If statements are used to switch the flow of control between alternate paths.

**The If-Then Statement**

The formal syntax for the if-then statement is:

```
if (Boolean Expression) Statement_T
```

Example:
```
if (grade < 50)
   cout << "F!"
```

Interpretation: if the value of grade is less than 50 "an F!" is put into the output stream

Remember that Statement_T can either be a single statement or a statement block.

Let’s use the if-then statement to determine the letter grade that should be assigned to a particular numeric mark. (Note, when the video was created, Memorial University reported grades to the nearest five marks. We’ve left the example to be as close to the video as possible.)

```
# include <iostream>
using namespace std;

int main()

   cout<< "Enter your mark: ";
   cin >> mark;
   cout << "\nThis is a"
   if (mark < 48) {
       cout << "n F."
   }
   // if (48 <= mark < 52)   ***** NOT LEGAL
   if (mark >= 48 && mark < 52) {
       cout << " D."
   }
   if (mark >= 52 && mark < 63) {
       cout << " C."
   }
   if (mark >= 63 && mark < 78) {
       cout << " B."
   }
   // if (78 <= mark < 90)   ***** NOT LEGAL
   if (mark >= 78 && mark < 90) {
       cout << " A."
   }
```

if (mark >= 78) {
    cout << "n A.\n";
} return 0;

Code Notes

1. We don't bother to initialize mark because we know that data will be entered into it before it will be used.

2. The line
   
   cout << "\nThis is a" ;
   
   precedes all the if statements because we want to output it no matter what the letter grade is

3. When checking whether a mark is between boundaries, the perfectly well-formed mathematical condition 48 <= mark < 52 is not legal as a C++ expression and must be reformulated as two separate boolean conditions joined by an and

   if (mark >= 48 && mark < 52)

4. Note that we have inserted the letter n before the A and the F. This is to make the output read correctly in English, i.e.
   
   This is a B but
   This is an F

Programs are for people. Get the details right and make them people friendly

You might consider point 4 fussy, but remember, though you write the code just once if gets used many times. This example exemplifies another important principle:

Here’s how our program looks diagrammatically.

Competent C++ programmers would complain about the program.

The larger decision diamonds were made bigger to hold the bigger expressions

They demonstrate graphically that each of these require two tests to be made

Note that a lot of the tests are redundant

if the result of mark<48 is false then we know it must be >=48. Why retest it?
The if-then-else Statement

With the if-then-else statement we add a second clause for the false case called the else clause.

if-then-else statement:

Form:
```
if (Boolean Expression) Statementₜ
else Statementₕ
```

Example:
```
if (number < 0)
    cout << -number;
else
    cout << number;
```

Interpretation: if number is negative it outputs -number (a positive quantity) otherwise it outputs number, i.e., outputs the absolute value of number.

The Statement subscripts signify the statements to be executed when the boolean expression is true and false. So Statementₜ is the then clause and Statementₕ the else clause.

Here's the control flow:
Let's see how we can use the if-then-else form of the if statement to improve our first example.

In this demo we sort marks into letter grade bins a little more efficiently than in the first demo.

```cpp
#include <iostream>

using namespace std;

int main() {
    int mark;
    cout << "Enter your mark: ";
    cin >> mark;
    cout << "\nThis is a";
    if (mark < 48) {
        cout << "n F.\n";
    } else {
        if (mark < 52) {
            cout << " D.\n";
        } else {
            if (mark < 63) {
                cout << " C.\n";
            } else {
                if (mark < 78) {
                    cout << " B.\n";
                } else {
                    cout << " A.\n";
                }
            }
        }
    }
    return 0;
}
```

Here, by using the else clause, and by nesting each subsequent if statement in its preceding statement's else clause, we have eliminated the double testing.

The strategy is actually very simple. In essence what we're doing is

- if the mark is less than 48
  - then the grade is an F
- otherwise go on to further testing

But haven't we generated spaghetti code? Not really. The flow of control is highly constrained and so very orderly.

Here's what the control flow looks like.
All the rest of the if statements are wholly embedded within that else clause.

The power of control structures really stems from our ability to nest them, one inside the other, to build almost arbitrarily complex structures.

Of course, we don't want arbitrarily complex, just complex enough to do the job (and not one jot more).

One of the most basic design skills good programmers have is an ability to get their control structures right.
A Complicated Example

Remember our complex number printing function?

```cpp
#include <iostream>

void printComplex(double re, double im){    // function DEFINITION
    cout << '(' << re << ' ' << im << 'j');
}
```

It isn't very good. Try it with -ve nos. or with one part set to zero. Here's a better version (it uses i instead of j because that's what we used in Math when the video was done).
By our standards this isn't nice example. The specialised nature of the printing task should be embedded in a function as we did a few lectures back. We kept it because we do have a video, that will let you review the detailed programming issues at a later date.

Here's an updated version that takes our earlier well-structured example and simply replaces the function with the better one from the video:

```cpp
/******* if demonstration *******

In this complex if demo we output properly formatted complex nos.

******************************************************************************/
#include <iostream> // info from standard
using namespace std;

// cout is in the std namespace

int main(){
    int real;
    int imag;

cout << "Enter the real part: ";
    cin >> real;

cout << "& the imaginary part: ";
    cin >> imag;

cout << "The complex no. is (";
    if (real == 0) {
        if (imag == 0) {
            cout << "0";
        } else {
            cout << imag << 'j';
        }
    } else {
        cout << real;
        if (imag < 0) {
            cout << " - ";
            cout << -imag << 'j';
        } if (imag > 0) {
            cout << " + ";
            cout << imag << 'j';
        }
    }
    cout << "\n";
    return 0;
}
```

Self-study: Try to identify each then and else clauses. Roll the mouse over each if to see its corresponding then clause and each else to see the else clause

Here they are diagrammed in the flow graph for the function
Notice how the other \texttt{if} statements are wholly nested with the then and else clauses of the first \texttt{if} statement.

\textbf{Rounding}

On the section on Expressions we briefly discussed \texttt{rounding} but said it was only good for positive numbers.

When C++ converts a \texttt{double} to an \texttt{int} it does it by \textit{truncation}. Thus $3.7 \rightarrow 3$, $3.4 \rightarrow 3$, $-3.7 \rightarrow -3$, and $-3.3 \rightarrow -3$.

So while adding $.5$ to a \texttt{double} before converting works for positive nos we need to add $-.5$ to negative ones. Sounds like a good job for a function!

\begin{verbatim}
int round(double x) {
    if (x < 0)
        return (int)(x-.5);
    return (int)(x+.5);
}
\end{verbatim}

\textbf{Code Notes}

1. Our function has two separate \texttt{return} statements. Some programming shops forbid this because it can make larger functions hard to maintain.

2. Notice there is no \texttt{else} even though we only want to execute one of the two \texttt{return}s. This is because if $x < 0$, the then clause is executed and it is a \texttt{return}. Remember, \texttt{return} not only sends back a value it exits as well. So the only way the second \texttt{return} can be reached is if $x \geq 0$.

3. The type cast (\texttt{int}) in the two \texttt{return} statements is not strictly necessary. The \texttt{round} function is contracted to return an \texttt{int} so the compiler will automatically insert a conversion into the object code even without the type cast.

Professional programmers will often insert such type casts simply to flag maintenance programmers that they intended a conversion to take place.
Here is another way to write the same function.

```c
int round(double x) {
  return x < 0 ? (int)(x-.5) : (int)(x+.5);
}
```

**Style Issues**

**Placement of the {**

There are two schools of thought on where to place the { (open block) operator

```c
if (im > 0) {
  cout << " + ";
  cout << im;
}
```

```c
if (im > 0) {
  cout << " + ";
  cout << im;
}
```

The second one makes it easy to line up the } with its own {.

The first one is more compact, taking one less line vertically.

We accept either in this course but you must be consistent. Whichever one you decide on use it always.

Other aspects are consistent.

1. indent statements inside the block (most program editors try to automate this anyway—set the indent to 4 spaces)
2. Don't indent the ). It should line up with the start of its own if (or else if it terminates an else clause).

**To Block or Not to Block**

One of our bin examples above includes the following lines of code

```c
if (mark < 48 ) {
  cout << "n F.\n";
}
```

This could equally well have been written

```c
if (mark < 48 )
  cout << "n F.\n";
```

Some programming shops routinely insist upon the first. Others prefer the second as being more compact. We will accept either.

This line is also legal C++

```c
if (mark < 48 ) cout << "n F.\n";
```

We will not accept it. Always put (or start) then and else clauses on their own line.
The else if Construct

Remember this fragment of code taken from our second example?

```cpp
int main()
    
    int mark;
    cout << "Enter your mark: ";
    cin >> mark;
    cout << "\nThis is a";
    if (mark < 48) {
        cout << " F.\n";
    } else {
        if (mark < 52) {
            cout << " D.\n";
        } else {
            if (mark < 63) {
                cout << " C.\n";
            } else {
                if (mark < 78) {
                    cout << " B.\n";
                } else {
                    cout << " A.\n";
                }
            }
        }
    }
    return 0;
```

All if statements subsequent to the first one are wholly embedded in the preceding else clause. In fact they are the else clause.

Some languages actually have a special elseif keyword.

There's no need of that, but many experienced programmers would simulate that by replacing else { if statement } by else if statement as below (we've actually removed all the brackets because the then clauses were also just single statements).

```cpp
int main()
    
    int mark;
    cout << "Enter your mark: ";
    cin >> mark;
    cout << "\nThis is a";
    if (mark < 48) {
        cout << " F.\n";
    } else if (mark < 52) {
        cout << " D.\n";
    } else if (mark < 63) {
        cout << " C.\n";
    } else if (mark < 78) {
        cout << " B.\n";
    } else {
        cout << " A.\n";
    }
    return 0;
```