

## Summary of Lecture 5

- Assigning and Re-assigning variables.
- Rules for naming variables.
- Mathematical operators.
- The `if`-statement.
- Input using `scanf`.
- Quadratic equations.

## This lecture

- General form of `if`-statement.
- Developing `quadratic.c` via nested `if`-statements.
- Boolean operators.
- Loops - the `while` and `for` statements.
- Fibonacci numbers.
- Prime numbers.

## Practical 1

- Practical 1 is **out today**. :-)  
Pick up a copy before leaving the lecture.
- **due** by **11am**, Tuesday 20 October.
- 4 Tasks:
  - Part A on (general) Imperial-to-Metric distance conversion.
  - Parts B-D are basic geometric tasks, when input is given through an interactive graphics tool.
- Should be able to attempt Parts A-C right away!
- We discuss Parts B-D in detail on Thursday 8 October.

---

# Computer Programming: Skills & Concepts (INF-1-CP1)

## Conditional Execution; Boolean operators; Loops

6th October, 2009



## if statement - general form

```
if (<condition-1>) {
    <statement-sequence-1>;
}
else if (<condition-2>) {
    <statement-sequence-2>;
}
...
else {
    <statement-sequence-n>;
}
```

- <condition-1>, ..., <condition-(n-1)> are all boolean expressions.
- <statement-sequence-1>, ..., <statement-sequence-n> are all sequences of C-programming statements.

## Boolean operators

Assume e1 and e2 are (usually arithmetic) expressions ...

We can apply boolean operators to form a boolean expression.

e1 == e2	e1 equal to e2
e1 != e2	e1 not equal to e2
e1 < e2	e1 less than e2
e1 <= e2	e1 less than or equal to e2
e1 > e2	e1 greater than e2
e1 >= e2	e1 greater than or equal to e2.

**note:** We can compare float expressions in this way - but int comparisons are *most reliable*.

## More complicated Boolean expressions

Assume e1 and e2 are boolean expressions ...

Can build more complicated boolean expressions iteratively.

0	false (always)
non-zero	true (always)
!e1	true if e1 is false
e1 && e2	true if (e1 is true and e2 is true)
e1    e2	true if (e1 is true or e2 is true)

The expressions e1, e2 are (formally) integer expressions.

Can think of integers as (informally) acting as boolean "type".

## Nested if-statements

- The <statement-sequence> place-holder in the general if-statement allows other if-statements to be part of the program fragment.
- This is a "nested" use of the if-statement.
- Example - refine our `quadratic.c` program further.

## quadratic.c - header and input code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>    // Need to include math.h to use sqrt.

int main(void) {
    int a, b, c;
    double x1, x2;

    printf("Input the x^2 co-efficient a: ");
    scanf("%d", &a);
    printf("Input the x co-efficient b: ");
    scanf("%d", &b);
    printf("Input the constant term c: ");
    scanf("%d", &c);
```

## quadratic.c - what if a=0

If  $ax^2 + bx + c$  is a quadratic, and  $a$  is 0, then we have a linear equation:

$$bx + c.$$

This has ...

- Exactly *one* root of value  $-c/(b)$ , if  $b \neq 0$ .
- No root at all, if  $b = 0$

*Now incorporate this case into our code:*

## quadratic.c - $a \neq 0$ case

```
if (b*b > 4*a*c) {
    x1 = (-(double)b - sqrt((double)(b*b -4*a*c)))/((double)(2*a));
    x2 = (-(double)b + sqrt((double)(b*b -4*a*c)))/((double)(2*a));
    ....
    return EXIT_SUCCESS;
}
else if (b*b == 4*a*c) {
    x1 = -((double)b)/((double)(2*a));
    ....
    return EXIT_SUCCESS;
} else {
    printf("There are 0 real solutions to %dx^2 + %dx +%d = 0.\n", a, b, c);
    return EXIT_SUCCESS;
}
```

## quadratic.c - all cases

```
if (a != 0) {
    if (b*b > 4*a*c) {
        x1 = (-(double)b - sqrt((double)(b*b -4*a*c)))/((double)(2*a));
        x2 = (-(double)b + sqrt((double)(b*b -4*a*c)))/((double)(2*a));
        ...
    }
    else if (b*b == 4*a*c) {
        x1 = -((double)b)/((double)(2*a));
        ....
    }
    else {
        printf("There are 0 real solutions to %dx^2 + %dx +%d = 0.\n", a, b, c);
        return EXIT_SUCCESS;
    }
}
```

```

}
else if (b != 0) {
    x1 = -((double)c)/((double)(b));
    printf("There is 1 real solution to %dx^2 +%dx +%d = 0.\n", a, b, c);
    printf("It is %lf.\n", x1);
    return EXIT_SUCCESS;
}
else {
    printf("There are 0 real solutions to %dx^2 + %dx +%d = 0.\n", a, b, c);
    return EXIT_SUCCESS;
}
}

```

## while

```

while (<condition>) {
    <statement_sequence>;
}

```

while means "repeat until failure" (of the <condition>).

<statement-sequence> must alter some parameters involved in <condition>.

WHY?

## Fibonacci Numbers

```

0      1
0 + 1 = 1
      1 + 1 = 2
          1 + 2 = 3
              2 + 3 = 5
                  3 + 5 = 8
                      5 + 8 = 13
                          8 + 13 = 21

```

## fibonnaci.c

```

int main(void) {
    int n, next, count;
    int previous = 0; /* Fibonacci -1 */
    int current = 1; /* Fibonacci 0 */
    ...
    count = 0;
    while (count != n) {
        next = previous + current; // i.e. 2 = 0 + 1
        previous = current;
        current = next; // after: 2 + 1
        ++count;
    }
    printf("Fibonacci %d is %d", n, current);
    return EXIT_SUCCESS;
}

```

## running fibonnaci.c

```
: ./a.exe
Calculate which Fibonacci number? 0
Fibonacci 0 is 1

: ./a.exe
Calculate which Fibonacci number? 1
Fibonacci 1 is 1

: ./a.exe
Calculate which Fibonacci number? 2
Fibonacci 2 is 2

: ./a.exe
Calculate which Fibonacci number? 7
Fibonacci 7 is 21
```

## while-statement: Repeat n-times

```
initialise_iterator;
while (<not_iterator_endpoint>) {
    <statement_sequence>;

    next_iterator_value;
}
```

## while-statement

### Counting-up:

```
count = 0;
while (count != n) {
    <statement_sequence>;
    ++count;
}
```

### Counting-down:

```
count = n;
while (count != 0) {
    <statement_sequence>;
    --count;
}
```

## The for-loop

```
for (count = n; count != 0; --count) {
    <statement_sequence>;
}
```

## Fibonacci using for

```
int n, next, count;
int previous = 0; /* Fibonacci -1 */
int current = 1; /* Fibonacci 0*/

for (count = n; count != 0; --count) {
    next = previous + current;
    previous = current;
    current = next;
}
```

## Prime Numbers

**Definition:** A prime number is any natural number which has no factors except itself and 1.

**Prime:** 3, 7, 11

**Not Prime:** 9 (3\*3), 10 (2\*5)

Simple test for primes:

n is prime if n=1 or if there is no integer k  
between 2 and  $\sqrt{n}$  such that  $n \% k = 0$ .

## prime.c

```
...
k = 2; // First divisor-attempted is 2
int prime = 1 ;
while ((k*k) <= n) && (prime)) { // finish at sqrt(n)

    if ((n % k) == 0) {
        printf("%d is %d * %d\n", n, n/k, k);
        prime = 0; // terminate the loop
    }
    ++k; // Test each value
}
if (prime)
    printf("%d is a prime number\n", n);
return EXIT_SUCCESS;
```

## scanf - erroneous input

*What if the user types a word, when an integer is required?*

Apart from the *action* performed by `scanf` (reading, or attempting to read, the object of the specified type), `scanf` returns an integer, which is the *number of input items assigned*. This may be fewer than provided for, or even zero, in the event of a matching failure.

This returned value can be used to test for a successful read:

```
scanf("%d", &a) == 1
```

if and only if an integer was successfully read into a.