Introduction to programming language using c

Unit-2 C Language Operators and Decision Making

by Shailee Shah
Assistant professor
President Institute of Computer application

2.1 Operators and Expression

- Any expression in C has 2 parts:

1. Operand
2. Operator

- The expression having only 1 operand and 1 operator are called as unary expression and operators are called unary operations.
- Ex: i++ ,i=operand and ++=operators.
- The expression having only 2 operand and 2 operator are called as binary expression and operators are called binary operations.
- Ex: a+b,a,b=operand and +=operators.

2.1.1 Types Of Operators

1. arithmetic
    1. Unary [+,-,++,--]
    2. binary[+,-,/,*,%]
2. Relational [<,>,<=,>=,!=]
3. Logical [&&, | |,!]
4. Conditional [?:]
5. assignment [=]
6. Size of
7. bitwise [&, | ,^,<<,>>]

arithmetic Operators

1. Unary operators:
   - + indicates a positive value of the operand or value.
     - +a,+5
   - - indicates a negative value of the operand or value.
     - -a,-5
   - Increment operator is used to increment value of the operand by 1.
     - It expanded internally as a=a+1.
     - Pre-increment(++a):

       Here the value is incremented first and then assigned to a variable.
     - Post-increment(a++):

       Here the value is assigned to a variable first and then it is incremented.
     - Pre-decrement(--a):

       Here the value is decremented first and then assigned to a variable.
     - Post-decrement(a--):

       Here the value is assigned to a variable first and then it is decremented.

```
/*C Program to demonstrate the working
      of increment and decrement
      operators */


#include <stdio.h>
int main()
{
   int a = 10, b = 100;
   float c = 10.5, d = 100.5;

   printf("++a = %d \n", ++a);

   printf("--b = %d \n", --b);

   printf("++c = %f \n", ++c);

   printf("--d = %f \n", --d);

   return 0;
}
```

Output

```
++a = 11
--b = 99
++c = 11.500000
++d = 99.500000
```

arithmetic Operators

2. binary operators:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denumerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

```
/* C Program to demonstrate the
      working of arithmetic operators */

#include <stdio.h>
int main()
{
   int a = 9,b = 4, c;

   c = a+b;
   printf("a+b = %d \n",c);

   c = a-b;
   printf("a-b = %d \n",c);

   c = a*b;
   printf("a*b = %d \n",c);

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

   c=a%b;
   printf("Remainder when a divided
by b = %d \n",c);

   return 0;
}
```

Output

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

Relational [<,>,<=,>=,!=] Operators

Operator	Example	Description
`>`	`a > b`	Returns `true` if `a` is greater than `b`, else `false`.
`<`	`a < b`	Returns `true` if `a` is less than `b`, else `false`.
`>=`	`a >= b`	Returns `true` if `a` is greater than or equal to `b`, else `false`.
`<=`	`a <= b`	Returns `true` if `a` is less than or equal to `b`, else `false`.
`==`	`a == b`	Returns `true` if `a` is equal to `b`, else `false`.
`!=`	`a != b`	Returns `true` if `a` is not equal to `b`, else `false`.

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

```c
/* C Program to demonstrate the working of
      Relational operators */
#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10;

   printf("%d == %d = %d \n", a, b, a == b); // true
   printf("%d == %d = %d \n", a, c, a == c); // false

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

   printf("%d < %d = %d \n", a, b, a < b); //false
   printf("%d < %d = %d \n", a, c, a < c); //true

   printf("%d != %d = %d \n", a, b, a != b); //false
   printf("%d != %d = %d \n", a, c, a != c); //true

   printf("%d >= %d = %d \n", a, b, a >= b);//true
   printf("%d >= %d = %d \n", a, c, a >= c); //false

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

   return 0;

}
```

Output

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```

Logical [&&, | |,!] Operators

Operator	Meaning	Example	Result
&&	Logical AND	(5<2)&&(5>3)	False
||	Logical OR	(5<2)||(5>3)	True
!	Logical NOT	!(5<2)	True

&&		
Operand 1	Operand 2	Result
True	True	True
True	False	False
False	True	False
False	False	False

||		
Operand 1	Operand 2	Result
True	True	True
True	False	True
False	True	True
False	False	False

!	
Operand	Result
False	True
True	False

X	Y	X && Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X || Y
0	0	0
0	1	1
1	0	1
1	1	1

X	! X
0	1
1	0

1=TRUE
0=FaLSE

```
// C Program to demonstrate the working of logical
      operators

#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10, result;

   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) equals to %d \n", result);

   result = (a == b) && (c < b);
   printf("(a == b) && (c < b) equals to %d \n", result);

   result = (a == b) || (c < b);
   printf("(a == b) || (c < b) equals to %d \n", result);

   result = (a !== b) || (c < b);
   printf("(a !== b) || (c < b) equals to %d \n", result);

   result = !(a != b);
   printf("!(a == b) equals to %d \n",
result);

   result = !(a == b);
  printf("!(a == b) equals to %d \n",
result);

   return 0;
}
```

Output

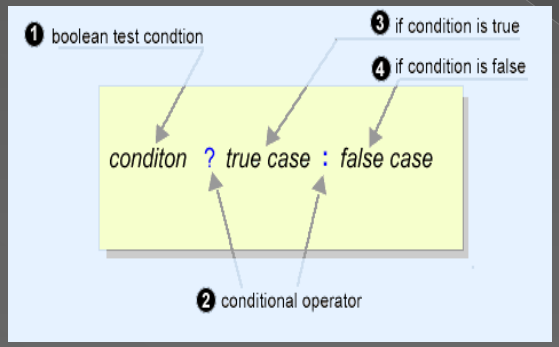
```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c < b) equals to 1
(a !== b) || (c < b) equals to 0
!(a !== b) equals to 1
!(a == b) equals to 0
```

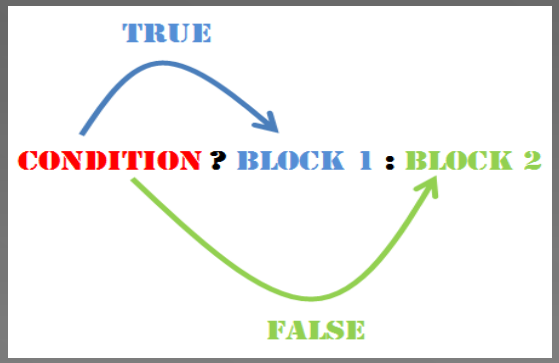
Explanation of logical operator program

- **(a = b) && (c > 5)** evaluates to 1 because both operands **(a = b)** and **(c > b)** is 1 (true).
- **(a = b) && (c < b)** evaluates to 0 because operand **(c < b)** is 0 (false).
- **(a = b) || (c < b)** evaluates to 1 because **(a = b)** is 1 (true).
- **(a != b) || (c < b)** evaluates to 0 because both operand **(a != b)** and **(c < b)** are 0 (false).
- **!(a != b)** evaluates to 1 because operand **(a != b)** is 0 (false). Hence, !(a != b) is 1 (true).
- **!(a == b)** evaluates to 0 because **(a == b)** is 1 (true). Hence, **!(a == b)** is 0 (false).

Conditional [?:] Operators

- It is also known as Ternary operator.
- The condition is checked , If the condition is true first expression will execute otherwise second expression will execute.





Conditional Operators Example

```
#include<stdio.h>
        void main()
        {
                int a, b, x;
                printf("Enter the values of a add b : ");
                scanf("%d %d", &a, &b);
                x=(a>b)?a:b;
                printf("Biggest Value is :%d",x);
        }
```

assignment Operators

- an assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

```
/* C Program to demonstrate the
      working of assignment operators */


#include <stdio.h>
int main()
{
   int a = 5, c;

   c = a;
   printf("c = %d \n", c);

   c += a;  // c = c+a
   printf("c = %d \n", c);

   c -= a;   // c = c-a
   printf("c = %d \n", c);

   c *= a;  // c = c*a
   printf("c = %d \n", c);

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

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

   return 0;
}
```

Output

```
c = 5
c = 10
c = 5
c = 25
c = 5
c=0
```

Size of Operators

- It is unary operator which gives the size of its arguments in terms of bytes.
- Ex: sizeof(int);
- This gives the bytes occupied by the int datatype ,that is 2 bytes.
- It is the only operator which takes arguments.

```
/* C Program to demonstrate the working of
      assignment operators */

#include <stdio.h>
Void  main()
{
   int a, e[10];
   float b;
   double c;
   char d;
   printf("Size of int=%lu bytes\n",sizeof(a));
   printf("Size of float=%lu bytes\n",sizeof(b));
   printf("Size of double=%lu bytes\n",sizeof(c));
   printf("Size of char=%lu byte\n",sizeof(d));
   printf("Size of integer type array
having 10 elements = %lu
bytes\n", sizeof(e));
}
```

Output

```
Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
Size of integer type array having
10 elements = 40 bytes
```

bitwise [&, | ,^,<<,>>] Operators

- The following operators are used to operate on bits so called as bitwise operator.
- The numbers are converted into binary and then operator carry out respective operations on bits.
- Ex: int x=7,y=5;
- binary of x and y:
  - 7=111
  - 5=101
  - x & y=101 so x & y =5
  - x | y=111 so x | y =7

Operator	Operation
&	Bitwise AND
|	Bitwise OR
~	One's Complement
>>	Shift right
<<	Shift left

X	Y	X & Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X | Y
0	0	0
0	1	1
1	0	1
1	1	1

bitwise [&, | ,^,<<,>>] Operators

- Left Shift :
  - a << b value of a is left shifted by b bit position.
  - Ex: a=7,b=1;
  - Printf("left shift a<<b = %d",a<<b);
  - 7 <<1
  - Binary of 7 is:00000111
  - After left shift:00001110
  - So now a =14

- Right Shift :
  - a << b value of a is left shifted by b bit position.
  - Ex: a=7,b=1;
  - Printf("left shift a>>b = %d",a<<b);
  - 7 >>1
  - Binary of 7 is:00000111
  - After left shift:00000011
  - So now a =3

2.1.2 precedence and Associativity

Operators	Description	Associativity
() [] -> . ++ --	Function Call Array Subscript Member Selectors Postfix Increment/Decrement	Left to Right
++ -- + - ! ~ (type) * & sizeof	Prefix Increment / Decrement Unary plus / minus Logical negation / bitwise complement Casting Dereferencing Address of Find size in bytes	Right to Left
* / %	Multiplication Division Modulo	Left to Right
+ -	Addition / Subtraction	Left to Right
>> <<	Bitwise Right Shift Bitwise Left Shift	Left to Right
< <= > >=	Relational Less Than / Less than Equal To Relational Greater / Greater than Equal To	Left to Right
== !=	Equality Inequality	Left to Right
&	Bitwise AND	Left to Right
^	Bitwise XOR	Left to Right
|	Bitwise OR	Left to Right
&&	Logical AND	Left to Right
||	Logical OR	Left to Right
?:	Conditional Operator	Right to Left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operators	Right to Left
,	Comma Operator	Left to Right

2.2 Console Based IO And Related Built-in IO Functions:

- The console based input/Output function are available under conio.h header file and standard input output functions are available under stdio.h.
- Keyboard treated as standard input device and monitor as standard output device.

(1) printf

- It is used to display the text and numeric values on the console.
- Syntax:
  - printf("control string",arg1,arg2,….arg);

Data Type		Format
Integer	Integer	%d
	Short	%d
	Short unsigned	%u
	Long	%ld
	Long assigned	%lu
	Hexadecimal	%x
	Long hexadecimal	%lx
	Octal	%O (letter 0)
	long octal	%lo
Real	float,double	%f, %lf, %g
Character		%c
String		%s

(1) printf()

- The control string indicates how many arguments are there and what their types are.
- The control string having 3 types:
  1. The character to be printed on screen

     printf("welcome to c lab")
  2. The format specifies that define the way the arguments are displayed

     printf("%d",a)
  3. The escape sequence characters.

     printf("\n")

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

void main()
{
    int l=10;
    clrscr();
    printf("simple use of printf");
    printf("\n value of I is = %d",i);
    getch();
}
```

Output

simple use of printf
value of I is = 10

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

void main()
{
    float n=10.2233445;
    clrscr();
    printf("\n value of n is = %0.2f",n);
    getch();
}
```

Output

value of n is = 10.22

Explain

%w.p=

- where w is an integer number that specifies the total number of columns for the output value
- p is the integer number that specifies the number of digits to the right of the decimal point.

(2) scanf()

- It is used to read the values from the console using the standard input and stores the values in the variable according to parameters anf format specifiers supplied.
- Syntax:
  - scanf("string with format specifiers",&arg1,&arg2,....);
  - scanf("%d",&a);

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

void main()
{
      int l;
      clrscr();
      printf(“enter the value for l :\n”);
      scanf(“%d”,&l);
      printf(“\n value of I is = %d”,l);
      getch();
}
```

Output

```
enter the value for l
15
value of I is = 15
```

(3) getch()

- It is used to read the character from the standard input and save it to a variable.
- It does not use any parameters but returns ascii value of the character entered by the user.
- The function is defined in conio.h header file.
- The character is not displayed on the screen.
- It is genreally used in c programs to hold the output on the screen till user press any key from the keyboard.
- Syntax:
  - Int getch(void);

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

void main()
{
      char c;
      clrscr();
      printf(“enter the character:\n”);
      c=getch();
      printf(“\n value of c is = %d”,C);
      printf(“\n value of c is = %c”,C);

      getch();
}
```

Output

```
enter the character :
value of c is =65
value of c is =A
```

(4) getchar()

- It is used to read the character from the standard input and save it to a variable.
- It does not use any parameters but returns ascii value of the character entered by the user.
- The difference between getch() and getchar()is that the character is displayed on the screen in getchar().
- Syntax:
  - Int getchar(void);

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

void main()
{
      char c;
      clrscr();
      printf(“enter the character:\n”);
      c=getchar();
      printf(“\n value of c is = %d”,c);
       printf(“\n value of c is = %c”,c);
      getch();
}
```

Output

```
enter the character :
A
value of c is =65
value of c is =A
```

(5) putchar()

- It is used to write a character to standard output.
- This character is passed as the parameter to this method.

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

void main()
{
    char c='A';
    clrscr();
    putchar(c);
    getch();
}
```

Output

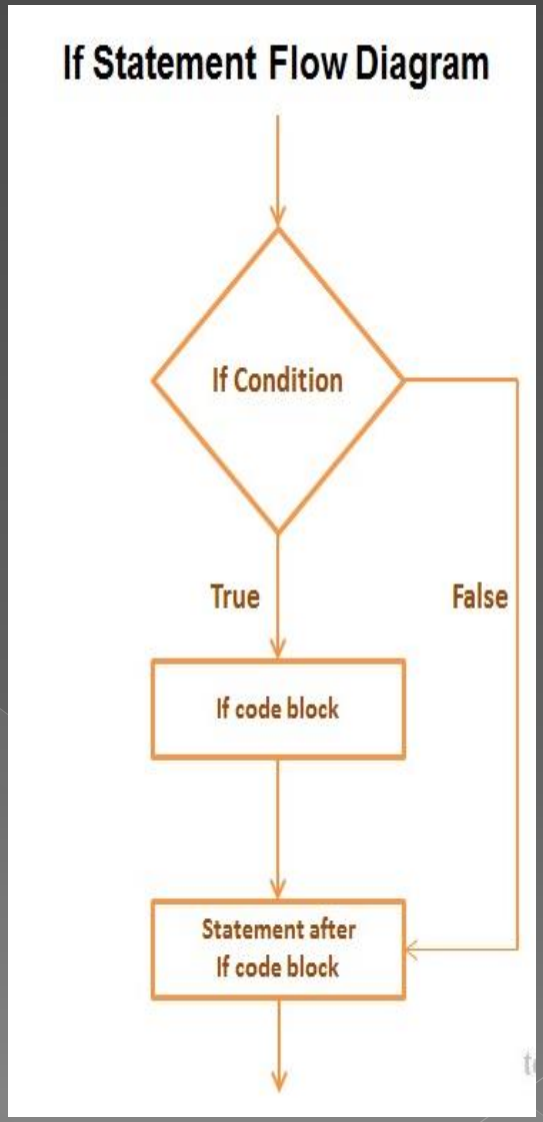
A

2.3 Decision making structure

2.3.1 **if statements :**

- Syntax:

```
if(expression)
{
    Statement block
}
```



// C Program to demonstrate the working of if statement.

```
#include <stdio.h>
void main()
{
   int  a = 2, b = 3;

      if(a<b)
      {
            printf(“a is smaller than b”);
      }
}
```

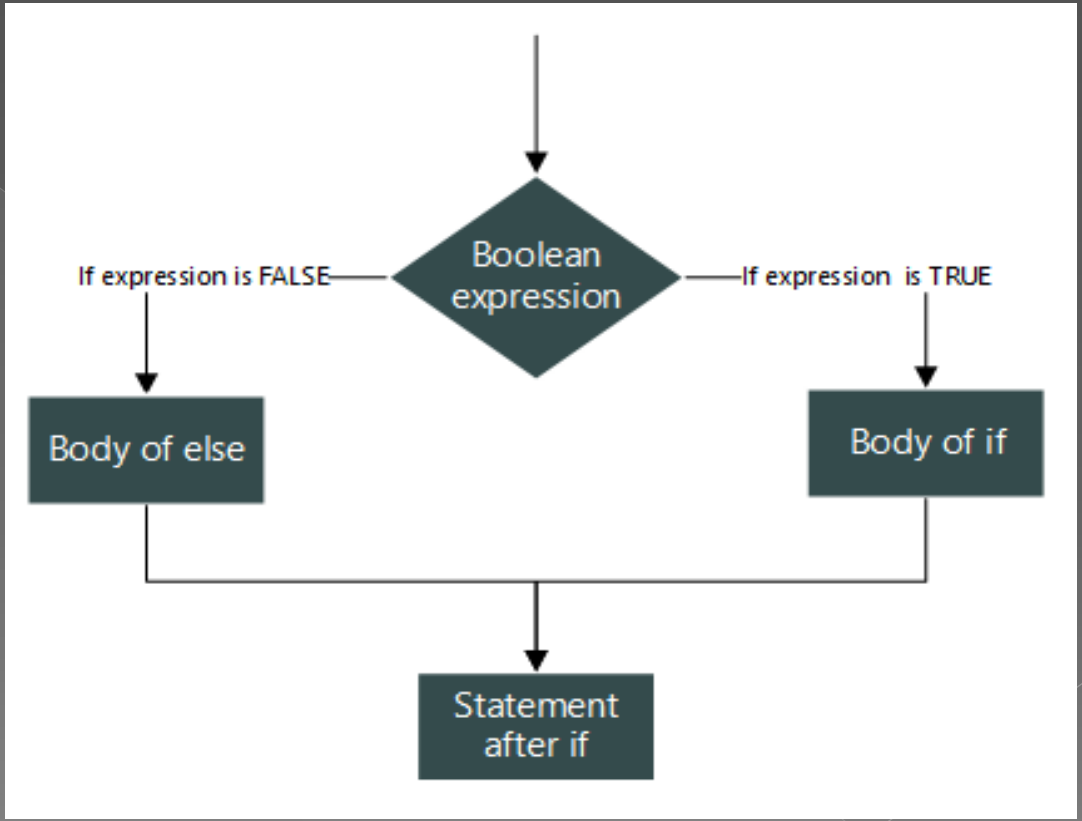
Output

a is smaller than b

2.3.2 **if..else statements :**

Syntax:

```
if(expression)
{
    Statement block
}
else
{
    Statement block
}
```



// C Program to demonstrate the working of if_else statement.

```
#include <stdio.h>
void main()
{
   int  a = 12, b = 10;

      if(a<b)
      {
            printf("a is smaller than b");
      }
      else
      {
            printf("b is smaller than a");
      }
}
```

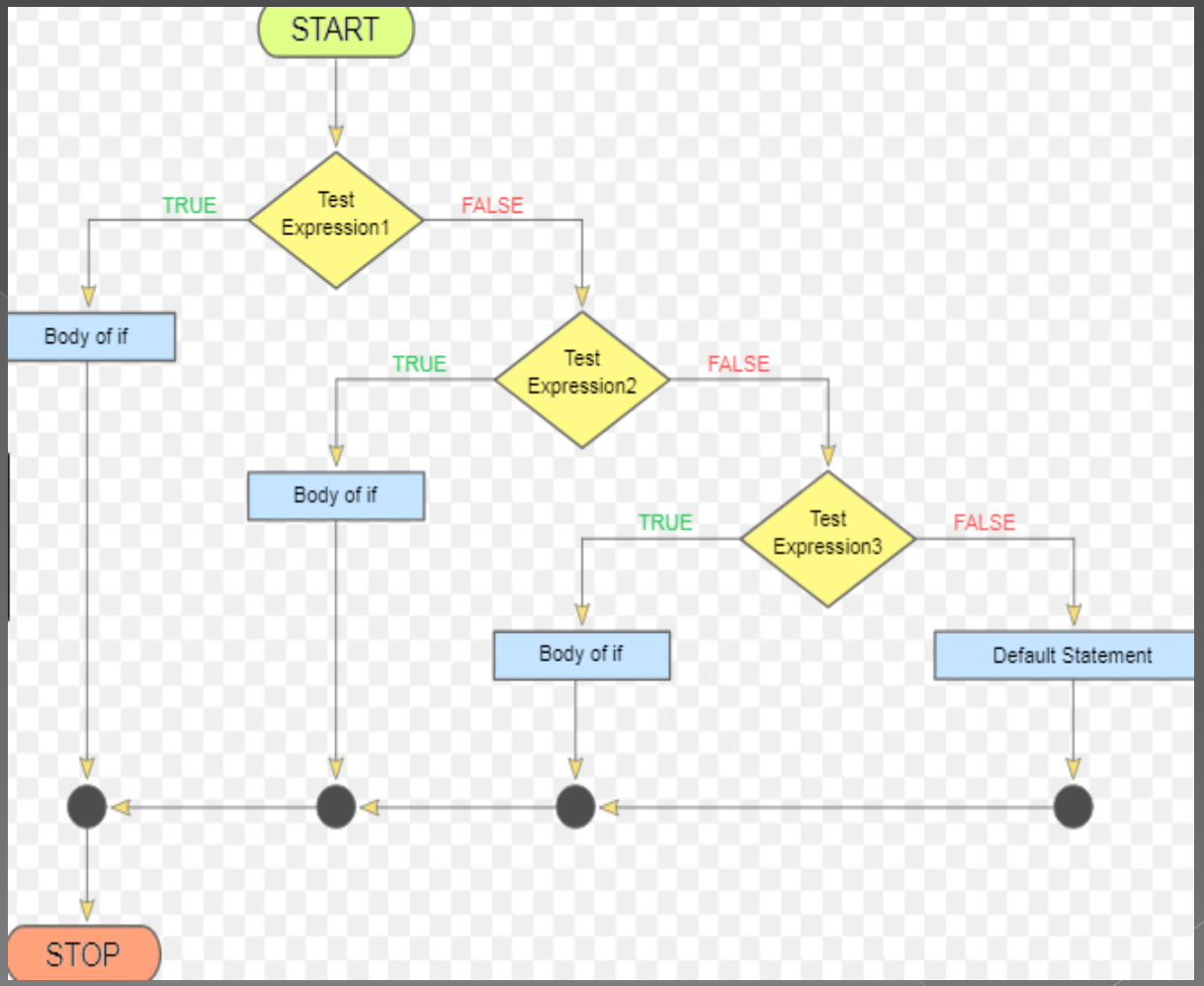
Output

a is smaller than b

2.3.3 **if..elseif…elseif..else statements**:

- Syntax:

```
if(expression)
{
    Statement block
}
elseif(expression)
{
    Statement block
}
elseif(expression)
{
    Statement block
}
else
{
    Statement block
}
```



```
// C Program to demonstrate the working of if_elseif_else statement.

#include <stdio.h>
void main()
{
	int  num;
	clrscr();
	printf("enter a number:  \n");
	scanf("%d",&num);

	if(num<0)
	{
		printf("Num is negative");
	}
	elseif(num>0)
	{
		printf("Num is positive");
	}
	else
	{
		printf("Num is zero");
	}

}
```

Output

- enter a number:

10

Num is positive

- enter a number:

-5

Num is negative

- enter a number:

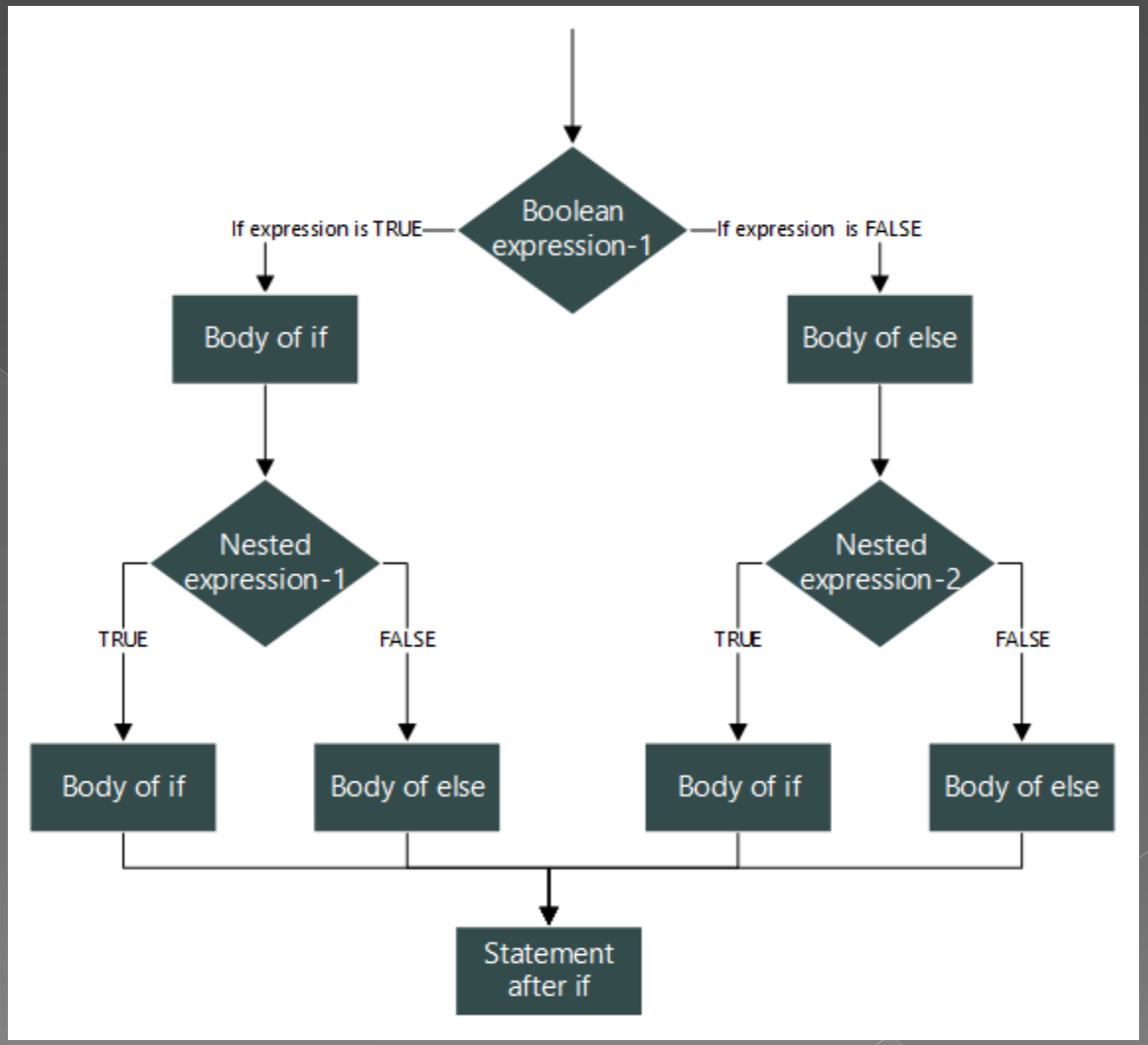
0

Num is zero

2.3.3 **Nested…if..else statements**:

- Syntax:

```
if(expression)
{
    if(expression)
    {
        Statement block
    }
    else (expression)
    {
        Statement block
    }
}
else
{
    Statement block
}
```



```
// C Program to demonstrate the working of
      nested _if_else statement.

#include <stdio.h>
void main()
{
      int  no1,no2,no3;
      clrscr();
      printf("enter three number:  \n");
      scanf("%d %d %d",&no1,&no2,&no3);

      if(no1>no2)
      {
            if(no1>no3)
           {
            printf(" \n Number 1 is maximum");
           }
           else
           {
            printf("Number 3 is maximum");
           }
      }
else
{
            if(no2>no3)
            {
            printf(" \n Number 2 is maximum");
            }
            else
            {
            printf("\n Number 3 is maximum");
            }
}
```

Output

enter three number:

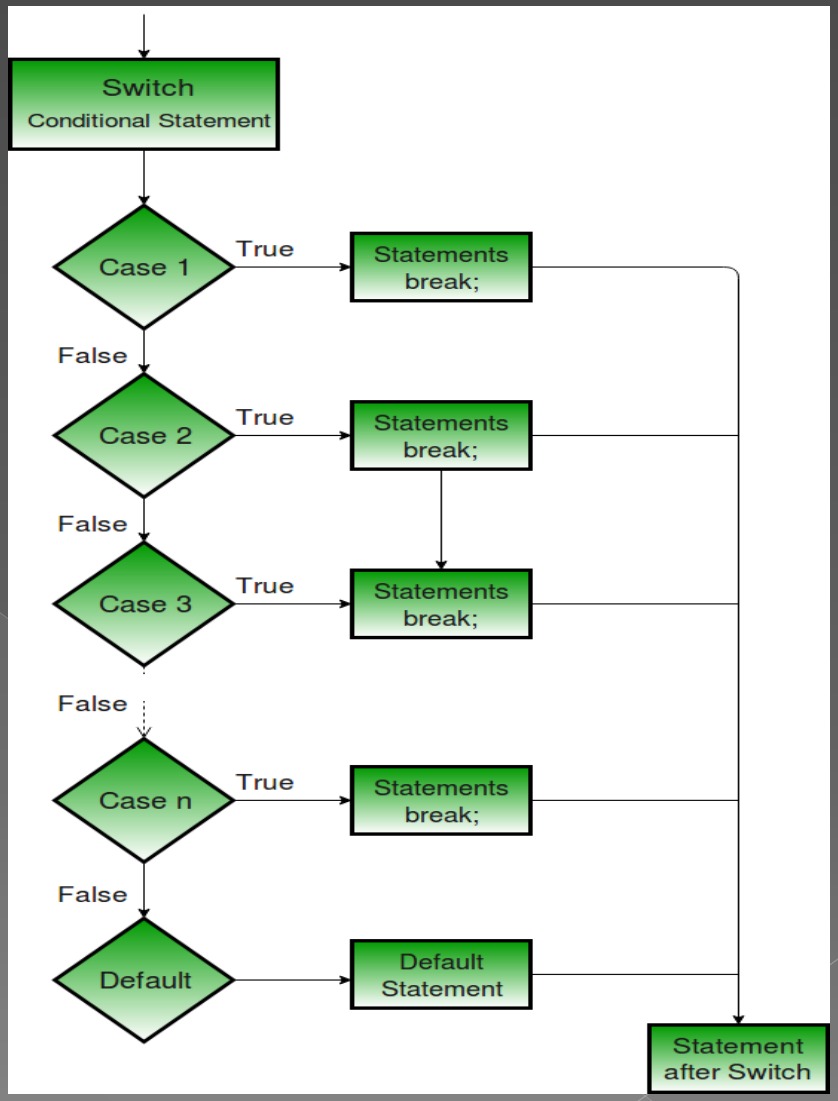
10 3 12

Number 3 is maximum

2.3.4 **Switchstatements**:

- Syntax:

```
switch(variable)
{
case 1:
    statement block1;
    break;
Case 2:
    statement block2;
    break;
case 3:
    statement block3;
    break;
default:
    default statement block;
}
```



```
// C Program to demonstrate the working of
      switch statement.

#include <stdio.h>
void main()
{
      int  no;
      clrscr();
      printf("enter a number:  \n");
      scanf("%d ",&no);

      switch(no)
      {
            case  1:
            printf("value of number is =1");
            break;

            case  2:
            printf("value of number is =2");
            break;

            case  3:
            printf("value of number is =3");
            break;

            default:
            printf("value is other than is 1,2,3");

            getch();

      }
}
```

Output

```
Enter a number:
3
value of number is 3
```

Concept of header file #include

- It is used to read the character from the standard input and save it to a variable.
- Header files contains number of library function.
- 2 types of header files:
  - User defined header files
  - Standard header files.
- Header file is a collection of constants , macro and functions.
- We can add the header files into program using #directives like #include<header file name>.
- #include<stdio.h>
  - is a standard header file which include library function like printf() and scanf().
- #include<conio.h>
  - is a standard header file which include library function like clrscr().

Concept of header file #define

- The c pre-processor is a macro pre-processor that transforms your program before it compiled.
- #define is another pre-processor directives.
- It is used to define a macro and its value can be used throughout the program.
- Macro definitions are not variables and its value can't be changed by your program.

```
// C Program to demonstrate the working of #define.

#include <stdio.h>
#include <conio.h>
#define pi 3.14

void main()
{
      int radius;
      float area;

      clrscr();

      printf("enter the radius :  \n");
      scanf("%d ",&radius);

       area= pi* radius * radius;

       printf("area is : %f,area);

            getch();
 }
```

Output

enter the radius :

5

area is : 78.5

Thank you