

Loops & Decisions

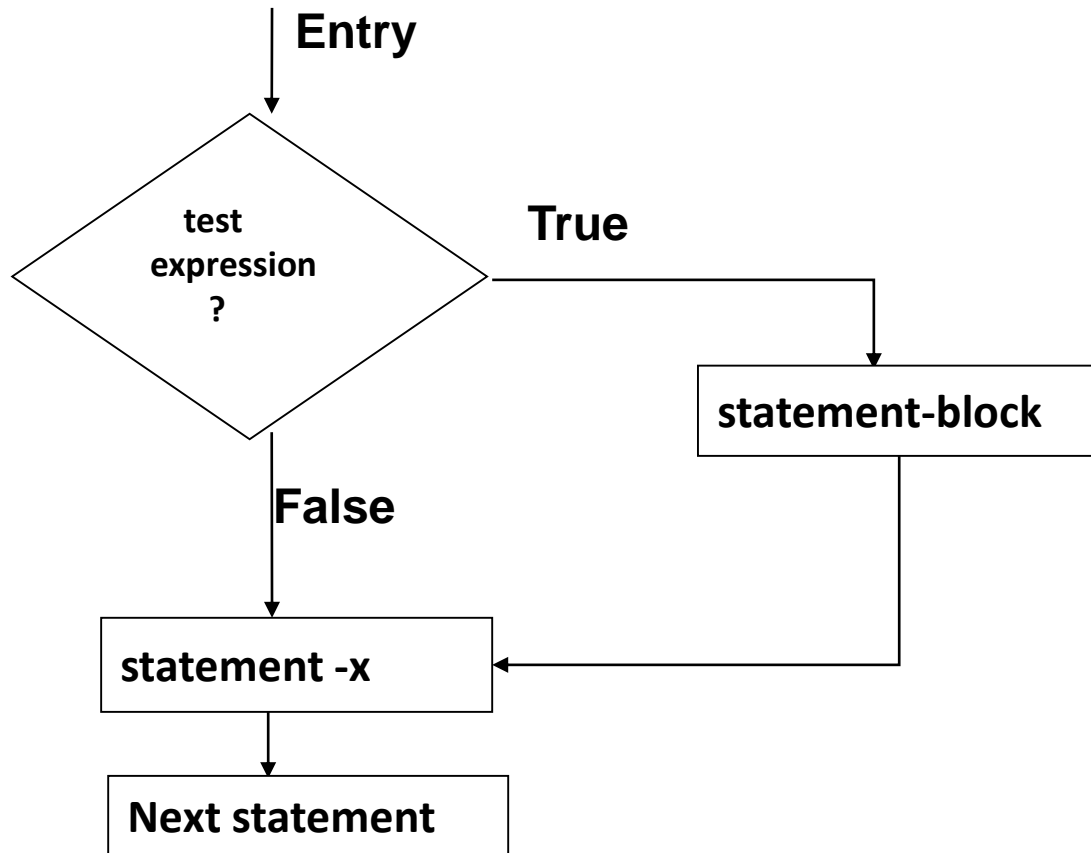
The “if” condition

```
// demonstrates IF statement
#include <iostream>
using namespace std;
int main()
{
    int x;
    cout << "Enter a number: ";
    cin >> x;
    if( x > 100 )
        cout << "That number is greater than 100\n";
    return 0;
}
```

General format:

```
if (test expression)
{
    statement-block;
}
statement-x;
```

Flow of Control of Simple if statement



Example:

.....

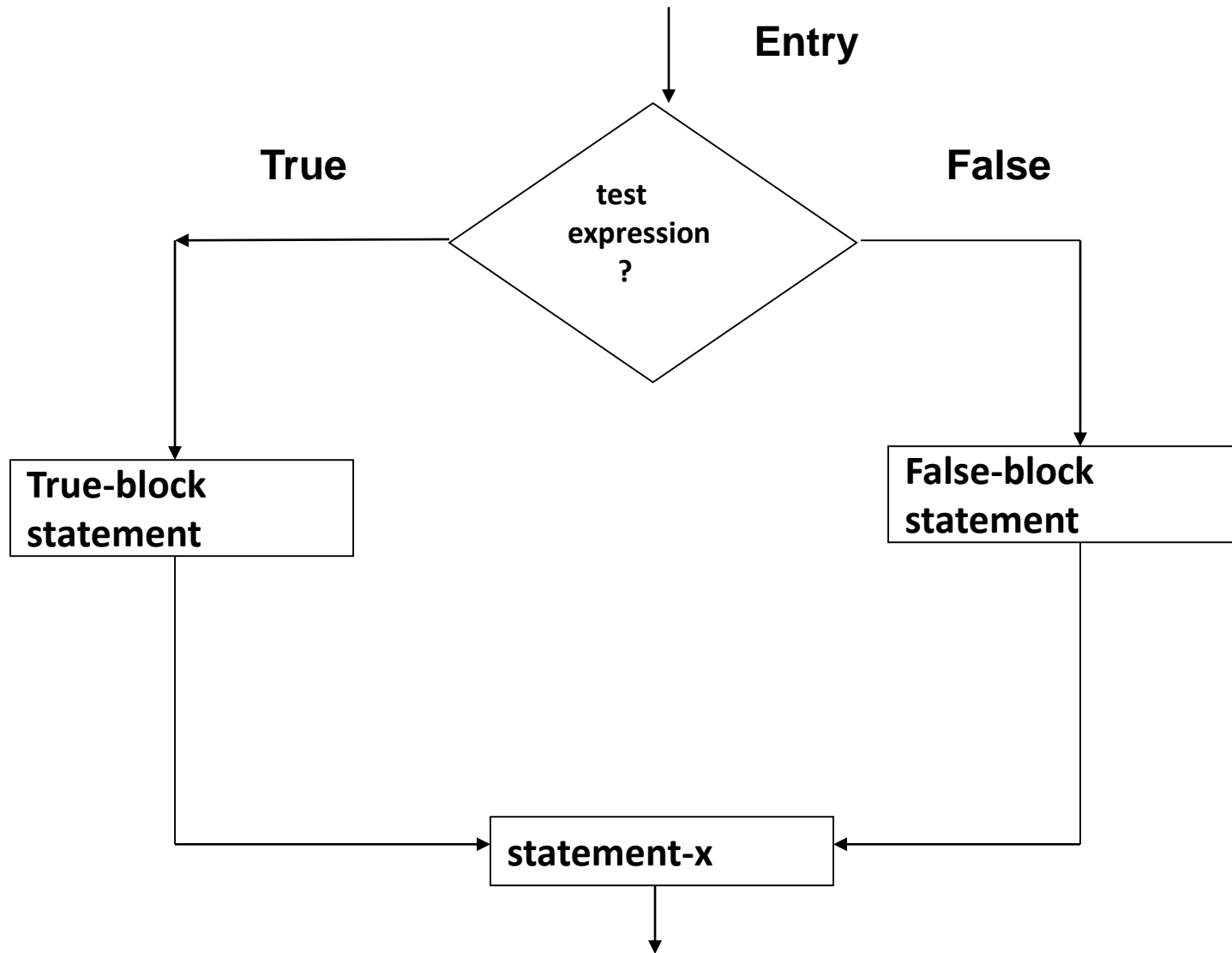
```
if (salary > 10000)
{
    salary = salary + commission;
    cout << salary;
}
cout << "end";
```

.....

Variations of if statement

- Simple if
- if...else
- Nested if...else
- else if ladder

If – else

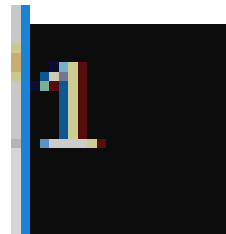


General form

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

```
3  int main()  
4  {  
5      int x = 10, y = 0;  
6      if( x > 10 )  
7          y = 10;  
8      y++;  
9      cout<<y;  
10     return 0;  
11 }
```

OUTPUT

A terminal window with a black background and a blue title bar. The number '1' is displayed in a light blue, monospaced font.

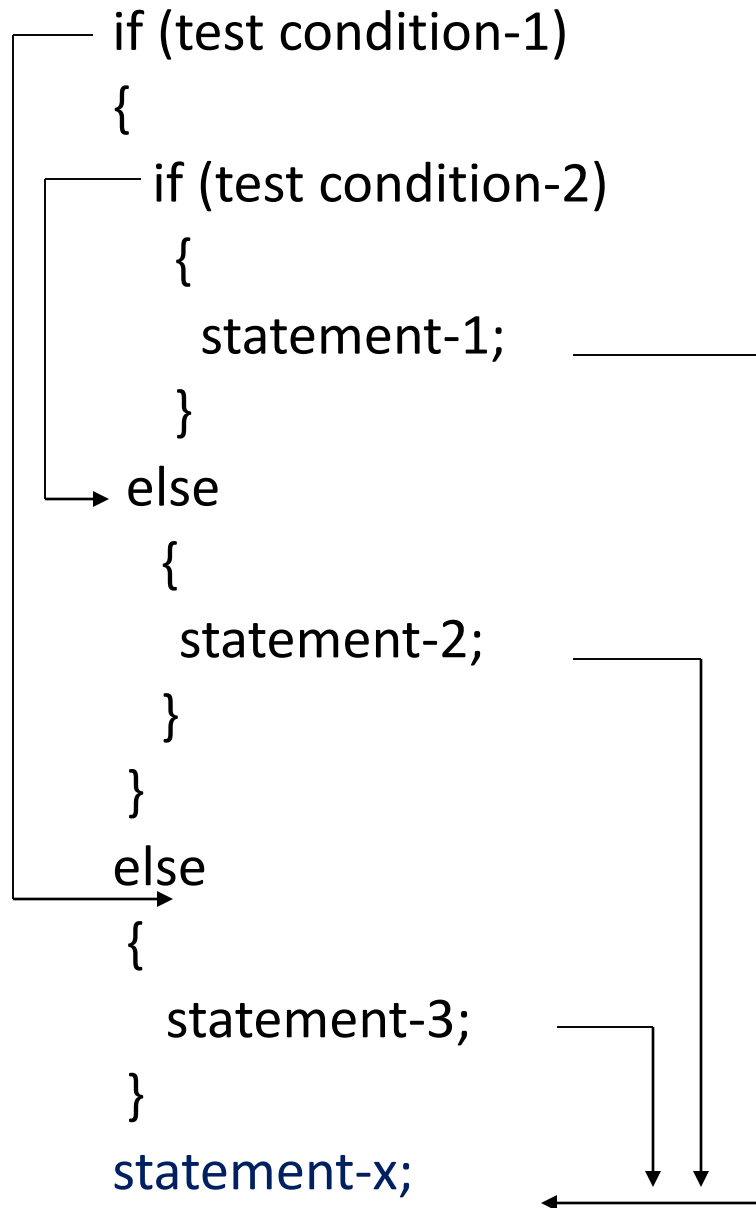
```
3  int main()  
4  {  
5      int x = 10, y = 0;  
6      if( x > 10 )  
7          y = 10;  
8      else  
9          y = 15;  
10     y++;  
11     cout<<y;  
12     return 0;  
13 }
```

OUTPUT

16

```
3  int main()  
4  {  
5      int x = 10, y = 0;  
6      if( x > 10 )  
7          y = 10;  
8      else  
9          y = 15;  
10     else  
11         y++;  
12     cout<<y;  
13     return 0;  
14 }
```

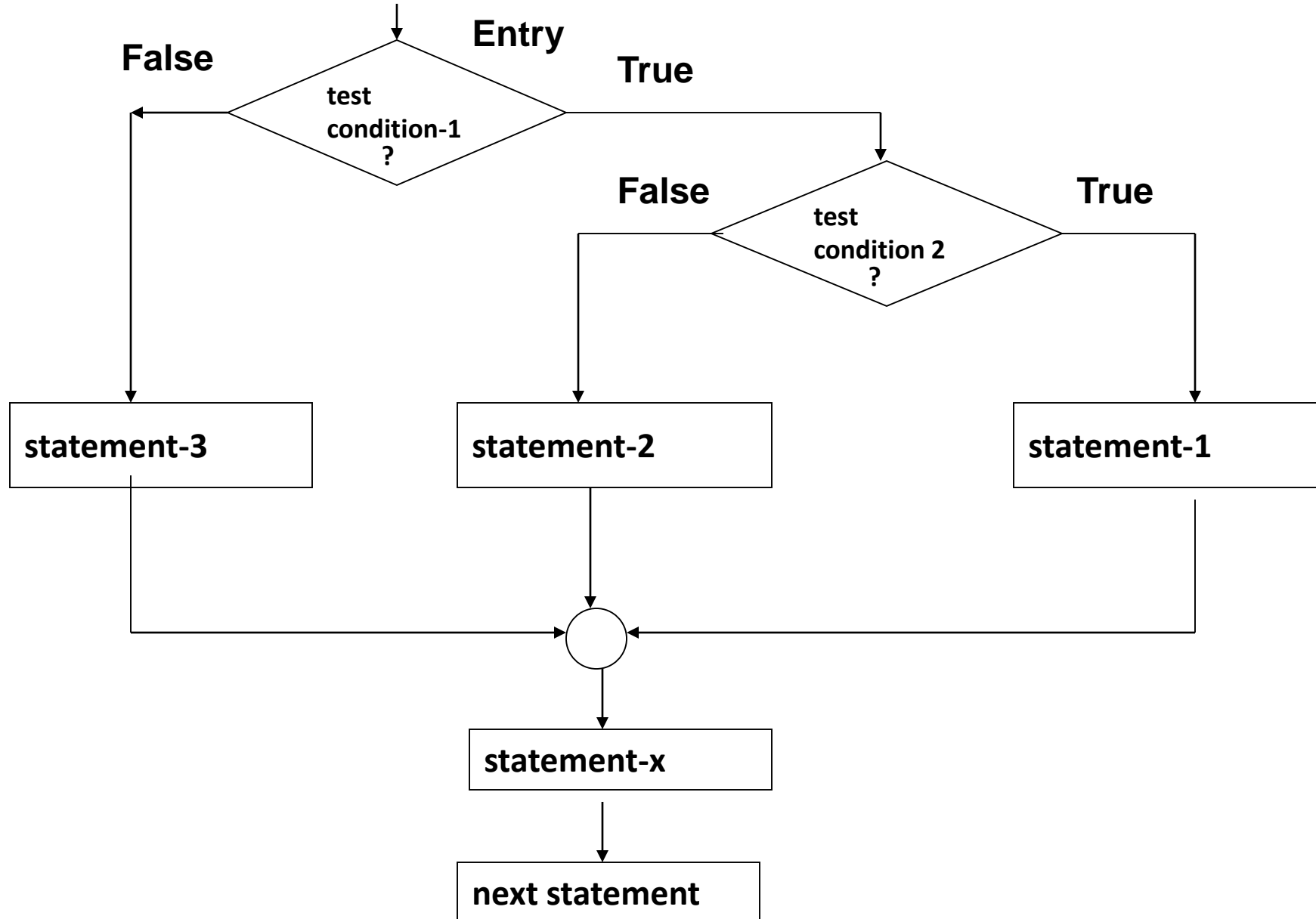
Syntax error !!



Nesting of if..else statements

- If the test condition-1 is false, statement-3 will be executed; otherwise it continues to perform the second test.
- If the test condition-2 is true, statement-1 will be evaluated; otherwise statement-2 will be evaluated and then the control is transferred to statement-x

Flow chart showing nesting of if..else statements



```
3  int main()  
4  {  
5      int a = 10, b = 5;  
6      if( a > 5)  
7          if( b > 5 )  
8              cout<<"a and b are >5";  
9      else  
10         cout<<"a is < 5";  
11     return 0;  
12 }
```

OUTPUT

```
a is < 5
```

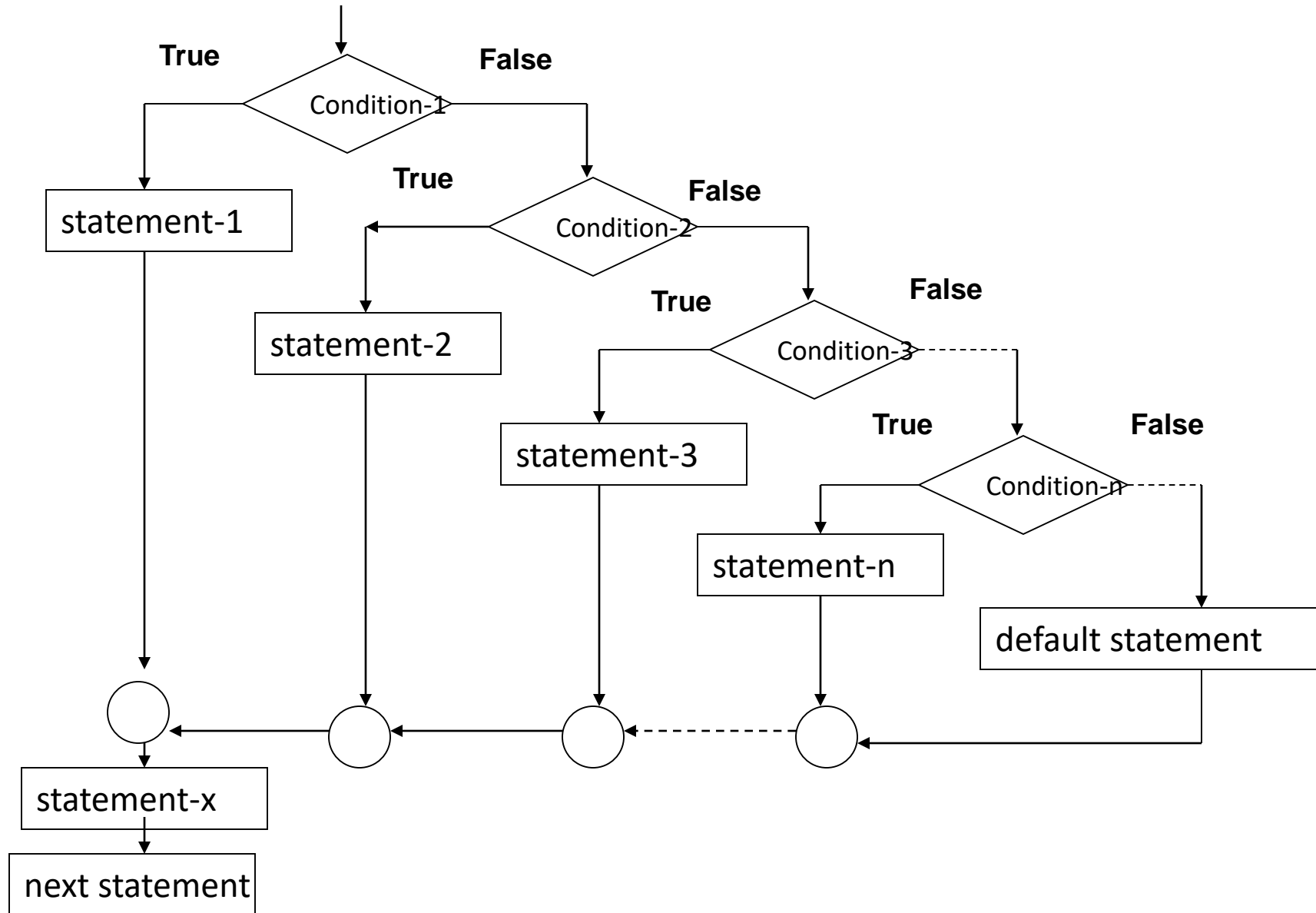
```
3  int main()
4  {
5      int a = 10, b = 5;
6      if( a > 5)
7      {
8          if( b > 5 )
9              cout<<"a and b are >5";
10     }
11     else
12         cout<<"a is < 5";
13     return 0;
14 }
```

else- if ladder

```
if (condition-1)
    statement-1;
else if (condition-2)
    statement-2;
else if (condition-3)
    statement-3;
else if (condition-n)
    statement-n;
else
    default-statement;
statement-x
```

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

Flow chart for else- if ladder



Example: To calculate the grade for the marks entered

avg -marks

grade

80-100

A

60-79

B

50-59

C

40-49

D

00-39

F

Program to calculate the grade for the marks entered

/* Program WITHOUT indentation */

```
int main()
{
    char grade;
    int marks;
    cout<<"enter marks";
    cin>>marks;
    if (marks>79)
        grade = 'A';
    else if (marks>59)
        grade = 'B';
    else if (marks>49)
        grade = 'C';
    else if (marks>39)
        grade = 'D';
    else
        grade = 'F';
    cout<<"\n grade= "<<grade;
}
```


Program to calculate the grade for the marks entered

/ Program WITH indentation */*

```
int main()
{
    char grade;
    int marks;
    cout<<"enter marks";
    cin>>marks;
    if (marks>79)
        grade = 'A';
    else if (marks>59)
        grade = 'B';
    else if (marks>49)
        grade = 'C';
    else if (marks>39)
        grade = 'D';
    else
        grade = 'F';
    cout<<"\n grade="<< grade;
}
```

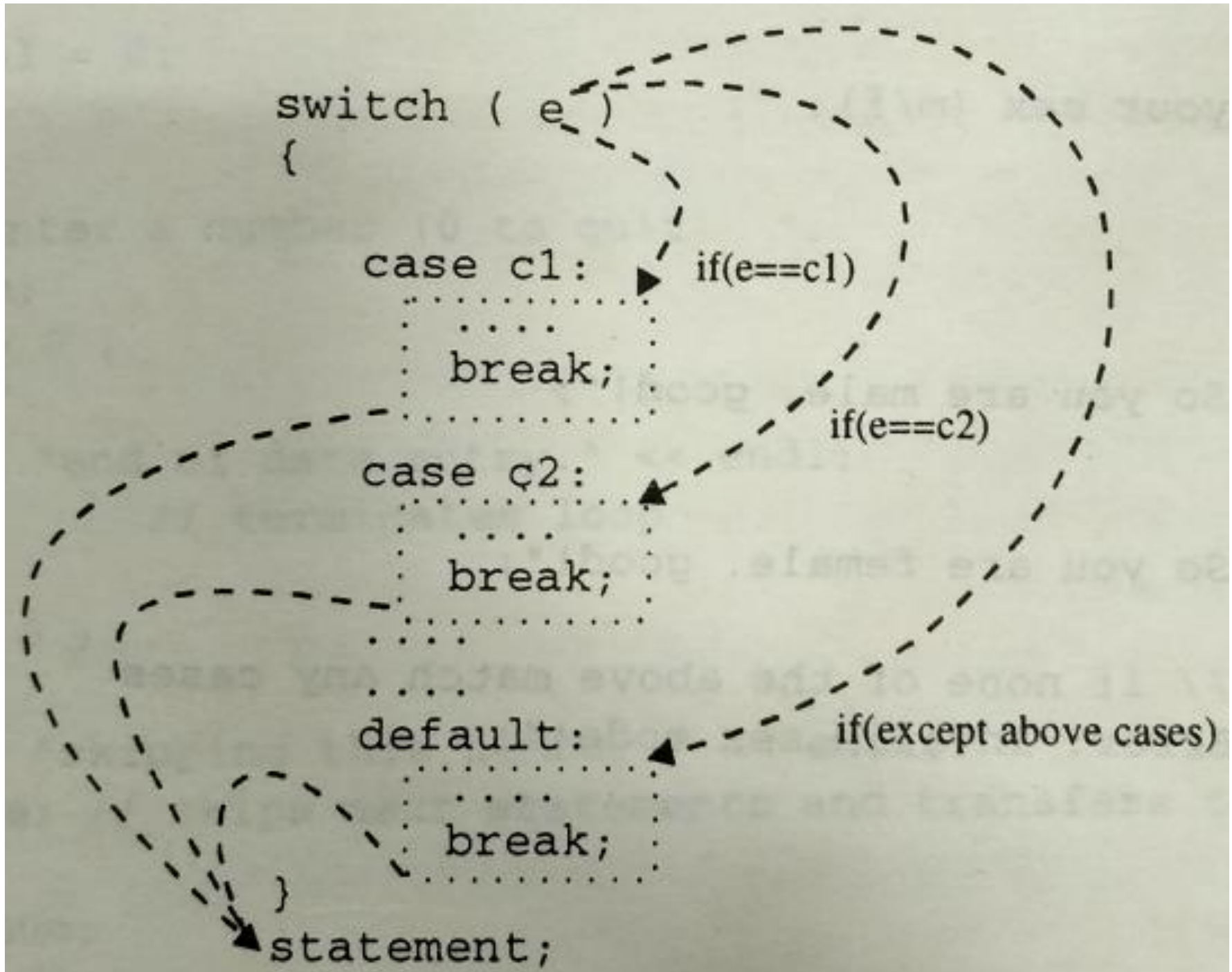
General form:

```
switch (expression)
{
    case value-1 : block-1;
        break;
    case value-2 : block-2;
        break;
    .....
    .....
    case value-n : block-n;
        break;
    default :
        default-block
}
```

statement-x;

The switch case statement

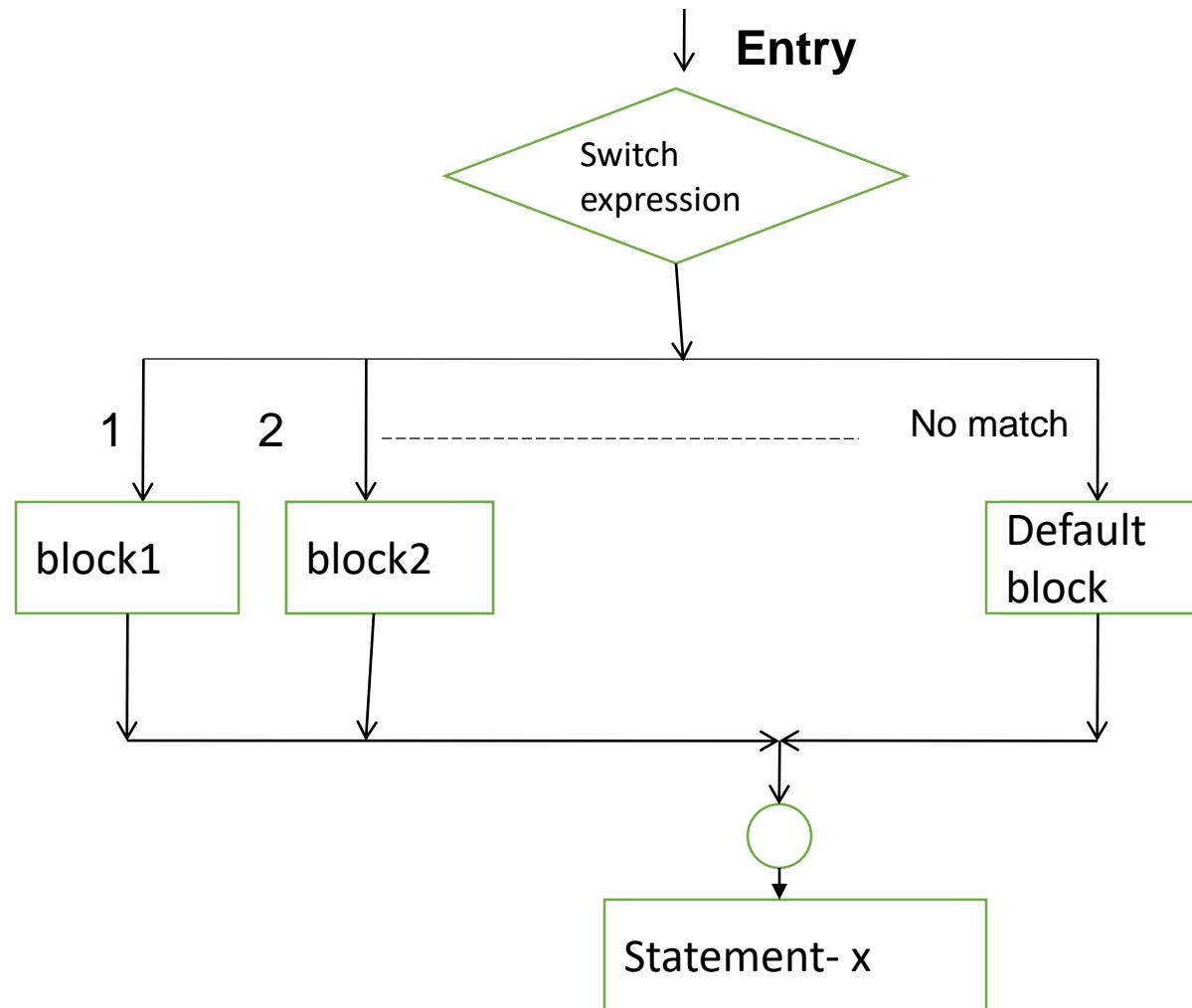
- When the **switch** is executed, value of the expression is successfully compared against the values value-1, value-2.
- If a case is found whose value matches with the value of the expression, then the block of statements that follows the case are executed
- The **break** statement at the end of each block signals the end of a particular case and causes an exit from the switch statement, transferring the control to statement-x following the switch



Rules for switch

- ❖ Case labels must be constants or constant expressions
- ❖ Case labels must be unique. No two labels should have same value
- ❖ **break** statement transfers the control out of the switch statement
- ❖ **break** is optional. Two or more case labels may belong to the same statements
- ❖ **default** label is optional. If present, it will be executed when the expression does not find a matching case label
- ❖ There can be at most one **default** label
- ❖ **default** may be placed anywhere but usually placed at the end
- ❖ Nesting of switch is possible

Flow chart for switch statement

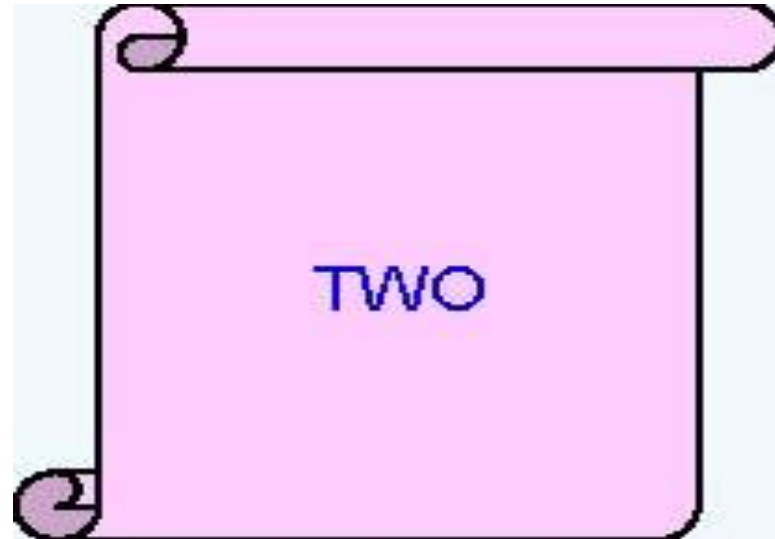


Calculation of grade using switch case

```
index=mark/10;
switch (index)
{
    case 10:
    case 9:
    case 8: grade='A';
            break;
    case 7:
    case 6:
            grade='B';
            break;
    case 5:
            grade='C';
            break;
    case 4:
            grade='D';
            break;
    default: grade='F';
            break;
} cout<<grade;
```

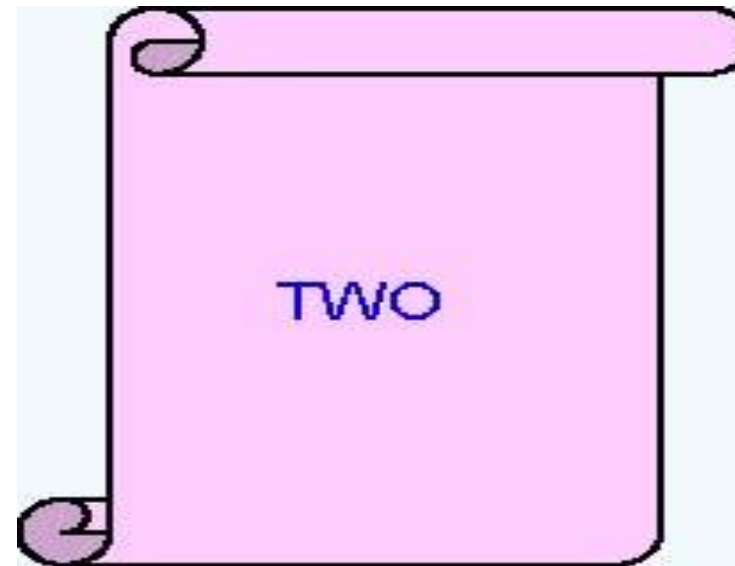
Predict the output -1

```
int iNum = 2;  
switch(iNum){  
    case 1:  
        cout<<" ONE";  
        break;  
    case 2:  
        cout<<" TWO";  
        break;  
    case 3:  
        cout<<" THREE";  
        break;  
    default:  
        cout<<"INVALID");  
}
```



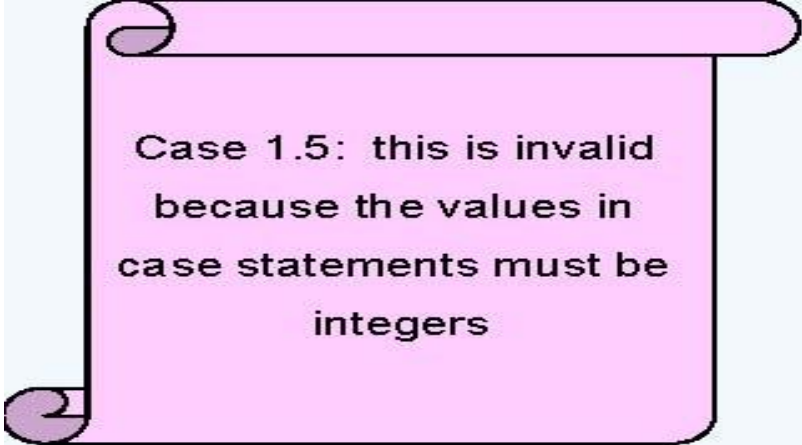
Predict the output -2

```
iNum = 2;  
switch(iNum) {  
default:  
    cout<<" INVALID");  
case 1:  
    cout<<" ONE");  
case 2:  
    cout<<" TWO");  
    break;  
case 3:  
    cout<<" THREE");  
}
```



Predict the output -3

```
int iNum = 2;  
switch(iNum) {  
case 1.5:  
    cout<<" ONE AND HALF";  
    break;  
case 2:  
    cout<<"TWO";  
  
case 'A' :  
    cout<<" A character";  
}
```



Case 1.5: this is invalid
because the values in
case statements must be
integers



Predict the output -4

```
const unsigned int iCountOfItems = 5;
```

```
switch ( iCountOfItems ) {
```

```
case iCountOfItems >=10 :
```

```
    cout<<" Enough Stock";
```

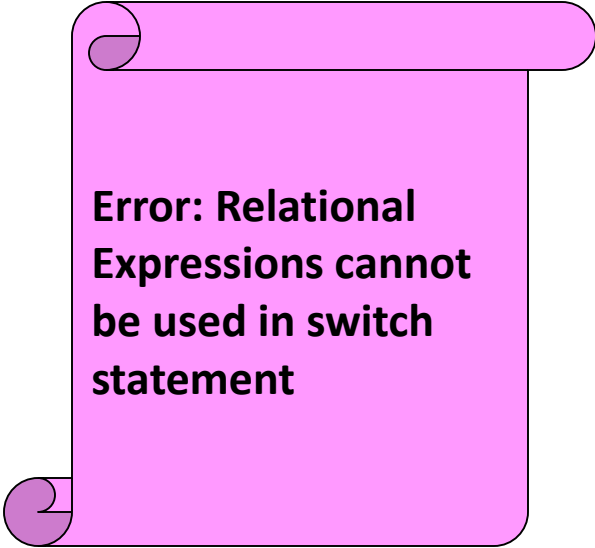
```
    break;
```

```
default :
```

```
    cout<<" Not enough stock";
```

```
    break;
```

```
}
```

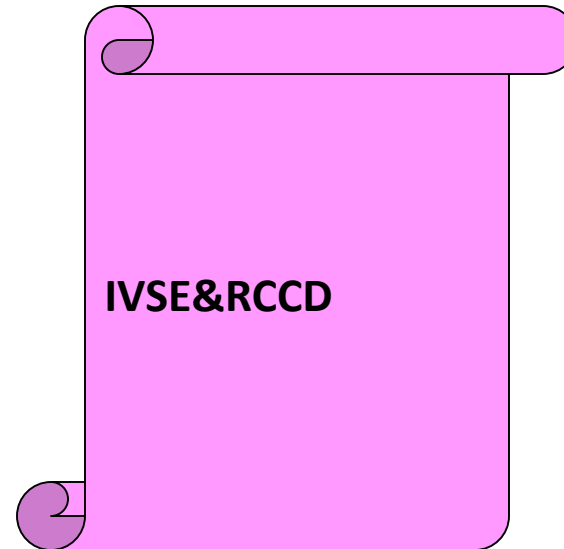


**Error: Relational
Expressions cannot
be used in switch
statement**

Predict the output -5

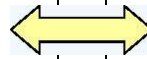
```
switch (iDepartmentCode){  
    case 110 : cout<<"HRD";  
    case 115 : cout<<"IVS";  
    case 125 : cout<<" E&R";  
    case 135 : cout<<" CCD";  
}
```

- Assume iDepartmentCode is 115 and find the output



An example for switch case

```
char ch='a';
switch(ch)
{
    case 'a' : cout<<" Vowel";
                break;
    case 'e' : cout<<" Vowel";
                break;
    case 'i' : cout<<" Vowel";
                break;
    case 'o' : cout<<" Vowel";
                break;
    case 'u' : cout<<" Vowel";
                break;
    default: cout<<" Not a Vowel";
}
}
```



```
char ch='a';
switch(ch)
{
    case 'a' :
    case 'e' :
    case 'i' :
    case 'o' :
    case 'u' :
        cout<<" Vowel";
        break;
    default :
        cout<<" Not a vowel";
}
}
```


Loops & Decisions

- Flow of control
- ***Control statements – statements that cause jump***

Relational Operators

<i>Operator</i>	<i>Meaning</i>
>	Greater than (greater than)
<	Less than
==	Equal to
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

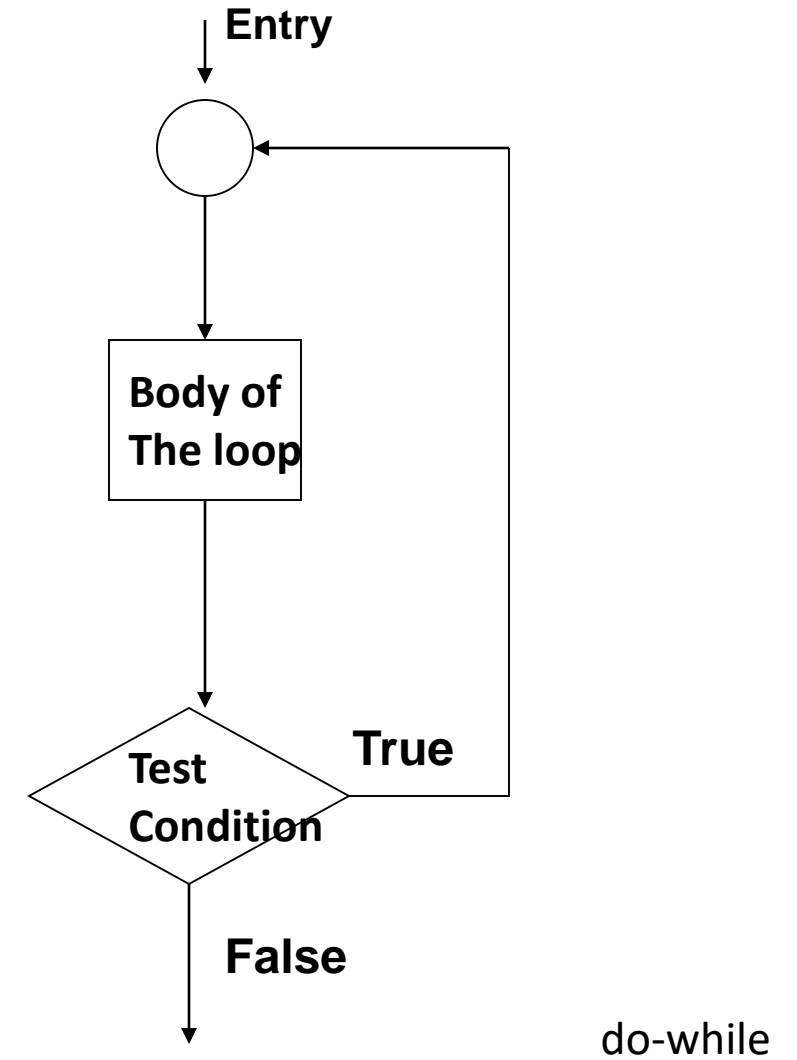
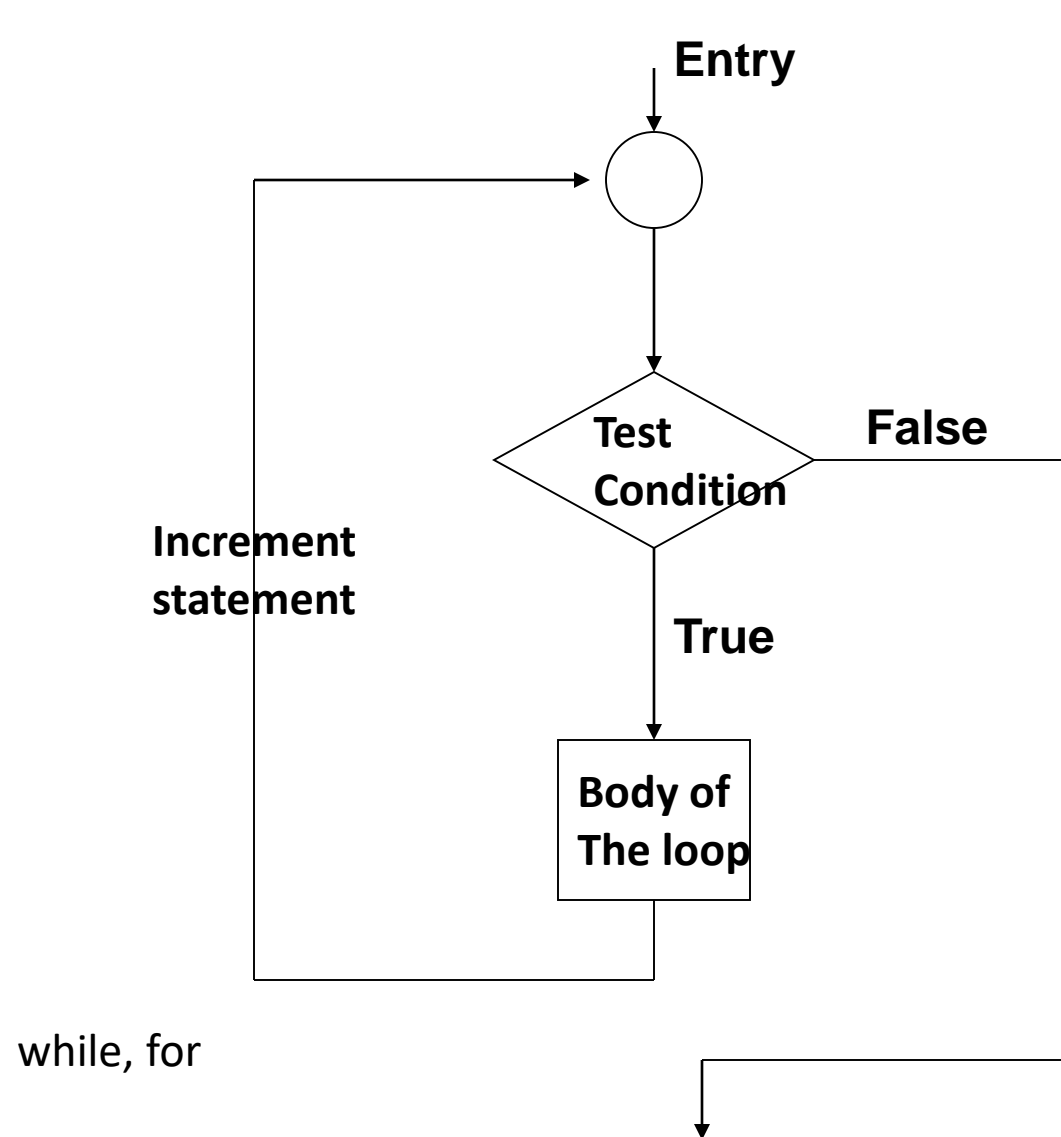
```
jane = 44;  
harry = 12;  
(jane == harry)  
(harry <= 12)  
(jane > harry)  
(jane >= 44)  
(harry != 12)  
(7 < harry)  
(0)  
  
(44)
```

C++ generates a 1 to indicate true, it assumes that any value other than 0 (such as -7 or 44) is true;
only 0 is false

Loops

- Cause a section of your program to be repeated a certain number of times.
- The repetition continues while a condition is true. When the condition becomes false, the loop ends and control passes to the statements following the loop.
- Depending on nature of control variable:
 - Counter-controlled loops (definite repetition loop, exact times)
 - Sentinel-controlled loops (indefinite repetition loop, -1 or 99 or y)

Entry and exit controlled loop



The **for** loop

- Fixed number of times
- The general form:

```
for (initialization; test condition; increment)
{
    statements
}
```

Example -1

```
3  int main()  
4  {  
5      for( int i = 0; i < 10; i++ )  
6      {  
7          cout<<i<<"\t";  
8          i *= 2;  
9      }  
10     return 0;  
11 }
```

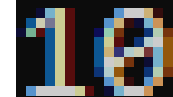
OUTPUT:

0 1 3 7

Example -2

```
3 int main()  
4 {  
5     int i;  
6     for( i = 0; i < 10; i++ );  
7         cout<<i<<"\n";  
8     return 0;  
9 }
```

OUTPUT:



Example -3

```
3  int main()  
4  {  
5      int i = 100;  
6      for( int i = 0; i < 5; i++ )  
7          cout<<i<<"\n";  
8      cout<<i;  
9      return 0;  
10 }
```

OUTPUT:

```
0  
1  
2  
3  
4  
100  
...
```

The **while** loop

General format:

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

Predict the output

```
unsigned int iCount = 1;  
while (iCount<10);  
{  
    cout<<iCount;  
}
```

Because of this;

NO OUTPUT!!!

Results in an infinite loop.. WHY???

Problems to be tried

- Find the sum of the digits of a number.
- To count how many 1's are present in the input binary number.

Example -4

```
3  int main()  
4  {  
5      int n, count=0, bit;  
6      cout<<"Enter a binary number:";  
7      cin>>n;  
8      while( n > 0 )  
9      {  
10         bit = n % 10;  
11         if( bit == 1 )  
12             count++;  
13         n = n / 10;  
14     }  
15     cout<<count;  
16     return 0;  
17 }
```

OUTPUT:

```
Enter a binary number: 11001  
3  
-----
```

The **do** loop

- Guarantee that the loop body is **executed at least once**, no matter what the initial state of the test expression

General form:

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

Nesting of loops

```
for ( i = 0; i < 5; i ++ )  
{  
    statement-1; // Executed 5 times  
    for ( j = 1 ; j < 3 ; j++ )  
    {  
        statement-2; // Executed 10 times  
    }  
}
```

Example -5

```
3 int main()  
4 {  
5     int i , j;  
6     for ( i = 0; i < 3; i ++ )  
7     {  
8         cout<<"\n\n i = "<<i<<"\n";  
9         for ( j = 10 ; j < 12 ; j++ )  
10        {  
11            cout<<"\t j = "<<j<<"\t";  
12        }  
13    }  
14 }
```

OUTPUT:

```
i = 0  
    j = 10    j = 11  
  
i = 1  
    j = 10    j = 11  
  
i = 2  
    j = 10    j = 11
```

Jumping out of for loop

- An early exit from a loop can be accomplished by using the **break** 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. i.e., the **break** will exit only a single loop.

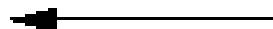
```
while ( loop-continuation-condition )
```

```
{  
    statement1  
    statement2  
    ....  
    break;  
    ....  
}
```

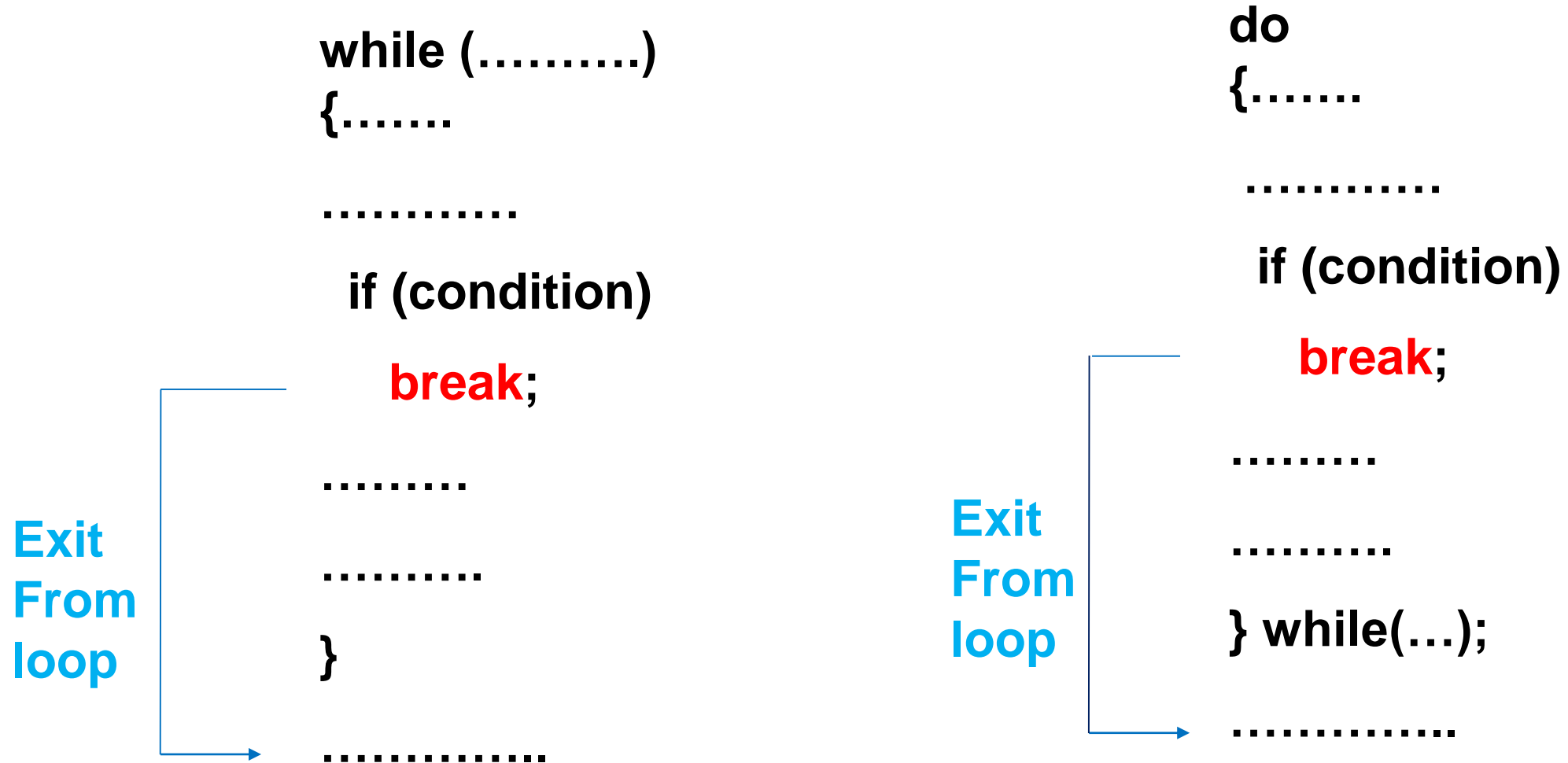
Execution proceeds to here

statement

statement following the while loop




Exiting a loop with break statement



Exiting a loop with break statements


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

Exit
From
loop



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

Exit
From
inner
loop



Example -6

```
3 int main()  
4 {  
5     int a = 10;  
6     while( a < 20 )  
7     {  
8         cout<<"a = "<<a<<"\n";  
9         a++;  
10  
11         if( a > 15)  
12             break;  
13     }  
14     return 0;  
15 }
```

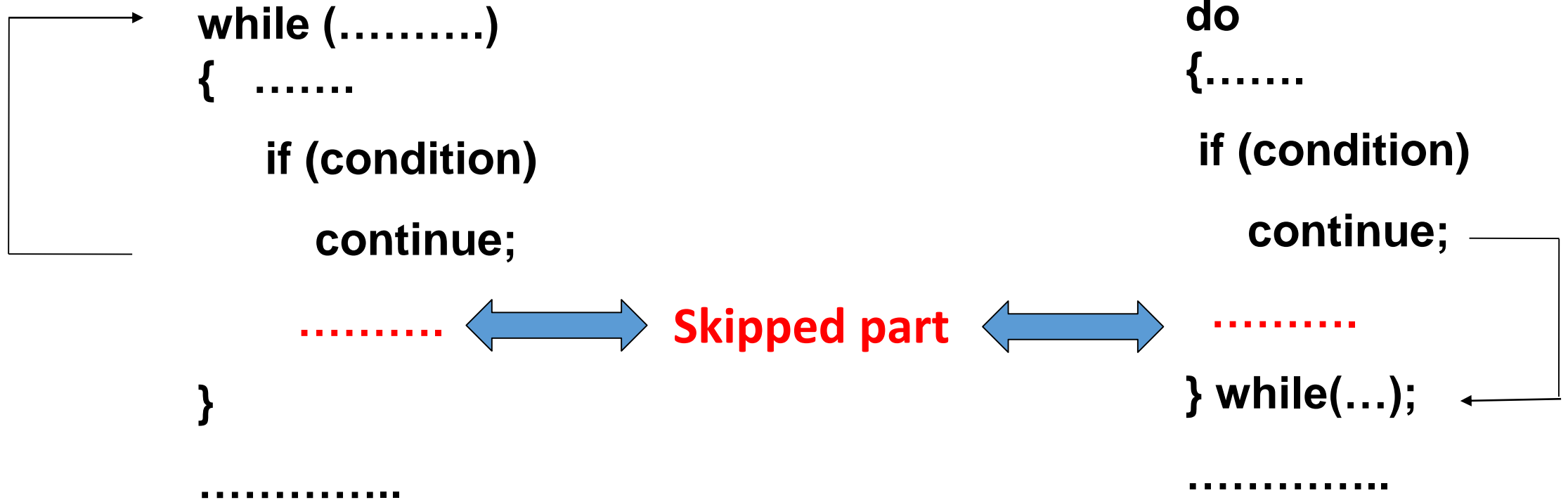
OUTPUT:

```
a = 10  
a = 11  
a = 12  
a = 13  
a = 14  
a = 15
```

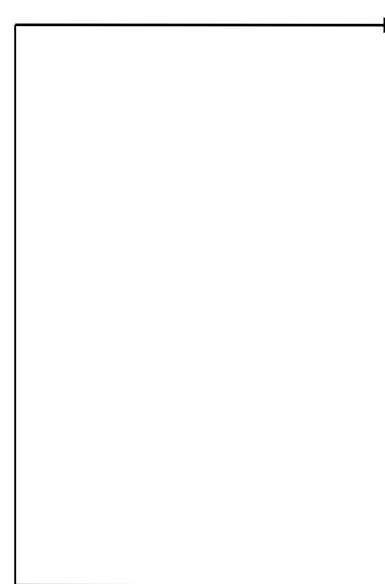

Skipping a part of the loop

- ✓ Skip a part of the body of the loop under certain conditions using continue statement.
- ✓ As the name implies, causes the loop to be continued with next iteration, after skipping rest of the body of the loop.
- ✓ In while and do loops, continue causes the control to go directly to the test-condition and then to continue the iteration process.
- ✓ In for loop, the increment section of the loop is executed before the test condition is evaluated

Skipping a part of the loop(contd.)



Skipping a part of the loop(contd.)



```
for ( initialization; test_condition; increment )
{.....
.....
    if (condition)
        continue;
    ..... // Skipped part
}
.....
```

The diagram illustrates a loop structure where a specific part of the loop body is skipped. A vertical line on the left side of the code block, with a horizontal arrow pointing to the start of the loop body, indicates the flow of execution. The code shows a 'for' loop with an initialization, a test condition, and an increment. Inside the loop, there is a block of code followed by an 'if' statement. If the condition is met, the 'continue' statement is executed, which skips the remaining code in the loop body (indicated by red text) and jumps back to the start of the loop body. The skipped part is represented by a red line of dots followed by the text '// Skipped part'.

Example -7

```
3  int main()
4  {
5      int a[] = { 11, -22, 33, -44, 55 };
6      for ( int i = 0; i < 5; i++ )
7      {
8          if ( a[i] < 0 )
9              continue;
10         cout<<a[i]<<"\n";
11     }
12     return 0;
13 }
```

OUTPUT:

11

33

55