Nested if

Nested if used when we need to test condition after another condition happened. It has the form:

```
if (condition1)
{
    If (condition2)
    {
    }
   Else
    {
    }
}
Else
{
}
```

Q) Write a program to check if the no. is even and accept division by 5 without remainder

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
cout << "Number:=";
cin >> a;

 if ((a % 2)==0)
  if ((a % 5)==0)
  cout << " even number accept division by 5";

 cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

Q) Write a program to check if the no. is odd and negative

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
```

```
cout << "Number:=";
cin >> a;

 if ((a % 2)!=0)
  if (a<0)
  cout << "--- negative odd number ---";

 cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

Q) Write a program to solve the following equations

When a>=5

b=a-1 when b>=20

b=a+1 when b<20

When a<5

b=a*5 when b>=20

b=a/5 when b<20

```
#include <iostream.h>
#include <conio.h>
int main()
{
float a,b;
clrscr();
cout << "a:=";
cin >> a;
cout << "b:=";
cin >> b;

 if (a >=5)
  if (b>=20)
   b=a-1;
  else
   b=a+1;
 else
  if (b>=20)
   b=a*5;
  else
   b=a/5.0;

  cout << " b:="<<b;
  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

Logical operator

1- Logical operator (&&, ||)

Turbo C++ offers these logical operators:

&& logical AND
|| logical OR

Syntax:
logical-AND-expr && inclusive-OR-expression
logical-OR-expr || logical-AND-expression

In these expressions, both operands must be of scalar type.

E1 && E2 E1 || E2

The usual arithmetical conversions are performed on E1 and E2.

Unlike the bitwise operators, && and || guarantee left-to-right evaluation.

E1 is evaluated first; if E1 is zero, E1 && E2 gives 0 (false), and E2 is not evaluated.

With E1 || E2, if E1 is nonzero, E1 || E2 gives 1 (true), and E2 is not evaluated.

2- Logical negation operator !

In the expression

! cast-expression

the cast-expression operand must be of scalar type.

The result is of type int and is the logical negation of the operand:

- 0 if the operand is nonzero
- 1 if the operand is 0

The expression !E is equivalent to (0 == E).

If with logical operator

Here is a more complicated Boolean expression:

```
if ((x==y) && (j>k))
    z=1;
else
    q=10;
```

This statement says, "If the value in variable x equals the value in variable y, and if the value in variable j is greater than the value in variable k, then set the variable z to 1, otherwise set the variable q to 10." You will use if statements like this throughout your C++ programs to

make decisions. In general, most of the decisions you make will be simple ones like the first example; but on occasion, things get more complicated.

Notice that C++ uses **= =** to **test for equality**, while it uses **=** to **assign a value** to a variable. The **&&** in C++ represents a Boolean AND operation.

Boolean expressions evaluate to integers in C++, and integers can be used inside of Boolean expressions. The integer value 0 in C++ is False, while any other integer value is True. The following is legal in C++:

```
#include <iostream.h>

int main()
{
    int a;

    cout << "Enter a number:";
    cin >> a;
    if (a)
    {
        cout << "The value is True";
    }
    return 0;
}
```

If **a** is anything other than 0, the printf statement gets executed.

In C++, a statement like **if (a=b)** means, "Assign **b** to **a**, and then test **a** for its Boolean value." So if **a** becomes 0, the if statement is False; otherwise, it is True. The value of **a** changes in the process. This is not the intended behavior if you meant to type **= =** (although this feature is useful when used correctly), so be careful with your **=** and **= =** usage.

Q) Write a program to check if the no. is even and accept division by 5 without remainder

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
cout << "Number:=";
cin >> a;

 if ((!(a % 2)) && (!(a % 5)))
  cout << " even number accept division by 5";

 cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

Q) Write a program to check if the no. is odd and negative

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
cout << "Number:=";
cin >> a;

 if ((a % 2) && (a<0))
  cout << "--- negative odd number ---";

 cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

Q) Write a program to solve the following equations

When a>=5

b=a-1 when b>=20

b=a+1 when b<20

When a<5

b=a*5 when b>=20

b=a/5 when b<20

```
#include <iostream.h>
#include <conio.h>
int main()
{
float a,b;
clrscr();
cout << "a:=";
cin >> a;
cout << "b:=";
cin >> b;

 if ((a >=5) && (b>=20))
   b=a-1;
 else if ((a >=5) && (b<20))
   b=a+1;
 else if ((a<5) && (b>=20))
   b=a*5;
 else if ((a<5) && (b<20))
   b=a/5.0;

  cout << " b:="<<b;
  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

Q) Write a program to check the mark of the student

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
cout << "Mark:=";
cin >> a;

 if (a >=90)
    cout << " Excellent";
 else if ((a >=80) && (a<90))
     cout << " Very good";
 else if ((a >=70) && (a<80))
     cout << " Good";
 else if ((a >=60) && (a<70))
     cout << " Medium";
 else if ((a >=50) && (a<60))
     cout << " Accepted";
 else
     cout << " Fail";

  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

The switch statement

The next branching statement is called a **switch statement**. A **switch statement** is used in place of many **if statements**.

If you want to test whether a variable takes one of a series of values, it's easier to use a switch statement than an if statement.

switch (<variable>) {

case <value 1>:

<one or more statements>

break;
case <value 2>:

<one or more statements>

break;
default:
<one or more statements>

break;
} //end switch

switch, case, and default (keywords)

Branches control

Syntax:
- switch (<expression>) <statement>
- case <constant expression> :
- default :

switch

Causes control to branch to one of a list of possible statements in the block defined by <statement>.

The branched-to statement is determined by evaluating <expression>, which must return an integral type.

case
The list of possible branch points within <statement> is determined by preceding sub statements with

case <constant expression> :

where <constant expression> must be an int and must be unique.

Once a value is computed for <expression>, the list of possible <constant expression> values determined from all case statements is searched for a match.

If a match is found, execution continues after the matching case statement and continues until a break statement is encountered or the end of <statement> is reached.

default

If a match is not found and the "default :" statement prefix is found within <statement>, execution continues at this point.

The **break** keyword means "jump out of the switch statement, and do not execute any more code." To show how this works, examine the following piece of code:

```
int value = 0;
switch(input){
  case 1:
    value=value+4;
  case 2:
    value=value+3;
  case 3:
    value=value+2;
  default:
    value=value+1;
}
```

If *input* is 1 then 4 will be added to *value*. Since there is no **break** statement, the program will go on to the next line of code which adds 3, then the line of code that adds 2, and then the line of code that adds 1. So *value* will be set to 10!

The code that was intended was probably:

```
int value = 0;
switch(input){
  case 1:
    value=value+4;
    break;
  case 2:
    value=value+3;
    break;
  case 3:
    value=value+2;
    break;
  default:
    value=value+1;
}
```

Q) Write a program to calculate the area or circumference of rectangle using menu

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
int l,w,r;
clrscr();
cout << "*********************************\n";
cout << "* 1: Area of rectangle          *\n";
cout << "* 2: Circumference of rectangle *\n";
cout << "*********************************\n\n";

cout << "Length:=";
cin >> l;
cout << "Width:=";
cin >> w;

cout << "\nChoice:=";
cin >> a;
switch (a)
{
 case 1:
   r=l*w;
   cout << "\n   Area of rectangle:="<<r;
 break;
 case 2:
   r=(l+w)*2;
   cout << "\n   Circumference of rectangle:="<<r;
 break;
}

  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

*** You can write the same previous program depending on character**

```
#include <iostream.h>
#include <conio.h>
#include <ctype.h>
int main()
{
char a;
int l,w,r;
clrscr();
cout << "*********************************\n";
cout << "* A: Area of rectangle          *\n";
cout << "* C: Circumference of rectangle *\n";
cout << "*********************************\n\n";

cout << "Length:=";
cin >> l;
cout << "Width:=";
cin >> w;

cout << "\nChoice:=";
cin >> a;
switch (toupper(a)) // to convert the character to upper case
{
 case 'A':
   r=l*w;
   cout << "\n   Area of rectangle:="<<r;
 break;
 case 'C':
   r=(l+w)*2;
   cout << "\n   Circumference of rectangle:="<<r;
 break;
 default:
  cout << "\n   Error Choice";
}

  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```