Loops

Darin Brezeale

The University of Texas at Arlington

Increment/Decrement Oper.

The following operators are available in C for incrementing and decrementing variables by a value of one:

      ++   increment

      --   decrement

Example: We could use

```
a = a + 1;
```

or

```
a++;
```

Increment/Decrement cont.

These operators can be placed before (prefix) or after (postfix) a variable:

```
int x = 5;

x--;

/*  x has a value of 4 here  */
```

or

```
int x = 5;

--x;

/*  x has a value of 4 here  */
```

Both reduce `x` by one. Later we will see examples where the choice of prefix or postfix matters.

Basic Concepts – Loops

C has the following loop constructs:

- `while`

- `for`

- `do-while`

Basic Concepts – Loops

Loops allow us to repeat a task. We need some way to determine when the loop should terminate. This could be

- after a predetermined number of iterations
- when some condition has been met

Basic Concepts – Loops

When the loop will terminate after a predetermined number of iterations, we need:

- a counting variable

- a test of that variable

- to increment/decrement that variable

`while` loop

The basic form of the `while` loop is

```
while(test)
     do_something;
```

As long as `test` is true, the loop will repeat.

To do multiple things in each iteration, we use curly braces:

```
while(test)
{
     do_something;
}
```

while loop

while loop that stops after predetermined number of iterations:

```c
#include <stdio.h>

int main(void)
{
    int i = 1, sum = 0;

    while( i <= 5 )
    {
        sum = sum + i;
        i++;
    }

    printf("sum of the integers 1 to %d is %d\n", i-1, sum);
}

/* sum of the integers 1 to 5 is 15 */
```

while loop

There are many occasions in which we don't know in advance how many times the loop should repeat, but we do know under what conditions the loop should terminate.

See `example-while.c` on the course webpage.

for loop

The `for` loop has the following form:

$$\texttt{for}(\text{expression}_1; \text{expression}_2; \text{expression}_3)$$
$$\text{do\_something;}$$

where

expression_1 initializes the counter

expression_2 is the condition for stopping

expression_3 is the method of incrementing the counter at the end of the loop

Note_1: Each expression is optional, but the semicolons are not.

Note_2: expression_3 is always evaluated at the bottom of the loop.

for loop example

```
int i;
for(i = 1; i <=3; i++)
    printf("i is %d\n", i);
```

which produces

```
    i is 1
    i is 2
    i is 3
```

for cont.

The counter variable doesn't have to be used in the statements that are part of the `for` loop.

```
int i;
for(i = 10; i > 6; i--)
    printf("Tick\n");
```

which produces

```
Tick
Tick
Tick
Tick
```

for cont.

Here is the `for` loop version of the `while` loop we saw earlier:

```c
#include <stdio.h>

int main(void)
{
    int i, sum = 0;

    for( i = 1; i <= 5; i++ )
        sum = sum + i;

    printf("sum of the integers 1 to %d is %d\n", i-1, sum);
}

/*  sum of the integers 1 to 5 is 15  */
```

Another `for` example

The probability distribution function for a normal distribution is

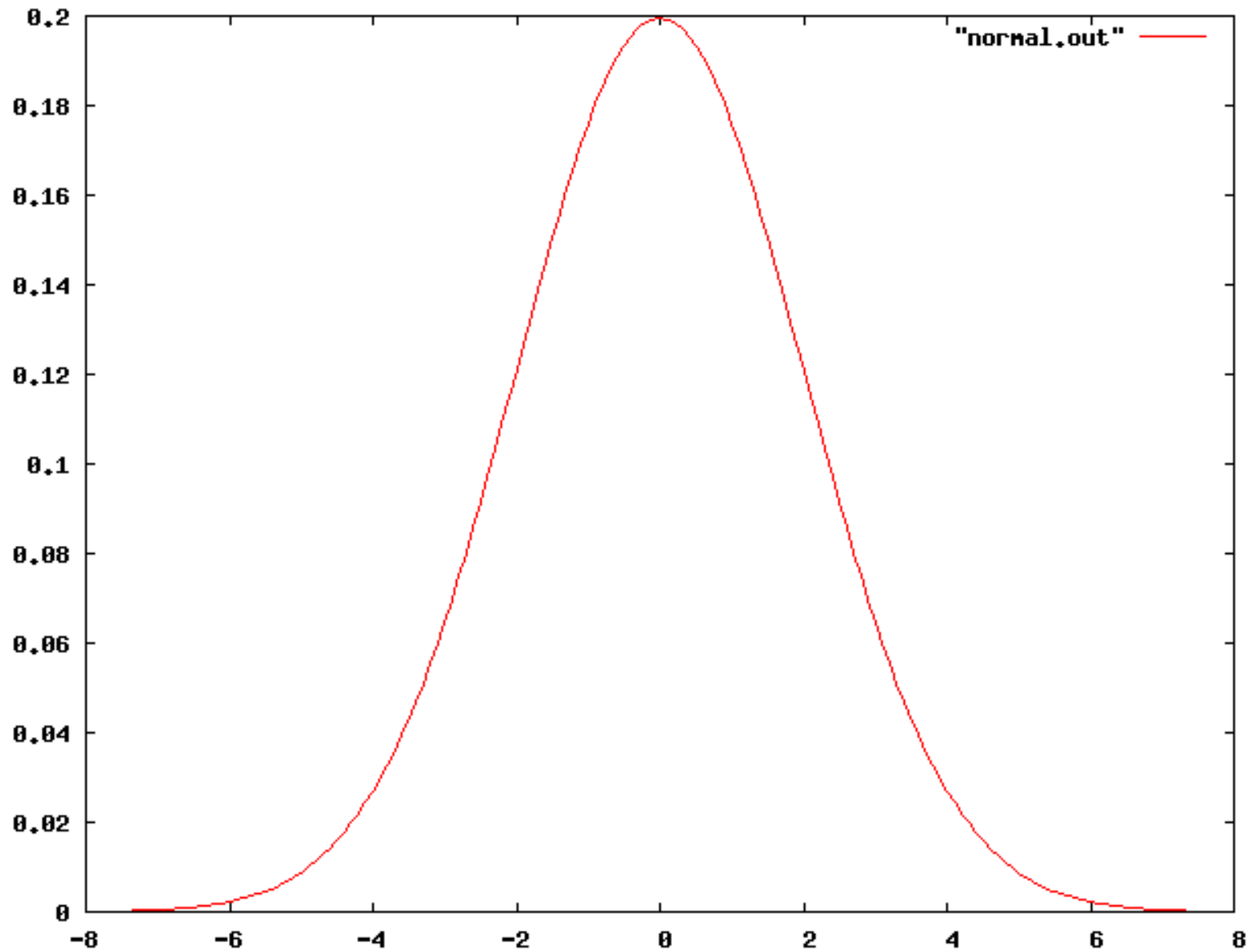$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

where

$\mu = \text{mean}$

$\sigma = \text{standard deviation}$

See `normal.c` for an implementation of this.

Note: This example shows that when multiple statements are part of the loop, we must use curly braces. It also shows that floating point numbers can be used for counting.

Normal distribution plotted

`do-while` loop

The `do-while` loop differs from the `while` loop in that the body will be visited once before the test is evaluated. It has the form:

```
do
    do_something;
while(test);
```
or
```
do
{
    do_something;
}
while(test);
```

See `example-dowhile.c` on the course webpage.

Changing loop behavior

Sometimes we want to end a loop early or move on to the next value. We have two ways of doing this:

1. `continue` – jump to the very end of the current loop

2. `break` – get out of the current loop completely

continue Statement

```c
#include <stdio.h>

int main(void)
{
    int i;
    for(i = 1; i <= 10; i++)
    {
        if ((i%2) != 0)
            continue;  /* skip over any remaining statements */
        printf("the square of %2d is %3d\n", i, i*i);
    }
}
```

produces

```
the square of  2 is    4
the square of  4 is   16
the square of  6 is   36
the square of  8 is   64
the square of 10 is  100
```

break Statement

```c
#include <stdio.h>
int main(void)
{
    int i, k;

    for(i = 1; i < 5; i++)
        for(k = 1; k < 5; k++)
            if(k == i)
            {
                printf("%d\n", k);
                break;
            }
            else
                printf("%d,", k);
}
```

produces

```
1
1,2
1,2,3
1,2,3,4
```