Unit-3
Handling Input Output
&
Control Statements

Course: BCA

Subject: Programming In C Language

CONTROL STATEMENTS

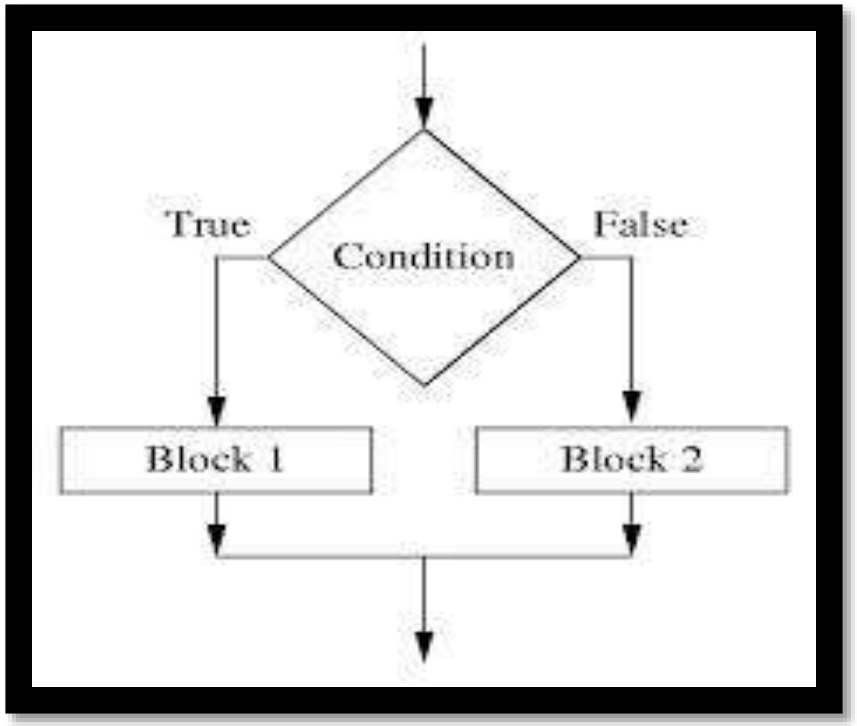
- C language supports the following statements known as *control or decision making* statements.

1. **if statement**
2. **switch statement**
3. conditional operator statement
4. **Go to statement**

IF STATEMENT

- The if statement is used to control the flow of execution of statements and is of the form
- **If(test expression)**

- Eg:
  if(bank balance is zero)
  Borrow money



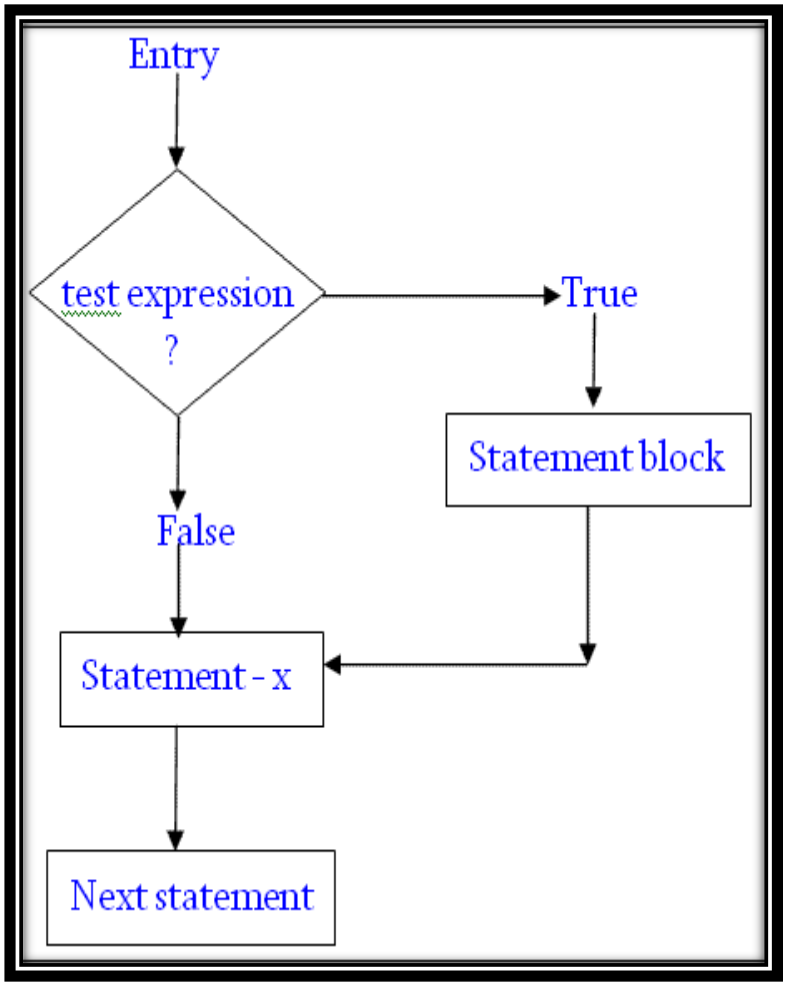
Cont..

- The **if statement may be implemented in different forms depending on the complexity of** conditions to be tested.

1. Simple **if statement**
2. **if.....else statement**
3. Nested **if.....else statement**
4. **elseif ladder**

Cont..

The general form of a simple **if statement is The 'statement-block' may be a single** statement or a group of statement



If(test exprn)
{
statement-block;
}
statement-x;

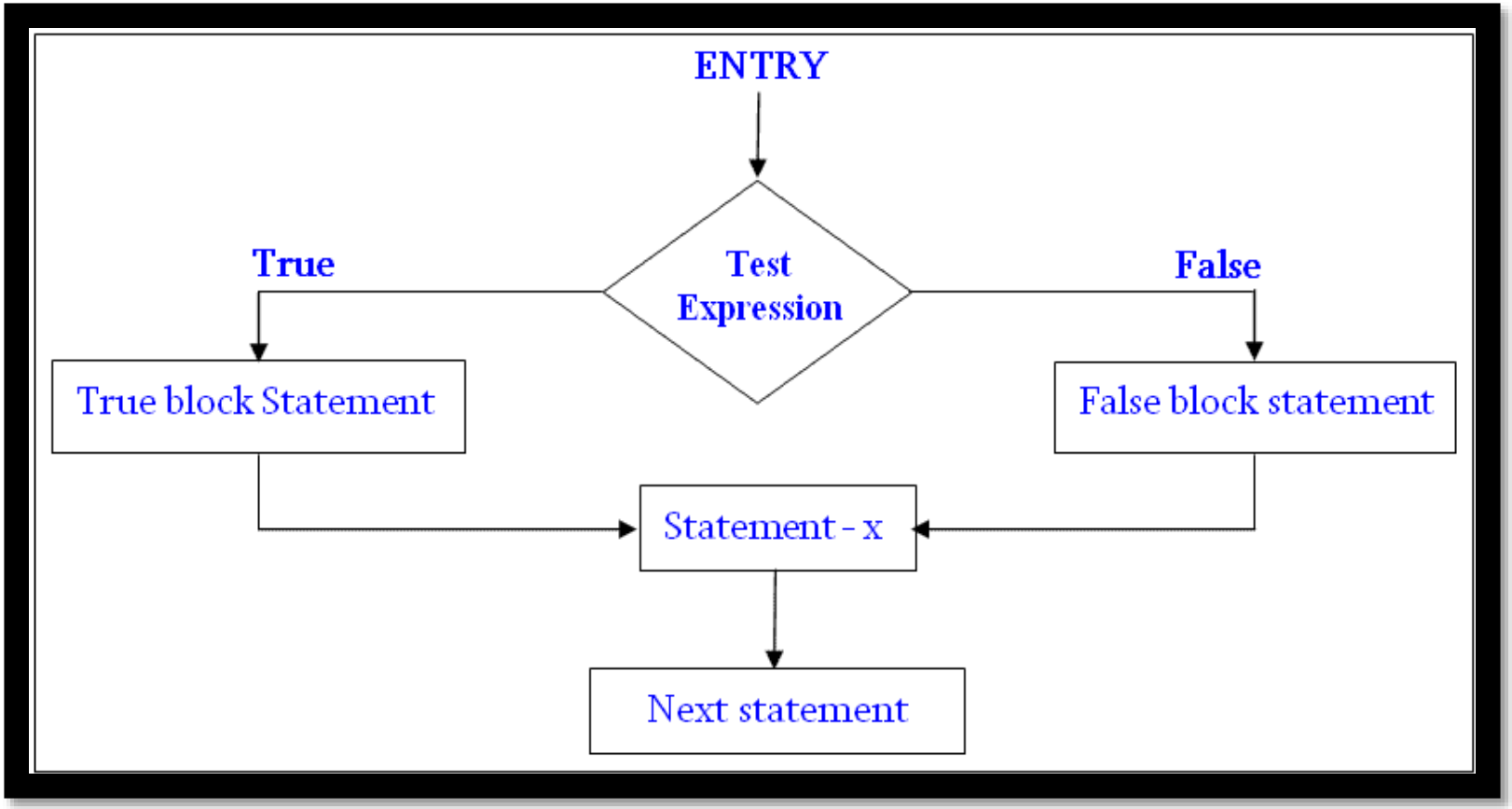
THE IF…ELSE STATEMENT

- The if….else statement is an extension of simple if statement. The general form is

```
If(test expression)
{
True-block statement(s)
}
else
{
False-block statement(s)
}
statement-x
```

Cont..

- If the *test expression is true, then the true block statements are executed; otherwise* the false block statement will be executed.



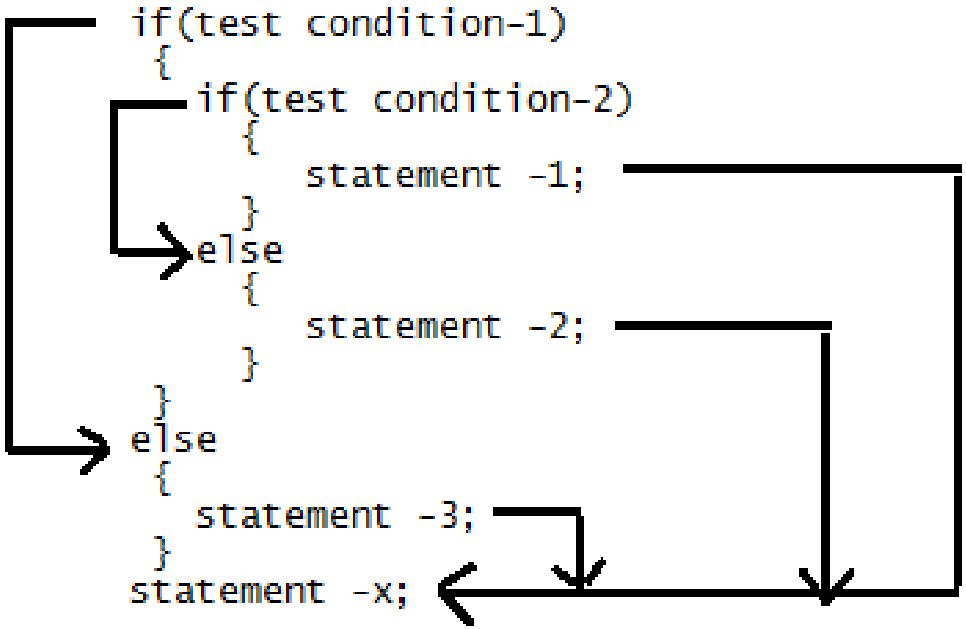
Cont..

- Eg:

```
.....
.....
if(code ==1)
boy = boy + 1;
if(code == 2)
girl =girl + 1;
.....
.....
```

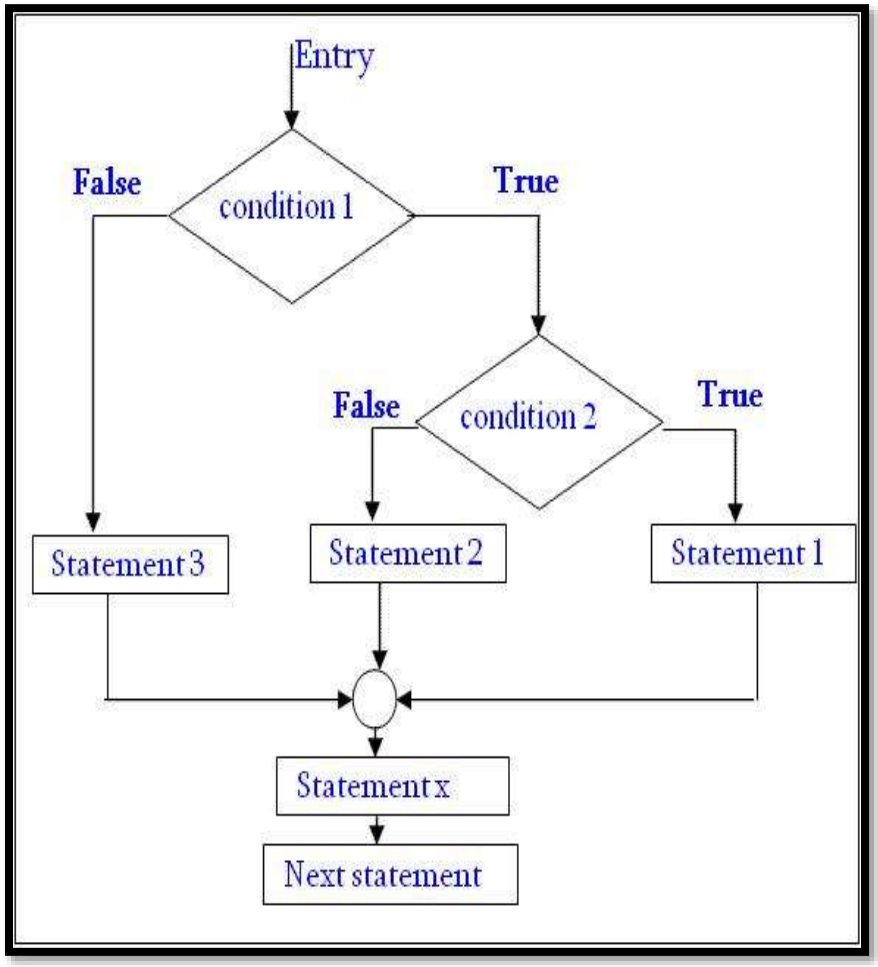
NESTING OF IF…..ELSE STATEMENTS

- When a series of decisions are involved, we may have to use more than one **if….else** statements, in *nested form as follows*



Cont..

- Here, if the condition 1 is **false** then it skipped to statement 3.
- But if the condition 1 is **true**, then it tests condition 2.
-
- If condition 2 is **true** then it executes statement 1 and if **false** then it executes statement 2.
- Then the control is transferred to the statement x.
- This can also be shown by the following flowchart,



Program
/*Selecting the largest of three values*/

```
main()
{
float A, B, C;
printf("Enter three values \n");
scanf("|%f %f %f",&A, &B, &C);
printf("\nLargest value is:");
if(A > B)
{ if(A > C)
printf("%f \n",A);
else
printf("%f \n",C);
}
else
{
if(C > B)
printf("%f \n",C);
else
printf("%f \n",B);
}
}
```

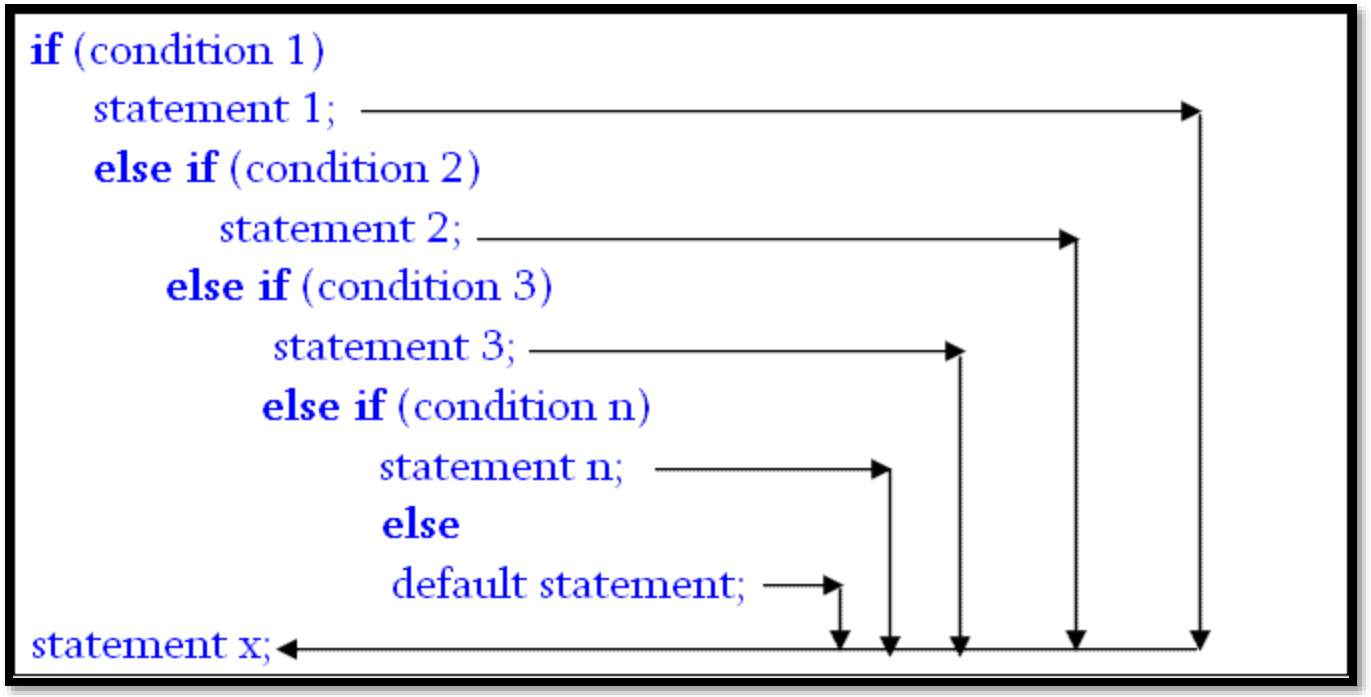
- OUTPUT
- Enter three values:
- 5 8 24
- Largest value is 24

The else if ladder

✓When a multipath decision is involved then we use **else if ladder.**

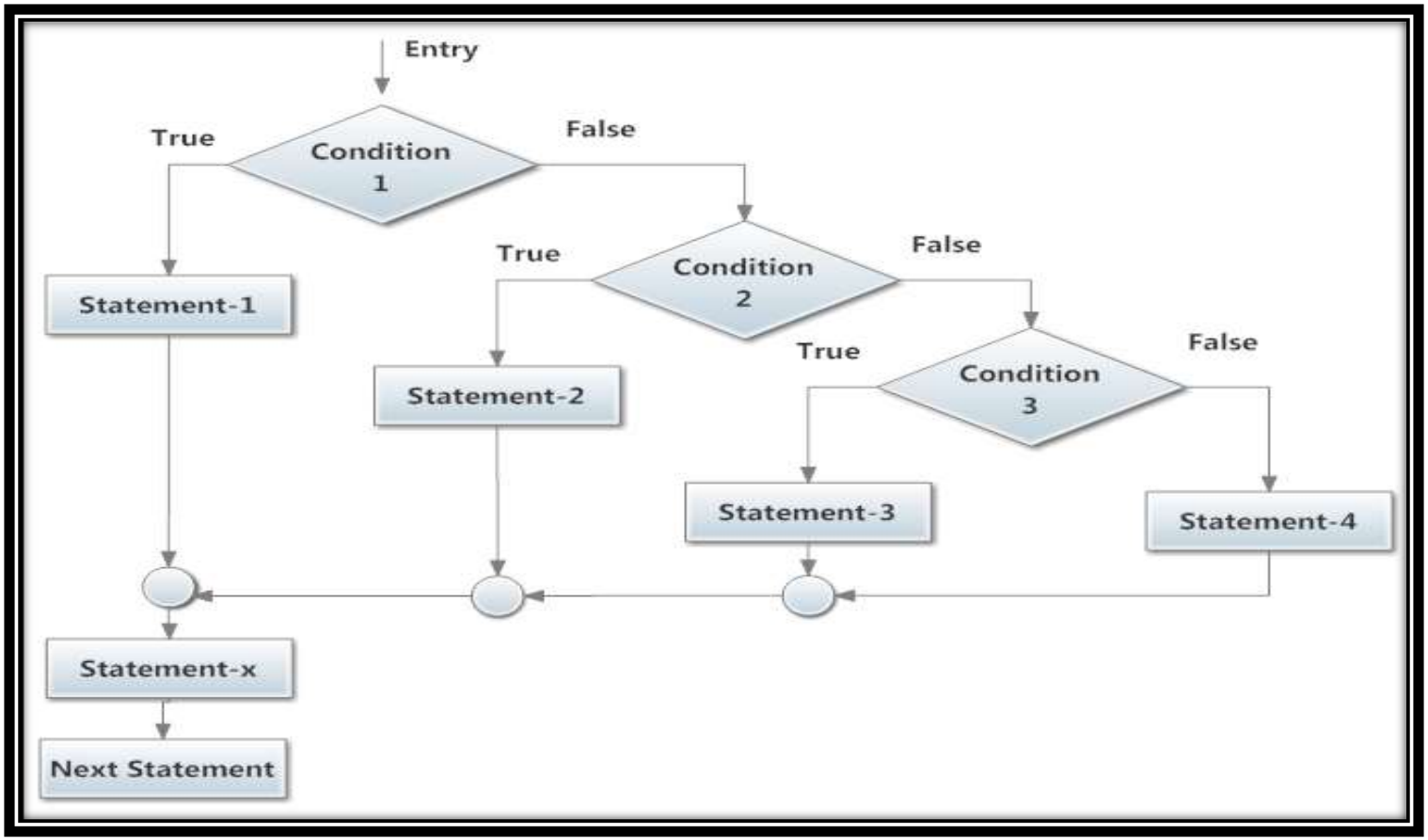
✓ A multipath decision is a chain of **if**s in which the statement associated with each **else** is an **if**.

✓It takes the following general form,



Cont..

- This construct is known as the **else if ladder**. The conditions are evaluated from the top, downwards. This can be shown by the following flowchart



THE SWITCH STATEMENT

- Switch statement is used for complex programs when the number of alternatives increases.

- The switch statement tests the value of the given variable against the list of **case** values and when a match is found, a block of statements associated with that case is executed.

SWITCH STATEMENT

- The general form of switch statement is

```
switch(expression)
{

        case value-1:
        block-1
        break;
        case value-2:
        block-2
        break;
        .......
        .......
        default:
        default-block
        break;
}
statement-x;
```

Example:

```
index = marks / 10;

switch(index)
{
case 10:
case 9:
case 8:
grade = “Honours”;
   break;
case 7:
case 6:
grade = “first division”;
   break;
case 5:
grade = “second division”;
break;
case 4:
grade = “third division”;
break;
default:
grade = “first division”; break
      }

printf(“%s \n”,grade);
.....
```

THE ?: OPERATOR

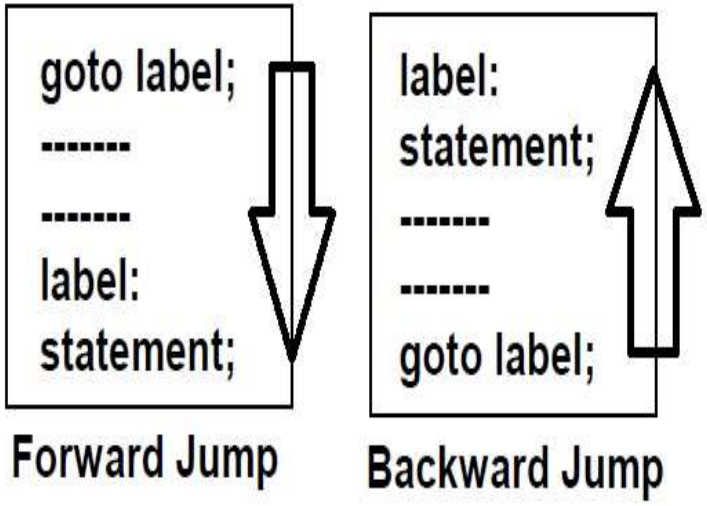
- The C language has an unusual operator, useful for making two-way decisions.
- This operator is a combination of **? and : and takes three operands.**
- **It is of the form** exp1?exp2:exp 3
- Here *exp1 is evaluated first. If it is true then the expression exp2 is evaluated and* becomes the value of the expression.
- If *exp1 is false then exp3 is evaluated and its value* becomes the value of the expression.

Eg:

```
if(x < 0)
flag = 0;
else
flag = 1;
can be written as
flag = (x < 0)? 0 : 1;
```

UNCONDITIONAL STATEMENTS - THE GOTO STATEMENT

- C supports the goto statement to branchunconditionally from one point of the program to another.
- The goto requires a *label in order to identify the place where the branch* is to be made.
- A **label is any valid variable name and must be followed by a colon.**



DECISION MAKING AND LOOPING

- In looping, a sequence of statements are executed until some conditions for termination of the loop are satisfied.

- In looping, a sequence of statements are executed until some conditions for termination of the loop are satisfied.

- A *program loop therefore consists of two segments, one* known as the ***body of the loop*** *and the other known as the* ***control statements.***

Cont..

- Depending on the position of the control statements in the loop, a control structure may be classified either as an ***entry-controlled loop or as the exit-controlled loop.***

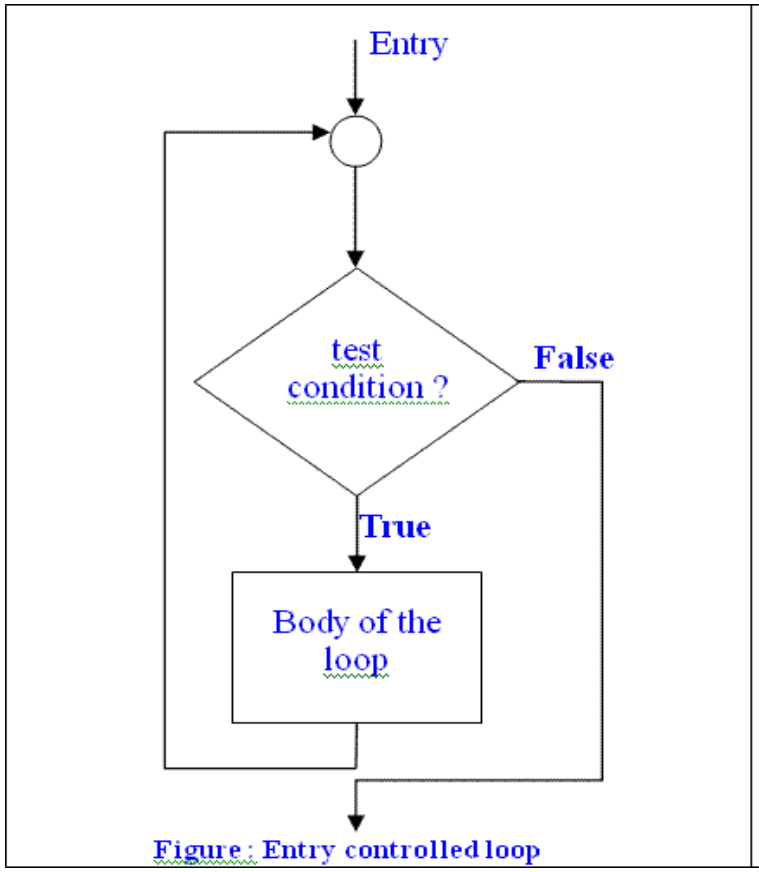


Figure : Entry controlled loop

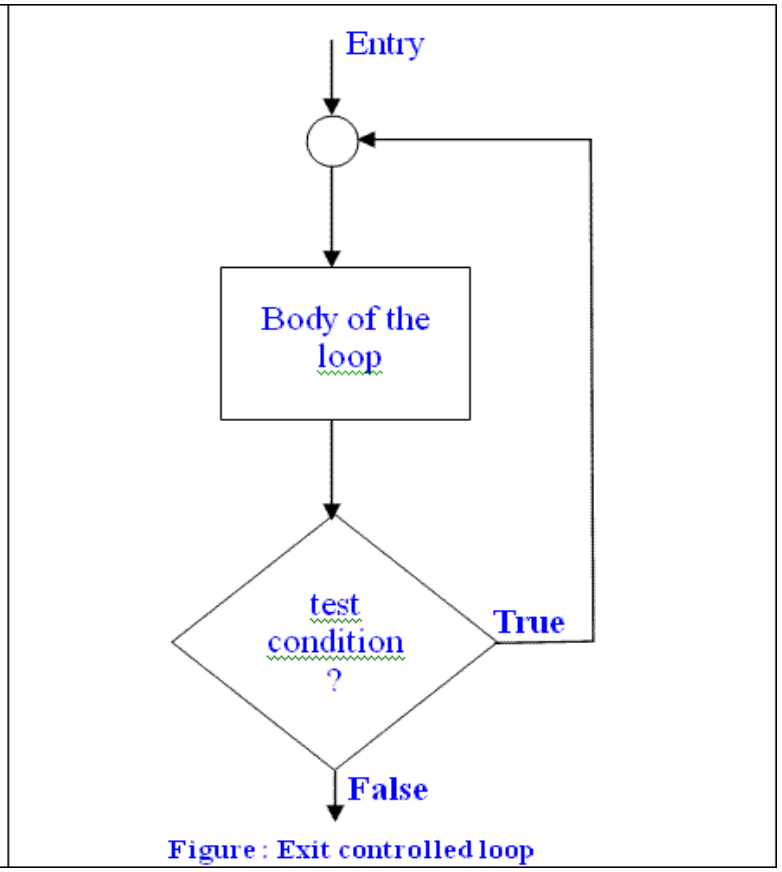


Figure : Exit controlled loop

Loops In C

- The C language provides for **three loop** constructs for performing loop operations.
- They are:

- The **while statement**
- The **do statement**
- The **for statement**

THE WHILE STATEMENT

- The basic format of the **while statement is**

```
while(test condition)
{
body of the loop
}
```

- The **while is an *entry–controlled loop statement*.**
- ***The test-condition is evaluated and if*** the condition is true, then the body of the loop is executed.
- After 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.*

Example of WHILE Loop

```
----------
----------
Body of the loop
test
condn?
while(test condition)
{
body of the loop
}
sum = 0;
n = 1;
while(n <= 10)
{
sum = sum + n* n;
n = n + 1;
}
printf("sum = %d \n",sum);
----------
----------
```

THE DO STATEMENT

- In while loop the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt.
- Such situations can be handled with the help of the **do** statement.

```
do
{
body of the loop
}
while(test condition);
```

Cont..

- Since the *test-condition is evaluated at the bottom of the loop, the* ***do.....while construct*** provides an *exit-controlled loop and therefore the body of the loop is always executed at least once.*

Eg:

```
----------
do
{
printf("Input a number\n");
number = getnum();
}
while(number > 0);
----------
```

THE FOR STATEMENT

- The for loop is another *entry-controlled loop that provides a more concise loop control* structure
- The general form of the **for loop is**

```
for(initialization ; test-condition ; increment
{
body of the loop
}
```

Cont..

- The execution of the **for statement is as follows:**

- *Initialization of the control variables is done first.*
- The value of the control variable is tested using the *test-condition.*

- *If the* condition is *true, the body of the loop is executed; otherwise the loop is* terminated and the execution continues with the statement that immediately follows the loop.

- When the body of the loop is executed, the control is transferred back to the **for statement after evaluating the last statement in the loop.**

- **Now, the control** variable is either incremented or decremented as per the condition.

For Statement

- Eg 1)

```
for(x = 0; x <= 9; x = x + 1)
{
printf)"%d",x);
}
printf("\n");
```

  - The multiple arguments in the increment section are possible and separated by *commas*.

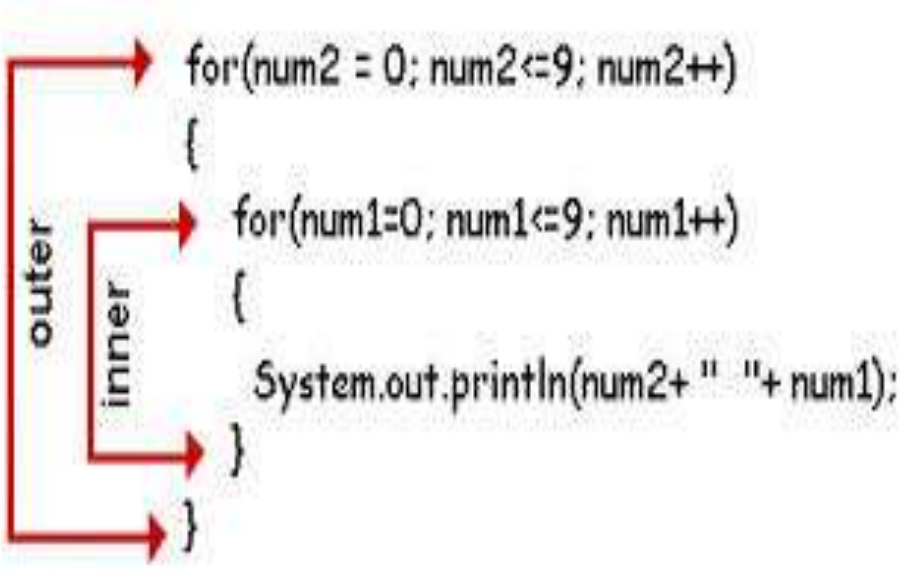
Cont..

- Eg 2)

```
 sum = 0;
for(i = 1; i < 20 && sum <100; ++i)
{
sum =sum + i;
printf("%d %d \n",sum);
}
for(initialization ; test-condition ; increment
{
body of the loop
}
```

Nesting of For Loops

- C allows one **for statement within another for statement.**



Cont..

- Eg:
- ----------
- ----------
- **for(row = 1; row <= ROWMAX; ++row)**
- **{**
- **for(column = 1; column < = COLMAX; ++column)**
- **{**
- **y = row * column;**
- **printf(“%4d”, y);**
- **}**
- **printf(“\n”);**
- **}**
- ----------
- ----------

JUMPS IN LOOPS

- C permits a jump from one statement to another within a loop as well as the jump outof a loop.

Jumping out of a Loop

- An early exit from a loop can be accomplished by using the break statement or the goto statement.
- When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.
- When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.

Skipping a part of a Loop

- Like the break statement, C supports another similar statement called the continue statement.
- However, unlike the break which causes the loop to be terminated, the continue, as the name implies, causes the loop to be *continued with the next iteration after skipping any* statements in between.

- The continue statement tells the compiler, "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION".
- The format of the continue statement is simply

```
continue;
```

Bypassing and continuing I Loops

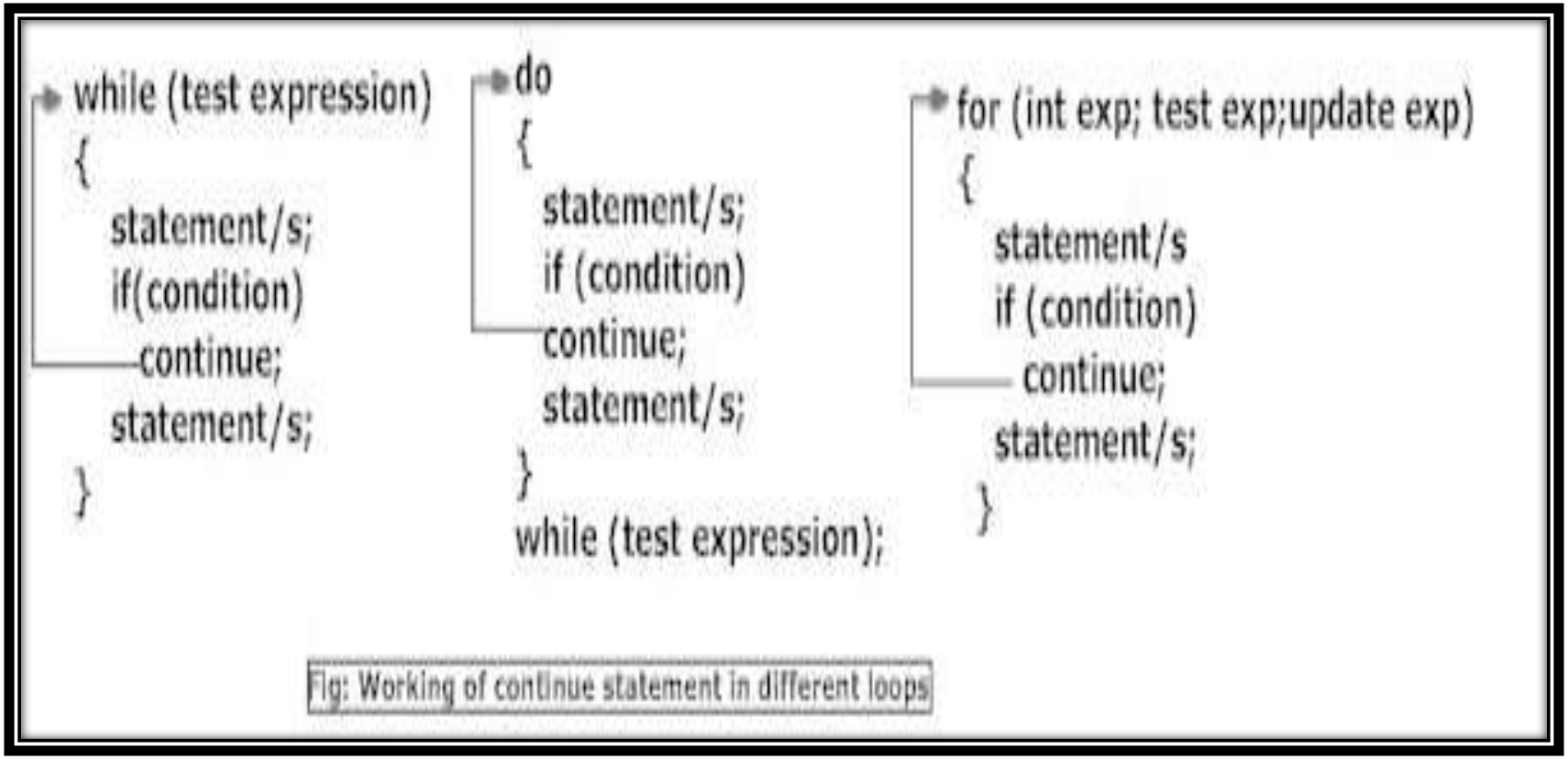


Fig: Working of continue statement in different loops

References

1. http://www.computer-books.us/c_0008.php
2. http://www.computer-books.us/c_0009
3. http://www.computer-books.us/c_2.php
4. www.tutorialspoint.com/cprogramming/cprogramming_pdf.
5. Programming in C by yashwant kanitkar
6. ANSI C by E.balagurusamy- TMG publication
7. Computer programming and Utilization by sanjay shah Mahajan Publication
8. www.**cprogramming**.com/**books**.html