- Short-circuit evaluation

- C++ does not use *full evaluation* of logical expressions

  * Each logical expression is evaluated from left to right
  * Evaluation stops as soon as the value of the logical expressions can be concluded

$$(i == 1)\ \&\&\ (j > 2)$$

  * if $i=5$ is *false* for example, the whole expression will be *false* independently of whether $j > 2$ is *true* or *false*

$$(c <= d)\ ||\ (e == f)$$

  * if $c <= d$ is *true* for example, the whole expression will be *true* independently of whether $e == f$ is *true* or *false*

- Precedence operators

- Like in the case of arithmetic expressions, logical expressions are evaluated by applying some operators (relational or logical) before some others

- The following rules apply

```
!                 Highest-precedence
*   /   %
+   -
<   <=   >   >=
==   !=
&&
||
=                 Lowest-precedence
```

- Operators at the same line in the list have the same precedence

- Operations with the same precedence are performed from left to right (like in the case of arithmetic expressions (*a/b*c* means *(a/b)*c*)

- Parentheses are used to override the order of evaluation in an expression

$x \ \&\& \ y \ || \ z$ (logical AND is applied first), $x \ \&\& \ (y \ || \ z)$ (logical OR is applied first)

- Changing english statements into logical expressions

- Example: computing final grades

Grading Scheme	
Raw score	Grade
raw >= 90	A
80 <= raw < 90	B
70 <= raw < 80	C
60 <= raw < 70	D
raw < 60	F

Implementation

```
#include <iostream.h>

int main()
{
   float raw;

   cout << "Enter raw score" << endl;
   cin >> raw;

   if(raw >= 90)
      cout << "Your grade is  A\n";
   else if ( (raw < 90) && (raw >= 80) )
      cout << "Your grade is  B\n";
   else if ( (raw < 80) && (raw >= 70) )
      cout << "Your grade is  C\n";
   else if ( (raw < 70) && (raw >= 60) )
      cout << "Your grade is  D\n";
   else
      cout << "Your grade is  F\n";

   return(0);
}
```

A more efficient implementation

```
#include <iostream.h>

int main()
{
   float raw;

   cout << "Enter raw score" << endl;
   cin >> raw;

   if(raw >= 90)
      cout << "Your grade is  A\n";
   else if (raw >= 80)
      cout << "Your grade is  B\n";
   else if (raw >= 70)
      cout << "Your grade is  C\n";
   else if (raw >= 60)
      cout << "Your grade is  D\n";
   else
      cout << "Your grade is  F\n";

   return(0);
}
```

What about this implementation ?
(incorrect !!)

```
#include <iostream.h>

int main()
{
   float raw;

   cout << "Enter raw score" << endl;
   cin >> raw;

   if(raw >= 90)
      cout << "Your grade is  A\n";
   if (raw >= 80)
      cout << "Your grade is  B\n";
   if (raw >= 70)
      cout << "Your grade is  C\n";
   if (raw >= 60)
      cout << "Your grade is  D\n";
   if (raw < 60)
      cout << "Your grade is  F\n";

   return(0);
}
```

- Boolean variables

- We have seen that we can store the value of an arithmetic expression to a variable

sum=x+y; (*x,y,sum* may be *int*, *float*, *double* etc.)

- The same is true for logical (or Boolean) expressions: the value of a logical expression can be stored to a (Boolean) variable

result = x > y; (*x,y* may be *int*, *float*, *double* etc., but *result* can only be a Boolean variable)

- Boolean variables can have only two values: *true* or *false*

- C++ does not have a Boolean data type (other languages, like Pascal for example, do)

- The reason is that a Boolean data type can be implemented using the *int* data type !!

- *Rule:* the value 0 represents *false* and every nonzero value represents *true*

If *x=10* and *y=20* in the above example, then *result=0*

- Logical expressions can then be substituted in the *if* statements by Boolean variables

Without using Boolean variables

```
#include <iostream.h>

int main()
{  int x, y;

   cout << "Enter two numbers";
   cin >> x >> y;

   if(x > y)
     cout << "x is greater than y\n";
   else
      cout << "x is less than or equal to y\n";

   return(0);
}
```

Using Boolean variables

```
#include <iostream.h>

int main()
{  int x, y, result;

   cout << "Enter two numbers";
   cin >> x >> y;
   result = x > y;
   if(result)     // true if result != 0
     cout << "x is greater than y\n";
   else
     cout << "x is less than or equal to y\n";

   return(0);
}
```

By defining a Boolean data type

new data type !!

```
#include <iostream.h>

typedef int Boolean;

int main()
{  int x, y;
   Boolean result';

   cout << "Enter two numbers";
   cin >> x >> y;
   result = x > y;
   if(result)     // true if result != 0
     cout << "x is greater than y\n";
   else
      cout << "x is less than or equal to y\n";

   return(0);
}
```

- Defining new data types

typedef ExistingTypeName NewTypeName;

- We can define a Bollean datatype as follows

typedef int Boolean;

- By defining the data type *Boolean*, we make clear for somebody to understand that certain identifiers are Boolean identifiers

- It should be noted that the compiler substitutes (during compilation) the word *Boolean* with the word *int* everywhere in the executable (i.e., machine code)

- Combining Boolean variables with logical operators

```
cond1= x > y;
cond2= x > z;
if (cond1 && cond2)
  x=y+z;
```

- C++ allows programmers to define new data types using the statement *typedef*

- More examples using Boolean variables

score=(finalScore > 90) && (midtermScore > 90);, *dataInvalid=(inputVal == 0);*

Testing the state of the I/O stream

- Detecting the *fail state* when invalid data is provided

```
cin >> i >> j >> k;
cout << "i: " << i << " j: " << j << " k: " << k;
```

* if we enter the following data *1234.56 7 89* the input stream will enter the *fail state* part4.7.ps

* The *fail state* can be detected by checking the value of *cin*
*If everything went ok, *cin* has a nonzero value, otherwise, it has the value zero

```
cin >> i >> j >> k;
if(cin)
  cout << "i: " << i << " j: " << j << " k: " << k;
else
  cout << "The fail state has been detected !!0;
```

- Detecting the *fail state* when trying to open an input file which doesn't exist

```
#include <iostream.h>
#include <fstream.h>

int main()
{
    int i,j,k;
    float avg;
    ifstream fin;

    fin.open("myfil.dat");
    fin >> i >> j >> k;
    avg=(i+j+k)/3.0;
    cout << "avg=" << avg;

    return 0;
}
```

```
#include <iostream.h>
#include <fstream.h>

int main()
{
    int i,j,k;
    float avg;
    ifstream fin;

    fin.open("myfil.dat");
    if(fin)
    {
        fin >> i >> j >> k;
        avg=(i+j+k)/3.0;
        cout << "avg=" << avg;
    }
    else
    {
        cout << "Can't open the input file !!\n";
        return 1;
    }
    return 0;
}
```

you can check fin again here to make sure that the data is read correctly !!!

terminates the execution of the program and returns a nonzero value to the operating system meaning that something went wrong !!