Flow Control

Control Flow Basics

- If statement allows for alternative paths of execution

```
char option;
cout << "Enter h to say hello, m to multiply by 2: ";
cin >> option;
if ( option == 'h' )
        cout << "Hello C++\n";
else if ( option == 'm' )
{
        int n;
        cout << "What is the number: ";
        cin >> n;
        cout << n << " times 2 is " << n * 2 << "\n";
}
else
        cout << "That wasn't one of the options!\n";
```

As '=' is used for assignment, e.g. x = 4;
'==' is used for comparison.

else can be used to make a further comparison if the 'if' condition failed

if the 'if' and any 'else if' conditions were false, the statement after else is used

Conditional Expressions

- Used in if statements

```
if ( x == 6 )
  ...
```
Use == not = for 'is equal to'.

```
if ( y != 4 )
  ...
```
!= means 'is not equal to'.

```
if ( x >= 4 && y < 2 )
  ...
```
&& means 'and'.

```
if ( x != 9 || y <= 7 )
  ...
```
|| means 'or'.

Assignment vs Comparison Gotcha

- Beware using assignment instead of comparison
  - = instead of ==

```
if ( option = 'h' )
   cout << "Hello C++\n";
```

The expression in the () actually puts the character 'h' into the variable option, overwriting its original value. Whether the statement after the if gets executed depends on the value within the (parentheses), which in this case will always by 'h'. C++ treats any non-zero value as being 'true'. As 'h' is not zero, the statement after the 'if' is always executed.

- Modern compilers warn about this

Conditional Expression

- Shorthand alternative to if statement when value is required

```
int n, m;
cout << "First: ";
cin >> n;
cout << "Second: ";
cin >> m;
cout << "Biggest was: ";
cout << ( n > m ? n : m );
cout << "\n";
```

The conditional expression consists of

condition ? value_if_true : value_if_false

In this example the whole expression is in (parentheses) as it is the result of the expression that is passed to cout.

Note that the condition itself, n > m in this example, does not need to be in (parentheses), but these are often used for clarity.

The switch Statement

- Used when different actions required based on value of an expression

```
char reply;
cin >> reply;
switch( reply )
{
case 'f':
case 'F':
        cout << "Factorial is " << factorial( n ) << "\n";
        break;
case 'm':
case 'M':
        multiply( val );
        break;
default:
        cout << "Not a valid option\n";
        break;
}
```

Value to be tested.

If the character was f or F, go here. Several values can be processed in the same way by putting case statements together.

Execution continues until a break statement.

If the value did not match any of the case statements, it will go to the default label if there is one.

The While Loop

- Repeat execution until condition evaluates to false

The block which follows is repeated while the condition is true.
while(num != 0) means 'while num is not equal to 0'.

```
double num = 1;
while ( num != 0 )
{
        cout << "What number would you like factorialized: ";
        double result = 1;
        cin >> num;
        for ( int i = 2; i <= num; i++ )
                result *= i;
        cout << "factorial is " << result << "\n";
}
```

result is declared within the loop so that it gets reinitialized to 1 each time.

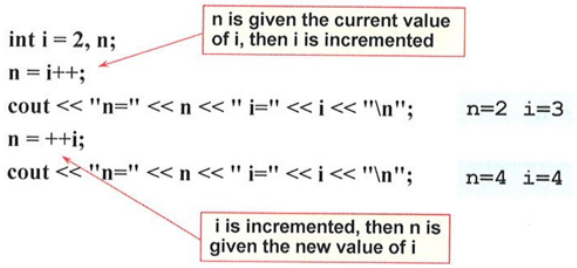
Exit Condition Loop – do-while

- Body of the loop guaranteed to be executed at least once

```
double num;
do
{
        cout << "What number would you like factorialized: ";
        double result = 1;
        cin >> num;
        for ( int i = 2; i <= num; i++ )
                result *= i;
        cout << "factorial is " << result << "\n";
} while ( num != 0 );
```

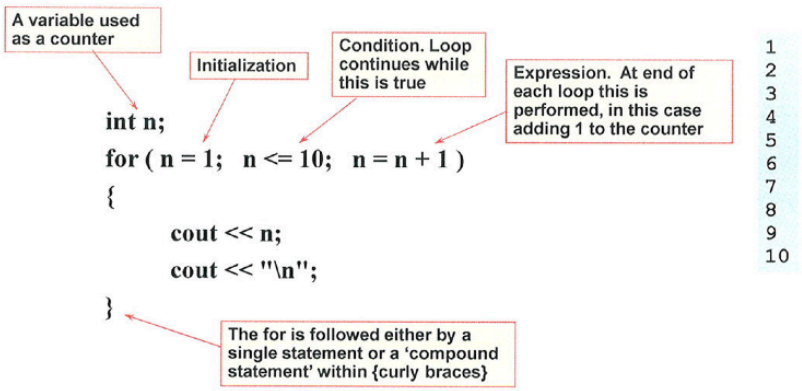
Increment and Decrement Operators

- May be prefix or postfix
  - i++ or ++i
  - i-- or --i
- Important to appreciate the difference
  - especially for C++



The for loop

- Used when number of iterations is predictable
  - Fixed number of iterations



The for loop

- Used when number of iterations is predictable
  - Fixed number of iterations

```
cout << "What number would you like factorialized: ";
double num, result = 1;
cin >> num;
for ( int i = 2; i <= num; i++ )
        result *= i;
cout << "factorial is " << result;
```

i is declared in the initializer.

i++ is used as a shorthand for i = i + 1.

result *= i is a shorthand for result = result * i.

The loop body is a single statement, so the { } are not required.