
Computer Programming: Skills & Concepts (CP1)

Strings

9th November 2009



Last lecture (5th Nov)

- Boolean expressions in detail;
- Expressions for incrementing;
- *Precedence* for evaluating expressions.

Today's lecture

- Strings.
- String I/O
- String Comparison.

Strings

A *string* is any 1-dimensional character array that is terminated by a null character.

- Null is `'\0'`.
- Strings are declared in function arguments either as `char *s` or `char s[]`.
eg, `void foo(char *s)` or `void foo(char s[])`
- In declaring a string, array length must be 1 greater than the longest string it will hold.
eg, `char[11]` can hold a 10-character string.

The string library

- Need to include it at the start:
 - `#include <string.h>`
- To copy a string `s2` into `s1`:
 - `strcpy(s1,s2);` `strcpy(s1,"Hello\n");`
- To add `s2` onto the end of `s1`:
 - `strcat(s1,s2)`
- Returns the length of `s1`:
 - `strlen(s1)`
- Many others . . .

The string library - types

```
char *strcpy(char *p1, const char *p2);
```

*Actually returns the pointer p1 which at return time holds the value of *p2.*

```
char *strcat(char *p1, const char *p2)
```

similar

```
size_t strlen(const char *p1)
```

the return type will be unsigned int or similar.

String I/O

(don't need `<string.h>` for these)

- To printf a string: `printf("%s", s1);`
- To read in a string:
 - `scanf("%s", s1);` */* ?why no & on s1? */*
- To print a float a into a string s1:
 - `sprintf(s1, "hello, num=%f", a);`
 - `sprintf` returns an integer, being the number of chars written.
- Similarly, we can read ints/floats etc; from a string via `sscanf`:
 - `int sscanf(s1, "%d Bellevue Road", &door);`
 - Value returned is the number of variables assigned to.

What about <, <=, == etc on strings?

```
int main(void) {
    char sone[] = "hiya";
    char stwo[] = "cp";
    char sthr[] = "coders";
    if (sone <= stwo)
        printf("'hiya' is less than or equal to 'cp'.\n");
    else
        printf("'cp' is less than 'hiya'.\n");
    if (stwo <= sthr)
        printf("'cp' is less than or equal to 'coders'.\n");
    else
        printf("'coders' is less than 'cp'.\n");
    return EXIT_SUCCESS;
}
```

<, <=, == don't work for strings

(sone <= stwo)

- sone and stwo are *pointers* to char variables (ie, are addresses in memory).
- comparison is true is and only if address of sone is less than stwo.

Output is *unpredictable*: compiler may allocate memory addresses for variables

- ... in order of declaration in the program, or maybe
- ... combination of declaration order and string length, or maybe
- ... in reverse order of declaration in program, or even
- ... in lexicographic order of initialization string (if given).

Better (non)-example for \leq

```
char sone[12], stwo[12];
printf("Input 1 please: ");
scanf("%s", sone);
printf("\nInput 2 please: ");
scanf("%s", stwo);
if (sone <= stwo)
    printf("%s is less than %s.\n", sone, stwo);
else
    printf("%s is less than %s.\n", stwo, sone);
```

No initialization bias on memory-allocation.

Can swap roles of input 1 and 2 to see result of comparison is non-lexicographic.

strcmp

```
int strcmp(const char *s1, const char *s2);
```

returns 0 if s1 and s2 are equal,

a negative int if string s1 is *lexicographically* less than s2

a positive int if string s1 is *lexicographically* greater than s2

```
...
```

```
if (strcmp(sone, stwo) <= 0)
    printf("'hiya' is less than or equal to 'cp'.\n");
else
    printf("'cp' is greater than 'hiya'.\n");
```

Comparing arrays of other types

A string is a char array. What about comparing arrays of ints or floats?

```
int memcmp (const void *a1, const void *a2, size_t size);
```

- `memcmp` compares the `size` bytes of memory beginning at `a1` against the `size` bytes of memory beginning at `a2`.
- Value returned has the same sign as the difference between the *first differing pair of bytes*.
- For this reason, only useful for testing *equality*, not relative order.

Assigned Reading (Kelley and Pohl)

For Strings: §6.10, §6.11, Appendix A.14