# C++ Programming for Engineers

Petar Jervic
Francisco Luro
Andrew Moss (course-resp)

May 2, 2012

# Loops defined

A loop is a control structure that causes **ONE STATEMENT** or a **GROUP OF STATEMENTS** to **repeat**.

```
int counter = 0;
cout « counter « ". Hello" « endl;
```

- General definition:

```
while (condition) {
  statement(s);
}
```

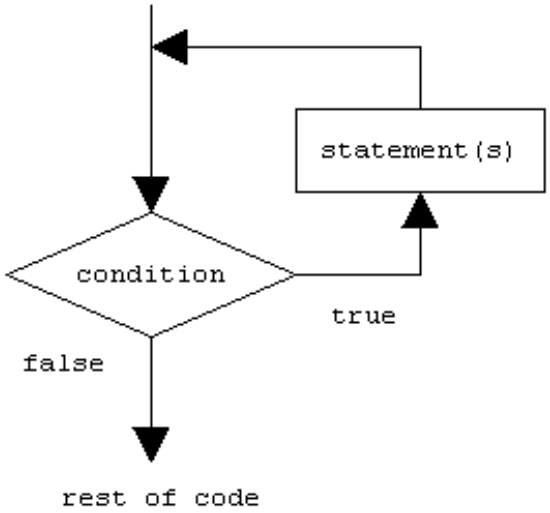
# How Loops Work

1. **Initialize the state-holding variable to initial value**
2. **The condition expression is tested** (can be evaluated as true or false)
3. If it is true EACH statement in the body of the loop is executed.
4. **Change the condition state** (Infinite loop???)
5. The condition is tested again
6. Repeat until condition is false

```
int counter = 0;
while (counter < 10) {
  cout « counter « ". Hello" « endl;
  counter++;
}
```

# How Loops Work

Thought process: I run my loop while condition is true




```
int counter = 0;
while (counter < 10) {
  cout « counter « ". Hello" « endl;
  counter++;
}
```

# One vs. Group of Statements

A run of a loop is called an **iteration**

- Group of statements:

```
int counter = 0;
while (counter < 10) {
  cout « counter « ". Hello" « endl;
  counter++; }
```

- One statement:

```
int counter = 0;
while (counter < 10)
  cout « counter++ « ". Hello" « endl;
```

**Output**:

0. Hello
1. Hello
2. Hello ...

# Infinite loop (1)

**Change the condition state** (Infinite loop???)

A loop must contain within itself a way to terminate, a piece of code that will change the condition to false.

If a loop does not have a way to stop it is called and infinite loop.

- Nothing to change the state:

```
int counter = 0;
while (counter < 10) {
  cout « counter « ". Hello" « endl;
}
```

- Semicolon after a while

```
int counter = 0;
while (counter < 10); {
  cout « counter « ". Hello" « endl;
  counter++;
}
```

# Infinite loop (2)

Most common mistake which results in an infinite loop is to use the assignment = operator when you intend to use the == operator.

```
int counter = 10;
while (counter = 10) {
  cout « counter « ". Hello" « endl;
  counter++;
}
```

Counter is set to 10 each iteration instead of testing it. This is an infinite loop.

**If your program hangs (is not responsive) for too long, it is probably stuck in an infinite loop.**

# Exploring a problem

User has three tries to guess (input) a random number between [1 - 10].

1. Generate a random number to guess
2. User input ((repeat???)
3. Loop condition modeling
4. Condition 1: Three tries
5. Condition 2: User guessed a number

# 1. Generate a random number to guess

Variable to hold the number

```
int rnd_number;
```

Rand() is pseudorandom, initialize the seed

```
//initialize random seed:
srand ( time(NULL) );
```

Generate the number from [1 - 10]

```
//generate random number:
rnd_number = rand() % 10 + 1;
```

# Condition 1: Three tries

We have to count the number of user input tries and stop the guessing (loop) after three tries.

```
int num;
int counter = 0;
while (counter < 3) {
  cout « "Guess the number: ";
  cin » num;
  counter++;
}
```

# Condition 2: User guessed a number

We have to stop the guessing (loop) after the user guessed the correct number.

```
int rnd_number;
// Genrate a random number
int num = 0;
while (num != rnd_number) {
  cout « "Guess the number: ";
  cin » num;
}
cout « "The number was " « rnd_number « endl;
```

# Loop condition modeling (1)

We have to loop while following conditions are both true.

- counter < 3
- num != rnd_number

We have to form a more elaborated logical expression by combining two simple test conditions.

- *condition 1 [insert logical operator] condition 2*
- *condition 1 && condition 2*
- *condition 1 || condition 2*
- *condition 1 ^condition 2*

# Loop condition modeling (2)

Boolean expression evaluates to true or false

- Condition 1 (C1): counter < 3
  - Three tries
- Condition 2 (C2): num != rnd_number
  - User guessed a number

| C1 | C2 | c1&&c2 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| C1 | C2 | c1\|\|c2 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| C1 | C2 | c1^c2 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Loop condition modeling (3)

The loop runs while the condition is true
*condition 1 && condition 2*
**(counter < 3) && (num != rnd_number)**

# Problem solution

```
int rnd_number;
int num = 0;
int counter = 0;

//initialize random seed:
srand ( time(NULL) );

//generate random number:
rnd_number = rand() % 10 + 1;


while ((counter < 3) && (num != rnd_number)) {
  cout « "Guess the number: ";
  cin » num;
  counter++;
}
```

# Exploring a problem - factorial

Factorial of a number

$$n! = (n-1)! * n$$
$$0! = 1$$
$$1! = 0! * 1$$
$$2! = 1! * 2$$
$$3! = 2! * 3 = (1 * 2) * 3 = 2 * 3$$
$$\ldots$$

Motivation: Permutations

- The number of ways of selecting a president, vice president, secretary and treasurer in a club consisting of 10 persons.

$${}_{10}P_4 = \frac{10!}{(10-4)!} = \frac{10*9*8*7*6!}{6!} = 10 * 9 * 8 * 7 = 5040$$

# Factorial (2)

- Initial state before the loop
  - We need to know n (user input)
  - A placeholder for the intermediate calculations
    - Initial value (0, 1, n) ?
- Termination condition in the loop
  - How many operations do we need to perform ?
    - Can we use a counter ?
    - Can we do it without an explicit counter ?
- Change of state during the loop
  - How to perform the calculations ?

# Factorial (3)

```
int n;
cin » n;
// initial state?
int total = ?;
int counter = ?;

while (condition) {
  //body of the loop
}
cout « "Result is: " « total;
```

# Factorial (4)

The calculation expanded

$$n * (n-1) * (n-2) * (n-3) * ... * 2 * 1$$

Two views of the same equation

- n decreases by one every step
- n gets substracted by a number that increases by one every step
- we never multiply by zero at the end!

# Factorial (5)

- Decrement n:

```
int total = 1;
while (n > 1) {
  total = total*n;
  n=n-1;
}
```

- Increment counter:

```
int total = 1;
int counter = 0;
while (counter < n) {
  total = total*(n-counter);
  counter++;
}
```

# Exploring a problem - Root finding

Find the square root of a number, without using the sqrt function.

- Many approaches!
- What constitutes a good solution ?
  - The square root of 2 is a irrational number...infinite decimals
  - 99/70 is a common used approximation. (error less than 1/10.000)
- Iterative solution
  - Perform a calculation to improve our solution
  - Measure the error obtained (distance from the solution)
    - Is it good enough ? stop.
    - Otherwise, repeat the process.

# Root finding (2)

Loop while the condition is true:

- Given the value of n, how to measure the error of its square root estimation:

$$abs(solution^2 - n) > delta$$

- *delta* is completely dependant on our needs!
- can we use this as a termination condition ?

# Root finding (3)

*Bisection* method, given *n* find its square root

1. Define lower and upper limits (1 and n)
2. Start with an initial guess of x
3. Repeat while termination condition is true
4. Take the average: x = (lower+upper)/2
5. If ($x^2$ >= n) then root between (lower,x)
   upper = x

5'. else root between (x,upper)
   lower = x

# Root finding (4)

## Bisection method, in C

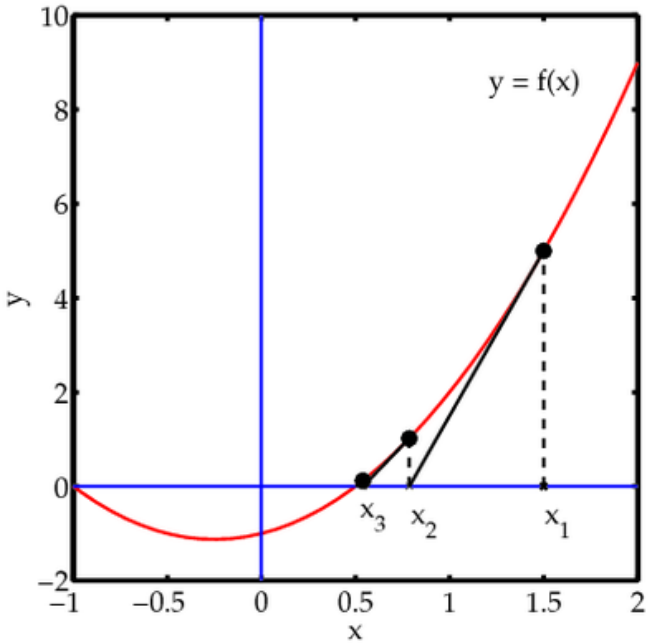
```
double guess = 1.0;
double lower=guess;
double upper=n;

while (fabs(guess*guess - n) > 0.001) {
  guess = (lower+upper)/2.0;
  if (guess*guess > n)
    upper = guess;
  else
    lower = guess;
}
cout « guess;
```

# Newton-Raphson method

Finds roots of real-valued functions in an iterative way.
Optimization in the search process, using derivatives

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$




Our problem can be reformulated as

$f(x) = x^2 - n$

$f'(x) = 2x$

$x_{n+1} = \frac{1}{2}(x_n + \frac{n}{x_n})$

# Newton-Raphson method (2)

- Condition for the loop is the same
- The main change is how we estimate the next guess

```
double guess = 1;
while (fabs(guess*guess - n) > 0.001) {
  guess = (guess + n/guess) / 2;
}
cout « guess;
```