A loop is the only control-flow construct that lets you go back to an earlier point in the code. Loops are designed to allow us to iterate—to execute the same piece of code over and over again.

We'll consider three different kinds of loops:

**While Loops**

A while loop is created by a while statement. Its formal syntax is:

```
while statement :
```

**Form:**

```
while (Boolean Expression ) Statement_B
```

**Example:**

```
cin >> grade;
while (grade > 100) {
    cout << "Grade can't exceed 100! ");
    cin >> grade;
}
```

**Interpretation:** A grade is entered and so long as its value exceeds 100 an error prompt is given and the grade is re-entered.

As always Statement_B can either be a single statement or a statement block.

We use the B subscript because its often called the body of the loop.

The control flow for the example in the syntax definition looks like this.

The >100 in the decision block is short for grade > 100 of the example

That boolean expression represents a loop continuation condition

If the condition is true the body of the loop is executed.

After the body is executed, we go back and test the loop continuation condition again

As soon as the condition is false, the flow of control moves to the statement immediately after the loop.

Here we use the while loop to generate a conversion table from degrees centigrade to degrees fahrenheit.

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

/***** a while loop *************/

Using a loop to generate a conversion table between degrees C and degrees F.

```
int main(){
    int tempC = 0;
    cout << "A temperature conversion table\n\n";
    cout << "centigrade\tfahrenheit\n";
    cout << "--------------------------------------\n";
    while (tempC <= 100) {
        cout << "   "  << tempC << " \t	\t" ;
        cout << (9*tempC )/5  + 32  << '\n' ;
        tempC += 5;
```

http://www.engr.mun.ca/~mpbl/teaching/2420/lectures/control_flow/loops.htm  2004.03.24

http://www.engr.mun.ca/~mpbl/teaching/2420/lectures/control_flow/loops.htm  2004.03.24
Loop Categories

There are a number of well-recognized loop categories, some of which we outline here.

Count-Controlled Loops

When you know how many times to iterate. Note that the first example (temperature conversion) is basically a variation on this theme.

Event-Controlled Loops

Iteration continues until some event occurs in the body of the loop. Here's an example:

In this case, the event is a decision by the user to stop entering numbers.

Sentinel-Controlled Loops

- The termination condition is a special value read from input.
- The special value is called a sentinel or trailer value.
- The sentinel must not be a valid input value.

Alternative (use side-effects and shortcut evaluation):

In this case, the event is a decision by the user to stop entering numbers.
Flag-Controlled Loops

flag—A Boolean variable used to control logical flow.

```cpp
bool positive = true; // flag, set to false when x < 0.
while (positive) {
    cin >> x; // ...
    if (x < 0) {
        positive = false;
    }
}
```

For Loops

The for statement is particularly well suited for **control-counted** loops.

The formal syntax for the for loop is

**Form:**

```
for statement :
```

```
for (init expression ;
    boolean expression ;
    update expression )
  Statement
```

**Example:**

```
for (int i = 0; i < 10; i++) {
    cout << i * i;
}
```

**Interpretation:** An int i is declared for the duration of the loop and its value initialised to 0. \(i^2\) is output in the body of the loop and then i is incremented. This continues until i is 7.

Again **Statement** is **called the body of the loop** and it can either be a **single statement** or a **statement block**.

Here's an example of a program that uses the for loop to create a table of factorials.

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

int main() {
    int factorial = 1;
    // Output a table heading
    cout << "Table of Factorials\n  i	  i!\n\n" ;
    for (int i = 0; i < 10; i++) {
        if (i != 0) factorial *= i;
        cout << i << "\t"  << factorial << '
' ;
    }
    cout << "That's all folks!\n" ;
    return 0;
}
```

Please note that in both the syntax definition block and the example the three internal expressions in the for loop have been written on separate lines.

This is not normal.

```cpp
for (int i = 0; i < 10; i++) {
    if (i != 0) factorial *= i;
    cout << i << "\t"  << factorial << '\n' ;
}
```

would be the normal way of starting the for loop

**NOTE:** The for statement's style is a little unusual it would normally be written out on one line

```cpp
for (int i = 0; i < 10; i++) {
    if (i != 0) factorial *= i;
    cout << i << "\t"  << factorial << '\n' ;
}
```

I've done it this way to focus on the three separate expressions in the statement.

---

http://www.engr.mun.ca/~mpbl/teaching/2420/lectures/control_flow/loops.htm  2004.03.24

http://www.engr.mun.ca/~mpbl/teaching/2420/lectures/control_flow/loops.htm  2004.03.24
Loop Design

1. The general case:
   - What should be done in the body?

2. The special cases:
   - Under what condition should the iteration stop?
   - How should the loop control condition be initialized?
   - How should the loop control condition be updated?
   - How should other variables be initialized?
   - How should other variables be updated?
   - What is the state when the loop exits?

Notice we have put the general case first. Although it may seem counter-intuitive (because when you read the code the while or the for precedes the body of the loop)

Design loops from the inside out (from the general to the specific).

Note the implication here is that

1. first design the loop, then
2. code the loop

### Pseudo Code

How can we design before we code?

Enter pseudo code, which we will introduce by example.

Let's consider a program to compute the mark for every student in a course. Here's an algorithm specified in pseudo code:

```
enter the number of students in the course
enter the midterm1, midterm2, assignments, labs and final max (or 0 if none) and percentage
set student to 1
while student <= students
    enter mark for each component
    compute course mark
    increment student
```

Now consider the problem of entering a mark for each component. There's a lot to that. As before, we need to refine this step.

```
enter mark for a component
    if max of component > 0
        prompt for mark
        enter mark
        while mark > max or < 0
            give error prompt
```

http://www. engr .mun.ca/~mpbl/teaching/2420/lectures/control_flow/loops.htm 2004.03.24
Pseudo code is just a refinement of the way we taught functional decomposition.

In pseudo code, we use the control structures of computer programming but we state what we want to do in plain English or using any convenient, understandable notation (e.g. mathematical).

The idea is to get the control structures right.

Let's take the pseudo code and use it to produce a program.