

ENCM Fall 2005: Handout for L02 (Dr. Norman's lecture) Fri., Sept. 30

Author: Dr. S. A. Norman. Electronic copies of handouts for L02 and T02 can be found at <http://www.enel.ucalgary.ca/People/Norman/encm339fall2005/>

Demonstration of the strcpy function (repeats Example 3 from Wed., Sept. 28) ...

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char s[6];
    strcpy(s, "foo");
    printf("BEGIN%sEND\n", s);
    return 0;
}
```

Read function prototypes for string-handling functions.

One of many correct ways to define strcpy ...

```
char *strcpy(char *dest, const char *src)
{
    int i = 0;

    /* point one */

    while (src[i] != '\0') {
        dest[i] = src[i];
        i++;
    }

    /* point two */

    dest[i] = '\0';
    return dest;
}
```

About the return value from strcpy: The C standard says that `strcpy` has a return type of pointer-to-char—see page 249 of K&R. (Confusingly, Section 5.5 of K&R shows implementations of a non-standard version of `strcpy`.) The return value is the address of the beginning of the destination string.

This return value is hardly ever used. It could be used to copy a string twice, as in

```
char s1[6];
char s2[6];
strcpy(s2, strcpy(s1, "hello"));
```

where the return value of the first (inner) call is used as an argument to the second (outer) call. However, it's more readable and no less efficient to do this instead:

```
char s1[6];
char s2[6];
strcpy(s1, "hello");
strcpy(s2, s1);
```

Example of declaring a function argument with array declaration syntax. *A key point of this example is that y is really being declared as a **pointer**.*

```
double average(const double y[], int count);

int main(void)
{
    double x[] = { 1.1, 1.2, 1.6 };
    double avg;

    avg = average(x, 3);
    return 0;
}

double average(const double y[], int count)
{
    double sum = 0.0;
    int i;

    for (i = 0; i < count; i++)
        sum += y[i];

    /* point one (after the for loop is finished) */

    return sum / count;
}
```

Because the above syntax for function arguments is widely used, it is important for all C programmers to understand what it really means. However, in ENCM 339 we will prefer the following syntax, because it is less misleading about the type of y .

```
double average(const double *y, int count);

int main(void)
{
    double x[] = { 1.1, 1.2, 1.6 };
    double avg;

    avg = average(x, 3);
    return 0;
}

double average(const double *y, int count)
{
    double sum = 0.0;
    int i;

    for (i = 0; i < count; i++)
        sum += y[i];
    return sum / count;
}
```