UNIT 3

Operators and Statements

Operators

- An operator is a symbol which helps the user to command the computer to do a certain mathematical or logical manipulation.
- Operators are used in C language program to operate on data and variables.
- C has a rich set of operators which can be classified as
  - ✓ Arithmetic operators
  - ✓ Relational Operators
  - ✓ Logical Operators
  - ✓ Assignment Operators
  - ✓ Increments and Decrement Operators
  - ✓ Conditional Operators
  - ✓ Bitwise Operators
  - ✓ Special Operators

Arithmetic Operators

- All the basic arithmetic operations can be carried out in C.
- All the operators have almost the same meaning as in other languages.
- Unary and binary operations are available in C language.
- Unary operations operate on a single operand.
- Example
  - ✓ -5 The number 5 when operated by unary – will have the value –5.

Operator	**Meaning**
+	Addition or Unary Plus
-	Subtraction or Unary Minus
*	Multiplication
/	Division
%	Modulus Operator - evaluates the remainder of the operands after division

- ✓ x + y
- ✓ x - y
- ✓ -x + y
- ✓ a * b + c
- ✓ -a * b

- here a, b, c, x, y are known as operands.

Integer Arithmetic

- When an arithmetic operation is performed on two whole numbers or integers than such an operation is called as integer arithmetic.
- It always gives an integer as the result.
- Let x = 27 and y = 5 be 2 integer numbers.
  - ✓ x + y = 32
  - ✓ x – y = 22
  - ✓ x * y = 115
  - ✓ x % y = 2
  - ✓ x / y = 5
- In integer division the fractional part is truncated.
  - ✓ op1/op2 and op1%op2
    - Both op1 and op2 are integers but the quotient is not an integer
    - If op1 and op2 have the same sign, op1/op2 is the largest integer less than the true quotient. op1%op2 will have the sign of op1.
    - If op1 and op2 have the opposite signs, op1/op2 is the smallest integer greater than the true quotient. op1%op2 will have the sign of op1.
- Ex: 15/4 = 3, 15%4 = 3
- -15/-4 = 3, -15%-4 = -3
- Ex: -15/4 = -3   -15 % 4 = -3
- 15/-4 = -3   15 % -4 = 3

Floating point arithmetic

- When an arithmetic operation is preformed on two real numbers or fraction numbers such an operation is called floating point arithmetic.
- The floating-point results can be truncated according to the properties requirement.
- **The remainder operator cannot be used for floating point arithmetic operands.**
- Let x = 14.0 and y = 4.0 then
  - ✓ x + y = 18.0
  - ✓ x – y = 10.0
  - ✓ x * y = 56.0
  - ✓ x / y = 3.50

Mixed Mode arithmetic

- When one of the operand is real and other is an integer and if the arithmetic operation is carried out on these two operands then it is called as mixed mode arithmetic.
- If anyone operand is of real type then the result will always be real.
- Example
  - ✓ 15/10.0 = 1.5

Relational Operators

- These operators are required to compare the relationship between operands and bring out a decision and program accordingly.

Operator	**Meaning**
<	is less than
<=	is less than or equal to
==	is equal to
>=	is greater than or equal to
>	is greater than
!=	is not equal to

- A simple relational expression contains only one relational operator and takes the following form.
  - ✓ exp1 relational operator exp2
    - Where exp1 and exp2 are expressions, which may be simple constants, variables or combination of them.
- Example
  - ✓ 6.5 <= 25 TRUE
  - ✓ -65 > 0 FALSE
  - ✓ 10 < 7 + 5 TRUE

LOGICAL OPERATORS

- They compare or evaluate logical and relational expressions.

Operator	**Meaning**
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical And &&

- This operator is used to evaluate 2 conditions or expressions with relational operators simultaneously.
- **If both the expressions to the left and to the right of the logical operator is true then the whole compound expression is true.**

- Example
  - ✓ a > b && x = = 10
    - The expression to the left is a > b.
    - The expression on the right is x == 10.
    - The whole expression is true only if both expressions are true i.e., if a is greater than b and x is equal to 10.

Logical Or ||

- The logical OR is used to combine two expressions or the condition evaluates to true if any one of the two expressions is true.
- Example
  - ✓ a < m || a < n
    - The expression evaluates to true if any one of them is true or if both of them are true.
    - It evaluates to true if a is less than either m or n and when a is less than both m and n.

Logical not !

- The logical not operator takes single expression and evaluates to true if the expression is false and evaluates to false if the expression is true.
- In other words it just reverses the value of the expression.
- Example
  - ✓ ! (x >= y)
    - The NOT expression evaluates to true only if the value of x is neither greater than or equal to y

ASSIGNMENT OPERATOR

- The Assignment Operator evaluates an expression on the right of the expression and substitutes it to the value or variable on the left of the expression.
- Example
  - ✓ x = a + b
    - Here the value of a + b is evaluated and substituted to the variable x.
    - Here the value of a + b is evaluated and substituted to the variable x.
  - ✓ var oper = exp;
    - Here var is a variable, exp is an expression and oper is a C binary arithmetic operator.
    - The operator = is known as shorthand assignment operator

Simple assignment	Shorthand Operator
a=a+1	a+=1
a=a-1	a-=1

a=a*(n-1)	a*=(n-1)
a = a / (n+1)	a /= (n+1)
a = a % b	a %= b

INCREMENT AND DECREMENT OPERATOR

- The increment and decrement operators are one of the unary operators which are very useful in C language.
- They are extensively used in for and while loops.
- The syntax of the operators is
  - ✓ ++ variable name
  - ✓ variable name++
  - ✓ – –variable name
  - ✓ variable name– –
- The increment operator ++ adds the value 1 to the current value of operand.
- The decrement operator – – subtracts the value 1 from the current value of operand.
- ++variable name and variable name++ mean the same thing when they form statements independently.
- They behave differently when they are used in expression on the right hand side of an assignment statement.
- Example
  - ✓ m = 5;
  - ✓ y = ++m; (prefix)
    - In this case the value of y and m would be 6
  - ✓ m = 5;
  - ✓ y = m++; (post fix)
    - The value of y will be 5 and that of m will be 6.
- A prefix operator first adds 1 to the operand and then the result is assigned to the variable on the left.
- A postfix operator first assigns the value to the variable on the left and then increments the operand.
- **These operators should not be used with floating point variables.**

CONDITIONAL OPERATOR

- The conditional operator consists of 2 symbols the question mark (?) and the colon (:).
  - ✓ exp1 ? exp2 : exp3

- exp1 is evaluated first.
- If the exp1 is true then exp2 is evaluated & its value becomes the value of the expression.
- If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.
- Only one of the expression is evaluated.

✓ Example

- a = 10; b = 15; x = (a > b) ? a : b
  - ∞ Here x will be assigned to the value of b.
  - ∞ The condition follows that the expression is false therefore b is assigned to x.

BITWISE OPERATOR

- A bitwise operator operates on each bit of data.
- Those operators are used for testing, complementing or shifting bits to the right on left.
- Bitwise operators may not be applied to a float or double.

Operator	**Meaning**
&	Bitwise AND
|	Bitwise Inclusive OR
^	Bitwise Exclusive OR
~	One's complement
<<	Shift Left
>>	Shift Right

Bitwise Operators – Shift Operators

- They move bits to the left and the right.
- When applied to unsigned numbers, these operators are implementation independent.
- When applied on signed numbers the implementation is left to the discretion of the software engineer who designs the system.
- Shift operator requires two operands.
  - ✓ The first is the value to be shifted.
  - ✓ The second operand specifies the number of bits to be shifted.
- Num1 = 0257, Num2 = 0463
  - ✓ 0257 & 0463 =000100011
  - ✓ 0257 | 0463 = 110111111
  - ✓ 0257 ^ 0463 = 110011100
  - ✓ ~0257 = 101010000
  - ✓ 49<<1 = 98   64>>1= 32

- ✓ 49<<2 = 196 64>>2= 16
- ✓ 49<<4 = 784 64>>4= 4
  - The left shift operator multiplies by a power of 2.
  - The right shift operator divides by a power of 2.

Shift operator	**Multiplies by**	**2nd shift value**	**Shift operator**	**Divides by**
<<1	2	1	>>1	2
<<2	4	2	>.2	4
<<3	8	3	>>3	8
<<4	16	4	>>4	16
<<5	32	5	>>5	32

SPECIAL OPERATORS

- C supports some special operators of interest such as comma operator, size of operator, pointer operators (& and *) and member selection operators (. and ->).

Comma operator

- The comma operator can be used to link related expressions together.
- A comma-linked list of expressions are evaluated left to right and value of right most expression is the value of the combined expression.
- The commas that separate function arguments, variables in declarations are not comma operators and do not guarantee left to right evaluation.
  - ✓ value = (x = 10, y = 5, x + y);
    - First assigns 10 to x and 5 to y and finally assigns 15 to value.
- Since comma has the lowest precedence in operators the parenthesis is necessary.
- In for loops
  - ✓ for (n=1, m=10, n <=m; n++,m++)
- In while loops
  - ✓ while (c=getchar(), c != ‘10’)
- Exchanging values
  - ✓ t = x, x = y, y = t;

Sizeof operator

- The operator ***sizeof*** gives the size of the data type or variable in terms of bytes occupied in the memory.

- The operand may be a variable, a constant or a data type qualifier.
- Example
  - ✓ m = sizeof (sum);
  - ✓ n = sizeof (long int);
  - ✓ k = sizeof (235L);
- The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer.
- It is also used to allocate memory space dynamically to variables during the execution of the program.

ARITHMETIC EXPRESSIONS

- An expression is a combination of variables constants and operators written according to the syntax of C language.
- In C every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable.

Algebraic Expression	C Expression
a x b – c	a * b – c
(m + n) (x + y)	(m + n) * (x + y)
(ab / c)	a * b / c
$3x^2$ +2x + 1	3*x*x+2*x+1
(x / y) + c	x / y + c

EVALUATION OF EXPRESSIONS

- Expressions are evaluated using an assignment statement of the form
  - ✓ Variable = expression;
    - Variable is any valid C variable name.
- When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left-hand side.
- All variables used in the expression must be assigned values before evaluation is attempted.
- Examples
  - ✓ x = a * b – c
  - ✓ y = b / c * a
  - ✓ z = a – b / c + d;

PRECEDENCE OF ARITHMETIC OPERATORS

- An arithmetic expression without parenthesis will be evaluated from left to right using the rules of precedence of operators.
- There are two distinct priority levels of arithmetic operators in C.

- ✓ High priority * / %
- ✓ Low priority + -

Rules for evaluation of expression

1. First parenthesized sub expression left to right are evaluated.
2. If parenthesis are nested, the evaluation begins with the innermost sub expression.
3. The precedence rule is applied in determining the order of application of operators in evaluating sub expressions.
4. The associability rule is applied when two or more operators of the same precedence level appear in the sub expression.
5. Arithmetic expressions are evaluated from left to right using the rules of precedence.
6. When Parenthesis are used, the expressions within parenthesis assume highest priority.

Type conversions in expressions:

- There are two types of conversions
  - ✓ Implicit type conversion
  - ✓ Explicit type conversion

Implicit type conversion

- C permits mixing of constants and variables of different types in an expression.
- C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance.
- This automatic type conversion is known as implicit type conversion
- During evaluation it adheres to very strict rules and type conversion.
- If the operands are of different types the lower type is automatically converted to the higher type before the operation proceeds.
- The result is of higher type.

Operand 1	**After conversion**	**Operand 2**	**After conversion**	**Result**
Short	Int	Short	Int	Int
Short	Int	Char	Int	Int
Char	Int	Short	Int	Int
Char	Int	Char	Int	Int
Long Double		Any datatype	Long Double	Long Double

Double		Any datatype except Long Double	Double	Double
Float		Any datatype except Double and Long Double	Float	Float
Unsigned Long Int		Any datatype except Double and Long Double and Float	Unsigned Long Int	Unsigned Long Int
Long int		Unsigned int	Long int	
Long int	unsigned long int	Unsigned int	unsigned long int	
Long int		Any datatype except Double and Long Double and Float and unsigned long int or unsigned int	Long int	
Unsigned int		Any datatype except Double and Long Double and Float and unsigned long int or unsigned int or long int	Unsigned int	Unsigned int

Explicit type conversion

- Many times, there may arise a situation where we want to force a type conversion in a way that is different from automatic conversion.
- Example
  - ✓ The calculation of number of female and male students in a class.

$$ratio = \frac{female\_students}{male\_students}$$

- If female_students and male_students are declared as integers, the decimal part will be rounded off and its ratio will represent a wrong figure.

- This problem can be solved by converting locally one of the variables to the floating point as shown below.

$$ratio = \frac{(float)\ female\_students}{male\_students}$$

- The operator float converts the female_students to floating point for the purpose of evaluation of the expression.
- Then using the rule of automatic conversion, the division is performed by floating point mode, thus retaining the fractional part of the result.

- The process of such a local conversion is known as explicit conversion or casting a value.
- The general form is
  - ✓ (type_name) expression

Operator precedence and Associativity:

- Each operator in C has a precedence associated with it.
- The precedence is used to determine how an expression involving more than one operator is evaluated.
- There are distinct levels of precedence and an operator may belong to one of these levels.
- The operators of higher precedence are evaluated first.
- The operators of same precedence are evaluated from right to left or from left to right depending on the level.
- This is known as **associativity property** of an operator.

Order	Operators	Associativity
Order 1	( ) , [ ], : : Function call , Array element reference	L → R , Left to Right
Order 2 - Unary	! : Logical NOT, ~ : Bitwise 1's complement, + : Unary plus, - : Unary minus, ++ :Pre or post increment, -- :Pre or post decrement, & :Address , * : Indirection, Sizeof :Size of operand in bytes, (type): type cast	R → L , Right -> Left
Order 3: Member Access	., → : Dereference	L → R
Order 4: Multiplication	* : Multiply, / : Divide, % : Modulos	L → R

Order 5: Additive	+ : Binary plus, - : Binary Minus	L → R
Order 6: Shift	<<: Shift left, >>: Shift right	L → R
Order 7: Relational	<: Less than, <=: Less than equal to, >: Greater than, >=: Greater than or equal to	L → R
Order 8: Equality	== : Equal to, != : Not equal to	L → R
Order 9: Bitwise AND	& : Bitwise AND	L → R
Order 10: Bitwise XOR	^ : Bitwise XOR	L → R
Order 11: Bitwise OR	|: Bitwise OR	L → R
Order 12: Logical AND	&& : Logical AND	L → R
Order	Operators	Associativity
Order 13: Logical OR	|| Logical OR	L → R
Order 14: Conditional	? : Ternary Operator	R → L
Order 15 Assignment	= : Assignment , *= : Assign Product, %= : Assign Remainder, /= : Assign Quotient, +=: Assign Sum, -= : Assign Difference, &=: Assign Bitwise AND, ^= Assign Bitwise XOR, |= : Assign Bitwise OR, <<= : Assign Left Shift, >>= : Assign Right Shift	R → L
Order 16: Comma	, Comma	L → R

Mathematical Functions

- Most C compilers support basic math functions.

- The math.h header file should be included to use the math functions.
- Note
  - ✓ The parameters x and y are double.
  - ✓ In trigonometric and hyperbolic functions x and y are in radians.
  - ✓ All functions return a double.
- log10(x) base-10 logarithm of x, x > 0
- pow(x,y) raise x to the power of y, xy
  - ✓ sin(x) sine of x
- sinh(x) hyperbolic sine of x
- sqrt(x) square root of x, x>=0
- tan(x) tangent of x
- tanh(x) hyperbolic tangent of x
- acos(x) inverse cosine of x,
- asin(x) inverse sine of x,
- atan(x) one-parameter inverse tangent of x,
- atan2(x,y) two-parameter inverse tangent of x/y
- ceil(x) the smallest integer not less than x
- cos(x) cosine of x
- floor(x) the largest integer not greater than x

Questions

- Find the values of x and a in the following program segments

int a;
float x;
a = 1600/1000 + 2.5;
x = 1600/1000 + 2.5;
printf("%d %f",a,x);
	int a;
	float x;
	a = 1600/1000.0 + 2.5;
	x = 1600/1000 .0+ 2.5;
	printf("%d %f",a,x);
		int a;
		float x;
		a = 4 * 3% 5 +8 / 3 * 2;
		x = 4 * 3% 5 +8 / 3 * 2;
		printf("%d %f",a,x);

int a = 25;
float x = .25;
printf("%d %d",a,x);		

- What are math functions? Write any six math functions.
- List the binary, unary and ternary operators in C. Give examples in each case.
- Evaluate the following segments
  - ✓ int K = 5, j = 7, i; i = K>j?K<3?0:1:K>3?2:3;
  - ✓ int K = 15, j = 7, i; i = K>j?K<3?0:1:K>3?2:3;
  - ✓ int i = 5, j; j = i++;
  - ✓ int i = 5, j; j = ++i;

STATEMENTS IN C

Selection statement

- It provides two options
  - ✓ If
  - ✓ switch

Decision making with if statement

- These statements are called control statements.
- These statements help to jump from one part of the program to another.
- The control transfer may be
  - ✓ Conditional
  - ✓ Unconditional

IF STATEMENT

- It is the simplest form of the control statement
- It is very frequently used in decision making and allowing the flow of program execution.
- The If structure has the following syntax
  - ✓ **if (condition)** // The condition is any valid C' language expression

    statement1; // The statement is any valid C' language statement.

    statement2;
- Logical operators are frequently used in the condition statement.
- The condition part should **not end with a semicolon,** since the condition and statement should be put together as a single statement.
- The command says if the condition is true then perform the following statement or if the condition is false the computer skips the statement and moves on to the next instruction in the program.

IF ELSE STATEMENT

- It is also known as the two-way selection.
- The syntax of the if else construct is as follows

```
✓ if (condition)
          statement1;
    else
          statement2;
  statement x;
```

- If the result of the condition is true, then statement1 is executed, otherwise statement2 will be executed.
- If any case either statement1 is executed or statement2 is executed but not both.

NESTING OF IF .. ELSE

- The if statement may itself contain another if statement is known as **nested if statement**.

```
✓ if (condition1)
     if (condition2)
         statement-1;
     else
         statement-2;
   else
      statement-3;
   statement-4;
```

```
✓ if (condition1)
      if (condition2)
         statement-1;
      else
          statement-2;
    else
       if (condition3)
           statement-3;
       else
           statement-4;
   statement-5;
```

- The if statement may be nested as deeply as you need to nest it.
- One block of code will only be executed if two conditions are true.
  - ✓ Condition 1 is tested first and then condition 2 is tested.
  - ✓ The second if condition is nested in the first.
- The second if condition is tested only when the first condition is true else the program flow will skip to the corresponding else statement.

ELSE IF LADDER

- When a series of many conditions have to be checked use the ladder **else if** statement which takes the following general form.

- ✓ **if (condition1) statement – 1;**

```
if (condition1) statement – 1;
    else if (condition2) statement2;
        else if (condition3) statement3;
            else if (condition) statement n;
                else default statement;
statement-x;
```

- This construct is known as **if else construct** or **ladder**.
- The conditions are evaluated from the top of the ladder to downwards.
- As soon on the true condition is found, the statement associated with it is executed and the control is transferred to the statement – x (skipping the rest of the ladder).
- When all the condition becomes false, the final else containing the default statement will be executed.

SWITCH CASE

- It is also known as **Multi-way selection**.
- Unlike the if statement which allows a selection of two alternatives the switch statement allows a program to select one statement for execution out of a set of alternatives.
- During the execution of the switch statement only one of the possible statements will be executed and the remaining statements will be skipped.
- The usage of multiple if else statement increases the complexity of the program since when the number of if else statements increase
  - ✓ it affects the readability of the program
  - ✓ makes it difficult to follow the program.
- The switch statement removes these disadvantages by using a simple and straight forward approach.
  - ✓

```
switch (expression)
{
        case label-1: block_1; break;
        case label-2: block_2; break;
        case label-n: block_n; break;
        ..................
        default: block_default;
}
```

- When the switch statement is executed the control expression is evaluated first and the value is compared with the case label values in the given order.

- If the label matches with the value of the expression then the control is transferred directly to the group of statements which follow the label.
- If none of the statements matches then the statement against the default is executed.
- The default statement is optional in switch statement.
- If no default statement is given and if none of the condition matches then no action takes place.
- In this case the control transfers to the next statement written after the switch statement.

Switch – points

- Tests for equality.
    - ✓ If evaluates any type of relational or logical expression.
- No two case statements in the same switch can have identical values.
- If character constants are used in switch they are automatically converted to integers.

The ?: operator

- This is the conditional operator.
    - ✓ **Conditional expression? Expression1 : Expression2;**
        - The **conditional expression** is evaluated first.
            - ∞ If the result is nonzero **expression1** is evaluated and is returned as the value of the conditional expression.
            - ∞ Otherwise expression2 is evaluated and its value is returned.
- Example
    - ✓ if(x<0) flag = 0;
      else flag = 1;
        - Can be written as **flag=(x<0)?0:1;**
- Nested conditional operators can also be used.
- Example
    - ✓ Find the largest of three numbers
        - **max=((a>b)&&(a>c)?a:(b>c)?b:c);**
        - Is equivalent to
            - ∞ if((a>b)&&(a>c)) printf("A is largest");
              else if(b>c) printf("B is largest");
              else printf("C is largest");

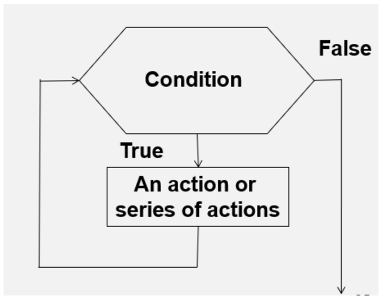
ITERATOR STATEMENT

- They are also called as loops.
- Allow a certain set of instructions to be repeatedly executed until a certain condition is reached.

- A program loop consists of two segments one known as **body of the loop** and other is the **control statement**.
- The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.
- The looping process in general would include the following four steps
  1. Setting and initialization of a counter
  2. Execution of the statements in the loop
  3. Test for specified conditions for the execution of the loop
  4. Incrementing or decrementing the counter
- The test may be either to determine whether the loop has repeated the specified number of times or to determine whether the particular condition has been met.
- There are two terminating condition tests
  - ✓ Pre-test
  - ✓ Post-test

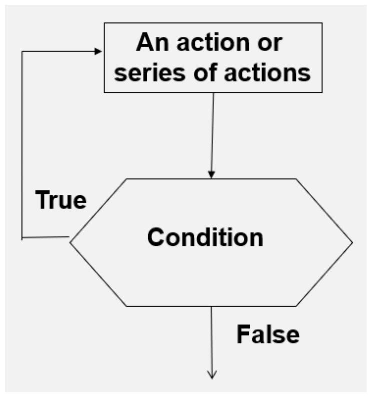
Pre-test Loop

- In each iteration the control expression is tested first.
- If it is true, the loop continues otherwise it is terminated.
- The action may be done zero, one or more times.



Post-test loop

- In each iteration, the loop actions are executed.
- The control expression is tested then.
- If it is true, a new iteration is started; else the loop terminates.
- The action is done one or more times.



LOOP INITIALIZATION

- It must be done before the first execution of the loop body.
- It sets the stage for the loop actions.

LOOP UPDATE

- The condition that controls the loop must be true for a while and then change to false.
- If this does not happen it would become an infinite loop.
- The action that causes the changes are known as loop updates.
- Updating is done in each iteration usually as the last action.
- If the body of the loop is repeated '***n***' times, the updating is normally done '***n+1***' times.

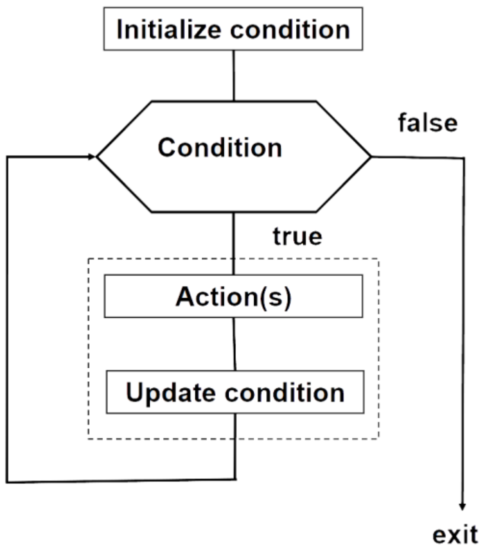
Types of loops

- There are two kinds of loops
  - ✓ Event-controlled loop
  - ✓ Counter-controlled loop

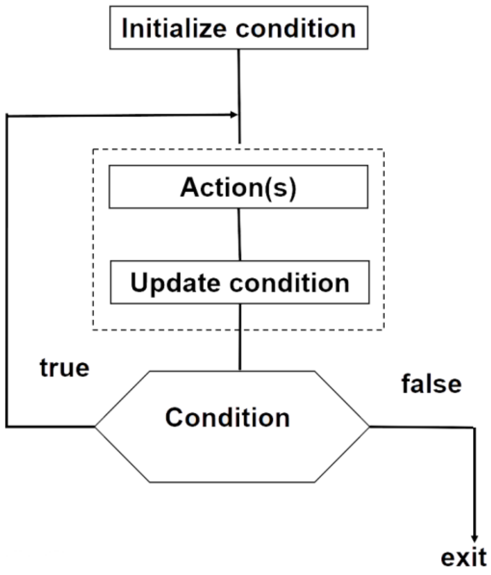
EVENT-CONTROLLED LOOPS

- An event changes the control expression from true to false.
- Example
  - ✓ When reading data, reaching the end of the data changes the expression from true to false.
  - ✓ The updating process can be implicit or explicit.

Event controlled PRE-TEST Loops

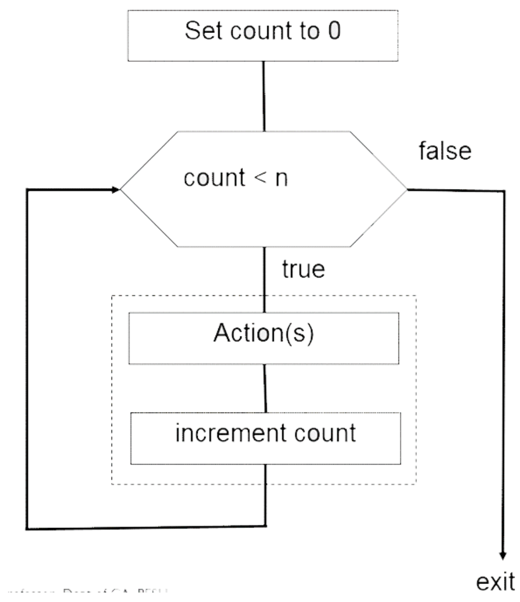


Event controlled – POST-TEST loops

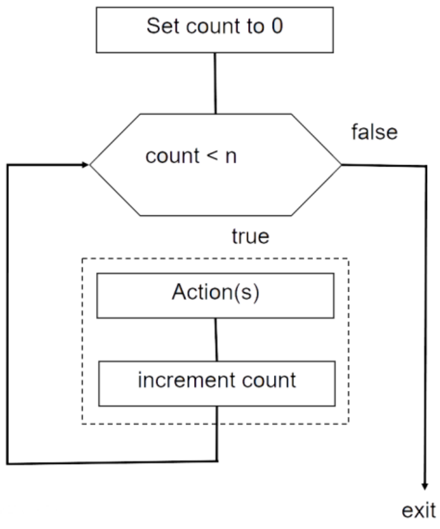


Counter controlled loops

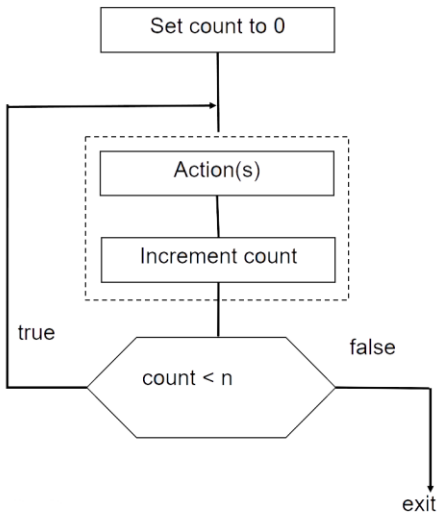
- When the number of times an action is to be executed is known counter-controlled loops are used.
- The number of times need not be a constant.
- It can be a variable or a calculated value.
- The update can be an increment or a decrement.



Counter-controlled Loops – Pre-test loops



Counter-controlled Loops – Post-test loops



WHILE STATEMENT

- It is a **pretest loop**.
- It uses an expression to control the loop.
- The general format of the while statement is

- **while (test condition)**

```
{
    body of the loop ;
}
```

- The given test condition is evaluated and if the condition is true then the body of the loop is executed.
- After the execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again.
- This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop.
- On exit, the program continues with the statements immediately after the body of the loop.
- The body of the loop may have one or more statements.
- The braces are needed only if the body contained two are more statements.

DO-WHILE STATEMENT

- It is a **post-test loop**.
- It uses an expression to control the loop.
- It tests the expression after the execution of the body.
- Since the body of the loop is executed first and then the loop condition is checked we can be assured that the body of the loop is executed at least once.
- The syntax of the do while loop is
  - ✓ **do**

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

- Here the statement is executed, then expression is evaluated.
- If the condition expression is true then the body is executed again and this process continues till the conditional expression becomes false.
- When the expression becomes false the loop terminates.

FOR STATEMENT

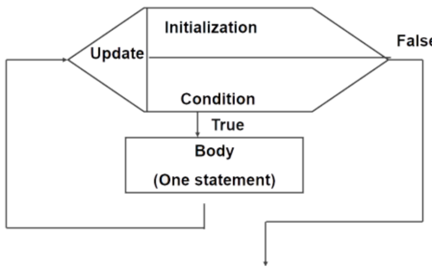
- Complex statement that has three parts
  - ✓ Loop initialization
    - Normally sets the counter
  - ✓ Limit test
  - ✓ End-of-loop action statements
    - Usually increment or decrement the counter
- Any of the parts can be null
- It is a pretest loop
  - ✓ If the terminating condition is true at the start, the body of the for is skipped.

- ✓ The general form of the for loop is:
  - **for (initialization; test condition; increment)**

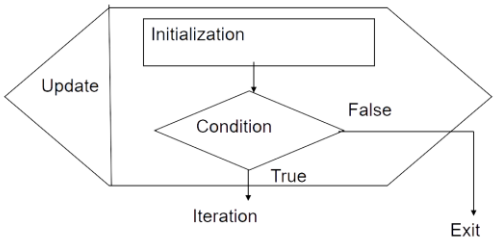
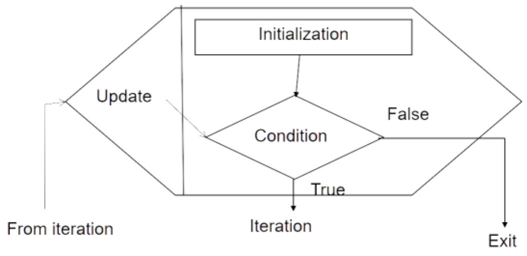
    {

    body of the loop;

    }



- 'For' loop uses three actions
  - ✓ Initialization
  - ✓ Test
  - ✓ Update
- Only two of these actions are required in each iteration flow.

✓ In the first flow When the loop is entered ∞ Only initialization and loop are used	 Initialization Update False Condition True Iteration Exit
✓ In all other iterations, only update and tests are used	 Initialization Update False Condition True From iteration Iteration Exit

- Additional features of the for loop:
  - ✓ Include multiple expressions in any of the fields of for loop provided that such expressions are separated by commas.
  - ✓ Example
    - for(i = 0, j = 100; i < 10;i++, j=j-10)
      - ∞ Sets up two index variables i and j.
      - ∞ i is initialized to zero and the latter to 100 before the loop begins.
      - ∞ Each time after the body of the loop is executed, the value of i will be incremented by 1 while the value of j is decremented by 10.
- A need may arise to omit on or more fields from the for statement.
- This can be done simply by omitting the desired filed, but by marking its place with a semicolon.

- The init_expression field can simply be “left blank” in such a case as long as the semicolon is still included.
    - ✓ for(;j!=100;++j)
        - The above statement might be used if j were already set to some initial value before the loop was entered.
- A for loop that has its looping condition field omitted effectively sets up an infinite loop, that is a loop that theoretically will be executed for ever.

NESTED LOOPS

- These loops are the loops which contain another looping statement in a single loop.
- Any loop can contain a number of loop statements in itself.
- If we are using a loop within another loop it is called a nested loop.

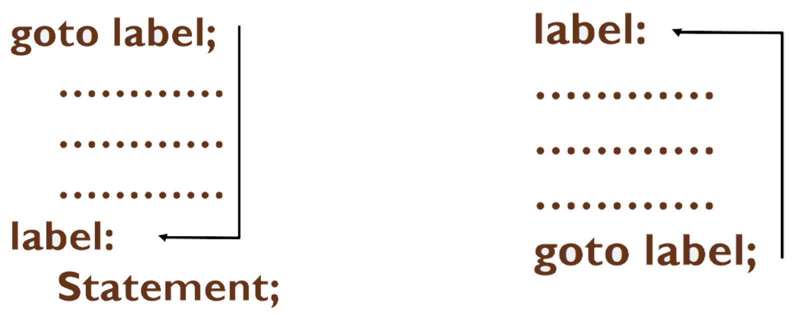
```
✓ for (initializing1 ; test condition1 ; update1)
  {
        statement;
        for (initializing2 ; test condition2 ; update2)
        {  body of inner loop;          }
        statement;
  }
```

- The inner loop will be executed in full for every execution of the outer loop.
- When working with nested loops, the outer loop changes only after the inner loop is completely finished (or is interrupted).
- The total number of iterations in a nested loop
    - ✓ **Iterations = outer loop iterations * inner loop iteration**

GOTO STATEMENT

- It is a simple statement that is used to transfer the program control unconditionally from one statement to another statement.
- Although it might not be essential to use the **goto** statement in a highly structured language like C, there may be occasions when the use of goto is desirable.

Forward Jump **Backward Jump**



- The goto requires a label in order to identify the place where the branch is to be made.
- A label is a valid variable name followed by a colon.
- The label is placed immediately before the statement where the control is to be transformed.
- The goto breaks the normal sequential execution of the program.
- A program may contain several goto statements that transfers control to the same place when a program executes.

BREAK STATEMENT

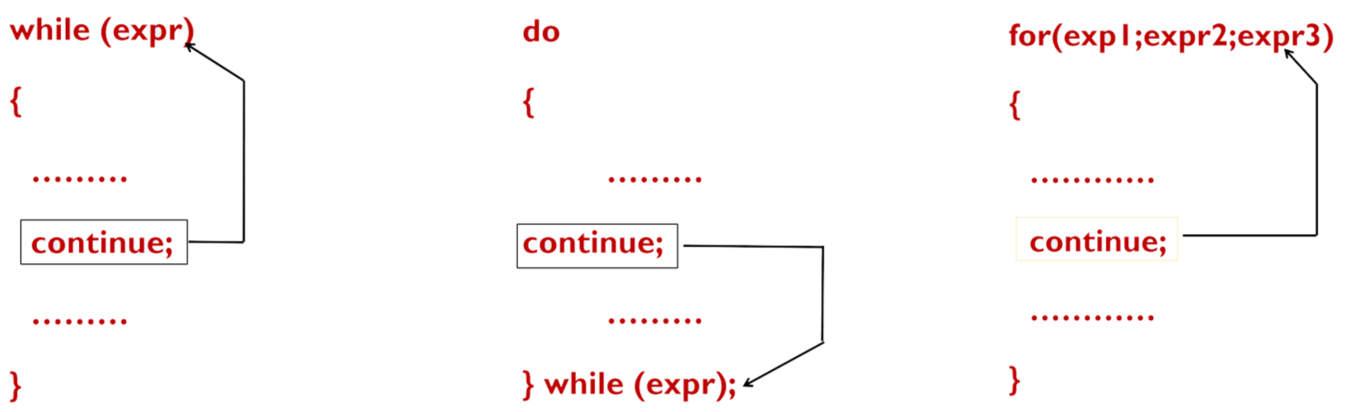
- It terminates the execution of the nearest enclosing switch statement in which it appears.
- Control passes to the statement that follows the terminated statement.
- Syntax
    - ✓ jump_statement: break;

JUMPS IN LOOPS

- When executing a loop it may be necessary to skip a part of the loop or to leave the loop as soon as a certain condition occurs.

CONTINUE STATEMENT

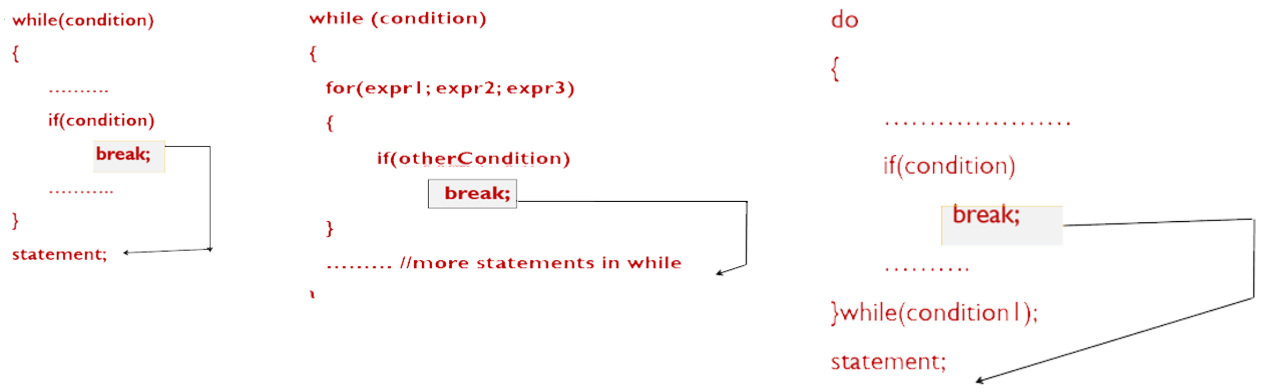
- Sometimes it is necessary to skip a part of the body of the loop under certain conditions.
- C provides a statement known as **continue**.
- It causes the loop to be continued with the next iteration after skipping any statements in between.
    - ✓ **continue;**
- It transfers to the testing expression in while and do..while loops.
- It transfers to the updating expression in the for statement.



```
while (expr)        do                      for(exp1;expr2;expr3)
{                   {                       {
  .........             .........             ............
  continue;         continue;                 continue;
  .........             .........             ............
}                   } while (expr);         }
```

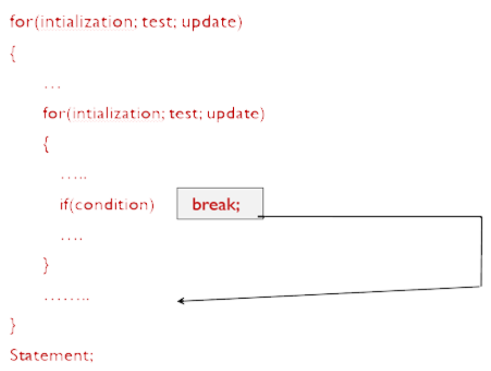
BREAK STATEMENT

- It has two uses.
  - ✓ Terminate a case in a switch.
  - ✓ Force immediate termination of a loop bypassing the normal loop conditional test.
- In nested loops break causes an exit from the loop that it has been written in.
- When written in switch that is inside a loop it only terminates the switch and not the loop.
- It causes a loop to terminate.
- It is equivalent to setting the loop's limit test to false.
- The program continues with the statement immediately after the loop.
- In a series of nested loops, break only terminates the inner loop.
- Break will exit only one loop.
- The break statement is used to break from an enclosing do, while, for, or switch statement.
- It is a compile error to use break anywhere else.



```
while(condition)        while (condition)                     do
{                       {                                     {
    ..........            for(expr1; expr2; expr3)                ....................
    if(condition)         {                                       if(condition)
        break;                if(otherCondition)                      break;
    ..........                    break;                          .........
}                         }                                   }while(condition1);
statement;                ......... //more statements in while  statement;
                        }
```

Dr. A Lekha, Associate Professor, Dept of CA, PESU


for(intialization; test; update)
{
...
for(intialization; test; update)
{
.....
if(condition)
break;
....
}
....
}
Statement;

- 'break' breaks the loop without executing the rest of the statements in the block.

EXIT FUNCTION

- An exit statement is not a program control statement.
- An exit statement is used to exit the program as a whole.
- It returns control to the operating system.
- After exit statement all memory and temporary storage areas are all flushed out and control goes out of program.
- In a program there can be only one exit statement.
- An exit statement is placed as the last statement in a program since after this program is totally exited.
- While a programmer uses exit to terminate the program if there is a error, C considers it a normal termination.
- The general formal of exit statement is
  - ✓ exit(status);
- Before termination exit
  - ✓ Closes all files
  - ✓ Writes buffered output
- A value of 0 for status indicates a normal exit.
- A non-zero value indicates some error.

EXPRESSION STATEMENT

- Expression statement is a valid expression followed by a semicolon.
- Examples
  - ✓ Func();
  - ✓ a=b+c;
  - ✓ i++;
  - ✓ ;   //Null statement

BLOCK STATEMENT

- Groups of related statements that are treated as a unit.
- Also called as **compound statements**.
- Begins with a { and terminated by a matching }.

PROGRAMMING EXAMPLES

- To print a triangle of the form

1	1	1	* * *
2 3	2 3	2 2	* *
4 5 6	4 5 6	3 3 3	*

- Generate 10 multiplication tables
- Writ a program to evaluate the following functions to 0.0001 accuracy
  - ✓ Sin(x) , Cos(x)
- Write a program using switch statement to declare the result based on marks as follows:

Marks	Result
<40	Fail
>=40 &<50	Pass
>=50 &<60	Second class
>=60 & <70	First Class
>=70	First Class Distinction

- Write a C program to find the sum of even numbers and odd numbers separately from 1 to N natural nos.
- Write a C program to input a number from 0 – 9 and display the number in words using if or if-else statement.
- Write a C program to find the least among three numbers using ternary operators.

QUESTIONS

- Explain the looping structures supported by C programming language. Differentiate the looping structure using flowcharts.
- Describe the continue, break and exit statements using examples.
- Mention the different loop structures available in C. Explain each of them briefly.
- Differentiate between normal and abnormal exit from a loop.
- What are nested loops? Give the rules for nested loops.
- With syntax and example explain the concept of a switch statement.
- Give the syntax of if..else statement and illustrate it with example.
- Explain simple if and if-else statements.