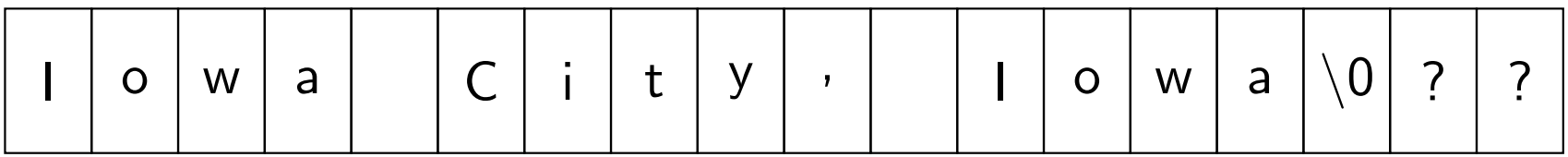
Strings

Class 35

C-Strings

- recall that a C string is an array of char with an embedded null character



C-String Library Functions

- there are several useful functions in the cstring library
  - strlen: the number of characters before the \0
  - strncat: concatenate two strings together
  - strncpy: overwrite one string with another

C-String Library Functions

- there are several useful functions in the cstring library
  - strlen: the number of characters before the \0
  - strncat: concatenate two strings together
  - strncpy: overwrite one string with another

- NEVER use strcpy or strcat as described on pages 569–570!
- they are crazy unsafe
- they do no bounds checking and will happily exceed the limits of the array, clobbering any memory in their way
- they have been the source of famous viruses, worms, and malicious code of all sorts

strcmp

- a very useful function
- `int result = strcmp(string1, string2);` returns
  - zero if the two strings are identical up to the null character
  - a negative if string1 is alphabetically before string2
  - a positive is string1 is alphabetically after string2

strcmp

- a very useful function
- `int result = strcmp(string1, string2);` returns
  - zero if the two strings are identical up to the null character
  - a negative if string1 is alphabetically before string2
  - a positive is string1 is alphabetically after string2

- confusing: when the strings are identical, strcmp returns zero, which normally indicates false

```
if (strcmp("foo", "foo"))
{
  cout << "they do NOT match!" << endl;
}
else
{
  cout << "they DO match!" << endl;
}
```

strcmp

- because of this, the code is normally written like this instead:

```
if (strcmp("foo", "foo") == 0)
{
  cout << "they DO match!" << endl;
}
else
{
  cout << "they do NOT match!" << endl;
}
```

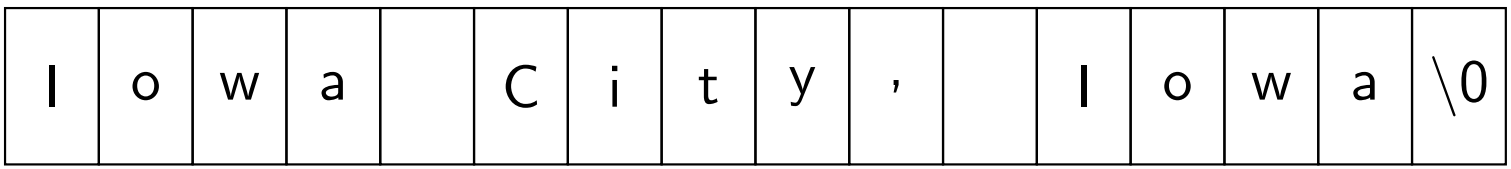
strstr

- strstr: find the location of a substring within a string
- return a pointer to the first character of the substring that matches
- return nullptr if the substring is not found

strstr

- let's say you have a string with a city name followed by a comma and space followed by a state name and are trying to find the state name

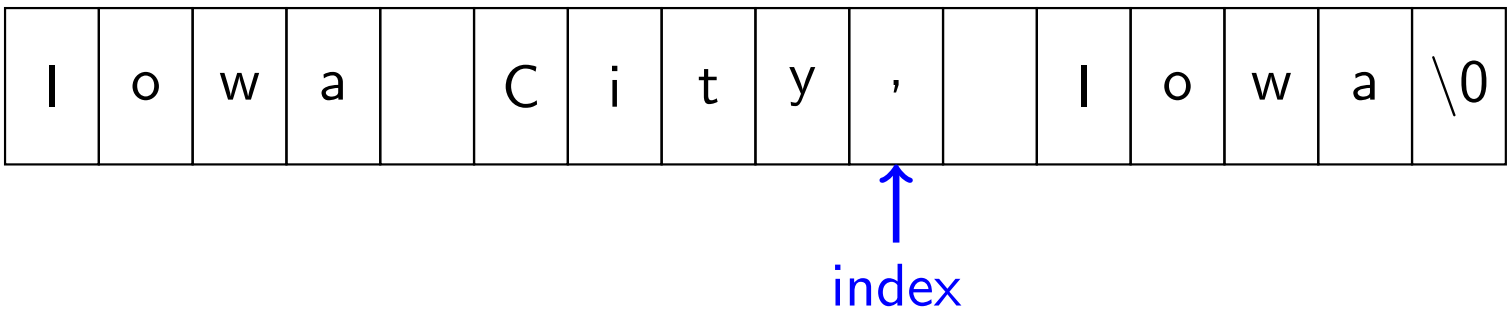
```
char* index = strstr(city_state, ", ");
```



strstr

- let's say you have a string with a city name followed by a comma and space followed by a state name and are trying to find the state name

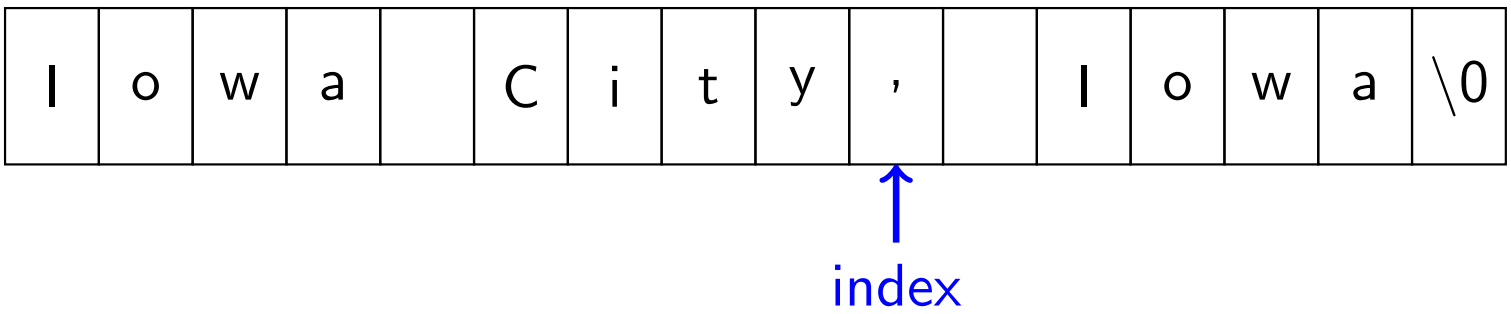
```
char* index = strstr(city_state, ", ");
```



strstr

- let's say you have a string with a city name followed by a comma and space followed by a state name and are trying to find the state name

```
char* index = strstr(city_state, ", ");
cout << (index + 2) << endl;
```



- the output is Iowa

Program 10-6

- this program (page 572) has several features that make it worth looking at
- before we start, note that it has several unacceptable style issues:
  - if statements without braces
  - a break in a for loop
  - int instead of size_t for an index
  - mixed endl and \n

Program 10-6

- this program (page 572) has several features that make it worth looking at
- before we start, note that it has several unacceptable style issues:
  - if statements without braces
  - a break in a for loop
  - int instead of size_t for an index
  - mixed endl and \n
- on line 13, it has a two-dimensional array, an array of arrays of chars
- i.e., an array of strings

- let's do two exercises with this code
- first, draw a picture of how the variable products is actually organized in memory

Program 10-6

- conceptually, there are five rows of twenty-seven characters each
- but physically in memory there are simply 135 contiguous memory locations


T V 3 2 7 3 1 - i n c h T e l e v i
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
s i o n \0 ? ? C D 2 5 7 C D P l a y
1
20 21 22 23 24 25 26 0 1 2 3 4 5 6 7 8 9 10 11 12
e r \0 ? ? ? ? ? ? ? ? ? ? ? T A 6 7 7
2
13 14 15 16 17 18 19 20 21 22 23 24 25 26 0 1 2 3 4 5
A n s w e r i n g M a c h i n e \0 ? ?
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

Program 10-6

- next, convert the for-loop-with-break into a proper while loop

```
for (size_t index = 0; index < NUM_PRODS; index++)
{
  char* str_ptr = strstr(products[index], look_up);
  if (str_ptr != nullptr)
    break;
}
```

Program 10-6

- better code:

```
size_t index = 0;
bool found = false;
char* found_location = nullptr;

while (!found && index < NUM_PRODS)
{
  found_location = strstr(products[index], look_up);
  if (found_location != nullptr)
  {
    found = true;
  }
  else
  {
    index++;
  }
}
```

Converting From C-strings to Numbers

- the cstdlib library contains functions to allow you to convert a C-string to a numeric value
- `int number = atoi("1234");`
  atoi stands for "ASCII to integer"
- `double pi = atof("3.14159");`

Converting From C-strings to Numbers

- the cstdlib library contains functions to allow you to convert a C-string to a numeric value
- `int number = atoi("1234");`
  atoi stands for "ASCII to integer"
- `double pi = atof("3.14159");`

- there is not a standard function to go the other direction
- some compilers have non-standard functions for this, and there are ways to do it, but not a simple built-in function

Arguments to main

- all of this semester's programs are console applications that are text based, not graphical
- they are designed to be run from a terminal
- we are using the Code::Blocks IDE to make life easier, but that is just for development
- for deploying and using the programs, you would use a terminal, not Code::Blocks
- you can run a console program manually in a terminal on Mac, Linux, or Windows
- or click the little green Run arrow in Code::Blocks to get a terminal window where your program runs

Initial Arguments

- a number of our programs have required user input to run
- recall the rectangle area program, for example:

```
$ ./rectangle_area
This program displays the area of rectangles
The widths are between 1 and 75 and the lengths, 1 and 500

How many rectangles to you want? 4
The area of a 40 by 276 rectangle is 11040
The area of a 45 by 400 rectangle is 18000
The area of a 68 by 376 rectangle is 25568
The area of a 28 by 338 rectangle is 9464
```

Initial Arguments

- it is possible in a terminal application to supply this number on the command line without having to prompt for it

```
$ ./rectangle_area_cl 4
This program will show the area of 4 random rectangles

The area of a 57 by 20 rectangle is 1140
The area of a 21 by 348 rectangle is 7308
The area of a 72 by 29 rectangle is 2088
The area of a 2 by 405 rectangle is 810
```

Initial Arguments

- it is possible in a terminal application to supply this number on the command line without having to prompt for it

```
$ ./rectangle_area_cl 4
This program will show the area of 4 random rectangles

The area of a 57 by 20 rectangle is 1140
The area of a 21 by 348 rectangle is 7308
The area of a 72 by 29 rectangle is 2088
The area of a 2 by 405 rectangle is 810
```

Command Line Arguments

- how does this work?
- remember that main can call any other function in your program
- but what calls main?

Command Line Arguments

- how does this work?
- remember that main can call any other function in your program
- but what calls main?

- the operating system calls main for you when you invoke the program, either by clicking the little green arrow or by entering the program's name at the terminal prompt

Command Line Arguments

- how does this work?
- remember that main can call any other function in your program
- but what calls main?

- the operating system calls main for you when you invoke the program, either by clicking the little green arrow or by entering the program's name at the terminal prompt

- when the operating system calls main, it passes the command line arguments to main as actual parameters

main Parameters

- so far, our main function has had no parameters
- but two formal parameters are in fact defined for it
  1. argc: an integer that is the count of arguments
  2. argv: an array of strings, i.e., an array of arrays of characters, which are the arguments themselves as strings
     argv stands for "vector of arguments"

```
int main(int argc, char* argv[])
{ ...
```

main Parameters

```
int main(int argc, char* argv[])
{ ...
```

- argc can be used to determine how many command line arguments were passed to main
- there is always one argument, which is the name of the program itself
- the other arguments (if any) can be addressed with the functions we have discussed above
- the entire array of strings is very similar to the array of strings in Program 10-6

main Parameters

a simple program to echo command line arguments to screen

```
int main(int argc, char* argv[])
{
  for (int index = 0; index < argc; index++)
  {
    cout << index << ": " << argv[index] << endl;
  }
  return 0;
}
```

Doing It Yourself

- if you know how to run programs from the command line in a terminal, you can do this yourself
- if you only know how to run programs from Code::Blocks, you need to tell Code::Blocks what command line arguments you want to use
- to set command line arguments in Code::Blocks, do

  Project $\rightarrow$ Set programs' arguments...

  and then enter the program arguments in the box labeled Program arguments: on the lower half of the box
- enter the arguments in one line, space separated, just as though the program were being run at the terminal

C++ String Functions

- the `string` library has many functions for dealing with C++ strings
- first is a family of functions to convert strings to numbers
- examples are stoi, stol, stoul, stof, stod for converting a string into an int, a long, an unsigned long, and a double, respectively
- there are a number of others
- strangely, there is no stoui, to convert directly from a string to an unsigned integer, so you have to typecast for that

From Number to String

- the to_string function can convert *any* of the numeric types to its string representation, e.g.,

```
string s = to_string(12.34); // returns "12.34"
```

C++ String Operators

- you can concatenate strings with +

```
string message = "Hello ";
message += "world!";
```

C++ String Operators

- you can concatenate strings with +

```
string message = "Hello ";
message += "world!";
```

- you can compare strings with the relops (we've seen this before)

  name1 == name2

  name1 <= name2

C++ String Operators

- you can concatenate strings with +
  ```
  string message = "Hello ";
  message += "world!";
  ```

- you can compare strings with the relops (we've seen this before)
  name1 == name2
  name1 <= name2

- to access a single character from a string, DO NOT use square brackets; use .at()
  ```
  char initial = name.at(0);
  ```

String Methods

- string is a class, so the functions that belong to a string object are called *methods*
- there are many useful string methods

- .length() tells you how many characters are in the string
- .substr() extracts a substring

- the table on page 598 lists two dozen more

npos

- there is a predefined constant `string::npos` which is the largest possible `size_t` value
- npos stands for “no position”
- it is used by methods such as find that return a position, if what is being searched for is not found