Computer Systems

CS107

Cynthia Lee
Topics

**STRINGS IN C:**

› Pointers and arrays in C  
  • Review from CS106B/X, building to a more intense level

› String structure in C  
  • Null terminating char

› String library functions  
  • strlen, strcmp, strcat, strcpy, strdup, …
Declaring an Array, Passing an Array to a Function

(CODE DEMO)
Starter code (needs work)

#include <stdio.h>
#include <stdlib.h>

double sum(double arr[])
{
    double total = 0;
    /* loop over array and sum */

    return total;
}

int main(int argc, char *argv[])
{
    double arr[] = {1.1, 2.2, 3.3, 4.4};
    double total = 0.0;

    /* want to call sum to calculate total of array values */
    total = sum(arr);

    printf("Sum = %g\n", total);

    return 0;
}
Key points from the code example:

```c
#include <stdio.h>
#include <stdlib.h>

double sum(double * arr, int length)
{
    double total = 0;
    for (int i=0; i<length; i++)
        total += arr[i];
    return total;
}

int main(int argc, char ** argv)
{
    double arr[] = {1.1, 2.2, 3.3, 4.4};
    double total = 0.0;

    /* want to call sum to calculate total of array values */
    total = sum(arr, 4);

    printf("Sum = %g\n", total);

    return 0;
}
```

- double arr[] and double *arr are equivalent for parameter types
  - Not quite true for local variable declarations
- Any time we pass an array ([] or * notation), we need to also pass along its accompanying array size
  - They’re always a pair
  - **Example:** argc to go with argv!
Strings

C-style strings
Strings in C: just an array of char but with a special ending sentinel value

"hello"

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>h</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>o</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\0</td>
</tr>
</tbody>
</table>

- \0 is the null-terminating character for strings
  - Always need to allocate one extra space in an array for it
- Special feature of char*/char[] arrays that are valid strings*, you don’t need to pass around the length in addition to the array!
  - strlen() function will loop over string and count how many chars until the null-terminating character
  - void strfun(char* str) { /* you don’t need to pass length! */
  - double sum(double* arr, int length) { /* here you do */
Strings in C: just an array of chars, but with a special ending sentinel value

```c
int main(int argc, char *argv[]) {
    int x = 4;
    char str[] = "hello";
}
```

- What does memory look like?
Code example: printArgsEndingInEnd
Prints all of its arguments that end in the suffix – “end"
Strings reference

- Some useful string functions:
  - `strcat(str1, str2)` // concat str2 to the end of str1
  - `strcmp(str1, str2)` // returns 0 if strings are equal, otherwise negative# or positive#, for < or > (alphabetically)
  - `strdup(str)` // returns a new (malloc’ed) copy of str
  - `strcpy(str1, str2)` // copies contents of str2 to str1
  - `strlen(str)` // finds the length of a str
  - `strstr(str1, str2)` // returns a ptr to the first occurrence of str2 in str1
Strings in C: passing them as arguments

(CODE DEMO, PART I)
Passing strings as arguments: code demo key points

```c
void lowercase(char *str) {

    // You don’t need to pass length (be careful with this)
    // You may alter the contents of a char* argument, but not redirect the pointer
    // For example, if you want to lengthen the string, you’re out of luck with char*
    // If you want to do this, add a level of indirection that gives you access to the char* pointer itself: char**, or return a char*
```
Strings in C: what does char* parameter passing look like in memory?

```c
void lowercase(char *str) {
    for (int i = 0; str[i] != '\0'; i++)
        str[i] = tolower(str[i]);
}

int main(int argc, char *argv[]) {
    int x = 4;
    char str[_______________];
    strcpy(str, "hello");
    lowercase(str);
}
```

What goes in the blank?
A. 4  
B. 5  
C. 6  
D. strlen("hello")  
E. Something else

What does memory look like?
Strings in C: what does char* parameter passing look like in memory?

```c
void lowercase(char *str) {
    for (int i=0; str[i] != '\0'; i++){
        str[i] = tolower(str[i]);
    }
}

int main(int argc, char *argv[]) {
    int x = 4;
    char str[6]; // or strlen("hello") + 1
    strcpy(str, "hello");
    lowercase(str);
}
```

- What does memory look like?
Strings in C: the nitty gritty details

```c
int main(int argc, char *argv[]) {
    int x = 4;
    char *str = "hello";
}
```

- What does memory look like?
Strings in C: even nittier, grittier details

```c
int main(int argc, char *argv[]) {
    int x = 4;
    char *str = "hello";
    str[4] = 'a'; /* not allowed - read only */
```