

```
• you can use cin to read integers and characters, as well as other simple data types (e.g., float, double)
```

- if you want to read in multiple values, you can use multiple calls to cin or one call with multiple parameters
- example of making multiple calls:

```
int i, j, k;
cout << "enter three numbers: ";
cin >> i;
cin >> j;
cin >> k;
```

- when you put this code in a program and run it, you can enter three numbers separated by *whitespace* (space, tab or return)
- example of making one call with multiple parameters:

```
int i, j, k;
cout << "enter three numbers: ";
cin >> i >> j >> k;
```

• as above, when you put this code in a program and run it, you can enter three numbers separated by *whitespace* (space, tab or return)

```
the std:: class scope notation

. both cout and cin belong to something called the std class
. you may see notation like this: std::cout or std::cin.
the std:: prefix uses something called the scope operator, which is the :: part.
the std:: prefix uses something called the scope operator, which is the :: part.
the std:: prefix uses something called the scope operator, which is the :: part.
the std:: prefix uses something called the scope operator, which is the :: part.
the std:: prefix uses something called the scope operator, which is the :: part.
the std:: prefix uses something called the scope operator, which is the :: part.
the std:: prefix uses something called the scope operator, which is the :: part.
the std:: prefix uses something called the scope operator, which is the :: part.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses something called the scope operator.
the std:: prefix uses somethi
```

- in most cases, std::cout means the same thing as cout and std::cin means the same thing as cin.
- the scope operator is used to disambiguate, just in case there were two things defined using the name cout or the name cin.
- you can think of it like having two people in the class with the same first name. we need their last names to figure out who we are talking about.
- the programs that you write for this class will NOT define anything else called cout or cin, so you don't need to worry about the std:: prefix from that standpoint. however, the textbook uses that notation; and it is good to know anyway.

cisc1110-fall2010-sklar-lecIV.1

• if the user enters *david ortiz* when the program asks please enter your name: then the value of s will be "david"

• HOWEVER, when reading a string variable using the getline() function, the input will stop as soon as the first *newline* character is read (i.e., the user hits the ENTER key on the keyboard), e.g.:

#include <iostream>
#include <string>
using namespace std;
int main() {
 string s;
 cout << "please enter your name:";
 getline(cin, s);
 cout << "s = " << s << endl;
} // end of main()</pre>

 here, if the user enters david ortiz when the program asks please enter your name: then the value of s will be "david ortiz"

cisc1110-fall2010-sklar-lecIV.1

reading strings from the keyboard

• there are two ways to read input values from the keyboard into a string variable: (1) using cin >>

(2) using the getline() function

- the first way, using the >> operator, will only read until the first whitespace character is read (the term "whitespace" refers to characters like blank spaces, tabs and newlines); this means that the input will stop as soon as the first whitespace character is read
- for example:

#include <iostream>
#include <string>
using namespace std;
int main() {
 string s;
 cout << "please enter your name:";
 cin >> s;
 cout << "s = " << s << endl;
} // end of main()</pre>

```
cisc1110-fall2010-sklar-lecIV.1
```

C style strings

- last class, we covered C++ style strings
- today, we'll talk about C style strings
- both types of strings allow you to store multiple characters in a single variable
- the underlying data type is still char, but only one variable name is used, and a *length* is associated with it
- in a C++ style string, you can use the string length() function
- in a C style string, the last character is a *terminator*: '\0' you can also use the constant called NULL
- with both styles, string constants are surrounded by *double* quotes: "
- example:
- string s1 = "ABCDE"; // c++ style string char s2[6] = "FGHIJ"; // c style string

```
cisc1110-fall2010-sklar-lecIV.1
```

 storage of the c style string: char s2[6] = "FGHIJ"; looks like this: FGHIJ"; looks like this: FGHIJ)(0) so with strings, you really only access the values stored at indeces 0 through length − 1, (or size - 2) since the value stored at length is always \0 NOTICE that you have to make your C style string one character longer than you need, to make room for the terminator character so in the example, we wanted to have 5 characters in our string ("FGHIJ"), but we declared the string s2 to be of length 6, in order to have room for the NULL (\0) terminator 	<pre>C string library • in addition to the C++ string library that we talked about last class, there is also a C string library • to use the C string library, include the header in your C++ source file: #include <cstring> • this will give you access to a lot of functions, including the following: strlen() strcmp() strcpy() strncpy()</cstring></pre>
• string length function: int strlen(char *s);	shortcut operators

11

- this function returns the number of characters in ${\bf s};$ note that this is NOT the same thing as the number of characters allocated for the string array
- string comparison function:

int strcmp(const char *s1, const char *s2);

"This function returns an integer greater than, equal to, or less than 0, if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2 respectively. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes that differ in the strings being compared."

• copying functions:

char *strcpy(char *dest, char *source);

- copies characters from source array into dest array up to NULL

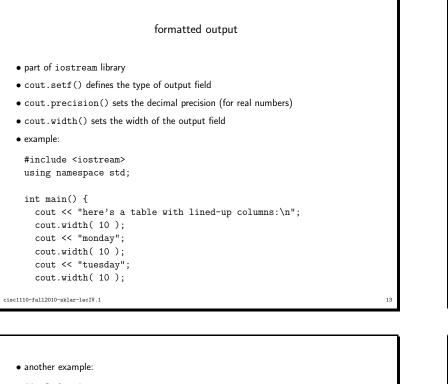
char *strncpy(char *dest, char *source, int num);

 copies characters from source array into dest array; stops after num characters (if no NULL before that); appends NULL also called "increment" and "decrement" operators
increment operator: ++

meaning: add one and assign
example: i++;
is the same as: i = i + 1;

 decrement operator: --meaning: subtract one and assign example: i--; is the same as: i = i - 1;

cisc1110-fall2010-sklar-lecIV.1



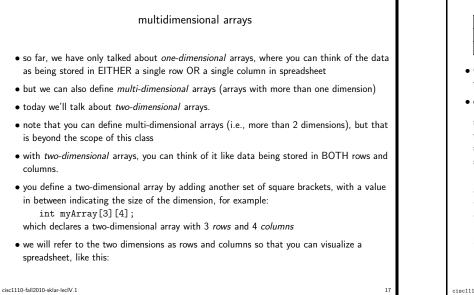
#include <iostream> #include <cmath> using namespace std; int main() { const int A = 5: const double B = 3.4568; double C; cout << "output using fixed precision, 2 decimal places:\n";</pre> cout.setf(ios::fixed); cout.precision(2); cout << "B=" << B << endl:</pre> cout << "output using width=10, left justified:\n";</pre> cout.setf(ios::left); cout.width(10); cout << "B=" << B << endl;</pre> cout << "output using width=10, right justified:\n";</pre> cout.setf(ios::right);

cisc1110-fall2010-sklar-lecIV.1

```
cout << "wednesday";</pre>
    cout << endl;</pre>
    cout.width( 10 );
    cout << "1":
    cout.width( 10 );
    cout << "2";
    cout.width( 10 );
    cout << "3":
    cout << endl;</pre>
 } // end of main()
• output:
 here's a table with lined-up columns:
      monday tuesday wednesday
          1
                      2
                                3
```

output using fixed precision, 2 decimal places: B=3.46 output using width=10, left justified: B= 3.46 output using width=10, right justified: B=3.46 you have to repeat the formatting if you want the same thing again: C=-0.31

```
cisc1110-fall2010-sklar-lecIV.1
```



 $\begin{array}{|c|c|c|c|c|}\hline 3 & 1 & 1 & 9 \\\hline 2 & 5 & 6 & 8 \\\hline 5 & 7 & 4 & 4 \end{array} \ there are 4 columns, going vertically$

 \bullet C++ defines arrays as "row major", which means that the row dimension comes first, then the column dimension

example:

#include <iostream>
using namespace std;
#include <time.h>
#include <stdlib.h>

// declare global variables
const int LENGTH = 3;
const int WIDTH = 4;

int main() {
 int myArray[LENGTH][WIDTH];
 srand(time(NULL));

cisc1110-fall2010-sklar-lecIV.1