APS105 Lecture 18
7.5, 7.4

Dan Zingaro

March 8, 2010

Feedback from Reading Quiz

- You requested that we work on a couple of textbook exercises
- We'll do one or two at the end of class
- In general, we don't have time in lecture to do very many
- But, there is a secret place for working on exercises ...

Arrays of Strings

- There are two ways to make an array of strings
- Method 1: Declare a two-dimensional array with `x` rows and `y` columns
  - `x` is the number of strings you want to store
  - `y` is one more than the length of the longest string you want to store
- e.g. to store the strings `cat` and `tiger`, we could use:

```
char animals[2][6];
```

Arrays of Strings...

```
char animals[2][6];
```

To store `cat`:

```
animals[0][0] = 'c';
animals[0][1] = 'a';
animals[0][2] = 't';
animals[0][3] = '\0';
```

Or use an initializer:

```
char animals[][6] = {"cat", "tiger"};
```

c	a	t	'\0'	'\0'	'\0'
t	i	g	e	r	'\0'

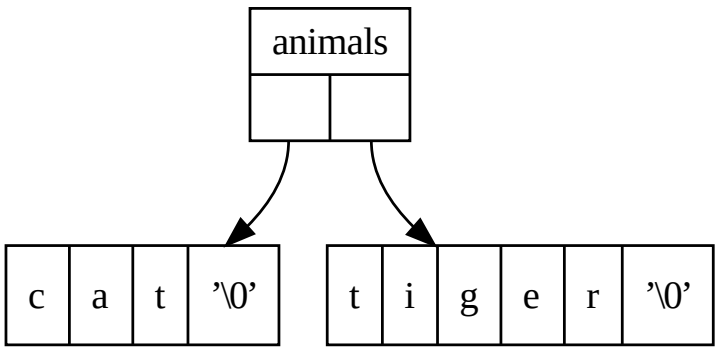
- This approach wastes space; short strings take up as much memory as long strings

Arrays of Strings...

- Method 2: Declare a one-dimensional array with `x` elements
  - `x` is the number of strings you want to store
  - Each array element is a pointer that points to a string constant
- e.g. to store the strings `cat` and `tiger`, we could use:

```
char *animals[] = {"cat", "tiger"};
```

- Each string is now stored using only the required space
- But, these are string constants; we cannot modify them!



ConcepTest

Assume we want to store the strings `Canada`, `Ontario`, and `flag`. Which of the following declarations could be used?

- A. `char words[3][7];`
- B. `char words[3][8];`
- C. `char words[3][9];`
- D. All of the above
- E. None of the above

Compact C Code

- C programmers are notorious for writing code that is extremely compact
- We should be familiar with several common idioms, so we can understand them when we read other people's code
- Here is a function that copies string `q` into string `p` (the same thing that `strcpy` does)
- It loops while we haven't found the `'\0'` at the end of `q`

```
void stringCopy (char *p, char *q) {
  while (*q != \0) {
    *p = *q;
    p++;
    q++;
  }
  *p = \0;
}
```

Idiom 1: combine assignment and comparison

- In C, the expression `2 + 5` has value 7
- An assignment `a = b` also has a value; it is whatever is assigned to `a`
- e.g. if variable `b` has value 9, then `a = b + 3` has value 12 (and it also assigns 12 to `a`, of course)
- The following if-statement causes `a` to get `b`'s value, and execute the `printf` if `a`'s new value is nonzero

```
if (a = b)
  printf ("Inside if\n");
```

ConcepTest

Consider the following function to copy string q into string p.

```
void stringCopy (char *p, char *q) {
  while ((*p = *q) != \0) {
    p++;
    q++;
  }
}
```

Does the code properly put a '\0' at the end of p?

- A. Yes, because the '\0' is copied just before the while-condition fails
- B. No, because there is no *p = \0; after the loop
- C. No, because no characters at all are copied into p

Idiom 2: combine assignment with increment

- When we say `*p`, we are accessing the data pointed to by `p`
- When we say `*(p++)`, we are still accessing the data pointed to by `p`, but then we are incrementing `p` to point to the next element

```
char a[6] = "hello";
char *p = a;
//Print h, make p point to next element
printf ("%c\n", *(p++));
//Print e, make p point to next element
printf ("%c\n", *(p++));
...
```

Idiom 2: combine assignment with increment...

- `*(p++)` is the same as `*p++`; `++` has higher precedence than `*`

```
void stringCopy (char *p, char *q) {
  while ((*p++ = *q++) != \0)
  ;
}
```

Exercise: Implementing Built-in Function

Implement `strcat` using `strlen` and `strcpy`.

Exercise: No Common Letters

Write a function `noCommon` that returns `true` if and only if the two string parameters have no letters in common.

```
bool noCommon (char *word1, char *word2)
```

```
noCommon ("first", "second") returns true
noCommon ("computer", "rain") returns false
```