UCLA PIC 10 B

Problem Solving using C++ Programming

- **Instructor**: **Ivo Dinov**, Asst. Prof. in Mathematics, Neurology, Statistics
- **Teaching Assistant:** , Suzanne Nezzar, Mathematics

University of California, Los Angeles, Summer 2001

http://www.math.ucla.edu/~dinov/10b.1.011/

UCLA PIC 10 B

Problem Soving using C++ Programming

Course Description,
Class homepage,
online supplements, VOH's etc.

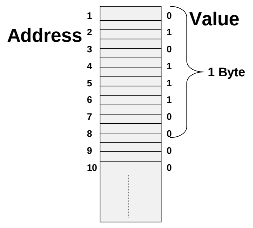
http://www.math.ucla.edu/~dinov/10b.1.011/

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- Chapter 1
  - Computer Systems (hardware and Software components, memory types)
    – PC, Workstation, Mainframe, Network, input/output
    – memory – primary, secondary, fixed, removable
    – CPU
    – Why 8Bits/9Bits = 1Byte?
  - Programming and Problem Solving using Computers
  - Introduction to C++
  - Program Editing, Compiling, Testing, Debugging, Re-designing, and program/algorithm analysis

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- Chapter 2
  - Variables, names, memory addresses, assignments



Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- Chapter 2
  - Variables, names, memory addresses, assignments
  - Standard I/O (input/output)
  - Data types, expressions, arithmetic operators
  - Simple flow of control (conditional statements, loops)
  - Programming style, comments, indenting, headers, naming conventions
  - American Standard Code for Information Interchange (ASCII Character set)

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- Chapter 3
  - Source Code HTML Documentation Generation
  - Top-down Algorithm and implementation design
  - Predefined Functions, header files
  - Procedural Abstraction
  - Local and Global Variables, scope of definition
  - Function, Constructor and Operator Overloading
  - Makefiles. Compile large packages without a builder.
    - Sorter example: This example illustrates the Top-Down design from scratch.

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- **Chapter 4**
  - **- Void type functions**
  - **- Formal parameters, function prototype and function header definitions**
  - **- Precondition and postcondition**
  - **- Function Call-by-value & Call-by-reference**
  - **- Driver (test) programs & Stubs (fake subroutines)**

Review, Monday, June 25, 2001 PIC 10 B

- Course Description
- Online support
- Homework assignments
- Textbook and coverage
- Reviewed Chapters 01-04, PIC 10A

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

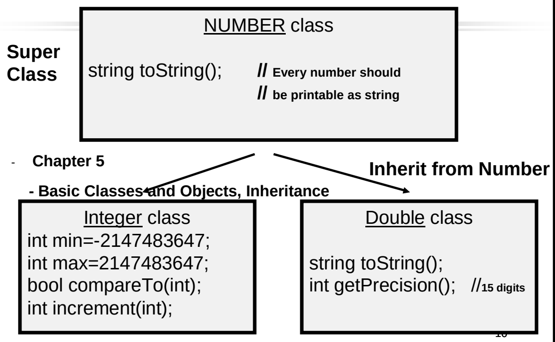
- **Chapter 5**
  - **- Basic File Input/Output, I/O Streams**
    - **- External file names and local stream names**

```
#include <fstream>
ifstream in_stream;
in_stream.open ("infile.dat");
if (in_stream.fail()) {    ...     exit(1);   }
else {  in_stream >> next;  while (next!=eof) { ...  }  }
in_stream.close():
```

- Character I/O; char c=cin.get(); cout.put(c);
endl vs. "\n" vs. '\n'

the **boolean functions** isalpha(char_expr), isdigit(char_expr), etc.

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department



```
string toString();     // Every number should
                       // be printable as string
```

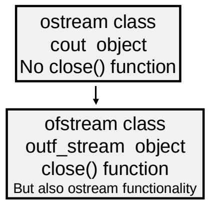
- **Chapter 5**
- **- Basic Classes and Objects, Inheritance**

```
int min=-2147483647;
int max=2147483647;
bool compareTo(int);
int increment(int);
```

```
string toString();
int getPrecision();   //15 digits
```

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- **Chapter 5**
  - **- Basic Classes and Objects, Inheritance**



- void say_hello(ostream& o_stream)
  { o_stream << "Hello!" << endl; }

Function call with diff streams:
say_hello(cout);
say_hello(outFile_stream);
Possible since, ofstream objects <u>ARE</u> also ostream objects.

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- **Chapter 6**

- Structures

```
struct CDAccount                            // structure tag
{       double balance;                     // Public  Member names
        Money cash;                         // Hierarchical Structures
        double interest_rate;
        int computeOneYearInterest();       // Public Method
    private:
        int term;                           //  months until maturity
};
```

- Classes, objects, members (variables and functions)
- Differences between classes and structures
- Inheritance
- String Encoding, Transmission, Decoding Example.

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- Chapter 7
- Extended Flow Control (if-else, which-case-break)
- Truth Tables (&& || !), boolean arithmetic/expressions
- Enumeration type

```
enum MonthLength { JAN_LEN = 31, FEB_LEN = 28,
                   MAR_LEN = 31, APR_LEN = 30,
                   . . .,   NOV_LEN = 30, DEC_LEN};
```

- All kinds of loops (for, do-while, while)
- Debugging nested loops (separate loops and print-test each)
- Loop termination criteria (List Headed by fixed-size; Ask before iterating; List ended by sentinel value; Running out of input (End of file).)
- Most common problems with loops (off-by-one error, infinite loops)

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- Chapter 9 & 8
- Declaring and Referencing Arrays
- Why use arrays? Why sort arrays?
- Arrays in Memory
  adr(a[k]) = adr(a[0]) + k*ElementByteSize
- Initializing Arrays
- Entire arrays (or elements-of-arrays) in Function calls
- The const Parameter Modifier
- Array descriptors: name, size, type, scope
- Operator overloading for ADT's
- Arrays of Classes (arrays of objects)
- Classes of arrays (classes having arrays as members)

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- Chapter 9 & 8
- **Friend Functions**
  Are ordinary functions and NOT members function. You do not use the dot operator to call a friend function, and you do not use a type qualifier (::) in the definition of a friend function
- **Overloading (Unary & Binary) Operations**
- **Overloading +, ==, >> and <<**

Money total, cost, tax; …
if (cost == tax) total = cost + tax;

E.g.,

```
Money operator+(const Money & lhs, const Money & rhs)
{  Money temp;
   temp.all_cents = lhs.all_cents + rhs.all_cents;
   return temp;
}
```

Review PIC 10 A, Problem Solving using C++ Spring 2001, UCLA, Mathematics Department

- What's Next?
- PIC 10 B (Intermediate Programming, C++)
  Dynamic data structures, including linked lists, stacks, queues, trees, and hash tables;
  applications;
  object-oriented programming and software reuse;
  recursion; algorithms for sorting and searching.

- PIC 10 C (Advanced Programming, C++)
  More advanced algorithms and data structuring techniques;
  additional emphasis on algorithmic efficiency;
  advanced features of C++, such as inheritance and virtual functions;
  graph algorithms.

Chapter 10

Strings and Multidimensional Arrays

10 Strings and Multidimensional Arrays

- String Basics
  - cstring Values, cstring Variables, and ANSI C++ string class
  - Predefined cstring Functions
  - Defining cstring Funcitons
  - cstring input and output
  - cstring-to-Number Conversions and Robust Input
- Multidimensional Arrays
  - Multidimensional Array Basics
  - Arrays of cstrings
- The C++ Standard string class
  - Interface for the Standard string class
  - Arrays of string revisited
  - Namespaces Revisited

10 Strings and Multidimensional Arrays

- We will refer to strings we have dealt with so far as cstrings (character arrays). ANSI C++ Library provides a string class which is introduced in this chapter. We refer to these as simply strings.
- In this chapter we study the ANSI C++ string class and arrays with more than one index.
- Arrays with more than one index called multidimensional arrays

(e.g., 3D arrays, 4D arrays, etc.)

10.1 String Basics
cstring Values and cstring Variables (1 of 5)

● ANSI C++ Library provides a string class which is introduced in this chapter. We refer to these as strings.

● We will refer to strings we have dealt with so far as cstrings.

● Members of the C++ Standard Library's a class string are declared in the header <string>.

● Technically, cstrings are null terminated char arrays.

● In the example,

char x[ ] = "Enter the input.";

"Enter the input:" is a cstring literal. The variable x is a cstring.

cstring Values and cstring Variables (2 of 5)

● A cstring variable is a partially filled array having base type char

● Any array uses positions having index values 0 through one less than the number used.

● The cstring variable signals the last used position by placing the special character, called the null character '\0' in the array one position beyond the last character of the cstring.

● If we declare and initialize a cstring variable s:

● char s[11] ;

● If s contains "Hi, Mom" then the array elements are:

● s[0] s[1] s[2] s[3] s[4] s[5] s[6] s[7] s[8] s[9] s[10]

H	i	,		M	o	m	!	\0	?	?

● The character '\0' is the sentinel marking the end of the cstring.

cstring Values and cstring Variables (3 of 5)

● It is possible to initialize a cstring variable at declaration:

char my_message[20] = "Hi there.";

● Notice that cstring variables need not fill the entire array.

● The compiler will count the characters in the initializing string and add one for the null character:

char short_string[ ] = "abc"; // 4-element char array

● This is equivalent to

char short_string[4] = "abc";

or

char short_string[4] = {'a', 'b', 'c', '\0'};

● You must leave space for the null character when you specify size.

Review, Tuesday, June 26, 2001
PIC 10 B

- **Reviewed Chapters 05-09, PIC 10A**

  (most importantly Ch. 05, Classes and basic inheritance, Ch. 8, operator overloading, friend functions).
- **cstrings:**

  char my_message[20] = "Hi there.";

  char short_string[ ] = "abc"; // 4-element char array

  char short_string[4] = "abc";

  char short_string[4] = {'a', 'b', 'c', '\0'};

cstring Values and cstring Variables (4 of 5)

● Do not confuse these situations:

char a_string[ ] = "abc"; // Inserts terminator '\0'
char not_a_string[4] = {'a', 'b', 'c'}; // Does not insert '\0'

● These are NOT equivalent.

● The first one of these initializations places the required null terminating character '\0' after the 'a', 'b', and 'c'. The result is a cstring.

● The second leaves space for the '\0' null character, but it does not insert the null character. The result is NOT a cstring.

The Null Character, '\0'

The null character, '\0', is used to mark the end of a cstring that is stored in an array of characters. When an array of characters is used in this way, the array is often called a cstring variable. Although the null character '\0' is written using two symbols, it is a single character that fits in one variable of type *char* or one indexed variable of an array of characters.

Cstring Variable Declaration

A **cstring variable** is the exact same thing as an array of characters, but it is used differently. A cstring variable is declared to be an array of characters in the usual way:

Syntax:

```
char Array_Name[Maximum_Cstring_Size + 1];
```

Example:

```
char my_cstring[11];
```

The + 1 allows for the null character '\0', which terminates any cstring stored in the array. For example, the cstring variable my_cstring in the above example can hold cstrings that are 10 or fewer characters long.

Initializing a Cstring Variable

A cstring variable can be initialized when it is declared, as illustrated by the following example:

```
char your_string[11] = "Do Be Do";
```

Initializing in this way automatically places the null character, '\0', in the array at the end of the cstring specified.

If you omit the number inside the square brackets [], then the cstring variable will be given a size one character longer than the length of the cstring. For example, the following declares my_string to have nine indexed variables (eight for the characters of the cstring "Do Be Do" and one for the null character '\0'):

```
char my_string[] = "Do Be Do";
```

cstring Values and cstring Variables (5 of 5)

- A cstring is an ordinary array with base type char, and may be processed one element at a time:
- This loop will change the cstring, our_string, into a cstring having the same length but with characters all 'X':

```
int index = 0;
while (our_string[index] != '\0')
{
  our_string[index] = 'X';
  index++;
}
```

- In processing cstrings take great care not to overwrite the null character. An array that was a cstring that has its terminating character overwritten is NO LONGER a cstring.
- In the loop above, if our_string has no null terminator, the loop will run off into memory, happily writing on every byte in memory beyond the end of our_string until a byte is found with zero value.

PITFALL: Using = and == with cstrings (1 of 5)

- Values and variables of type cstring when used with = (assignment) and == (comparison for equality) do not behave like built-in (primitive) data types.
- Assigning a value to a cstring in the obvious way is illegal:

  char a_string[10];

  a_string = "hello";   // ILLEGAL
- Initializing at declaration is straight forward:

  char a_string[10] = "DoBeDo";

  The = does not mean assignment, in spite of the appearance.
- In Chapter 11 we will see that in C++, assignment and initialization can have very different behavior.

PITFALL: Using = and == with cstrings (2 of 5)

- Assignment can be done barehanded, an element at a time:

```
char a_string[10] = "Hello";
char b_string[10];
int i = 0;
while(a_string[i] != '\0')
 {   b_string[i] = a_string[i];
      i++;
 }
     b_string[i] = '\0';
```

- There is a predefined function, strcpy, to assign cstrings:

```
char a_string[10] = "Hello";
char b_string[10];
```

PITFALL: Using = and == with cstrings (3 of 5)

- Comparision of cstrings cannot be done with the == operator. The attempt to compare cstrings with == compiles, it does not get the results you expect.
- Array names carry the address value of the first array element. The result of using == depends on where in memory the cstrings are stored.
- Use of the predefined comparison function strcmp.

```
char a_string[10] = "aeolean";
char b_string[10] = "aeonian";
if (strcmp(b_string, a_string))
  cout << "The strings are NOT the same.";
else
  cout << "The strings are the same.";
```

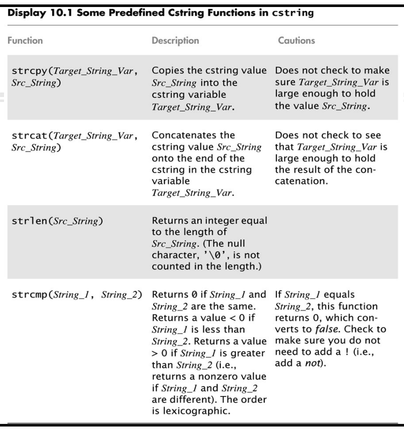
- The strcmp function is declared in the <cstring> header.

PITFALL:
Using = and == with cstrings (4 of 5)

- strcmp compares cstrings is in lexicographic order:
- For successive values of i starting at 0, cstring1[i] and cstring2[i] are compared:
  - If the characters are different:
    - If cstring1[i] < cstring2[i] strcmp returns a negative number.
    - If cstring1[i] > cstring2[i] , strcmp returns a positive number.
    - The number may be -1 or +1, or the difference of the encoding (cstring1[i] - cstring2[i]), or some other value.
      The actual value returned depends on the implemenation.
      Do not write code that depends on the value returned.
    - Testing then stops.
  - If the cstrings are equal up to the end of one of them, the value returned indicates the longer string is greater than the shorter string.
  - If the strings are <u>equal in length</u> and <u>have the same characters</u>, the strings are <u>equal</u>.

Predefined cstring Functions (5 of 5)

- Display 10.1 (next slide) contains a few of the functions from the cstring library.
- You must #include <cstring> to gain access to these functions.
- strcpy(target, source) replaces target with source. Be sure there is enough space in target to hold all of source.
- strcat(target, source) <u>appends</u> source to target. The first character of source is copied into the null terminator of target, and all successive characters of source are copied into target. Be sure there is enough space in target for all of source's characters, including source's null terminator.
- strlen(source) returns the number of characters up to but not including the null terminator.
- strcmp(str1, str2) We discussed this in an earlier slide. Refer to Display 10.1 for detail.



Display 10.1 Some Predefined Cstring Functions in `cstring`

Function	Description	Cautions
`strcpy(`*Target_String_Var*`,` *Src_String*`)`	Copies the cstring value *Src_String* into the cstring variable *Target_String_Var*.	Does not check to make sure *Target_String_Var* is large enough to hold the value *Src_String*.
`strcat(`*Target_String_Var*`,` *Src_String*`)`	Concatenates the cstring value *Src_String* onto the end of the cstring in the cstring variable *Target_String_Var*.	Does not check to see that *Target_String_Var* is large enough to hold the result of the concatenation.
`strlen(`*Src_String*`)`	Returns an integer equal to the length of *Src_String*. (The null character, `'\0'`, is not counted in the length.)	
`strcmp(`*String_1*`,` *String_2*`)`	Returns `0` if *String_1* and *String_2* are the same. Returns a value < 0 if *String_1* is less than *String_2*. Returns a value > 0 if *String_1* is greater than *String_2* (i.e., returns a nonzero value if *String_1* and *String_2* are different). The order is lexicographic.	If *String_1* equals *String_2*, this function returns 0, which converts to *false*. Check to make sure you do not need to add a ! (i.e., add a *not*).

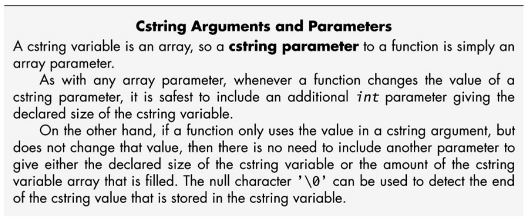
PITFALL:
Dangers in Using Functions from `<cstring>`

- There is a very real danger associated with the functions strcpy and strcat.
- Both these functions copy characters until a null character is found in the source string, without regard to whether space is available in the target.
- If there is no space in the target, strcpy and strcat will happily overwrite any variables in memory beyond the target array.
- This may be some of your variables, or it could be something that your system depends on to run correctly.
- There could be no effect what so ever.
- There could be a segmentation violation or illegal operation error, with your program crashing, and no further problems.
- The operating system could crash and burn.
- Nothing apparent may happen. But the next application started could crash and burn on loading. So, be careful!

Defining cstring functions

- The strcpy and strcat functions have problems.
- The Standard Library defines versions that have an additional parameter that can avoid some of these problems.
- To learn to write safe cstring functions, we write a string_copy function with an additional parameter to make the function safer.
- The added parameter takes an argument that is the declared size of the target argument.

617



Cstring Arguments and Parameters

A cstring variable is an array, so a **cstring parameter** to a function is simply an array parameter.

As with any array parameter, whenever a function changes the value of a cstring parameter, it is safest to include an additional *`int`* parameter giving the declared size of the cstring variable.

On the other hand, if a function only uses the value in a cstring argument, but does not change that value, then there is no need to include another parameter to give either the declared size of the cstring variable or the amount of the cstring variable array that is filled. The null character `'\0'` can be used to detect the end of the cstring value that is stored in the cstring variable.

Display 10.2 The function string_copy (1 of 2)

```
//      Program to demonstrate the function string_copy
#include <iostream>
#include <cstring>

void string_copy(char target[ ], const char source[ ], int target_size);
//    Precondition: target_size is the declared size of the cstring variable target.
//        The array source contains a cstring value terminated with '\0'.
//    Postcondition: The value of target has been set to the cstring value in source,
//        provided the declared size of target is large enough. If target is not large
//        enough to hold the entire cstring, a cstring equal to as much of the value of
//        source as will fit is stored in target.

int main( )
{
    using namespace std;
    char short_string[11]; //Can hold cstrings of up to 10 characters.
    string_copy(short_string, "Hello", 11);
    cout << short_string << "STRING ENDS HERE.\n";

    char long_string[ ] = "This is rather long.";
    string_copy(short_string, long_string, 11);
    cout << short_string << "STRING ENDS HERE.\n";
    return 0;
}
```

Display 10.2 The fucntion string_copy (2 of 2)

```
//Uses header file cstring or string.h:
void string_copy(char target[ ], const char source[ ], int target_size)
{
     using namespace std;
    int new_length = strlen(source);
    if (new_length > (target_size - 1))
        new_length = target_size - 1;      //    That is all that will fit.
    int index;
    for (index = 0; index < new_length; index++)
    {        target[index] = source[index];  }
    target[index] = '\0';
}
```

cstring Input and Output (1 of 3)

- **cstrings may be output using the <u>insertion</u> operator <<**
  cout << short_string << "STRING ENDS HERE.\n";
- **cstrings may receive input using the <u>extraction</u> operator >>**
  cin >> short_string >> some_other_string;
- **HOWEVER: Remember that extraction ignores all white space, and that extraction from istream objects stops at whitespace.**
- **Whitespace is blanks, tabs, and line breaks.**
- **The code**

```
char a[80], b[80];
cin >> a >> b ;
cout << a << b << "END OF OUTPUT.\n";
```

produces a dialog like:

```
Do be do to you!
DobeEND OF OUTPUT.
```

cstring Input and Output (2 of 3)

- **To get an entire line, you can write a loop to extract the line a word at a time, but this won't read the blanks.**
- **To get an entire line, you can use the predefined member getline.**
- **getline has two arguments: a cstring and a number of characters to extract to the cstring, allowing for the null terminator.**
- **Typically this is the declared size of the variable**

Example: This code

```
char a[80];
cin.getline(a, 80);
cout << a  << "END OF OUTPUT.\n";
```

produces a dialog like:

```
Do be do to you!
Do be do to you!END OF OUTPUT.
```

cstring Input and Output (3 of 3)

- **The getline member function stops reading when a number of characters equal to the second argument have been read:**

Example: This code

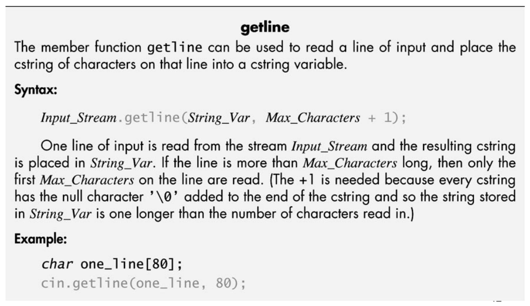
```
char a[80];
cin.getline(a, 5);
cout << a  << "END OF OUTPUT.\n";
```

produces a dialog like:

```
Dobedo to you!
DobeEND OF OUTPUT.
```

- **These cstring i/o techniques work the same for file i/o:**
- **If in_stream has been declared and connected to a file, this code will input 79 or fewer characters (up to the end of line) into cstring variable a.**

```
char a[80];
in_stream.getline(a, 80);
```



getline

The member function `getline` can be used to read a line of input and place the cstring of characters on that line into a cstring variable.

Syntax:

```
Input_Stream.getline(String_Var, Max_Characters + 1);
```

One line of input is read from the stream *Input_Stream* and the resulting cstring is placed in *String_Var*. If the line is more than *Max_Characters* long, then only the first *Max_Characters* on the line are read. (The +1 is needed because every cstring has the null character `'\0'` added to the end of the cstring and so the string stored in *String_Var* is one longer than the number of characters read in.)

Example:

```
char one_line[80];
cin.getline(one_line, 80);
```

cstring-to-number Conversions and Robust Input (1 of 3)

- **'1', "1" and 1 are all different.**
- **1 is a int constant, also called a literal.**
- **'1' is a char constant. It occupies one byte and is represented by some encoding. In C++ the value is the ASCII encoding, which has the decimal value 49. Recall we talked about ASCII encoding in PIC10A.**

  (There is a new encoding called unicode characters. The C++ type that holds unicode is wchar_t.)
- **"1" is a cstring constant. It occupies two bytes, one for the encoding of the character 1 and one for the null terminator.**
- **In a program in any language, you cannot ignore the difference between these objects.**
- **Robust numeric input may be written by inputting a cstring, extracting the digit characters and building the number from the digits.**

43

cstring-to-number Conversions and Robust Input (2 of 3)

- **Once you have a cstring containing the digits that represent an `int`, use the predefined function atoi**
- **atoi is named and pronounced: Ascii TO Integer)**
- **atoi takes a cstring argument and returns the int value represented by the digit characters in cstring.**
- **atoi returns 0 if the cstring contains a non-digit character. Example: atoi("#37") returns 0.**
- **The atoi function is declared in the <cstdlib> header.**
- **Display 10.3 has two utility functions:**
  - **read_and_clean that inputs a string, ignoring any non-digits entered.**
  - **new_line that discards all input remaining on the line.**

44

cstring-to-number Conversions and Robust Input (3 of 3)

- **The function atof is named and pronounced Ascii TO Floating point.**
- **atof is similar to atoi. It converts its cstring argument to the double value the cstring represents. Like atoi, the function atof returns 0.0 if the cstring argument does not represent to a double.**
- **Display 10.3 demonstrates read_and_clean, and Display 10.4 is demonstrates Robust Input Functions.**

45

Display 10.3 cstrings to Integers (1 of 3)

```
//      Demonstrates the function read_and_clean.
#include <iostream>
#include <cstdlib>
#include <cctype>
void read_and_clean(int& n);
//      Reads a line of input. Discards all symbols except the digits. Converts
//       the cstring to an integer and sets n equal to the value of this integer.
void new_line( );
//      Discards all the input remaining on the current input line.
//     Also discards the '\n' at the end of the line.
int main( )
{   using namespace std;
    int n;
    char ans;
    do
    {   cout << "Enter an integer and press return: ";
        read_and_clean(n);
        cout << "That string converts to the integer " << n << endl;
        cout << "Again? (yes/no): ";
        cin >> ans;
        new_line( );
    } while ( (ans != 'n') && (ans != 'N') );
    return 0;
}
```

46

Display 10.3 cstrings to Integers (2 of 3)

```
//    Uses iostream, cstdlib, and cctype:
void read_and_clean(int& n)
{   using namespace std;
    const int ARRAY_SIZE = 6;
    char digit_string[ARRAY_SIZE];

    char next;
    cin.get(next);
    int index = 0;
    while (next != '\n')
    {   if ( (isdigit(next)) && (index < ARRAY_SIZE - 1) )
         {
            digit_string[index] = next;
            index++;
         }
        cin.get(next);
    }
    digit_string[index] = '\0';

    n = atoi(digit_string);
}
```

47

Display 10.3 cstrings to Integers (3 of 3)

```
//      Uses iostream:
void new_line( )
{
    using namespace std;
    char symbol;
    do
    {
        cin.get(symbol);
    } while (symbol != '\n');
}
```

48

Display 10.4 Robust Input Functon (1 of 4)

```
//      Demonstration program for improved version of get_int.
#include <iostream>
#include <cstdlib>
#include <cctype>

void read_and_clean(int& n);
//    Reads a line of input. Discards all symbols except the digits. Converts
//    the cstring to an integer and sets n equal to the value of this integer.

void new_line( );
//    Discards all the input remaining on the current input line.
//    Also discards the '\n' at the end of the line.

void get_int(int& input_number);
//    Gives input_number a value that the user approves of.

int main( )
{   using namespace std;
    int input_number;
    get_int(input_number);
    cout << "Final value read in = " << input_number << endl;

    return 0;
}
```

Display 10.4 Robust Input Functon (2 of 4)

```
//      Uses iostream and read_and_clean:
void get_int(int& input_number)
{   using namespace std;
    char ans;
    do
    {   cout << "Enter input number: ";
        read_and_clean(input_number);
        cout << "You entered " << input_number
             << " Is that correct? (yes/no): ";
        cin >> ans;
        new_line( );
    } while ((ans != 'y') && (ans != 'Y'));
}
```

Display 10.4 Robust Input Functon (3 of 4)

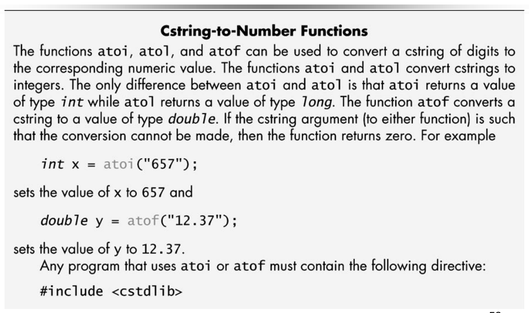
```
//     Uses iostream, cstdlib, and cctype:
void read_and_clean(int& n)
{
    using namespace std;
    const int ARRAY_SIZE = 6;
    char digit_string[ARRAY_SIZE];
    char next;
    cin.get(next);
    int index = 0;

    while (next != '\n')
    {   if ( (isdigit(next)) && (index < ARRAY_SIZE - 1) )
        {
            digit_string[index] = next;
            index++;
        }
        cin.get(next);
    }
    digit_string[index] = '\0';

    n = atoi(digit_string);
}
```

Display 10.4 Robust Input Functon (4 of 4)

```
//Uses iostream:
void new_line( )
{
    using namespace std;
    char symbol;
    do
    {   cin.get(symbol);
    } while (symbol != '\n');
}
```



Cstring-to-Number Functions

The functions `atoi`, `atol`, and `atof` can be used to convert a cstring of digits to the corresponding numeric value. The functions `atoi` and `atol` convert cstrings to integers. The only difference between `atoi` and `atol` is that `atoi` returns a value of type *int* while `atol` returns a value of type *long*. The function `atof` converts a cstring to a value of type *double*. If the cstring argument (to either function) is such that the conversion cannot be made, then the function returns zero. For example

```
int x = atoi("657");
```

sets the value of x to 657 and

```
double y = atof("12.37");
```

sets the value of y to 12.37.

Any program that uses `atoi` or `atof` must contain the following directive:

```
#include <cstdlib>
```

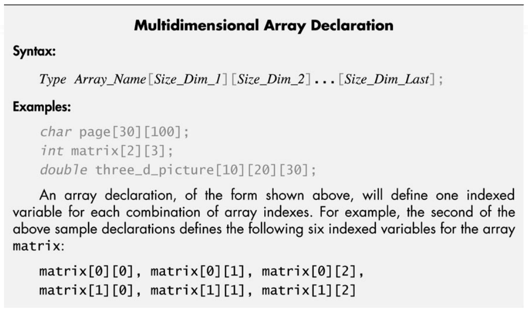
10.2 Multidimensional Arrays

Multidimensional Array Basics (1 of 2)

- **It is useful to have an array with more than one index. In C++, this is implemented using an array with an array type as base type.**
- **Such an array is declared as following:**
  char page[30][100];
- **There are 30*100 indexed variables for this array. The indexed variables for this array are:**

```
Row     page[0][0],   page[0][0], . . . .   page[0][99]
index   page[1][0],   page[1][1], . . . .   page[1][99]
[k][ ]  page[2][0],   page[2][1], . . . .   page[2][99]
            :             :                     :
            :             :                     :
        page[29][0], page[29][1], . . .   page[29][99]

              Column index [ ][k]
```



Multidimensional Array Declaration

Syntax:

Type Array_Name[*Size_Dim_1*][*Size_Dim_2*]...[*Size_Dim_Last*];

Examples:

```
char page[30][100];
int matrix[2][3];
double three_d_picture[10][20][30];
```

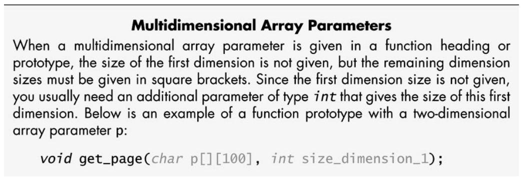
An array declaration, of the form shown above, will define one indexed variable for each combination of array indexes. For example, the second of the above sample declarations defines the following six indexed variables for the array `matrix`:

```
matrix[0][0], matrix[0][1], matrix[0][2],
matrix[1][0], matrix[1][1], matrix[1][2]
```

Multidimensional Array Basics (2 of 2)

- We said that a two-dimensional array is an array with a base type that is an array type. In other words, two-dimensional array is an array of arrays.
- The array
  char page[30][100];
  is a one dimensional array of size 30, whose base type is an array of size 100 with base type char.
- Each entry in the array of size 30 is an array of char of size 100.
- Most of the time the programmer can treat a two-dimensional array as if it were an array with two indices.
- There are two situations where being an arrays of arrays is evident:
- One is when a function with an array parameter for a two dimensional array:
  void display(const char p[ ][100], int size);
- With a two-dimensional array parameter the first dimension is ignored even if specified, *and the compiler does not use it*. This necessitates a size parameter.
- This makes sense if you think of the multidimensional array parameter as an array of arrays. The first dimension is the index, the rest describe the base type.
- With a higher-dimension array parameters the first dimension is (usually) not specified, but all the rest of the dimensions must be specified.

630



Multidimensional Array Parameters

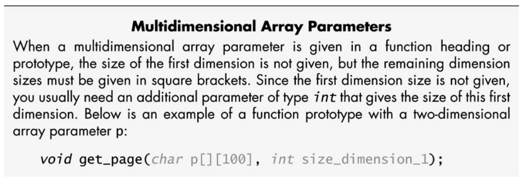
When a multidimensional array parameter is given in a function heading or prototype, the size of the first dimension is not given, but the remaining dimension sizes must be given in square brackets. Since the first dimension size is not given, you usually need an additional parameter of type `int` that gives the size of this first dimension. Below is an example of a function prototype with a two-dimensional array parameter p:

```
void get_page(char p[][100], int size_dimension_1);
```

A Programming Example
A Two-Dimensional Grading Program.

- Display 10.5 presents a program that uses a two-dimensional array named grade to store then display grade records for a small class.
- The first index designates a student, the second designates a grade.
- The grade of student 4 on quiz 1 is recorded in grade[3][0]
- The program has an array, quiz_ave to hold a list of class averages for each quiz over all student grades in the class, and an array st_ave to hold a list of student averages over the quizes that student has taken.

630



Multidimensional Array Parameters

When a multidimensional array parameter is given in a function heading or prototype, the size of the first dimension is not given, but the remaining dimension sizes must be given in square brackets. Since the first dimension size is not given, you usually need an additional parameter of type `int` that gives the size of this first dimension. Below is an example of a function prototype with a two-dimensional array parameter p:

```
void get_page(char p[][100], int size_dimension_1);
```

Display 10.5 Two-dimensional Array (1 of 5)

```
//    Reads quiz scores for each student into the two-dimensional array grade
//   Computes the average score for each student and the average score for
//     each quiz. Displays the quiz scores and the averages.
#include <iostream>
#include <iomanip>
const int NUMBER_STUDENTS = 4, NUMBER_QUIZZES = 3;
void compute_st_ave(const int grade[][NUMBER_QUIZZES], double st_ave[]);
//   Precondition: Global constant NUMBER_STUDENTS and NUMBER_QUIZZES
//        are the dimensions of the array grade. Each of the indexed variables
//        grade[st_num-1, quiz_num-1] contains the score for student st_num on
//       quiz quiz_num.
//    Postcondition: Each st_ave[st_num-1] contains the average for student number
//         stu_num.
```

Display 10.5 Two-dimensional Array (2 of 5)

```
void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double quiz_ave[ ]);
//   Precondition: Global constant NUMBER_STUDENTS and NUMBER_QUIZZES
//      are the dimensions of the array grade. Each of the indexed variables
//      grade[st_num-1, quiz_num-1] contains the score for student st_num
//       on quiz quiz_num.
//   Postcondition: Each quiz_ave[quiz_num-1] contains the average for quiz
     numbered
//        quiz_num.

void display(const int grade[][NUMBER_QUIZZES],
                         const double st_ave[ ], const double quiz_ave[ ]);
//    Precondition: Global constant NUMBER_STUDENTS and
//          NUMBER_QUIZZES are the dimensions of the array grade.
//          Each of the indexed variables grade[st_num-1, quiz_num-1] contains
//          the score for student st_num on quiz quiz_num. Each st_ave[st_num-1]
//          contains the average for student stu_num. Each quiz_ave[quiz_num-1]
//          contains the average for quiz numbered quiz_num.
//    Postcondition: All the data in grade, st_ave, and quiz_ave have been output.
```

Display 10.5 Two-dimensional Array (3 of 5)

```
int main( )
{
    using namespace std;
    int grade[NUMBER_STUDENTS][NUMBER_QUIZZES];
    double st_ave[NUMBER_STUDENTS];
    double quiz_ave[NUMBER_QUIZZES];

    grade[0][0] = 10; grade[0][1] = 10; grade[0][2] = 10;
    grade[1][0] = 2;  grade[1][1] = 0;  grade[1][2] = 1;
    grade[2][0] = 8;  grade[2][1] = 6;  grade[2][2] = 9;
    grade[3][0] = 8;  grade[3][1] = 4;  grade[3][2] = 10;

    compute_st_ave(grade, st_ave);
    compute_quiz_ave(grade, quiz_ave);
    display(grade, st_ave, quiz_ave);
    return 0;
}
```

Display 10.5 Two-dimensional Array (4 of 5)

```
void compute_st_ave(const int grade[ ][NUMBER_QUIZZES], double st_ave[ ])
{
    for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
    {         //Process one st_num:
        double sum = 0;
        for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
            sum = sum + grade[st_num-1][quiz_num-1];
        //    sum contains the sum of the quiz scores for student number st_num.
        st_ave[st_num-1] = sum/NUMBER_QUIZZES;
        //    Average for student st_num is the value of st_ave[st_num-1]
    }
}
void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double quiz_ave[])
{
    for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
    {         //Process one quiz (for all students):
        double sum = 0;
        for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
            sum = sum + grade[st_num-1][quiz_num-1];
        //    sum contains the sum of all student scores on quiz number quiz_num.
        quiz_ave[quiz_num-1] = sum/NUMBER_STUDENTS;
        //    Average for quiz quiz_num is the value of quiz_ave[quiz_num-1]
    }
}
```

Display 10.5 Two-dimensional Array (5 of 5)

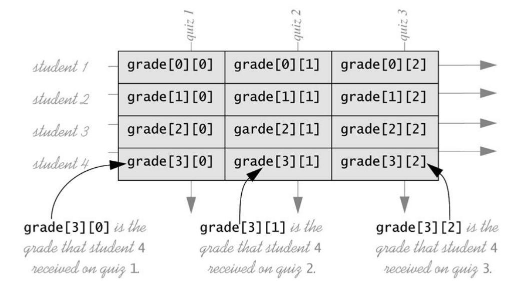
```
//          Uses iostream and iomanip:
void display(const int grade[ ][NUMBER_QUIZZES],
                   const double st_ave[ ], const double quiz_ave[ ])
{   using namespace std;
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(1);

    cout << setw(10) << "Student"
          << setw(5) << "Ave"
          << setw(15) << "Quizzes\n";
    for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
    {   //          Display for one st_num:
        cout << setw(10) << st_num
              << setw(5) << st_ave[st_num-1] << " ";
        for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
            cout << setw(5) << grade[st_num-1][quiz_num-1];
        cout << endl;
    }

    cout << "Quiz averages = ";
    for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
        cout << setw(5) << quiz_ave[quiz_num-1];
    cout << endl;
}
```

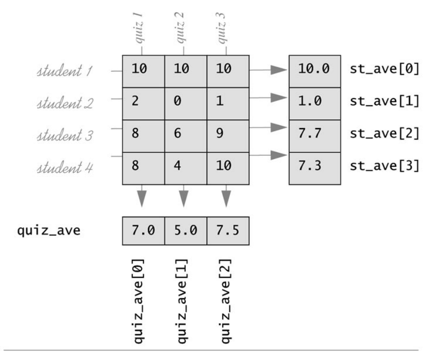
634_01

Display 10.6 The Two-dimensional Array grade

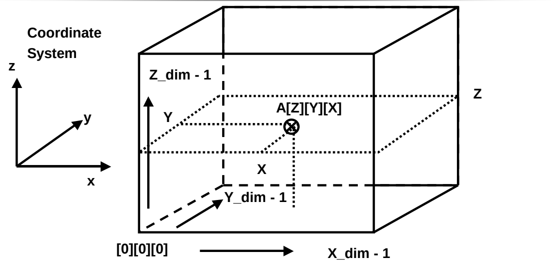


634_02

Display 10.7 The Two-dimensional Array grade



3D Arrays



int A[Z_dim][Y_dim][X_dim];

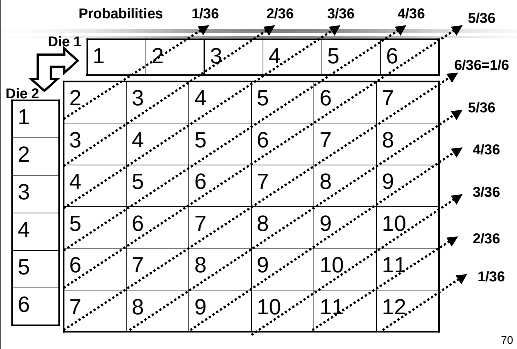
Slowest-varying index Fastest-varying index

3D Arrays – demo LONI_Viz

2 Dice Modeling Program

Write a program that simulates the rolling of two dice.The program should use rand to roll the first die and should use rand again to roll the second die.The sum of the two values should then be calculated. *<u>Note:</u>*Since each die can show an integer value from 1 to 6,then the sum of the two values will vary from 2 to 12,with 7 being the most frequent sum and 2 and 12 being the least frequent sums. The figure below shows the 36 possible combinations of the two dice. Our program should <u>roll the two dice 36,000 times</u>. Use a single-subscripted array to <u>tally the numbers</u> of times each possible sum appears. <u>Print the results</u> in a tabular format. Also, determine if the <u>totals are reasonable</u> (i.e.,there are six ways to roll a 7,so approximately one sixth of all the rolls should be 7).

2 Dice Modeling Program



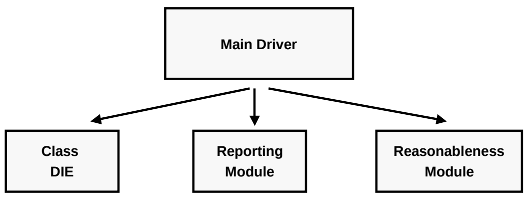
Probabilities
1/36
2/36
3/36
4/36
5/36
6/36=1/6
5/36
4/36
3/36
2/36
1/36

Die 2 \ Die 1	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

2 Dice Modeling Program – Problem Understanding

1. **<u>roll the two dice 36,000 times</u>.**
2. **<u>Tally the numbers</u> of times each possible sum appears.**
3. **<u>Print the results</u> in a tabular format.**
4. **Determine if the <u>totals are reasonable.</u>**

2 Dice Modeling Program – Top-Down Algorithmic Design



2 Dice Modeling Program –
Modular Specifications - Main

Main Driver

-Instantiate two objects of type Die
-Request 36,000 die rolls from both objects
-Tally the observed sums and send them to Reporter-Module for Printing
- Return 0, if all is Okay.

73

2 Dice Modeling Program –
Modular Specifications – Class Die

-Abstract class which allows us to roll any number of times a regular die and obtain the observed values.

Class DIE

-<u>Methods</u>: int rollDie();
-<u>Variables</u>: int die_roll_value;
-<u>Constructors</u>: default Die();
  Non-trivial: Die(int value);

74

2 Dice Modeling Program –
Modular Specifications – Reporting Module

```
void reportModule(int i2, int i3, int i4, . . . , int i12)
{   // Called with observed frequencies of Sums
    // Reporting in tabular form these and the
    // Expected sums for a pair of regular dice

}
```

Reporting Module

75

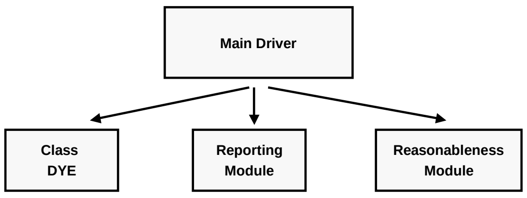
2 Dice Modeling Program –
Modular Specifications – Reasonableness Module

Deals with and discusses how reasonable are the OBSERVED Dice sums and the EXPECTED ones. Are the Dice fair?!?

Reasonableness Module

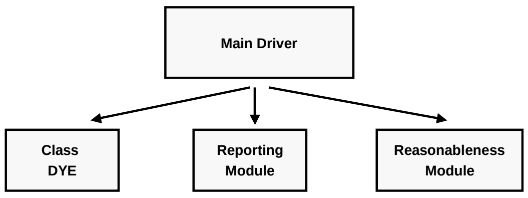
76

2 Dice Modeling Program –
Modular Specifications - Implementation



77

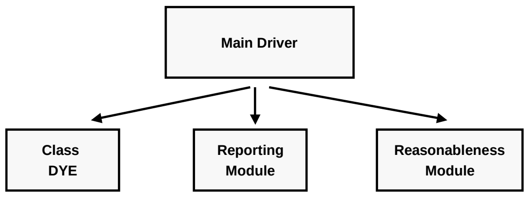
2 Dice Modeling Program –
Modular Specifications - Testing



78

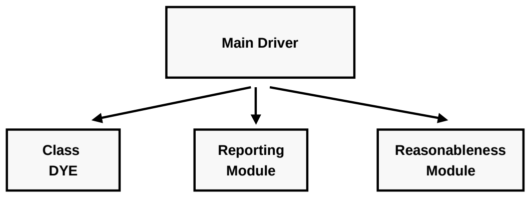
2 Dice Modeling Program –

Modular Specifications – Analysis and ReSpecification



2 Dice Modeling Program –

Modular Specifications



Arrays of cstrings

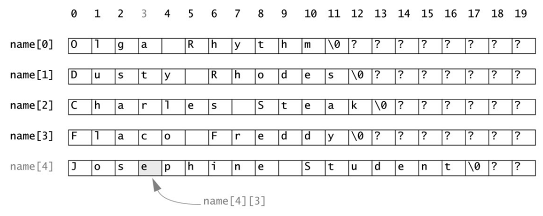
- A cstring is an array of base type char.
- Consequently an array of cstrings is a two-dimensional array of base type char.
- A cstring must hold a null terminator, '\0', so each element of this array of 5 cstrings can hold at most 19 characters:

  char name[5][20];
- Like any array, you can manipulate an array of cstrings by using both index values in nested loops.
- It is nicer to treat the cstrings as entities:

```
cout << "Enter 5 names, one per line:\n ";
for (int index = 0; index < 5; index++)
  cin.getline(name[index], 20);
```
- Output to the screen is also straightforward:

```
for (int index = 0; index < 5; index++)
  cout << name[index] << endl;
```

Display 10.8 An Array of Strings



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
name[0]	O	l	g	a		R	h	y	t	h	m	\0	?	?	?	?	?	?	?	?
name[1]	D	u	s	t	y		R	h	o	d	e	s	\0	?	?	?	?	?	?	?
name[2]	C	h	a	r	l	e	s		S	t	e	a	k	\0	?	?	?	?	?	?
name[3]	F	l	a	c	o		F	r	e	d	d	y	\0	?	?	?	?	?	?	?
name[4]	J	o	s	e	p	h	i	n	e		S	t	u	d	e	n	t	\0	?	?

10.3 The C++ Standard `string` class

- Using cstrings with predefined cstring functions is not as safe as we would like.
- Using strcpy to copy a longer cstring to another (shorter) cstring will overwrite memory that may be important to your program. If you are fortunate, it will be only your program that is the casualty. Your operating system may crash, or someone else's program running on the same system could generate errors.

Interface for the Standard Class string (1 of 4)

- The Standard Library supplied class, string, provides far more utility than the cstrings C++ gets by way of its C heritage.
- Class strings behave very much like built-in data types and are far safer than cstrings.
- Let s1, s2, and s3 be objects of class string, and suppose s1 and s2 have string values. Then + may be used for concatenation:

  s3 = s1 + s2;
- Additional space needed is allocated for s3 automatically.
- The default constructor (???) generates an empty string
- There is a constructor that takes a cstring argument:

```
string phrase, word1("Hello   "), word2("World");
phrase = word1 + word2;
cout << phrase << endl;
```
- The output will be

```
Hello   World
```

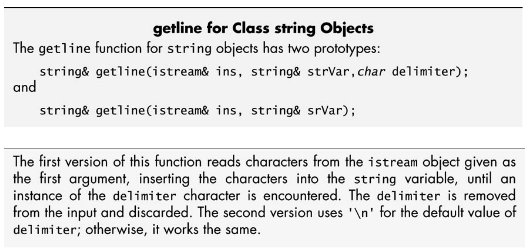
Interface for the Standard Class string (2 of 4)

- You can concatenate one string literal with a class string object:
  string phrase, word1(“Hello“), word2(“World”);
  phrase = word1 + “ “ + word2;
  cout << phrase << endl;
- The output will be
  Hello World
- This works because there is a constructor that converts from cstring to class string objects. C++ sees word1 + “ “, sees a string on the left of + looks for a string on the right of the +. Failing to find it, C++ looks and finds another overloading that has a cstring on the right. Finding such an overloading, it proceeds.
- If such an overloading were not found, C++ would look for a constructor to convert from cstring to string.
- An attempt such as
  phrase = “Hello “ + “World “ + word2;
  fails. The + operator groups from left to right, “Hello “ + “ World“ is done first. This fails because there is no concatenation operator for cstrings.

Interface for the Standard Class string (3 of 4)

- The class string overloads the << (insertion) and >> (extraction) operators with stream left arguments and string right hand arguments with familiar behavior.
- Overloaded >> operator skips leading whitespace and reads nonwhite characters up to the next white space.
- To get an entire line of input for cstrings, we used the getline member of the istream class.
- To get an entire line of input for class string objects, we use a stand alone version of getline.

642


getline for Class string Objects

The getline function for string objects has two prototypes:

```
string& getline(istream& ins, string& strVar,char delimiter);
```
and
```
string& getline(istream& ins, string& srVar);
```

The first version of this function reads characters from the istream object given as the first argument, inserting the characters into the string variable, until an instance of the delimiter character is encountered. The delimiter is removed from the input and discarded. The second version uses '\n' for the default value of delimiter; otherwise, it works the same.

Interface for the Standard Class string (4 of 4)

- Characteristic use of the getline function follow:

```
#include <iostream>
#include <string>
using namespace std;
//. . .
string str1;
getline(cin, str1);
//insert into str1 all input up to ‘\n’
//getline discards the ‘\n’
```

- NOTE THAT class string objects do not range check index values.
- If you want range checked indexing into strings, use the string member function at(int_index).

```
str1.at(9); //Checks index value 9 for legality in str1.
            //If legal,returns the character at index value 9.
```

Pitfall

Code That Depends on Order of Evaluation is Illegal. (1 of 2)

- ANSI C++ does not specify the order of evaluation for terms in an expression. Writing code that depends on the order of evaluation is illegal. Unfortunately, most compilers do not catch this error. The reason is code parallelization and optimization.
- Example: `(a + b) * (c + d)`
- There is no guarantee whether a + b or c + d is evaluated first, nor does it make any difference in this case.
- HOWEVER -- There is a considerable difference here:

```
int i = 0;
cout << i << “ “ << i++ << endl;
// Some compilers evaluate the  expressions i and i++ right
// to left before calling the operator << overloading,
// giving the result  1 0
// A different compiler might give the result 0 1
```

Pitfall

Code That Depends on Order of Evaluation is Illegal. (2 of 2)

- If you need such code, write it so that the sequence of evaluations of the operations can be guaranteed:

```
int i = 0;
cout << i << “ “ ;
i++;
cout << i++ << endl;
// Some compilers evaluate the  expressions i and i++ right
// to left before calling the operator << overloading,
// giving the result  1 0
```

Programming Tip
The Ignore Member Function

- **With cin >> intVariable, everything entered beyond the integer just read in will still be available on the input stream, ready for further extraction. This includes the return key pressed to make the line of data available.**
- **This data will cause the getline function to misbehave.**
- **We presented one fix, the new_line function, remember?**
- **A standard fix is to use the predefined cin member function ignore, whose prototype is**

  istream& ignore(int count, char delimiter);
- **This function will read count characters unless it reads a delimiter character first. All the characters are discarded.**

Pitfall
Mixing `cin >> variable` and `getline` can lose input.

- **Careless mixed use of cin >> variable and getline can lose data in strange ways.**
- **cin >> variable skips leading whitespace and leaves the newline ('\n') character on the input stream.**
- **getline reads everything up to and including the '\n', keeps the data and discards the '\n'.**
- **Use of cin >> variable leaves a '\n' that makes a getline see an empty string.**
- **Use the new_line function from the text or**

  cin.ignore(10000, '\n');

  to discard up to 10,000 characters or up to the newline.

Programming Example
Palindrome Testing (1 of 2)

- **In PIC 10A we had a HW problem that determines whether a string is a palindrome.**
- **A palindrome has the same characters read front to back as it does read back to front. Examples (ignore punctuation, case, and blanks):**

  Able was I ere I saw Elba.

  Madam, I'm Adam.

  Rats live on no evil star.
- **Back then we did not have access to the following string facilities:**

```
string str;          // default constructor - defines empty string
getline(cin, str);   // fetches an entire line of input
isPal(str)           // boolean function that tests for palindrome
```

Programming Example
Palindrome Testing (2 of 2)

- **The isPal funtion defines a string containing the characters we want removed (punctuation and space)**
- **Makes a working copy (Str) of the reference parameter**
- **forces working copy to all lower case**
- **makes a copy (lowerStr) of working string with punctuation removed**
- **returns the results of comparing work reverse(working string)**
- **makeLower cycles through all the characters in its parameter, returning a string all of whose characters has any uppercase characters replaced by corresponding lowercase letters.**
- **removePunct uses the string member function substr and find.**

Display 10.10 Palindrome Testing Program (1 of 5)

```
//          test for palindrome property
#include <iostream>
#include <string>
#include <cctype>
using namespace std;
void swap(char& lhs, char& rhs);
//          swaps char args corresponding to parameters lhs and rhs
string reverse(const string& str);
//          returns a copy of arg corresponding to parameter
//          str with characters in reverse order.
string removePunct(const string& src,
                   const string& punct);
//          returns copy of string src with characters
//          in string punct removed
string makeLower (const string& s);
//          returns a copy of parameter s that has all upper case
//          characters forced to lower case, other characters unchanged.
//          Uses <string>, which provides tolower
bool isPal(const string& this_String);
//          uses makeLower, removePunct.  If  this_String is a palindrome,
//          return true;  else  return false;
```

Display 10.10 Palindrome Testing Program (2 of 5)

```
int main()
{   string str;
    cout << "Enter a candidate for palindrome test "
         << "\nfollowed by pressing return.\n";
    getline(cin, str);
    if (isPal(str))     cout << "\"" << str + "\" is a palindrome ";
    else                cout << "\"" << str + "\" is not a palindrome ";
    cout << endl;
    return 0;
}

void swap(char& lhs, char& rhs)
{   char tmp = lhs;
    lhs = rhs;
    rhs = tmp;
}
```

Display 10.10 Palindrome Testing Program (3 of 5)

```
string reverse(const string& str)
{   int start = 0;
    int end = str.length();
    string tmp(str);

    while (start < end)
    {  end--;
       swap(tmp[start], tmp[end]);
       start++;
    }
    return tmp;
}

//     Returns arg that has all upper case characters forced to lower case,
//     other characters unchanged. makeLower uses <string>, which
//     provides tolower
string makeLower(const string& s)  //uses <cctype>
{   string temp(s); //This creates a working copy of s
    for (int i = 0; i < s.length(); i++)           temp[i] = tolower(s[i]);
    return temp;
}
```

Display 10.10 Palindrome Testing Program (4 of 5)

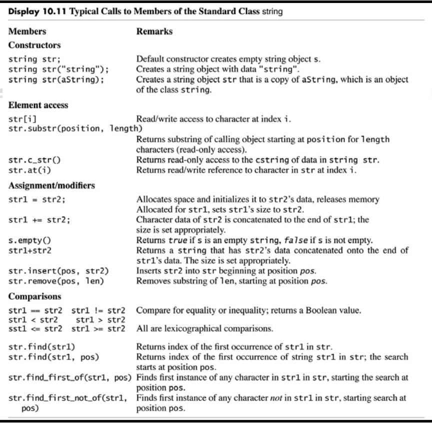
```
//   Returns a copy of src with characters in punct removed
string removePunct(const string& src,  const string& punct)
{
    string no_punct;
    int src_len = src.length();
    int punct_len = punct.length();
    for(int i = 0; i < src_len; i++)
    {    string aChar = src.substr(i,1);
         int location = punct.find(aChar, 0);
         //              find location of successive characters of src in punct
      if (location < 0 || location >= punct_len)
         no_punct = no_punct + aChar; //aChar not in punct -- keep it
    }
    return no_punct;
}
```

Display 10.10 Palindrome Testing Program (5 of 5)

```
//  uses functions makeLower, removePunct.  Returned
//  value: if this_String is a palindrome,  return true;
//  else   return false;
bool isPal(const string& this_String)
{
    string punctuation(",;:.?!'\" ");  //includes a blank
    string str(this_String);
    str = makeLower(str);
    string lowerStr = removePunct(str, punctuation);
    return lowerStr == reverse(lowerStr);
}
```

650

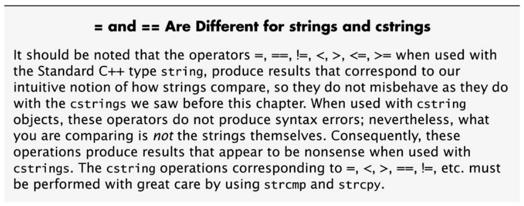
Class String



Display 10.11 Typical Calls to Members of the Standard Class string

Members	Remarks
Constructors	
`string str;`	Default constructor creates empty string object `s`.
`string str("string");`	Creates a string object with data `"string"`.
`string str(aString);`	Creates a string object `str` that is a copy of `aString`, which is an object of the class `string`.
Element access	
`str[i]`	Read/write access to character at index `i`.
`str.substr(position, length)`	Returns substring of calling object starting at `position` for `length` characters (read-only access).
`str.c_str()`	Returns read-only access to the `cstring` of data in `string str`.
`str.at(i)`	Returns read/write reference to character in `str` at index `i`.
Assignment/modifiers	
`str1 = str2;`	Allocates space and initializes it to `str2`'s data, releases memory Allocated for `str1`, sets `str1`'s size to `str2`.
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`; the size is set appropriately.
`s.empty()`	Returns *`true`* if `s` is an empty `string`, *`false`* if `s` is not empty.
`str1+str2`	Returns a `string` that has `str2`'s data concatenated onto 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, len)`	Removes substring of `len`, starting at position *pos*.
Comparisons	
`str1 == str2  str1 != str2`	Compare for equality or inequality; returns a Boolean value.
`str1 < str2   str1 > str2`	
`sst1 <= str2  str1 >= str2`	All are lexicographical comparisons.
`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)`	Finds first instance of any character in `str1` in `str`, starting the search at position `pos`.
`str.find_first_not_of(str1, pos)`	Finds first instance of any character *not* in `str1` in `str`, starting search at position `pos`.

651



= and == Are Different for strings and cstrings

It should be noted that the operators =, ==, !=, <, >, <=, >= when used with the Standard C++ type `string`, produce results that correspond to our intuitive notion of how strings compare, so they do not misbehave as they do with the `cstrings` we saw before this chapter. When used with `cstring` objects, these operators do not produce syntax errors; nevertheless, what you are comparing is *not* the strings themselves. Consequently, these operations produce results that appear to be nonsense when used with `cstrings`. The `cstring` operations corresponding to =, <, >, ==, !=, etc. must be performed with great care by using `strcmp` and `strcpy`.

Arrays of string Revisited

- **Remember, string is a type that acts exactly like any other type.**
- **You can have arrays whose base type is string:**
  string list[20];
- **This is an arrray of 20 string objects.**
- **This array can be filled as follows:**

```
cout << “Enter 20 names, one per line: \n”;
for (int i = 0; i < 20; i++) getline(cin, list[i]);
```

- **Output is the same as for cstrings**

```
    for (int i = 0; i < 20; i++)
      cout << list[i] << endl;
```

Namespaces Revisited

- Display 10.12 is a version of Display 10.10 where we have handled the namespace issues differently.
- Display 10.10 has only one using directive that applies to the entire file: using namespace std;
- In Display 10.12, we keep the scope of the using directives to a single function, and do not place using directives in swap because none is needed there.
- Names in function headers are qualified with std::, as in std::string.

Display 10.12 Careful Namespace Usage (1 of 6)

```
//   test for palindrome property
#include <iostream>
#include <string>
#include <cctype>
void swap(char& lhs, char& rhs);
//   swaps char args corresponding to parameters lhs and rhs
std::string reverse(const std::string& str);
//   returns a copy of arg corresponding to parameter
//   str with characters in reverse order.
std::string removePunct(const std::string& src,  const std::string& punct);
//   returns copy of string src with characters in string punct removed
std::string makeLower (const std::string& s);
//   returns a copy of parameter s that has all upper case
//   characters forced to lower case, other characters unchanged.
//   Uses <string>, which provides tolower
bool isPal(const std::string& this_String);
//   uses makeLower, removePunct.  If this_String is a palindrome,
//    return true;  else return false;
```

Display 10.12 Careful Namespace Usage (2 of 6)

```
int main()
{   using namespace std;
    string str;
    cout << "Enter a candidate for palindrome test "
         << "\n followed by pressing return.\n";
    getline(cin, str);
    if (isPal(str))
         cout << "\"" << str + "\" is a palindrome ";
    else
         cout << "\"" << str + "\" is not a palindrome ";
    cout << endl;
    return 0;
}
```

Display 10.12 Careful Namespace Usage (3 of 6)

```
void swap(char& lhs, char& rhs)
{  char tmp = lhs;
    lhs = rhs;
    rhs = tmp;
}


std::string reverse(const std::string& str)
{   using namespace std;
    int start = 0;
    int end = str.length();
    string tmp(str);
    while (start < end)
    {   end--;
       swap(tmp[start], tmp[end]);
       start++;
    }
    return tmp;
}
```

Display 10.12 Careful Namespace Usage (4 of 6)

```
//  Returns arg that has all upper case characters forced
//  to lower case, other characters unchanged. makeLower
//  uses <string>, which provides tolower
std::string makeLower(const std::string& s)  //uses<cctype>
{
    using namespace std;
    string temp(s); //This creates a working copy of s
    for (int i = 0; i < s.length(); i++)
        temp[i] = tolower(s[i]);
    return temp;
}
```

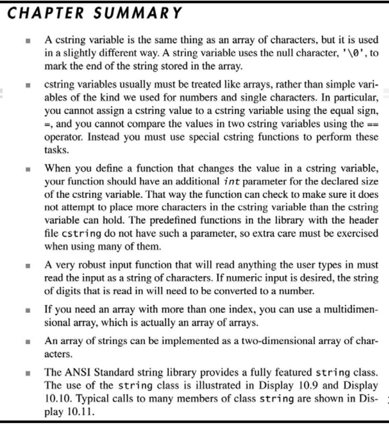
Display 10.12 Careful Namespace Usage (5 of 6)

```
//   Returns a copy of src with characters in punct removed
std::string removePunct(const std::string& src,  const std::string& punct)
{
    using namespace std;
    string no_punct;
    int src_len = src.length();
    int punct_len = punct.length();
    for(int i = 0; i < src_len; i++)
    {    string aChar = src.substr(i,1);
         int location = punct.find(aChar, 0);
         //    find location of successive characters of src in punct
      if (location < 0 || location >= punct_len)
         no_punct = no_punct + aChar; //aChar not in punct -- keep it
    }
    return no_punct;
}
```

Display 10.12 Careful Namespace Usage (6 of 6)

```
//    uses functions makeLower, removePunct.  Returned value:
//    if this_String is a palindrome, return true;  else  return false;
bool isPal(const std::string& this_String)
{
    using namespace std;
    string punctuation(",;:.?!'\" ");     //includes a blank
    string str(this_String);
    str = makeLower(str);
    string lowerStr = removePunct(str, punctuation);
    return lowerStr == reverse(lowerStr);
}
```

657-58



CHAPTER SUMMARY

- A cstring variable is the same thing as an array of characters, but it is used in a slightly different way. A string variable uses the null character, `'\0'`, to mark the end of the string stored in the array.
- cstring variables usually must be treated like arrays, rather than simple variables of the kind we used for numbers and single characters. In particular, you cannot assign a cstring value to a cstring variable using the equal sign, =, and you cannot compare the values in two cstring variables using the == operator. Instead you must use special cstring functions to perform these tasks.
- When you define a function that changes the value in a cstring variable, your function should have an additional *int* parameter for the declared size of the cstring variable. That way the function can check to make sure it does not attempt to place more characters in the cstring variable than the cstring variable can hold. The predefined functions in the library with the header file `cstring` do not have such a parameter, so extra care must be exercised when using many of them.
- A very robust input function that will read anything the user types in must read the input as a string of characters. If numeric input is desired, the string of digits that is read in will need to be converted to a number.
- If you need an array with more than one index, you can use a multidimensional array, which is actually an array of arrays.
- An array of strings can be implemented as a two-dimensional array of characters.
- The ANSI Standard string library provides a fully featured `string` class. The use of the `string` class is illustrated in Display 10.9 and Display 10.10. Typical calls to many members of class `string` are shown in Display 10.11.