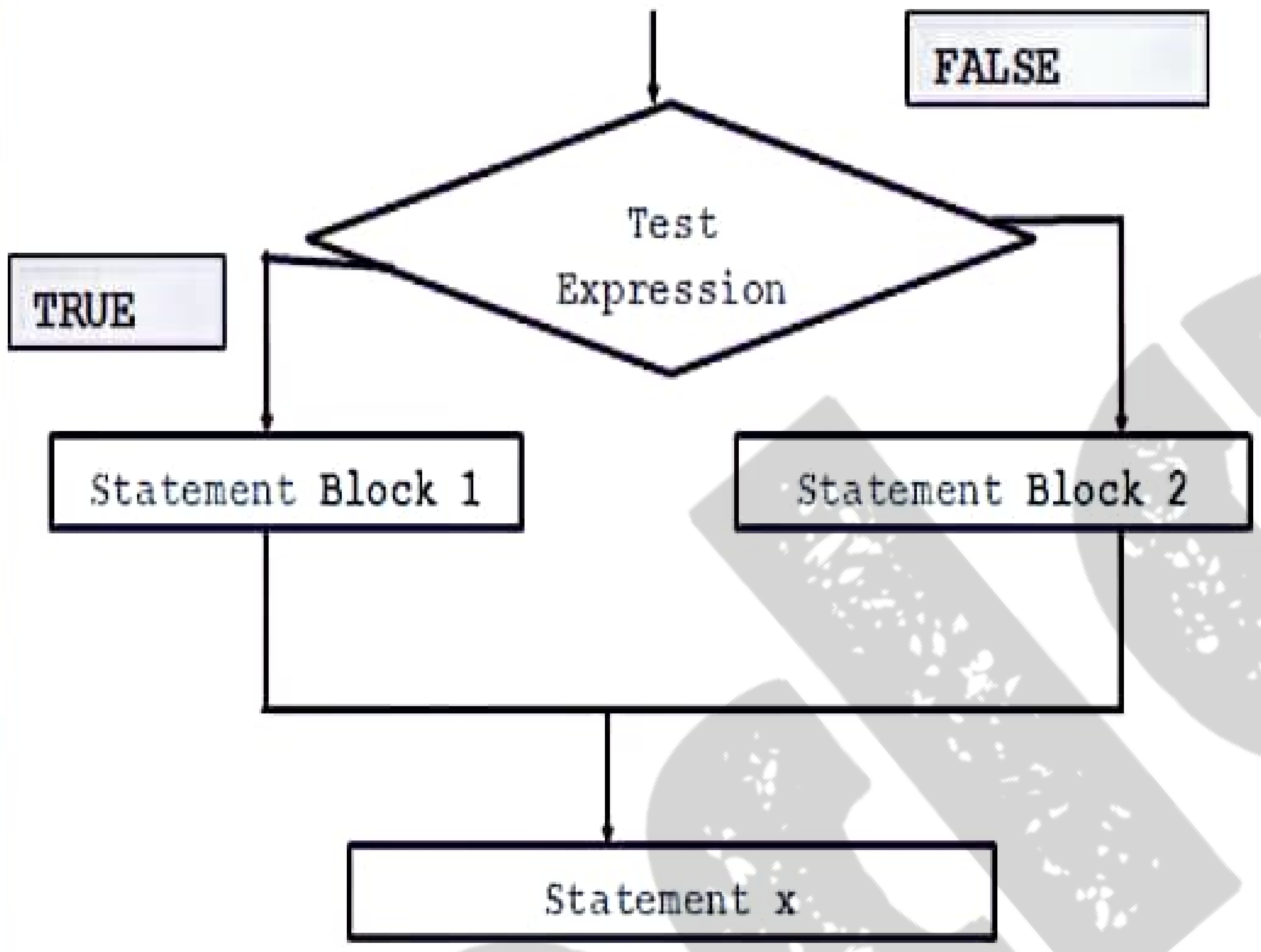
if-else Statement

In the if-else construct, first the test expression is evaluated. If the expression is true, statement block 1 is executed and statement block 2 is skipped. Otherwise, if the expression is false, statement block 2 is executed and statement block 1 is ignored. In any case after the statement block 1 or 2 gets executed the control will pass to statement x. Therefore, statement x is executed in every case.



SYNTAX OF IF STATEMENT

```
if (test expression)
{
    statement_block 1;
}
else
{
    statement_block 2;
}
statement x;
```



WAP to find if the given number is even or odd.

```
#include <stdio.h>
void main()
{
int a;
printf("Enter the number\n");
scanf("%d",&a);
if(a%2==0)
{       printf("the number is even");    }
else
{       printf("the number is odd");    }
}
```

WAP to find if the given character is a vowel or a consonant.

```
#include <stdio.h>
void main()
```

```
{
    char ch = 'C';
    if (ch == 'a' || ch == 'A' || ch == 'e' || ch == 'E'|| ch == 'i' || ch == 'I' || ch == 'o' ||
ch == 'O'|| ch == 'u' || ch == 'U')
     { printf("The character %c is a vowel.\n", ch); }
    else
     { printf("The character %c is a consonant.\n", ch); }
}
```

WAP to find largest of 2 numbers.

```
#include <stdio.h>
void main()
{
int a=100,b=20;
if(a<b)
printf("b is largest of 2 numbers");
else
printf("a is largest of 2 numbers");
}
```

WAP to enter any character. If entered in upper case then display in lower case and vice versa.

```
#include<stdio.h>
void main()
{
    char ch;
    printf("Enter the character\n");
    scanf("%c",&ch);
    if(ch>='A'&&ch<='Z')
    {
        printf("The entered character in lower case is %c",(ch+32));
    }
    else
    {
        printf("The entered character in upper case is %c",(ch-32));
    }
}
```

WAP to find if the given year is leap year or not.

The year number must be divisible by four – except for end-of-century years, which must be divisible by 400.

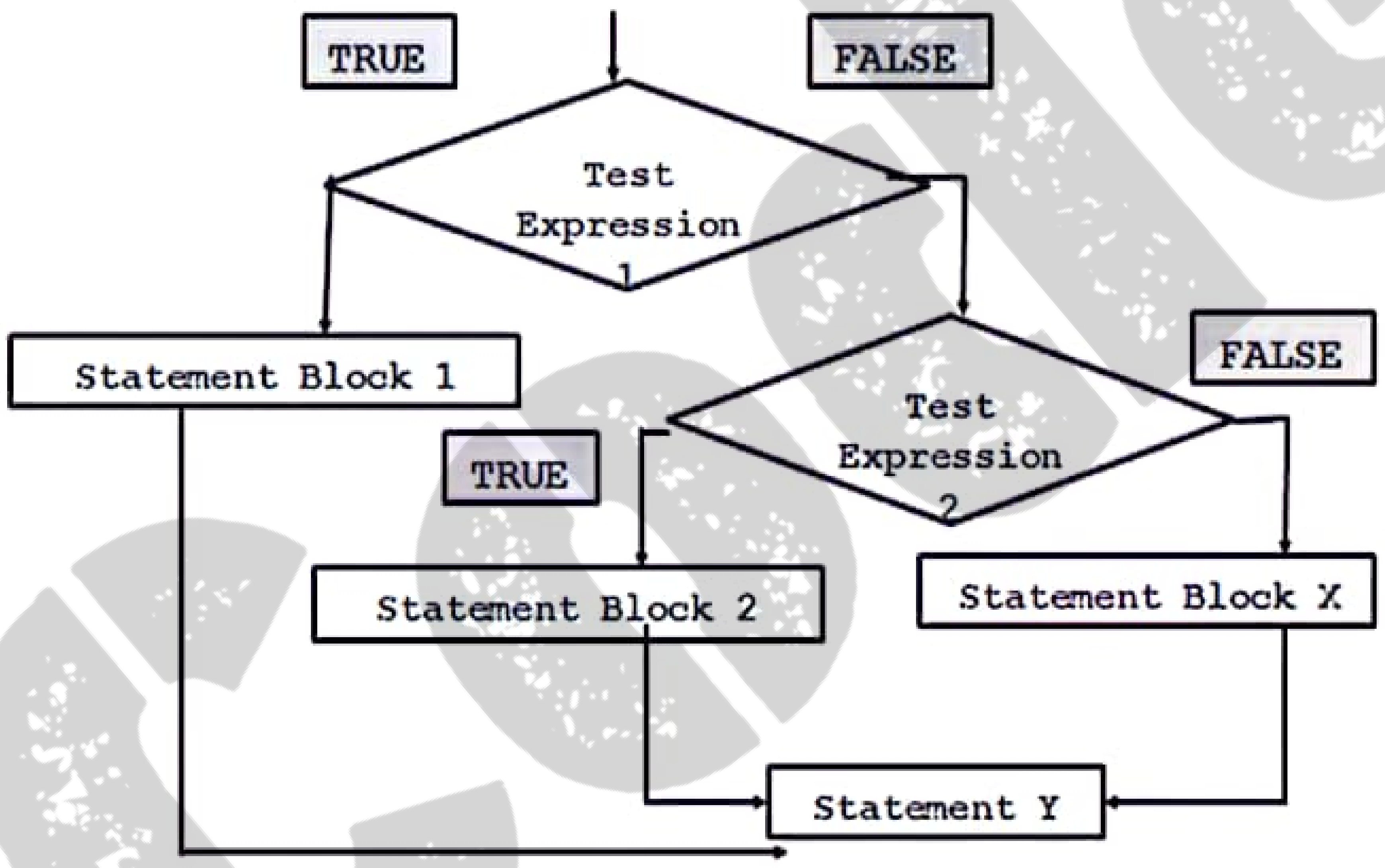
Example: This means that the year 2000 was a leap year, although 1900 was not

```
#include<stdio.h>
void main()
{
    int year;
    printf("Enter any year\n");
    scanf("%d",&year);
    if((year%4==0)&&(year%100!=0)||(year%400==0))
    {
        printf("Leap year");
    }
    else
    {
        printf("Not a leap year");
    }
}
```

if-else if Statement

- C language supports if else if statements to test additional conditions apart from the initial test expression. The if-else-if construct works in the same way as a normal if statement.
- This process continue until there is no if statement in the last block. if one of the condition is satisfy the condition other nested "else if" would not executed. But it has disadvantage over if else statement that, in if else statement whenever the condition is true, other condition are not checked. While in this case, all condition is checked.

```
SYNTAX OF IF-ELSE STATEMENT

if ( test expression 1)
{
        statement block 1;
}
else if ( test expression 2)
{
        statement block 2;
}
..........................
else if (test expression N)
{
        statement block N;
}
else
{
        Statement Block X;
}
Statement Y;
```



WAP to demonstrate if-else-if statement.

```
void main()
{
    int a,b;
    printf("Enter the value of a and b\n");
    scanf("%d,%d",&a,&b);
    if(a==b)
    printf("a and b are equal");
    else if(a>b)
    printf("a is greater than b");
    else
    printf("a is less than b");     }
```

Condition called Pitfall

It is a condition where in while using if statement, instead of using equal to(==) assignment operator, we use = to sign. Here instead of checking the condition, the value is assigned to the variable.

Example:

```
void main()
{
    int a;
    printf("Enter the value\n");
    scanf("%d",&a);
    if(a=10)//The condition should e a==10. Here instead of checking the condition ,the
value is being assigned to a.
    printf("Equal");
    else
    printf("not equal");
}
```

WAP to check if the number is positive,negative, or equal to zero

```
void main()
{
    int a;
    printf("Enter the value of a and b\n");
    scanf("%d",&a);
    if(a>0)
    printf("The number is positive");
    else if(a<0)
    printf("The number is negative");
    else
    printf("The numer is eqaul to zero");   }
```

A company decides to give bonus to its employee. If the employee is male he gets 5% bonus, if female she gets 10% bonus. If salary is less than 10,000, he/she gets extra 2% bonus on salary. WAP to calculate the bonus and display the salary which has to be given to the employee

```
#include<stdio.h>
void main()
{
   char ch;
   float sal,bonus,amt;
   printf("Enter the sex of emp\n");
   scanf("%c",&ch);
   printf("Enter the salary\n");
   scanf("%f",&sal);
   if(ch=='m')
   bonus=0.05*sal;
   else
   bonus=0.10*sal;
   if(sal<10000)
   bonus=bonus+0.20*sal;
   amt=sal+bonus;
   printf("The salary which employee will get is %f",amt);
}
```

WAP to display the examination result:

Marks>=75=Distinction

<75 and >=60=First class

<60 and >=50=Second class

<50 and >=40=Third class

<40=Fail

```
void main()
{     int marks;
printf("Enter the marks\n");
scanf("%d",&marks);
if(marks>=75)
printf("Distinction");
else if(marks>=60&&marks<75)
printf("First class");
else if(marks>=50&&marks<60)
printf("Second class");
else if(marks>=50&&marks<40)
printf("Third class");
else
printf("Fail");     }
```

WAP to find largest of 3 numbers.

```
void main(){
    int a,b,c;
    printf("Enter a,b,c\n");
    scanf("%d,%d,%d",&a,&b,&c);
    if(a>b)
    {
        if(a>c)
        printf("a is greater");
        else
        printf("c is greater");
    }
    else  if(b>c)
    printf("b is greater");
    else
    printf("c is greater");
}
```

WAP to find largest of 3 numbers using logical and operator

```
#include<stdio.h>
void main()
{
    int a,b,c;
    printf("Enter a,b,c\n");
    scanf("%d,%d,%d",&a,&b,&c);
    if(a>b&&a>c)
    printf("a is greater");
    if(b>a&&b>c)
    printf("b is greater");
    if(c>a&&c>b)
    printf("c is greater");
}
```

WAP to enter the marks of 4 subject out of 100 and then calculate total,average and display the grade obtained.

```
void main() {
    int m1,m2,m3,m4,total;
    float avg;
    printf("Enter the marks\n");
    scanf("%d,%d,%d,%d",&m1,&m2,&m3,&m4);
    total=m1+m2+m3+m4;
    avg=total/4;
    printf("The total of 4 sujects are %d\n",total);
    printf("The average of 4 subjects are %f\n",avg);
    if(avg>=75)
    printf("Distinction");
    else if(avg>=60&&avg<75)
    printf("First class");
    else if(avg>=50&&avg<60)
    printf("Second class");
    else if(avg>=50&&avg<40)
    printf("Third class");
    else
    printf("Fail");
}
```

Rules to use Decision control statements

- **Rule 1: The expression must be enclosed in parentheses.**
- **Rule 2: No semicolon is placed after the if/if-else/if-else-if statement. Semicolon is placed only after inside the statement block if required.**
- **Rule 3: A statement block starts and ends with curly braces. No semicolon is placed after the opening/closing of braces.**

Switch case

A switch case statement is a multi-way decision statement.

- Switch statements are used:

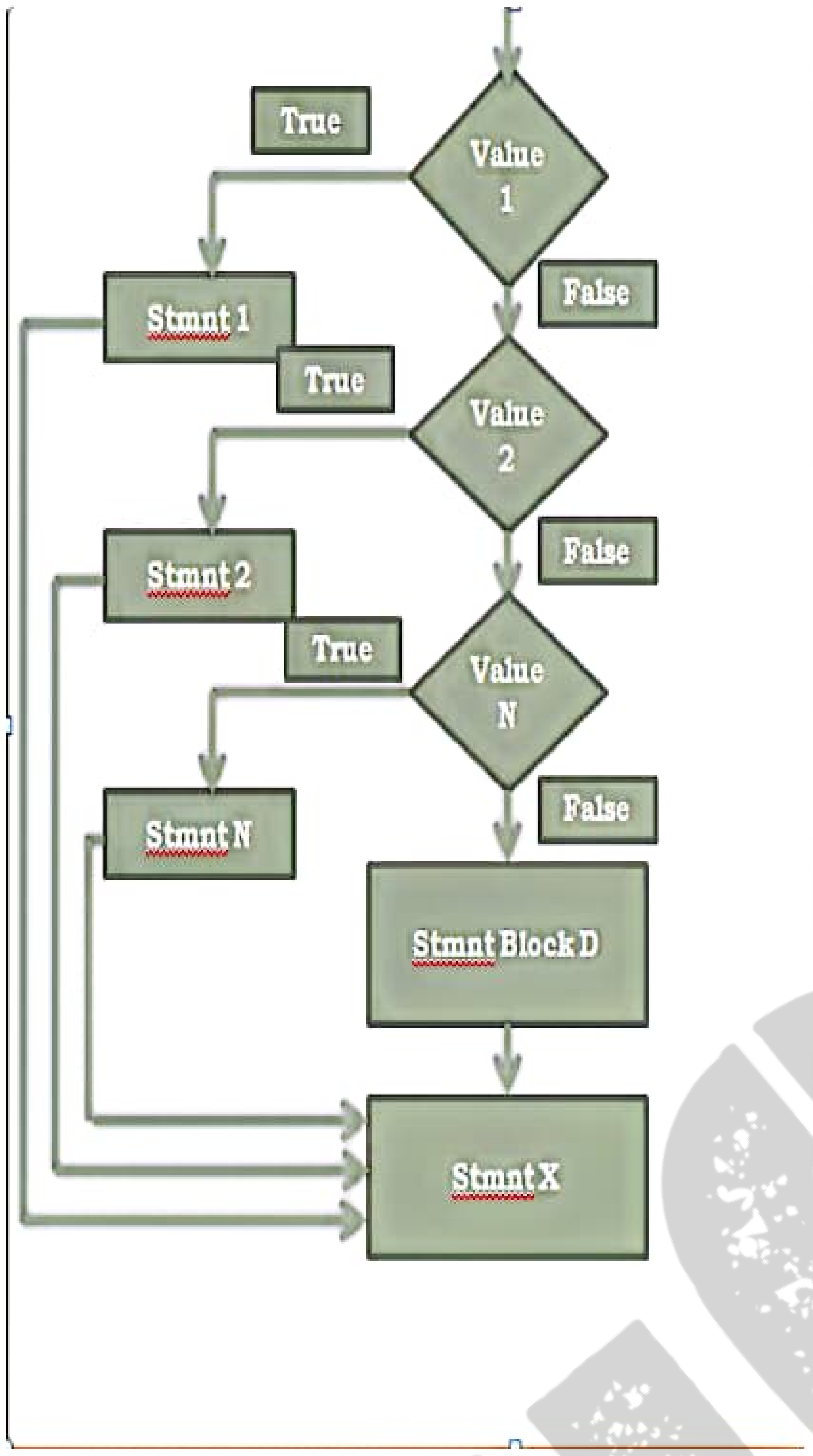
✓ When there is only one variable to evaluate in the expression

✓ When many conditions are being tested for

- Switch case statement advantages include:

1. Easy to debug, read, understand and maintain

2. Execute faster than its equivalent if-else construct



Syntax of Switch statement:

```
switch(variable)
{
case value1:
        Stmnt Block 1;
        break;
case value2:
        Stmnt Block 2;
        break;
........
case valueN:
        Stmnt Block N;
        break;
default:
        Stmnt block D;
        break;
}
Statement x;
```

How switch case works?

Step 1: The switch variable is evaluated.

Step 2: The evaluated value is matched against all the present cases.

Step 3A: If the matching case value is found, the associated code is executed.

Step 3B: If the matching code is not found, then the default case is executed if present.

Step 4A: If the break keyword is present in the case, then program control breaks out of the switch statement.

Step 4B: If the break keyword is not present, then all the cases after the matching case are executed.

Step 5: Statements after the switch statement are executed.

Example:

```
char grade='C';
 switch(grade)
                  {         case  'A':
                                    printf("\n Excellent");
                                    break;
                            case  'B':
                                    printf("\n Good");
                                    break;
                            case 'C':
                                    printf("\n Fair");
                                    break;
                            default:
                                    printf("\n Invalid Grade");
                                    break;
            }
```

WAP to check if character is vowel or not.

```
char ch;
printf("Enter the character\n");
scanf("%c",&ch);
switch(ch)
{
case  'A':
case 'a':printf("%c is vowel",ch);
        break;
case 'E':
```

```
case 'e':printf("%c is vowel",ch);
        break;
case  'I':
case 'i':printf("%c is vowel",ch);
        break;
case 'O':
case 'o':printf("%c is vowel",ch);
        break;
case 'U':
case 'u':printf("%c is vowel",ch);
        break;
default:printf("%c is not a vowel",ch);
}
```

WAP to enter a number from 1-7 and display the corresponding day of the week using switch case.

```
void main() {
int day;
printf("Enter any number from 1 to 7: \n");
scanf("%d",&day);
switch(day)     {
      case 1:printf("\nSunday");
             break;
   case 2:printf("\nMonday");
            break;
   case 3:printf("\nTuesday");
            break;
   case  4:printf("\nWednesday");
```

```
                break;
    case 5:printf("\nThursday");
               break;
    case 6:printf("\nFriday");
               break;
    case 7:printf("\nSaturday");
          break;
    default:printf("You have entered wrong number");
}
```

WAP to check if the number is even or odd.

```
#include<stdio.h>
void main()
{
int num,rem;
printf("Enter any number: \n");
scanf("%d",&num);
rem=num%2;
switch(rem)
{
case 0:printf("\nEven");
break;
case 1:printf("\nOdd");
break;
}}
```

Advantages of using SWITCH case:

- **Easy to debug.**

- **Easy to read and understand**
- **Ease of maintenance as compared with its equivalent if-else.**
- **switch case can also be nested.**
- **Executes faster than if-else.**

Iterative statements

Iterative statements are used to repeat the execution of a list of statements, depending on the value of an integer expression. In this section, we will discuss all these statements.

- While loop
- Do-while loop
- For loop

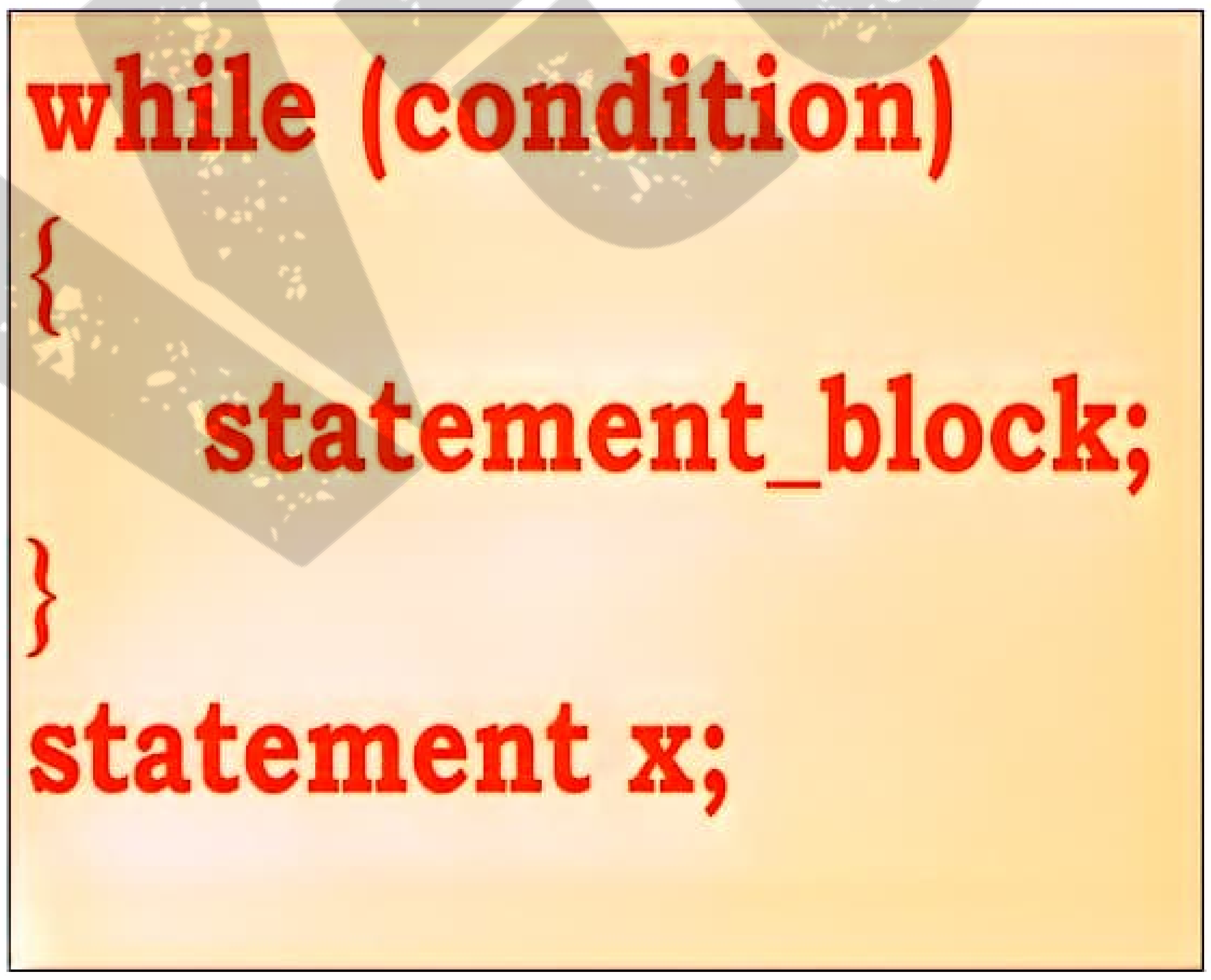
while loop

The while loop is used to repeat one or more statements while a particular condition is true.

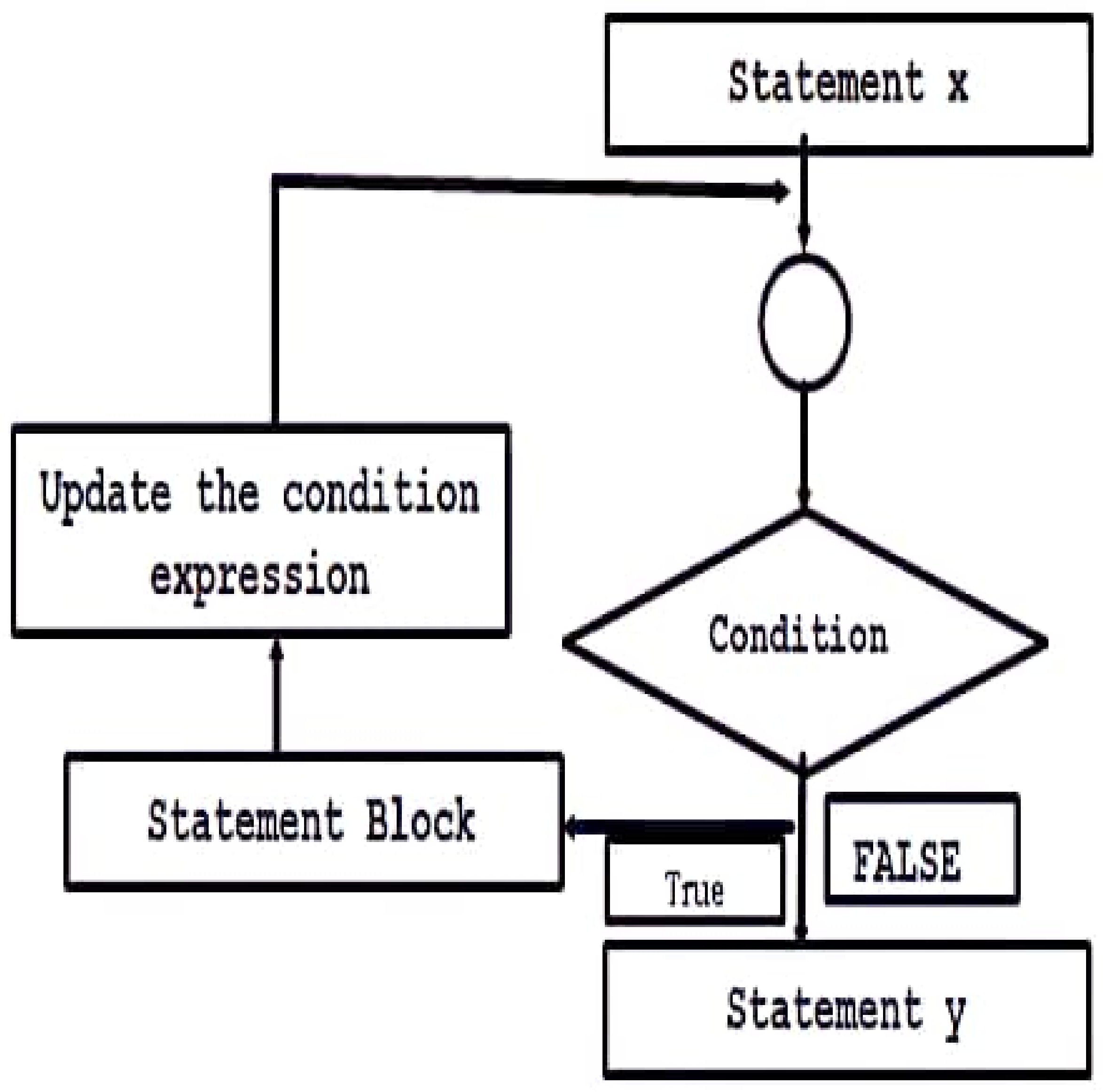
In the while loop, the condition is tested before any of the statements in the statement block is executed.

If the condition is true, only then the statements will be executed otherwise the control will jump to the immediate statement outside the while loop block.

We must constantly update the condition of the while loop.



```
while (condition)
{
        statement_block;
}
statement x;
```



WAP to calculate the sum of first 10 numbers.

```
#include<stdio.h>
void main()
{
int i=1,sum=0;
while(i<=10)
{
sum=sum+i;
i++;
}
printf("\nsum=%d",sum);
}
```

WAP to print 20 horizontal asterisks.

```
#include<stdio.h>
void main()
{
int i=1;
while(i<=20)
{
printf("*");
 i++;
}
}
```

WAP to calculate the sum of numbers from m to n

```
{
int i,j,sum=0;
printf("Enter value of i: ");
scanf("%d",&i);
printf("Enter value of j: ");
scanf("%d",&j);
while(i<=j)
{
sum=sum+i;
i++;
}
printf("\nsum=%d",sum);
}
```

WAP to display the largest of 5 numbers using ternary operator.

```
{
int i=1,large,num;
while(i<=5)
{
printf("Enter the number");
scanf("%d",&num);
large=num>large?num:large;
i++;
}
printf("\n The largest of 5 numbers is:%d",large);
}
```

WAP to calculate the average of numbers entered by the user.

```
void main()
{ int num,sum,count;
float avg;
printf("Enter the number. Enter -1 to stop:");
scanf("%d",&num);
while(num!=-1)
{
count++;
sum+=num;
printf("Enter the number. Enter -1 to stop:");
scanf("%d",&num);
}
avg=(float)sum/count;
printf("sum=%d\n",sum);
printf("Avg=%f",avg); }
```

do-while loop

The do-while loop is similar to the while loop. The only difference is that in a do-while loop, the test condition is tested at the end of the loop.

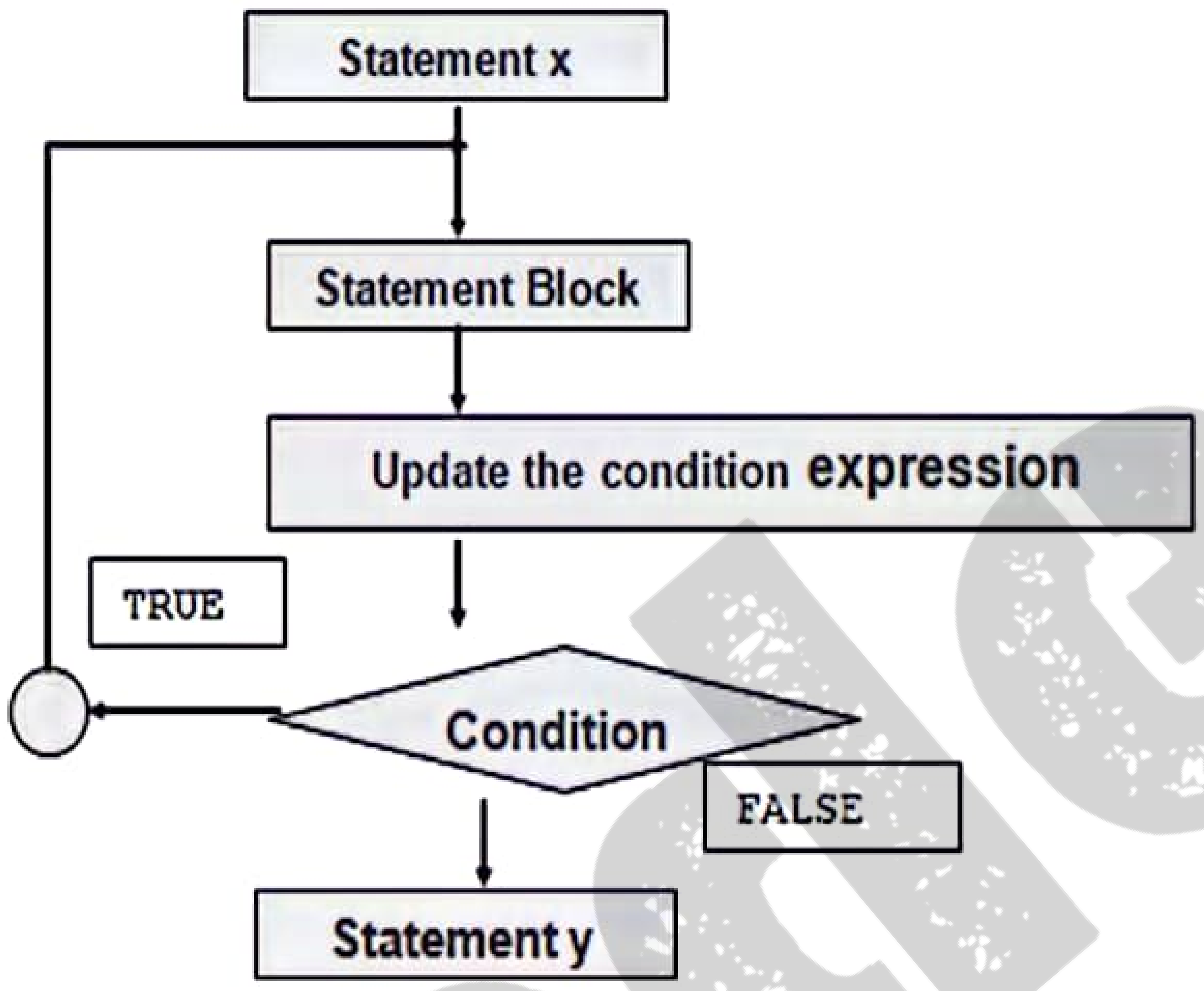
The body of the loop gets executed at least one time (even if the condition is false).

The do while loop continues to execute whilst a condition is true. There is no choice whether to execute the loop or not. Hence, entry in the loop is automatic there is only a choice to continue it further or not.

The major disadvantage of using a do while loop is that it always executes at least once, so even if the user enters some invalid data, the loop will execute.

do-while loops are widely used to print a list of options for a menu driven program.

```
Statement x;
do
{
      statement_block;
} while (condition);

statement y;
```



WAP to calculate the average of first n numbers.

```
Void main()
{   int n,i=1,sum;
float avg;
printf("\nEnter the value of n:");
scanf("%d",&n);
do
{
sum+=i;
i++;
}while(i<=n);
avg=(float)sum/n;
printf("Sum=%d\n",sum);
printf("Avg=%f",avg); }
```

WAP to display square and cube of first n natural numbers.

```
void main()
{   int i,n;
printf("Enter the value of n:");

scanf("%d",&n);

i=1;

do

{

printf("\n|\t%d\t|\t%d\t|\t%d\t|",i,i*i,i*i*i);

i++;

}while(i<=n);      }
```

WAP to list the leap year between 1900 to 1920.

```
void main()

{   int  m=1900,n=1920;

do

{

if(((m%4==0)&&(m%100!=0))||(m%400==0))

printf("%d is Leap year\n",m);

m++;

}while(m<=n);    }
```

WAP to read a character until * is encountered. Also count the number of upper case,lower case and numbers entered.

```
void main()

{   char  ch;

int lowers=0,uppers=0,numbers=0;

printf("Enter any charater\n");

scanf("%c",&ch);

do

{

if(ch>='A'&&ch<='Z')
uppers++;
if(ch>='a'&&ch<='z')
lowers++;
if(ch>='0'&&ch<='9')
numbers++;
printf("\nEnter another character.Enter * to exit");
scanf("%c",&ch);
}while(ch!='*');
printf("\nCount of upper case entered is %d",uppers);
printf("\nCount of lower case entered is %d",lowers);
printf("\nCount of numbers entered is %d",numbers);          }
```

for loop

- For loop is used to repeat a task until a particular condition is true.
- The syntax of a for loop

```
for (initialization; condition; increment/decrement/update)
{
    statement block;
}
Statement Y;
```

- When a for loop is used, the loop variable is initialized only once. With every iteration of the loop, the value of the loop variable is updated and the condition is checked. If the condition is true, the statement block of the loop is executed else, the statements comprising the statement block of the for loop are skipped and the control jumps to the immediate statement following the for loop body. Updating the loop variable may include incrementing the loop variable, decrementing the loop variable or setting it to some other value like, i +=2, where i is the loop variable.

WAP to print n natural numbers

```
void main()
{   int i,n;
```

```
printf("Enter the value of n: ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("%d\t",i);
}
```

WAP to print n whole numbers

```
void main()
{   int i,n;
printf("Enter the value of n: ");
scanf("%d",&n);
for(i=0;i<=n;i++)
{
printf("%d\t",i);
}
```

WAP to print multiples of 4 until 40.

```
void main()
{   int i,n;
printf("Enter the value of n: ");
scanf("%d",&n);
for(i=4;i<=n;i+=4)
{
printf("%d\t",i);
} }
```

WAP to print the following pattern.

Pass 1: 1 2 3 4 5

```
Pass 2: 1 2 3 4 5

Pass 3: 1 2 3 4 5

Pass 4: 1 2 3 4 5

Pass 5: 1 2 3 4 5
```

```
void main()

{

int i,j;

for(i=1;i<=5;i++)

{

printf("\nPass %d",i);

for(j=1;j<=5;j++)

{         printf("\t%d",j);         }

}}
```

WAP to print following pattern.

```
* * * * * * * * * *

* * * * * * * * * *

* * * * * * * * * *

* * * * * * * * * *

* * * * * * * * * *
```

```
void main()

{

   int i,j;

for(i=1;i<=5;i++)

{

printf("\n");

for(j=1;j<=10;j++)
{          printf("*  ");       }
}}
```

WAP to print the following pattern.

```
*
**
***
****
*****
```

```
void main()
{
int i,j;
for(i=1;i<=5;i++)
{
printf("\n");
for(j=1;j<=i;j++)
{
printf("*");
}}}
```

WAP to print following pattern

```
1
12
123
1234
12345
```

```
void main()
{
```

```
int i,j;
for(i=1;i<=5;i++)
{
printf("\n");
for(j=1;j<=i;j++)
{        printf("%d",j);      }}}
```

WAP to print following pattern

```
1
22
333
4444
55555
```

```
void main()
{
int i,j;
for(i=1;i<=5;i++)
{
printf("\n");
for(j=1;j<=i;j++)
{        printf("%d",i);      }}}
```

WAP to print following pattern.

```
0
12
345
6789
```

```
void main()
```

```
{
int i,j,count=0;
for(i=1;i<=4;i++)
{
printf("\n");
for(j=1;j<=i;j++)
{         printf("%d",count++);        }}}
```

WAP to print following pattern.

A

AB

ABC

ABCD

ABCDE

ABCDEF

```
void main()
{
char i,j;
for(i=65;i<=70;i++)
{
printf("\n");
for(j=65;j<=i;j++)
{         printf("%c",j);        }}}
```

WAP to print following pattern.

```
    1
   12
  123
 1234
12345
```

```
void main()
{
int N = 5,i,j,k;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < 2 * (N - i) - 1; j++)
        {
            printf("  ");
        }
         for (k = 0; k <= i; k++)
        {
            printf("%d ",k+1);
        }
        printf("\n");
    }
}
```

WAP to print the multiplication table of n, where n is entered by the user

```
void main()
{   int n,i;
printf("Enter any number");
scanf("%d",&n);
for(i=0;i<=10;i++)
printf("\n%d X %d = %d",n,i,i*n);      }
```

WAP to print all numbers from m to n, thereby classifying them as even or odd.

```
void main()
{   int m,n,i;
printf("Enter the value of m: ");
scanf("%d",&m);
printf("Enter the value of n: ");
scanf("%d",&n);
for(i=m;i<=n;i++)
{
if(i%2==0)
printf("\n%d is even",i);
else
printf("\n%d is odd",i);
}
```

break statement

- The break statement is used to terminate the execution of the nearest enclosing loop in which it appears.
- When compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears. Its syntax is quite simple, just type keyword break followed with a semi-colon.

```
            break;
```

- In switch statement if the break statement is missing then every case from the matched case label to the end of the switch, including the default, is executed.

```
while(...)
{
.....
if(condition)
break;
.....
}
```

Transfers the control out of while loop

```
do
{
.....
if(condition)
break;
.....
}while(...);
```

Transfers the control out of th do-while loop

```
for(...)
{
.....
if(condition)
break;
.....
}
```

Transfers the control out of the loop

```
for(...)
{
.....
for(...)
{
if(condition)
break;
.....
}
Stmnt y
}
```

Transfers the control out of inner for loop

continue statement

- The continue statement can only appear in the body of a loop.

- When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop. Its syntax is quite simple, just type keyword continue followed with a semi-colon.

```
          continue;
```

- If placed within a for loop, the continue statement causes a branch to the code that updates the loop variable. For example,

```
int i;

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

{       if(i==5)

     continue;

printf("\t  %d",i);     }
```

```
while(...)

{

.....

if(condition)

continue;

.....

}
```

Transfers the control to the condition expression of the while loop

```
do

{

.....

if(condition)

continue;

.....

}while(...);
```

Transfers the control to the condition expression of the do-while loop

```
for(...)
{
.....
if(condition)
continue;
.....
}
```

Transfers the control to the condition expression of the for loop

```
for(...)
{
.....
for(...)
{
if(condition)
continue;
.....
}
}
```

Transfers the control to the condition expression of the inner for loop

goto statement

- The goto statement is used to transfer control to a specified label.
- Here label is an identifier that specifies the place where the branch is to be made. Label can be any valid variable name that is followed by a colon (:).
- Note that label can be placed anywhere in the program either before or after the goto statement. Whenever the goto statement is encountered the control is immediately transferred to the statements following the label.
- Goto statement breaks the normal sequential execution of the program.
- If the label is placed after the goto statement then it is called a forward jump and in case it is located before the goto statement, it is said to be a backward jump.
- It is not necessary to use goto statement as it can always eliminated by rearranging the code.
- Using the goto statement violates the rules of structured programming.

- It is a good programming practice to use break, continue statements in preference to goto whenever possible.
- Goto statements make the program code complicated and render the program unreadable.

Example:

```
int num, sum=0;
read:     // label for go to statement
    printf("\n Enter the number. Enter 999 to end : ");
    scanf("%d", &num);
    if (num != 999)
    {
        if(num < 0)
            goto read;  // jump to label- read
        sum += num;
        goto read;          // jump to label- read
    }
    printf("\n Sum of the numbers entered by the user is = %d", sum);
```

*****End*****