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

11th October, 2010
Summary of Lecture 8

- for and while statements
- programs for Fibonacci and prime numbers
This Lecture

▶ Practical demonstration of writing a program
▶ basic debugging with printf
▶ scanf and erroneous input (!)
The Task

(From an exam question.) Using the Descartes package, write a program which takes three points from the user, draws the resulting triangle, computes the centroid of the triangle and draws rays from the centroid to each vertex.

(The centroid of a polygon is the average of its vertices – i.e. take the average of the $x$-coordinates and the average of the $y$-coordinates.)
First Step

Stop!

Think!
Plan

- Set up Makefile and skeleton program – copy and modify existing;
- develop program incrementally;
- at each stage, insert debugging information;
- at each stage, test.
Setting Up

Writing a Makefile from scratch is rather rare. And you need to understand how they work . . .
You can go a long way by copying and tweaking.
So we’ll copy the Makefile (and Descartes package) from the previous lectures.
Then edit in “the obvious way” for a program called triangle.
Skeleton Program

As usual ...
And on with the job
Tips to remember

If you don’t understand what your program is doing, add printfs and trace what’s happening to your variables.
(Advanced: use a debugger – but they have a steep learning curve.)

*Edit–compile–run* should be thought of as *edit–compile–test*.

To detect uninitialized variables, add `-O1` to the C flags along with `-Wall`. 
scanf - erroneous input

This bit not relevant for Practical 1

... 

EXCEPT for Part A
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`. 
Suppose we want to read in an integer to \( x \):

We can *test* for success by saving the returned value of `scanf`:

```c
read_succ = scanf("%d", &x);
if (read_succ == 1) {
    ....
}
else {
    ....
}
```

What about the else branch?

- Print an error message and terminate?
- Can give the user a second try.

*CP1–9 – slide 13 – 11th October, 2010*
scanf error-checking - first attempt

printf("Please input an integer: ");
read_succ = scanf("%d", &x);
if (read_succ == 1) {
    ....
}
else { /* read_succ must have been 0 */
    printf("That wasn't an integer! Try again: ");
    read_succ = scanf("%d", &x);
    ....
}

PROBLEM: Guaranteed to fail . . .
WHY?
scanf error-checking - “skipping over”

scanf("%*s"); - means “skip over” first item in read-buffer from standard input (the s is for ‘string’, the * for ‘don’t save’).

printf("Please input an integer: ");
read_succ = scanf("%d", &x);
if (read_succ == 1) {
    ....
}
else { /* read_succ must have been 0 */
    scanf("%*s"); /* scan the bad-input, don’t save */
    printf("That wasn’t an integer! Try again: ");
    read_succ = scanf("%d", &x);
    ....
}
scanf error-checking - loops

printf("Please input an integer: ");
read_succ = scanf("%d", &x);
if (read_succ != 1) { /* read_succ must have been 0 */
    while (read_succ != 1) {
        scanf("%*s"); /* scan bad-input, don’t try to save */
        printf("That wasn’t an integer! Try again: ");
        read_succ = scanf("%d", &x);
    }
}
    /* Now we definitely have an int; do the work */
}

Examples - with fibonacci.c, fibonacci1.c, fibonacci2.c, fibonacci3.c,