Arrays and Strings

CS 2022: Introduction to C

Instructor: Hussam Abu-Libdeh

Cornell University
(based on slides by Saikat Guha)

Fall 2011, Lecture 5

Announcement

Lab meetings

Fridays 8:00am – 8:50am in **B7** Upson Hall

Arrays

- Contiguous memory
- Type is same as element-pointer
  - Accessing array elements is syntactic sugar for pointer arithmetic
- On the stack
  - Fixed-size (at compile time)
  - Compiler allocates
  - Compiler deallocates
- On the heap
  - Variable size (malloc)
  - Explicit allocation/deallocation

Declaring Arrays

```
void foo(int x) {
    int a[100];
    int b[] = {0, 1, 0, 2, 3, 1};
    int c[x]; // ERROR: Size must be const.

    a[0] = 10;
    a[5] = b[2];
    a[100] = 10; // BAD: Clobbering stack!!

    *(a + 1) = 20;   // same as a[1] = 20;
    *b = *(a + 5);   // same as b[0] = a[5];
}
```

Declaring Arrays

```
#include<stdlib.h>

void foo(int x) {
    int *a = malloc(x * sizeof(int));

    a[0] = 10;        // same as *a = 10;
    a[1] = a[0];      // same as *(a+1) = *a;

    free(a);
}
```

Relevant Library

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

- Set all elements to 0:

  `memset(array, 0, bytes)`
- Copy elements:

  `memcpy(dst, src, bytes)`
- Note:
  - bytes = number of elements * sizeof(int) for integer arrays.

Array Problems

- No array-bound checks. No warnings.
  - Can clobber stack or heap
  - **especially** with array-to-array copy when the destination array doesn't have enough space.
- `sizeof(array)` returns:
  - number of *bytes*, when exact size can be determined
  - size of pointer, when size cannot be determined at compile time and is treated as a pointer. Avoid this!

Characters

- Type for character: `char`
- 1-byte in size
- Enclosed in single-quotes
- `printf` format: %c
- ASCII character
  - Alpha: `'a'`
  - Digit: `'4'`
  - Special: `'\t'`
  - Null: `'\0'`
- Type for unicode charcter: `wchar_t`

Strings

- Just an array of characters
- String: `char *` or `char []`
- Terminated by *Null character* (`'\0'`)
- Literals enclosed in double-quotes
- `"Hello"` is the same as

```
char str[] = {'H', 'e', 'l', 'l', 'o', '\0'}
```

Strings

- `printf` format: `%s`
- `(str + 5)`:
  - type is `char *`;
  - value: substring starting at 6th character
- `*(str + 5)` or `str[5]`
  - the 6th character

`string.h` Library Functions

- Many functions for common string manipulation tasks.
  - . . . use them, they will make your life a lot easier
- Library functions expect null-terminated strings.
- When joining/copying/splitting strings, library inserts null-character where appropriate.

Getting Help on Library Functions

To quickly check the manual pages,

```
man func_name
```

on a Unix/Linux system. Example:

- `man strlen`

`string.h` Library Functions

- `strlen(s)`
  Length, **not** including '\0'
- `strncpy(dst, src, n)`
  Copies 'n' characters from src to dst (incl. '\0')
- `strncat(dst, src, n)`
  Copies characters from src to end of dst until dst has 'n' characters (incl. '\0')

`string.h` Library Functions

- `int strcmp(char *s1, char *s2)`
  Compares strings. Returns 0 when strings **are equal**.
  Positive when `s1` greater, negative when `s1` smaller.
  ASCII order.

Note: Cannot use == to check string equality since it compares pointers. Points to two different copies of the same string will be different.

`string.h` Library Functions

- `char *strstr(char *haystack, char *needle)`
  Search for a substring in a string
- `char *strdup(char *str)`
  Allocates space on the heap (with malloc) and copies the argument into the allocated space.
  Caller MUST free the returned string when done.
- `char *strtok_r(char *str, char *delim, char **sav)`
  Used to break apart a string into pieces. See man-page for details.

Multi-Dimentional Arrays

```
void foo(void) {
    int a[3][5];
    int b[7][3][6];

    a[1][2] = 10; // same as *(a + 1*5 + 2) = 10
    b[1][2][3] = 5; // same as *(b + 1*18 + 2*6 + 3)=5
}
```

Arrays of Strings

- `char **` or `char *a[]`
- e.g. command-line arguments
- `a[0]` is a string (type `char *`)