Flow Control

There are three broad kinds of flow control instructions:

- selection (if)
- multi-way selection (switch)
- loops

If-Then-Else Statements

```
if( condition ) {
        // Some statements
} else if( condition2 ) {
        // More statements
} else if( condition3 ) {
        // More statements
} else {
        // Other statements
}
```

Switch Statements

Switch statements can only act on ordinal values
Control flow falls through cases unless the break statement is used
The default case also exists

```
switch(expression) {
case value1:
        // Some Statements
        break;
case value2:
        // More Statements
        break;
case value3:
        // More Statements
```

```
        break;
default:
        // Some Other Statements
}
```

If you deliberately omit a break statement between cases I strongly recommend you make it obvious, eg:

```
switch(expression) {
case value1:
        printf( "Value1 case\n" );
        // FALLTHRU: fall through to following statements
case value2:
        printf( "Value2 case\n" );
        break;
default:
        printf( "Default case\n" );
}
```

`while` and `do..while()` loops

```
while( condition ) {
        // Some statements (executed zero or more times)
}

do {
        // Some statements (executed at least once)
} while( condition );
```

for Loops

```
for( initialiser; condition; nextstep ) {
        // Some statements
}
```

The initial statement, the condition (which defaults to true) and the increment expression may all be omitted.

It is important to use idiomatic forms of loops so that other C programmers may easily understand your code

The conventional repeat N times loop is:

```
for( int i=0; i<N; i++) {
        // Some Statements
}
```

Execution of code inside a loop (definitely a for loop, and perhaps do or while) can be manipulated by the following statements:

- `break`
  - Break out of the current loop.
  - Any statements in the loop following the `break` are ignored and the loop condition automatically evaluates to false, ending the loop
- `continue`
  - Jump to the end of the current loop (effectively ignoring everything below the continue statement)
  - The `nextstep` is executed immediately after the `continue` statement
  - The loop condition is reevaluated - and the loop either continues executing, or terminates, depending on the result

Note that the combination of an infinite for loop and one or more conditional break statements means that you can simulate the most general loop