C String Library Routines

Wing Laboratory 3

Handed out: 20 September 2001
Due: 27 September 2001

This problem asks you to write formal specifications of the interfaces to two standard C string library routines. It also foreshadows the problem of reverse engineering code.

Write pre-/post-condition specifications for `strcat` and `strncpy`, whose informal descriptions are appended. The implementations of these routines are taken verbatim from /afs/cs/mach/src/usr/cs/lib/libc):

```
char *strcat (s1, s1)
register char *s1, *s2;
{
        register char *os1;
        os1 = s1;
        while (*s1++);
        --s1;
        while (*s1++ = *s2++);
        return(os1);
}

char *strncpy(s1, s2, n)
register char *s1, *s2;
{
        register i;
        register char *os1;

        os1 = s1;
        for (i = 0; i < n; i++)
                if ((*s1++ = *s2++) == '\\0') {
                        while (++i < n)
                                *s1++ = '\\0';
                        return(os1);
                }
        return(os1);
}
```

Specify as precisely as you can the behavior of `strcat` and `strncpy` exhibited by the implementations. Use Z notation for sets, relations, sequences, etc. as the assertion language for the predicates in your pre/post-conditions

```
#include <ctype.h>
int _tolower(c)
  char c;
```

If c is an uppercase letter, then _tolower returns the corresponding lowercase letter. The result of _tolower is undefined for all other arguments.

See also tolower (section 11.1.18), which is slower but produces a defined result for any character or EOF.

11.1.21 _toupper

```
#include <ctype.h>
int _toupper(c)
  char c;
```

If c is a lowercase letter, then _toupper returns the corresponding uppercase letter. The result of _toupper is undefined for all other arguments.

See also toupper (section 11.1.19), which is slower but produces a defined result for any character or EOF.

11.2 STRING PROCESSING

By convention, strings in C are of variable length and are terminated by a null character (that is, '\0'). The compiler automatically supplies an extra null character after all string constants, but it is up to the programmer to make sure that strings created in string variables (that is, character arrays) end with a null character.

All the characters in a string, not counting the terminating null character, are together called the *contents* of the string. An empty string contains no characters and is represented by a pointer to a null character. Note that this is *not* the same as a null character pointer (NULL), which is a pointer that points to no character at all. When we speak of a "pointer to the first character of a string," we mean a pointer to the terminating null character if the string is empty and to the first character of the contents if the string is not empty.

All of the string-handling facilities described here assume that strings are terminated by a null character. When characters are transferred to a destination string, no test is made for overflow of the destination. It is up to the programmer to make sure that the destination area in memory is large enough to contain the result string, including the terminating null character.

All of the facilities described here are declared by the library header file string.h.

11.2.4 strcpy

```
#include <string.h>
char *strcpy(s1, s2)
  char *s1, *s2;
```

strcpy copies the contents of the string s2 to the string s1, overwriting the old contents of s1. The entire contents of s2 are copied, plus the terminating null, even if s2 is longer than s1. A pointer to the first character of s1 is returned.

The results are unpredictable if the two string arguments overlap in memory.

See also strncpy (section 11.2.9), which copies a specified number of characters.

11.2.1 strcat

```
#include <string.h>
char *strcat(s1, s2)
  char *s1, *s2;
```

strcat appends the contents of the string s2 to the end of the string s1. A pointer to the first character of s1 is returned. The null character that terminates s1 (and perhaps other characters following it in memory) is overwritten with characters from s2 and a new terminating null character. Characters are copied from s2 until a null character is encountered in s2.

The results are unpredictable if the two string arguments overlap in memory. In particular, it does not necessarily work to supply the same string as both arguments in an attempt to double its length.

See also strncat (section 11.2.7), which appends a limited number of characters.

11.2.9 strncpy

```
#include <string.h>
char *strncpy(s1, s2, n)
  char *s1, *s2;
  int n;
```

strncpy copies exactly n characters to s1. It copies up to n characters from s2. If there are fewer than n characters in s2 before the terminating null character, then null characters are written into s1 as padding until exactly n characters have been written. If there are n or more characters in s2, then only n characters are copied, and so only a truncated copy of s2 is transferred to s1. It follows that the copy in s1 is terminated with a null by strncpy only if the length of s2 (not counting the terminating null) is less than n.

If the value of n is zero or negative, then this function makes no net change to the contents of memory.

The results are unpredictable if the two string arguments overlap in memory.

See also strcpy (section 11.2.9), which copies an unlimited number of characters.