# COMPUTER SCIENCE 1: STARTING COMPUTING CSCI 1300

The image part with relationship ID rId3 was not found in the file.

Ioana Fleming / Vipra Gupta
Spring 2018
Lecture 10

University of Colorado
Boulder

# Agenda

- Today and next time
  - While loops, cont.
  - Strings
  - Traversing strings using while loops

# Announcements

- Rec 4 due on 2/10
- Hmwk 3 due 2/11
- Practicum 1 has been scheduled:
  - February 21st, 2018
  - In lecture. 50 minutes. Bring a laptop!
  - Review: in lecture on 2/19
  - Visible in *Tentative Schedule* on Moodle

# While loops

- Cloud9 examples: *liftoff.cpp*
  - loop that is executed many many many many times = <u>infinite loop</u>: if nothing inside the loop body can turn the loop condition from True to False
  - loop that is executed 0 (zero) times: if the condition is False the first time

# Strings: Learning Objectives

- Referencing characters
  - indexing
- Operations with strings

  =, +, ==, <, >
- Processing Strings
  - .length()
  - other string functions
- Traversing strings
  - using while loops

# Strings

- Strings are sequences of characters:

```
"Hello world"
```

- If you include the string header,
  you can create variables to hold literal strings:

```
#include <string>
using namespace std;
...
string name = "Harry";
        // literal string "Harry" stored
```

# Strings

- String variables are guaranteed to be initialized even if you don't initialize them:

```
string response;
        // literal string "" stored
```

- `""` is called the empty string.

# Standard Class string

- Defined in library:

```
#include <string>
using namespace std;
```

- String variables and expressions
  - Treated much like simple types
- Can assign, compare, add:

```
string s1, s2, s3;
s3 = s1 + s2;   //Concatenation
s3 = "Hello Mom!" //Assignment
```

# Program Using the Class string

**Display 9.4 Program Using the Class `string`**

```
//Demonstrates the standard class string.
#include <iostream>
#include <string>
using namespace std;

int main( )
{
    string phrase;
    string adjective("fried"), noun("ants");
    string wish = "Bon appetite!";

    phrase = "I love " + adjective + " " + noun + "!";
    cout << phrase << endl
         << wish << endl;

    return 0;
}
```

*Initialized to the empty string.*

*Two equivalent ways of initializing a string variable*

**SAMPLE DIALOGUE**

```
I love fried ants!
Bon appetite!
```

# Common Error – Concatenation of literal strings

```
string greeting = "Hello, " + " World!";
                // will not compile
string exclamation = “!”;
string greeting = “Hello, “ + “World” +
  exclamation;
                // is allowed
```

Literal strings cannot be concatenated. But if one operand is a string, then it’s ok.

# I/O with Class string

- Just like other types!

```
string s1, s2;
cin >> s1;
cin >> s2;
```

- Results:
  User types in:
  May you live long and prosper!

- Extraction still ignores whitespace:
  s1 receives value "May"
  s2 receives value "the"

# String Functions

- The **`length`** *member function* yields the number of characters in a string.
- Unlike the **`sqrt`** or **`pow`** function, the **`length`** function is *invoked* with the *dot notation*:

```
int n = name.length();
```

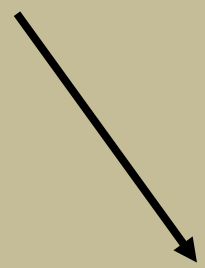
# String Functions

- Once you have a string, you can extract substrings by using the **substr** member function.
- **`s.substr(start, length)`**
  returns a **string** that is made from the characters in the **string s**, starting at character **start**, and containing **length** characters. (**start** and **length** are integer values).

```
string greeting = "Hello, World!";
string sub = greeting.substr(0, 5);
    // sub contains "Hello"
```

# String Functions

starting at character 0 ?



```
string sub = greeting.substr(0, 5);
```

# String Functions

```
string greeting = "Hello, World!";
string w = greeting.substr(7, 5);
    // w contains "World" (not the !)
```

**"World"** is 5 characters long but…

why is 7 the position of the “W” in **"World"**?

0 ?

# String Functions

| H | e | l | l | o | , | | W | o | r | l | d | ! |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

In most computer languages, the starting position 0 means "start at the beginning."

The first position in a string is labeled 0, the second one 1, and so on. And don't forget to count the space character after the comma—<u>but the quotation marks are **not** stored</u>.

# String Functions

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **H** | **e** | **l** | **l** | **o** | **,** | | **W** | **o** | **r** | **l** | **d** | **!** |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |

The position number of the last character
is always one less than the length of the string

The **`!`** is at position 12 in **`"Hello, World!"`**.
The length of **`"Hello, World!"`** is 13.

(C++ remembers to count the 0 as one of the
positions when counting characters in strings.)

# String Functions

```
H  e  l  l  o  ,     W  o  r  l  d  !
0  1  2  3  4  5  6  7  8  9 10 11 12
```

```
string greeting = "Hello, World!";
string w = greeting.substr(7);
   // w contains "World!"
```

If you do not specify how many characters to take, you get all the rest.

# String Functions

```
 H  e  l  l  o  ,     W  o  r  l  d  !
 0  1  2  3  4  5  6  7  8  9 10 11 12
```

```
string greeting = "Hello, World!";
string w = greeting.substr();
// w contains "Hello World!"
```

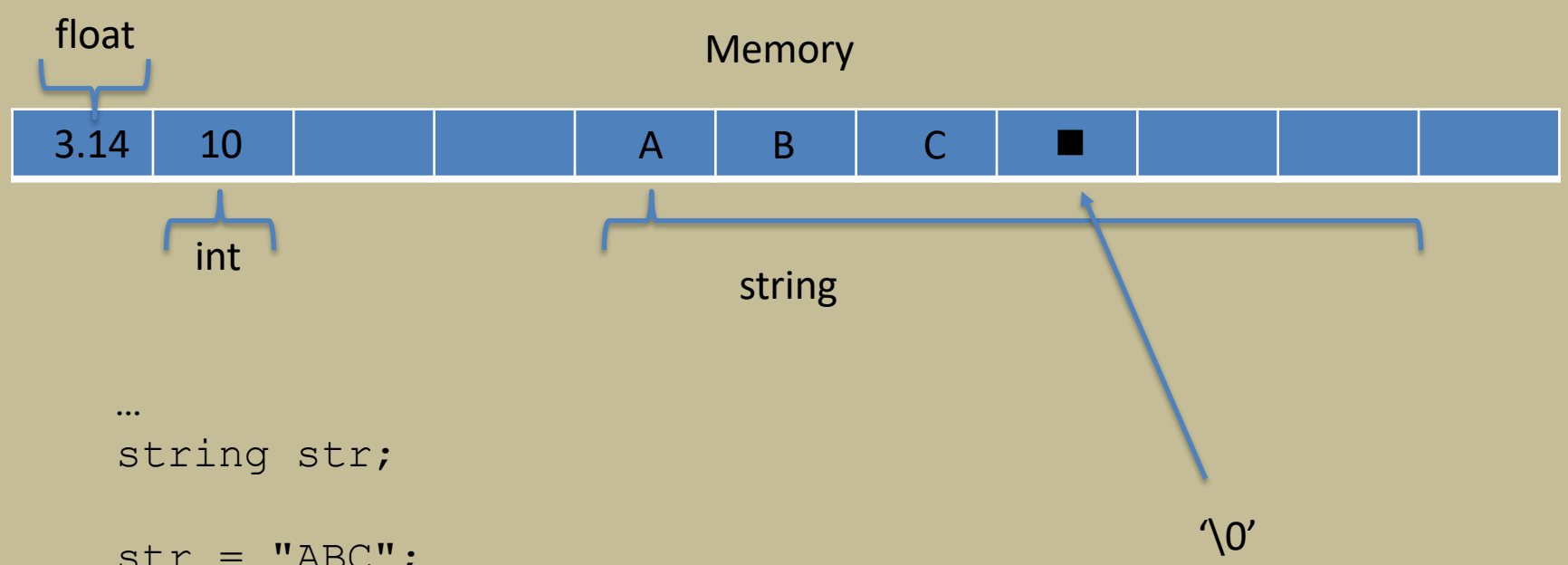
If you omit the starting position and the length, you get all the characters

(not much of *sub*string!)

# Examples

- Cloud9 demo:
  - Rachel_string1.cpp

# Declarations assign names to memory locations



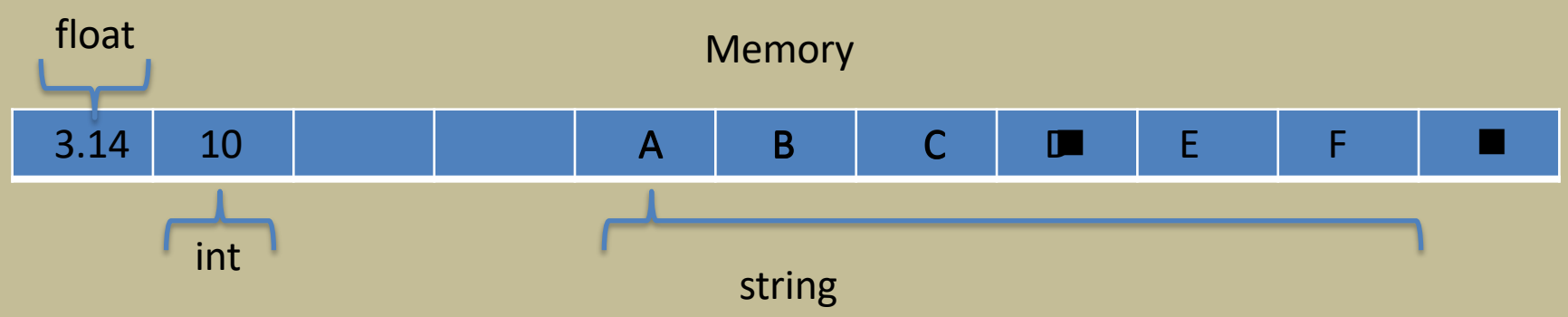

```
…
string str;

str = "ABC";



…
```

# Declarations assign names to memory locations



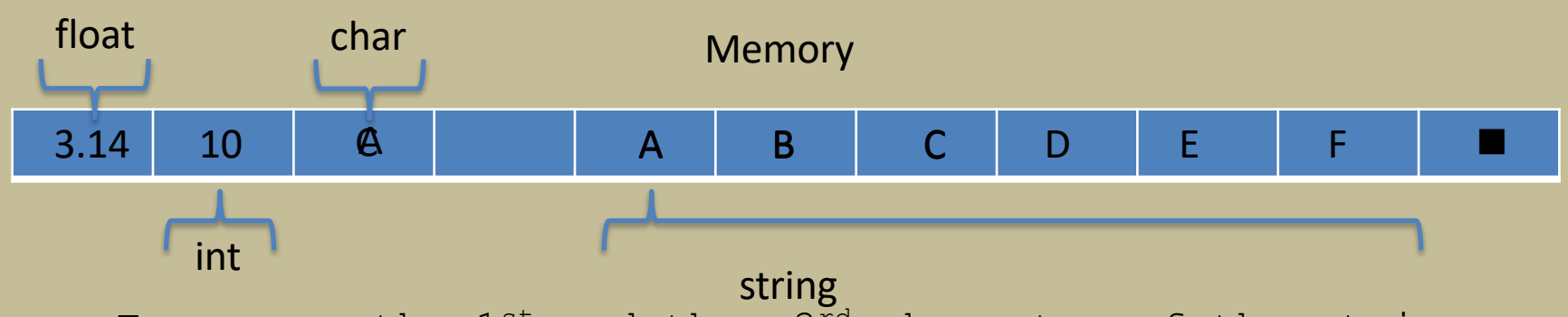

```
…
string str;

str = "ABC";

str = str + "DEF";

…
```

# Declarations assign names to memory locations



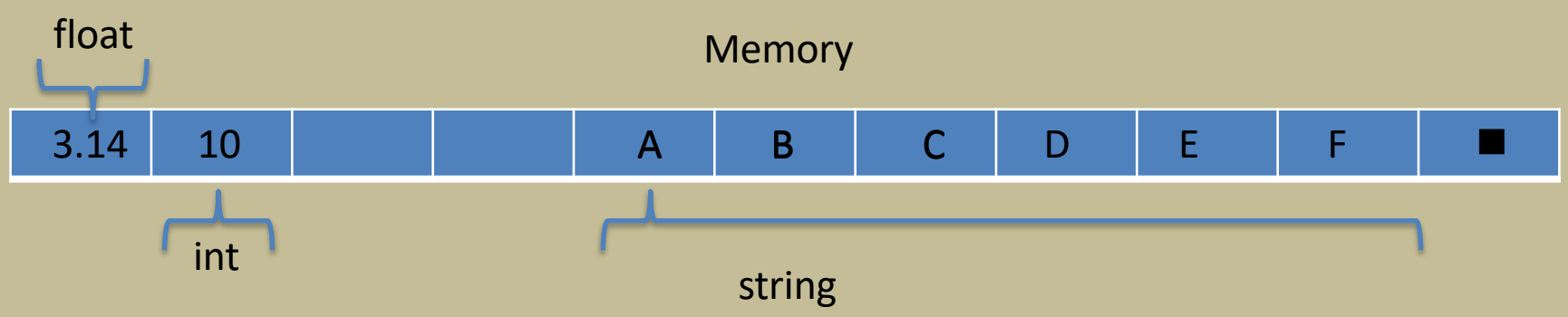

To access the 1st and then 3rd character of the string

```
…
string str;
char c;

str = "ABC";
str = str + "DEF";
c = str[0];
c = str[2];
…
```

The index number specifies how many entries from the beginning to skip

# String Manipulation

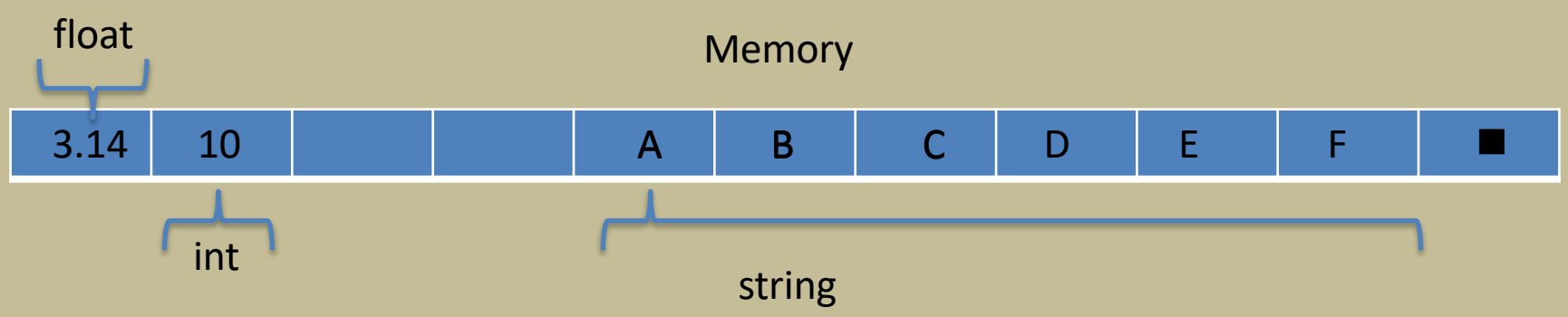



```
To count number of characters in a string
…
string str;
str = "ABC";

int i = 0;
while (str[i] != '\0')
    i = i + 1;
 …
```

Cloud9 demo str_len.cpp

# String Manipulation




```
To count number of characters in a string
…
string str;
str = "ABC";

int i = 0;
while (str[i] != 0)
    i = i + 1;
 …
```

There is already a function built into the string that will give us the number of characters:

```
int n = str.length();
```

# String Operations

| Statement | Result | Comment |
|---|---|---|
| `string str = "C";`<br>`str = str + "++";` | `str` is set to `"C++"` | When applied to strings, + denotes concatenation. |
| `string str = "C" + "++";` | **Error** | **Error:** You cannot concatenate two string literals. |
| `cout << "Enter name: ";`<br>`cin >> name;`<br>(User input: Harry Morgan) | `name` contains `"Harry"` | The >> operator places the next word into the `string` variable. |
| `cout << "Enter name: ";`<br>`cin >> name >> last_name;`<br>(User input: Harry Morgan) | `name` contains `"Harry"`, `last_name` contains `"Morgan"` | Use multiple >> operators to read more than one word. |
| `string greeting = "H & S";`<br>`int n = greeting.length();` | `n` is set to 5 | Each space counts as one character. |

# String Operations

| Statement | Result | Comment |
|---|---|---|
| `string str = "Sally";`<br>`string str2 = str.substr(1, 3);` | `str2` is set to `"all"` | Extracts the substring of length 3 starting at position 1. (The initial position is 0.) |
| `string str = "Sally";`<br>`string str2 = str.substr(1);` | `str2` is set to `"ally"` | If you omit the length, all characters from the position until the end are included. |
| `string a = str.substr(0, 1);` | `a` is set to the initial letter in `str` | Extracts the substring of length 1 starting at position 0. |
| `string b = str.substr(`<br>`  str.length() - 1);` | `b` is set to the last letter in `str` | The last letter has position `str.length() - 1`. We need not specify the length. |

# Member Functions of the Standard Class string

**Display 9.7 Member Functions of the Standard Class `string`**

| EXAMPLE | REMARKS |
|---|---|
| **Constructors** | |
| `string str;` | Default constructor; creates empty `string` object `str`. |
| `string str("string");` | Creates a `string` object with data `"string"`. |
| `string str(aString);` | Creates a `string` object `str` that is a copy of `aString`. `aString` is an object of the class `string`. |
| **Element access** | |
| `str[i]` | Returns read/write reference to character in `str` at index `i`. |
| `str.at(i)` | Returns read/write reference to character in `str` at index `i`. |
| `str.substr(position, length)` | Returns the substring of the calling object starting at `position` and having `length` characters. |
| **Assignment/Modifiers** | |
| `str1 = str2;` | Allocates space and initializes it to `str2`'s data, releases memory allocated for `str1`, and sets `str1`'s size to that of `str2`. |
| `str1 += str2;` | Character data of `str2` is concatenated to the end of `str1`; the size is set appropriately. |
| `str.empty( )` | Returns `true` if `str` is an empty `string`; returns `false` otherwise. |

(continued)

# Member Functions of the Standard Class string

**Display 9.7 Member Functions of the Standard Class `string`**

| EXAMPLE | REMARKS |
|---|---|
| `str1 + str2` | Returns a `string` that has `str2`'s data concatenated to the end of `str1`'s data. The size is set appropriately. |
| `str.insert(pos, str2)` | Inserts `str2` into `str` beginning at position `pos`. |
| `str.remove(pos, length)` | Removes substring of size `length`, starting at position `pos`. |
| **Comparisons** | |
| `str1 == str2  str1 != str2` | Compare for equality or inequality; returns a Boolean value. |
| `str1 < str2  str1 > str2` | Four comparisons. All are lexicographical comparisons. |
| `str1 <= str2  str1 >= str2` | |
| `str.find(str1)` | Returns index of the first occurrence of `str1` in `str`. |
| `str.find(str1, pos)` | Returns index of the first occurrence of string `str1` in `str`; the search starts at position `pos`. |
| `str.find_first_of(str1, pos)` | Returns the index of the first instance in `str` of any character in `str1`, starting the search at position `pos`. |
| `str.find_first_not_of (str1, pos)` | Returns the index of the first instance in `str` of any character *not* in `str1`, starting search at position `pos`. |

# String Functions

Cloud 9 demo:

1. Rachel_string2.cpp

# String Functions

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   cout << "Enter your first name: ";
   string first;
   cin >> first;
   cout << "Enter your significant other's first name: ";
   string second;
   cin >> second;
   string initials = first.substr(0, 1)
      + "&" + second.substr(0, 1);
   cout << initials << endl;

   return 0;
}
```

# Comparison Operators

```
string str;
str = "My name is Inigo Montoya.";

if ((str[0] == 'M') && (str[1] == 'y'))
   cout << "First two letters are \'My\' " << endl;
```

# ==, >, <, <=, >= with full strings

```
string str1, str2;
str1 = "My name is Inigo Montoya.";
str2 = "My name is Montoya Inigo.";

if (str1 == str2)
   cout << ”Same name" << endl;
if (str1 < str2)
   cout << ”Larger" << endl;
```

Cloud9 demo comparison.cpp

# ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

# Practice Problem

Read string from the user.  If string is email address display “You have mail!”

- looking for the @ character