

# PIC 10A

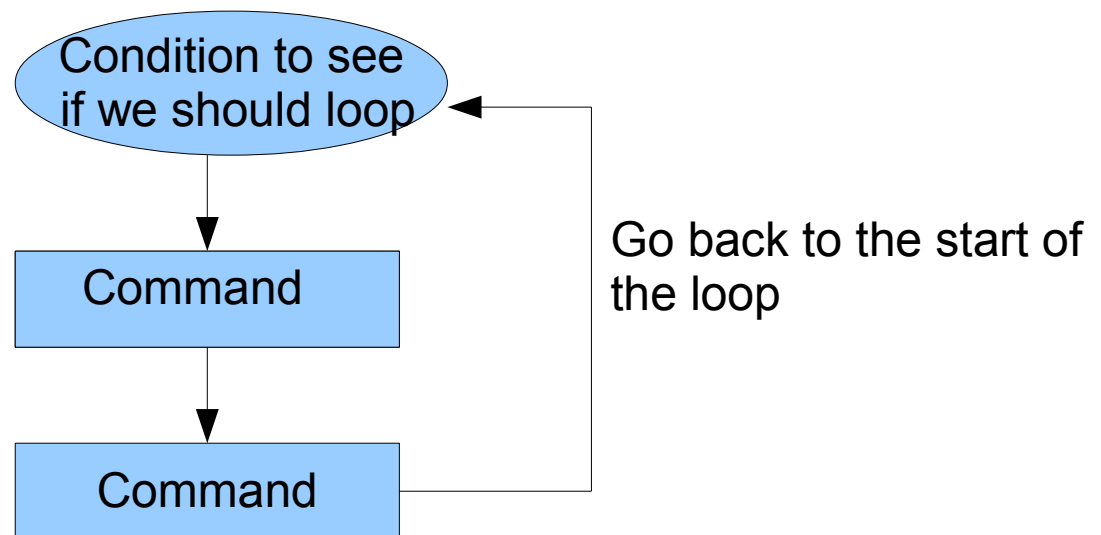
## Lecture 9: `while` loop

# Lets get loopy

Often we want to repeat a lines of code over and over again. We could rewrite the lines a few times if they are short. But what about when we want to repeat the code hundreds or millions of times?

We need an easy way to do this. The answer is a loop.

# Idea of a loop in code



# Infinite loops

We must be careful not to make infinite loops!

Whether or not the loop runs is determined by the condition set forth in the beginning of the loop.

This condition might be something like  $X < 5$ . So the loop will run while value of  $X$  is less than 5. In order for the loop to end the code inside the loop must be changing the value of  $X$ . So at some point  $X$  will be 5 or bigger and the loop terminates.

# The while loop

```
while (boolean expression)
{
    //CODE
}
```

The braces { } enclose the code to repeat.

Each repetition of the code inside { } is called iteration.

The while loop acts like a repeating if statement, repeating the code below it as long as the boolean statement is true.

By indenting the code inside the { } we make it clear what is being repeated.

# Example

```
int bottles_beer = 1000;

while (bottles_beer > 0)
{
    cout << bottles_beer << " bottles of beer on the  
wall..\n";
    bottles_beer--;
}
```

# Another example

What does this code do?

```
int x=1;
int sum=0;
while (x <=100)
{
    sum +=x;
    x++
}

cout << "Sum is: " << sum;
```

# { } or not?

Just as with an if statement we could omit the braces.

```
while(x > 0)  
    x--;
```

If you omit braces only the next line is considered to be under the while statement.

Usually this is not what you want!

Unlike if statement with while we almost always want braces.



# Careful!

What does the following code do?

```
int x=0;
while (x < 5);
{
    cout << "The value of x is " << x << "\n";
    x++;
}
```

# Common mistake

Do not put a ; after a while condition.

If the condition is true you have created an infinite loop!

```
int x=0;
while (x < 5); ← ; is a mistake here that creates an
                infinite loop!
{
    cout << "The value of x is " << x << "\n";
    x++;
}
```

The computer thinks of the semi-colon after the condition as an empty statement. The braces after the semicolon are now considered not to be part of the while loop. The only command under the while is the empty statement. So there is nothing changing value of x and the loop runs forever.

# Example

Every year Homer invests in the stock market he loses 20% of his investment. If he starts with an initial investment of \$20,000, how many years will it take until he has less than \$100 left?

```
int money = 20000;
int count = 0;

while (money > 100)
{
    money = money * .8;
    count++;
}
```

# Example

The following code is supposed to output positive even numbers up to 100.

```
int n=0;
while (n <= 100)
{
    n +=2;
    cout << n;
}
```

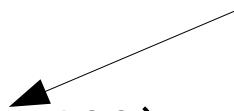
It does not quite work. What goes wrong?

# Stopping condition

The stopping condition in previous example was wrong.

```
int n=0;
while (n < 100)
{
    n +=2;
    cout << n;
}
```

We need to have <  
not <=



Pay especially close attention to what happens on the first and last iteration of your loop.

# bool variables in loops

```
string response;  
bool loop=true;  
  
while (loop)  
{  
  
.....  
  
cout << "Do you want to loop some more?";  
cin >> response;  
  
if (response=="Y" || response =="y")  
    loop = true;  
else loop = false;  
  
.....  
  
}
```

# Remark

Of course in the previous example we could have done:

```
cin >> response;  
while (response == "Y" || response == "y")  
{  
  
etc.  
  
}
```

We will soon see more realistic examples where it makes sense for the looping condition to be a boolean variable.

# `cin` and `if`

Recall that if `cin` receives input that doesn't match what it is expecting it goes into fail state.

`cin` actually returns `true` if it successfully read a value and `false` if input was invalid

```
cin >> number; // This line not only gets input for number but it also has a  
                // true or false value. In most cases this is not important.
```

But it does mean we can use `cin` in an `if` statement

Alternatively you could use the member function `cin.fail( )` described in Sec 4.4.



# Looping until the input is not valid

The loop terminates when user enters anything but an int.

```
int number = 0;
int sum = 0;
cout << "Gimme a number!\n";
while (cin >> number)
{
    sum+=number;
    cout << "Gimme a number!\n";
}

cout << "The sum is: " << sum;
```

# Nested loops

Example:

```
int x=1;
int y=1;

while ( x <= 10)
{
    while (y <= 10)
    {
        cout << x << " X " << y << " = " << x*y << "\n";
        y++;
    }
    y=1;
    x++;
}
```

How many times does the inner loop run?

# Big example

See examples page

# do - while

Syntax:

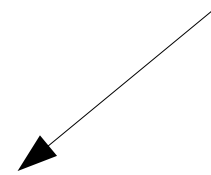
```
do
```

```
{
```

```
    /* code */
```

```
} while( boolean expression);
```

Semicolon!!



Main difference between while and do loop is that the do loop runs always at least once.

The condition is checked after the code is run.

# Example

```
string response;

cout << "Do you want to play?";
cin >> response;

while (response == "Y" || response == "y")
{
    /* code for a game */

    cout << "Do you want to play?";
    cin >> response;

}
```

# Example

```
string response;
```

```
do  
{
```

```
    /* code for a game */
```

```
    cout << "Do you want to play?";  
    cin >> response;
```

```
} while (response == "Y" || response == "y");
```

# Preview of a `for` loop

We have seen that in many of our examples with loops we had a counter that counted how many times we have looped.

In a `for` loop the counter is central part of its syntax.

```
for (int i=0; i < 100; ++i)
{

    /* code */

}
```

# for loop example

```
for (int i = 1; i < 100; ++i)
{
    cout << "Wow! This is the " << i << "th line of
    output\n";
}
```



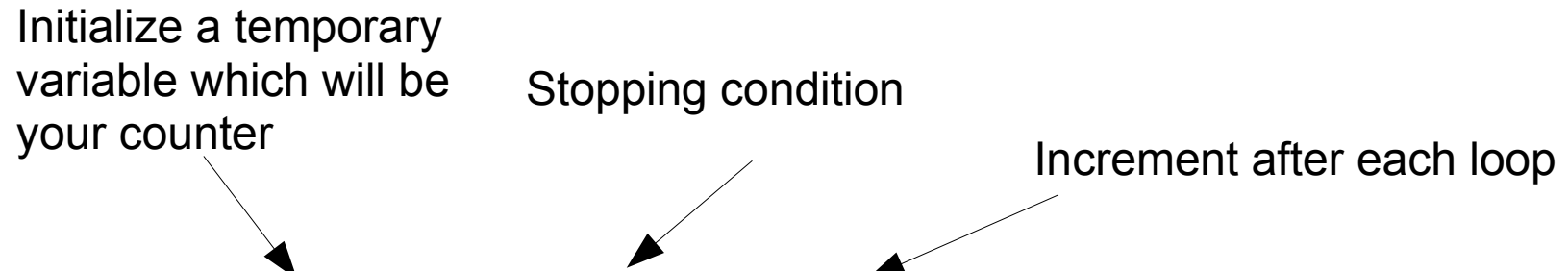
# Preview of a `for` loop

In the `for` loop we have a few possible options, but most common way of using the `for` loop is as in this example:

Initialize a temporary variable which will be your counter

Stopping condition

Increment after each loop



```
for (int i=0; i < 100; ++i)
{

/* code */

}
```