CSE 303
Lecture 14

Strings in C

reading: *Programming in C* Ch. 9; Appendix B

slides created by Marty Stepp
http://www.cs.washington.edu/303/

Type char

- `char` : A primitive type representing single characters.
  - literal `char` values have apostrophes: `'a'` or `'4'` or `'\n'` or `'\''`

```
char letter = 'S';
printf("%c", letter);    // S
```

- you can compare `char` values with relational operators

```
'a' < 'b'  and  'X' == 'X'  and  'Q' != 'q'
```

  - An example that prints the alphabet:

```
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```

char and int

- chars are stored as integers internally (*ASCII* encoding)

```
'A' is 65,     'B' is 66,     ' ' is 32,     '\0' is 0
'a' is 97,     'b' is 98,     '*' is 42,     '\n' is 10

char letter = 'S';
printf("%d", letter);     // 83
```

  - mixing char and int causes automatic conversion to int

    `'a' + 2` is 99, `'A' + 'A'` is 130

  - to convert an int into the equivalent char, type-cast it

    `(char) ('a' + 2)` is `'c'`

Strings

- in C, strings are just arrays of characters (or pointers to `char`)

- the following code works in C:

```
char greet[7] = {'H', 'i', ' ', 'y', 'o', 'u'};
printf(greet);                  // output:  Hi you
```

- the following versions also work and are equivalent:

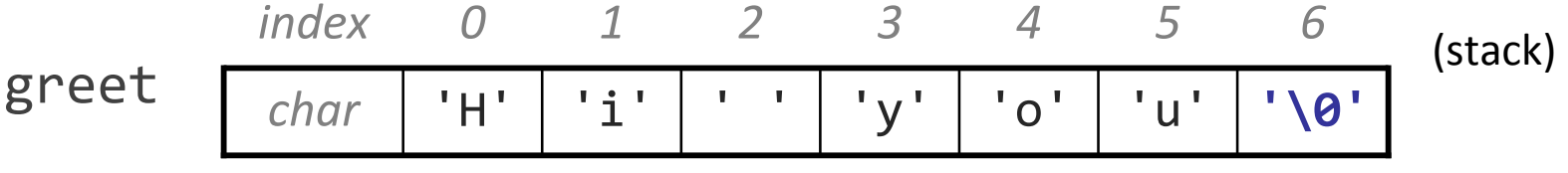
```
char greet[7] = "Hi you";
char greet[] = "Hi you";
```

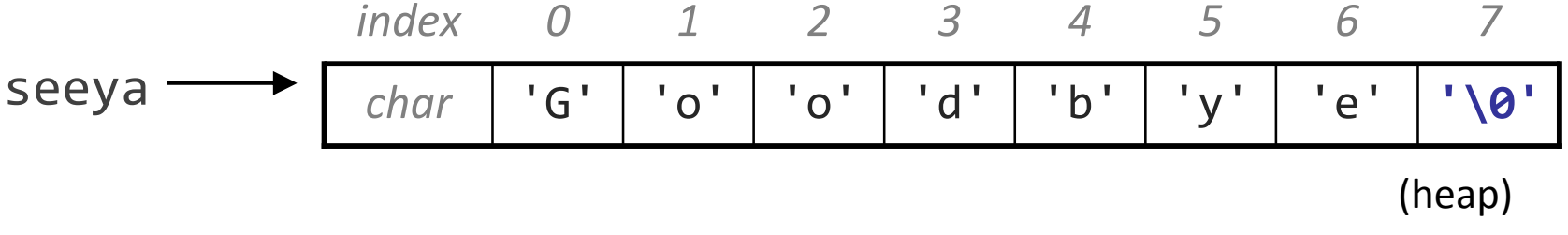
  - Why does the `word` array have 7 elements?

Null-terminated strings

- in C, strings are **null-terminated** (end with a 0 byte, aka `'\0'`)
- string literals are put into the "code" memory segment
  - technically `"hello"` is a value of type `const char*`

```
char greet[7] = {'H', 'i', ' ', 'y', 'o', 'u'};
char* seeya = "Goodbye";
```





String input/output

```
char greet[7] = {'H', 'i', ' ', 'y', 'o', 'u'};
printf("Oh %8s!", greet);    // output: Oh   hi you!

char buffer[80] = {'\0'};    // input
scanf("%s", buffer);
```

- `scanf` reads one *word* at a time into an array (note the lack of &)
- if user types more than 80 chars, will go past end of buffer (!)

- other console input functions:
  - `gets(char*)` - reads an entire line of input into the given array
  - `getchar()` - reads and returns one character of input

Looping over chars

- don't need `charAt` as in Java; just use `[ ]` to access characters

```
int i;
int s_count = 0;
char str[] = "Mississippi";
for (i = 0; i < 11; i++) {
    printf("%c\n", str[i]);
    if (str[i] == 's') {
        s_count++;
    }
}
printf("%d occurrences of letter s\n", s_count);
```

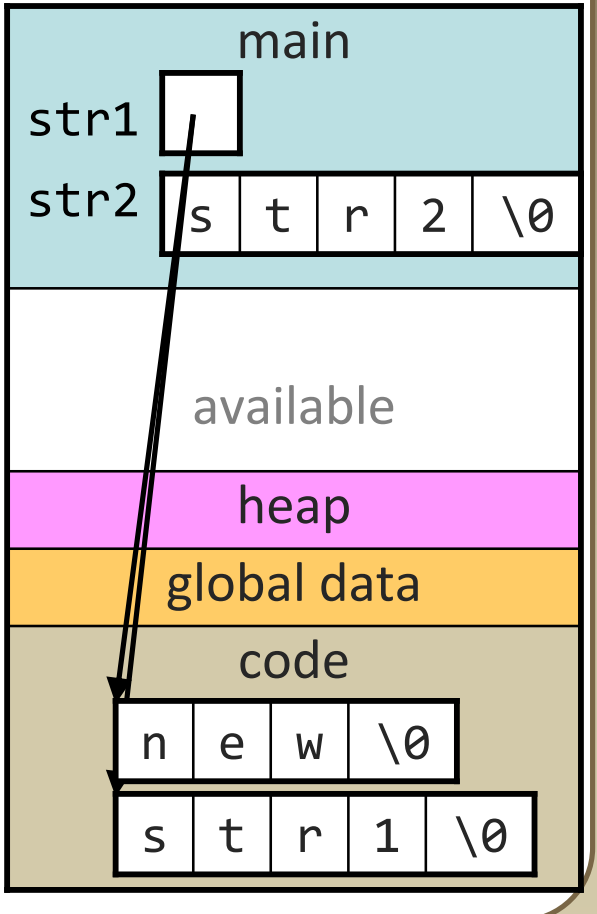
String literals

- when you create a string literal with "***text***", really it is just a `const char*` (unchangeable pointer) to a string in the code area

```
// pointer to const string literal
char* str1 = "str1";    // ok
str1[0] = 'X';          // not ok

// stack-allocated string buffer
char str2[] = "str2";   // ok
str2[0] = 'X';          // ok


// but pointer can be reassigned
str1 = "new";           // ok
str2 = "new";           // not ok
```



Pointer arithmetic

- adding/subtracting *n* from a pointer shifts the address by *n* times the size of the type being pointed to
  - Example: Adding 1 to a `char*` shifts it ahead by 1 byte
  - Example: Adding 1 to an `int*` shifts it ahead by 4 bytes

```
char[] s1 = "HAL";
char* s2 = s1 + 1;              // points to 'A'

int a1[3] = {10, 20, 30, 40, 50};
int* a2 = a1 + 2;               // points to 30
a2++;                           // points to 40

for (s2 = s1; *s2; s2++) {
    *s2++;                      // what does this do?
}
```

Strings as user input

```
char buffer[80] = {0};
scanf("%s", buffer);
```

- reads one *word* (not line) from console, stores into buffer

- *problem* : possibility of going over the end of the buffer
  - fix: specify a maximum length in format string placeholder

```
scanf("%79s", buffer);     // why 79?
```

- if you want a whole line, use `gets` instead of `scanf`
- if you want just one character, use `getchar` (still waits for \n)

String library functions

- `#include <string.h>`

function	description
`int strlen(s)`	returns length of string *s* until `\0`
`strcpy(dst, src)`	copies string characters from *src* into *dst*
`char* strdup(s)`	allocates and returns a copy of *s*
`strcat(s1, s2)`	concatenates *s2* onto the end of *s1 (puts `\0`)*
`int strcmp(s1, s2)`	returns < 0 if s1 comes before s2 in ABC order; returns > 0 if s1 comes after s2 in ABC order; returns 0 if s1 and s2 are the same
`int strchr(s, c)`	returns index of first occurrence of *c* in *s*
`int strstr(s1, s2)`	returns index of first occurrence of *s2* in *s1*
`char* strtok(s, delim)`	breaks apart ***s*** into tokens by delimiter ***delim***
`strncpy, strncat, strncmp`	length-limited versions of above functions

Comparing strings

- relational operators (==, !=, <, >, <=, >=) do not work on strings

```
char* str1 = "hello";
char* str2 = "hello";
if (str1 == str2) {          // no
```

- instead, use `strcmp` library function (0 result means equal)

```
char* str1 = "hello";
char* str2 = "hello";
if (!strcmp(str1, str2)) {
    // then the strings are equal
    ...
}
```

More library functions

function	description
`int atoi(s)`	converts string (ASCII) to integer
`double atof(s)`	converts string to floating-point
`sprintf(s, format, params)`	writes formatted text into *s*
`sscanf(s, format, params)`	reads formatted tokens from *s*

- `#include <ctype.h>` (functions for `char`s)

function	description
`int isalnum(c)`, `isalpha`, `isblank`, `isdigit`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `isxdigit`, `tolower`, `toupper`	tests info about a single character

  - `isalpha('A')` returns a nonzero result (`true`)

Copying a string

- 1. copying a string into a stack buffer:

```
char* str1 = "Please copy me";
char str2[80];       // must be >= strlen(str1) + 1
strcpy(str2, str1);
```

- 2. copying a string into a heap buffer (you must free it):

```
char* str1 = "Please copy me";
char* str2 = strdup(str1);
```

- 3. do it yourself (hideous, yet beautiful):

```
char* str1 = "Please copy me";
char str2[80];
while (*s2++ = *s1++);   // why does this work?
```