Computer Systems

CS107

Cynthia Lee
Topics

(MORE) STRINGS IN C:
› String library functions, with a focus on those that deal with storage allocation
  • strcat, strcpy, strdup
› More on how passing strings to functions works
  • Strings as pointers (char *)
  • New: pointers-to-strings (char **)
› More on where strings are stored
  • Previously focused on the stack
  • New: heap and read-only data segment

› New today: let’s do our multiple-choice question votes for real!
  • Not for a grade, just to help me calibrate topics and pace

💡 When poll is active, respond at PollEv.com/cs107
📞 Text CS107 to 22333 once to join
Test question:

**WHAT ARE YOUR CORE VALUES OR DEFINITIONS OF SUCCESS? (LIST AS MANY AS YOU LIKE; EX: LOYALTY, INDEPENDENCE, CREATIVITY, SPIRITUALITY/RELIGION, ...)**

When poll is active, respond at PollEv.com/cs107

Text **CS107** to **22333** once to join
Allocation of strings: stack vs heap

Where are strings stored?
Strings in C: what does char* parameter passing look like in memory?

When poll is active, respond at PollEv.com/cs107
Text CS107 to 22333 once to join

---

What goes in the blank?
A. 4
B. 5
C. 6
D. strlen("hElLo")
E. More than one of the above would work
F. Something else

```c
int main(int argc, char *argv[]) {
    int x = 4;
    char str[______________];
    strcpy(str, "hElLo");
    lowercase(str);
    printf("%s\n", str);
    return 0;
}
```

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[strlen("hElLo") + 1]; // i.e., 6
    strcpy(str, "hElLo");
    lowercase(str);
    printf("%s\n", str);
    return 0;
}
```

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

// BUG!
void make_adverb(char *str) {
    strcat(str, "ly");
}

int main(int argc, char *argv[]) {
    int x = 4;
    char str[strlen("quick") + 1];
    strcpy(str, "quick");
    make_adverb(str); // want to change to "quickly"
    printf("%s\n", str);
    return 0;
}

- What does memory look like?
Passing strings as arguments: “make_adverb” code demo key points

- You may alter the contents of a char* argument, but not redirect the pointer
Passing strings as arguments:
“adverb” code demo key points

- We can store strings on the **stack** by declaring them as an array of a fixed size on the stack, like these examples:

  ```c
  char str1[10];
  char str2[strlen("hello") + 1];
  ```

- We can store strings on the **heap** by declaring a pointer and then requesting memory from the heap, like these examples:

  ```c
  char * str1 = strdup("hello");
  char * str2 = malloc(strlen("hello") + 1);
  ...
  free(str1); // don't forget to free!
  free(str2); // don't forget to free!
  ```
Important detail: string literals

Where are strings typed into the code stored?
Strings in C: the nitty gritty details

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 */
```