Review for Final

Larry Caretto

Computer Science 106

Computing in Engineering and Science

May 18, 2006

The Final

- Thursday, Tuesday, May 23, 12:45 to 2:45 pm in this room
- Closed book, except for C++ guide
- Problems like the first quiz, midterm and homework assignments
  - Interpret code and give the results of the code (show your reasoning!)
  - Write code to accomplish a simple task

Outline

- Data types, operators, expressions, assignment, and type conversion
- Simple programs with sequential statements and input/output
- Use of if statements for choice
- Looping commands
- Functions
- Arrays

Data Types

- All variables must be declared as belonging to a certain data type
  - Good idea to initialize data at the same time that it is declared
  - Beware of scope rules
- Know int, double, char, and string
- Types (classes) for file variable names: fstream, ofstream, ifstream

Assignment Operator (=)

- ***<variable>*** = ***<expression>***
- Assignment operator not an equality
- Expression is a constant, a variable, or combination of constants, variables and operators
- Have arithmetic, relational and logical operators
  - Arithmetic operators in order of precedence (unary–) (* % /) (+ –)

Operator Precedence

- Determines how operators are applied
- Can use parentheses to overcome normal rules of precedence
  - Important for getting correct equations
- Precedence order: arithmetic, relational, logical
  - Relational precedence (<, >, <=, >=) (== !=)
  - Logical precedence ! && ||

Mathematical Statements

- Must be in correct sequential order
  - Input data
  - Do calculations in order: quantities on left sides of = operator must be calculated first
  - Write output
- Can use functions in <cmath> library such as exp, pow, log, sin, cos, atan, …
- Use type double for calculations, reserve type int for counting

Types of Statements

- Sequential
- Choice
  - Given by if statements
- Looping (repetition)
  - Test before versus test after
  - for loop for counting loops
- Function
  - Transfer control and data from one function to another and back

Conditions

- Choice and loops use conditions, expressions that have (Boolean) values of true or false
- Use relational operators <, >, <=, >=, ==, and !=
- Combine conditions with Boolean (bool) operators: not(!), and(&&), or(||)
  - Examples: (x < 3), (hours > 40), (age >= 16 && age < 75)

Simple if Statement

```
if ( <condition> )
{
    < true statement block >
}
<next statement executed if
condition is false>
```

Simple if-else Statement

```
if ( <condition> )
{
    < true statement block >
}
else
{
    < false statement block >
}
<next statement executed after
true or false block>
```

if-else if Statement

```
if ( <condition 1> )
{< statement block 1 >}
else if ( <condition 2> )
{< statement block 2 >}
..................
else if ( <condition n> )
{< statement block n >}
else
{<allConditionsFalse block> }
<next statement executed after the
selected block is executed>
```

- Only one block executes
- Final else is optional

Nested If Statements

- Can have one if block inside another
- Example: Find number of days in month
  - If the number of the month is 4, 6, 9, or 11 the answer is 30
  - If the number of the month is 2
    - If it is a leap year, the answer is 29
    - Otherwise the answer is 28
  - For all other month numbers (1, 3, 5, 7, 8, 10, and 12) the answer is 31

Simple while Statement

```
while ( <condition> )
{
      < loop body >
}
< next statement >
```

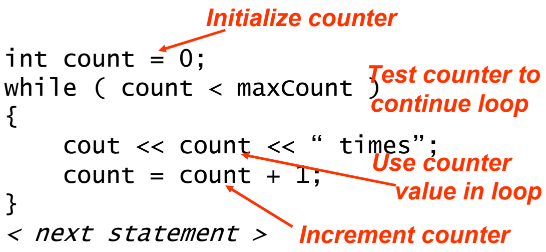
- Loop body statements are executed repeatedly if condition is true
- May never be executed once
- Control transfers when condition is false

do-while Statement

```
do
{
      < loop body >
}
while ( <condition> );
< next statement >
```

- Loop body statements are executed repeatedly if condition is true
- Loop body will be executed first time
- Control transfers when condition is false

Count-controlled while Loop



```
int count = 0;
while ( count < maxCount )
{
     cout << count << " times";
     count = count + 1;
}
< next statement >
```

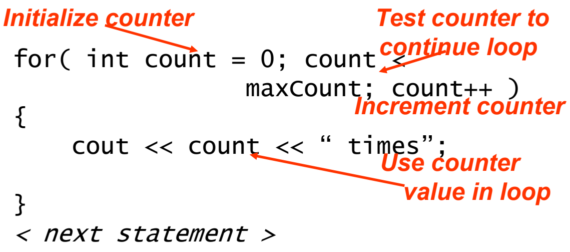
Initialize counter

Test counter to continue loop

Use counter value in loop

Increment counter

Count-controlled for Loop



```
for( int count = 0; count <
                    maxCount; count++ )
{
     cout << count << " times";

}
< next statement >
```

Initialize counter

Test counter to continue loop

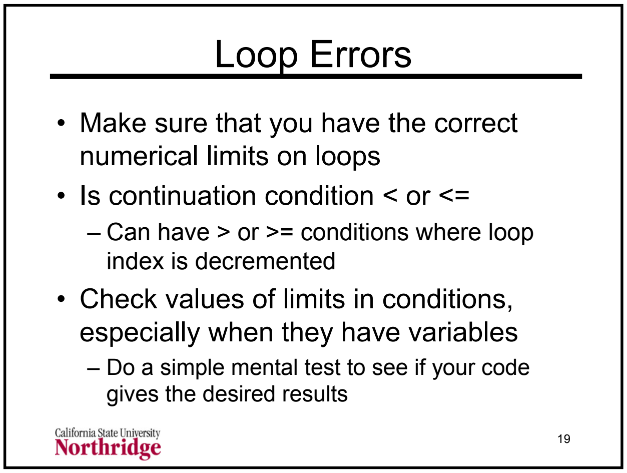
Increment counter

Use counter value in loop

Nested Loops

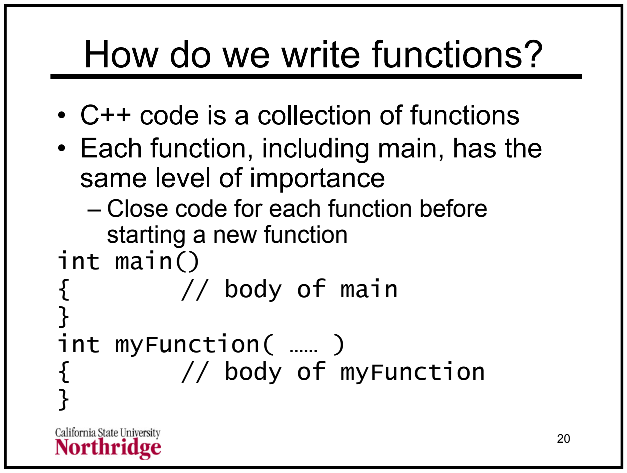
- Nested loop example of printing a table

```
// print column headers
for ( v1 = a; v1 < b; v1 += c)
{ cout << "\nv1 = " << v1;
   for ( v2 = d; v2 < e; v2 += f )
      cout << setw(12) << v3(v1, v2)
}
// watch roundoff in loop controls
```



Loop Errors

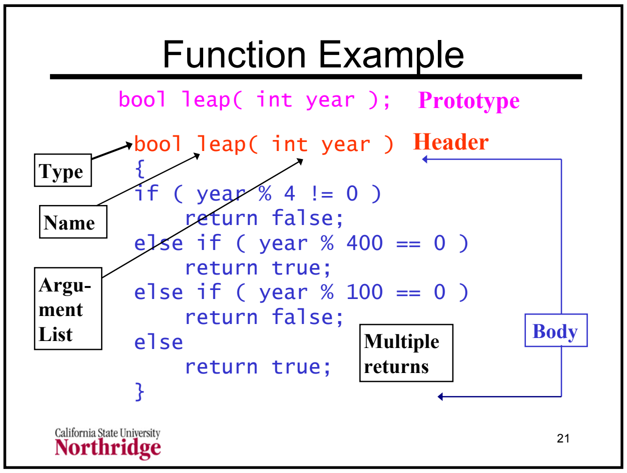
- Make sure that you have the correct numerical limits on loops
- Is continuation condition < or <=
  - Can have > or >= conditions where loop index is decremented
- Check values of limits in conditions, especially when they have variables
  - Do a simple mental test to see if your code gives the desired results



How do we write functions?

- C++ code is a collection of functions
- Each function, including main, has the same level of importance
  - Close code for each function before starting a new function

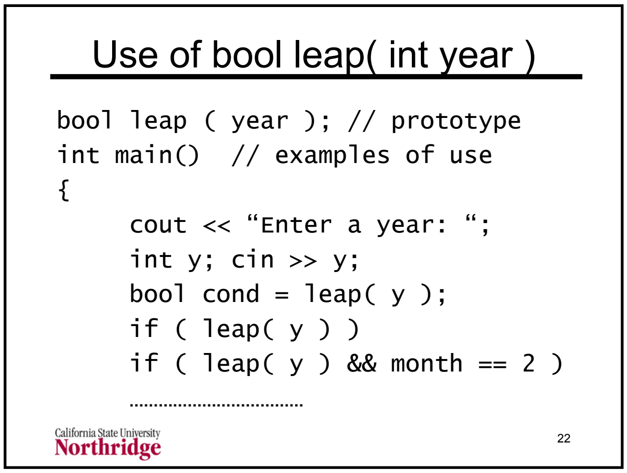
```
int main()
{        // body of main
}
int myFunction( …… )
{        // body of myFunction
}
```



Function Example

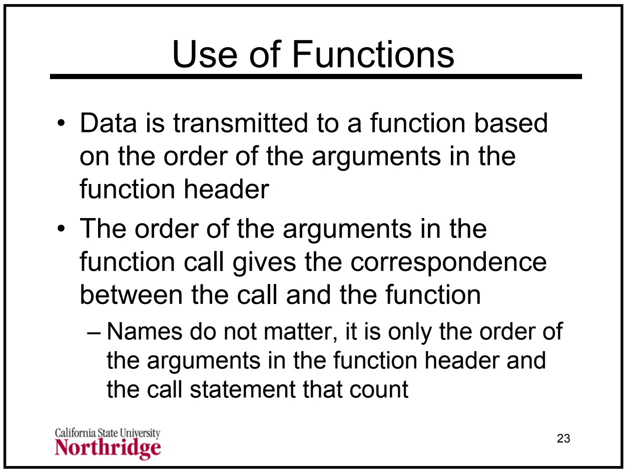
```
bool leap( int year );  Prototype

bool leap( int year )  Header
{
if ( year % 4 != 0 )
      return false;
else if ( year % 400 == 0 )
      return true;
else if ( year % 100 == 0 )
      return false;
else
      return true;
}
```



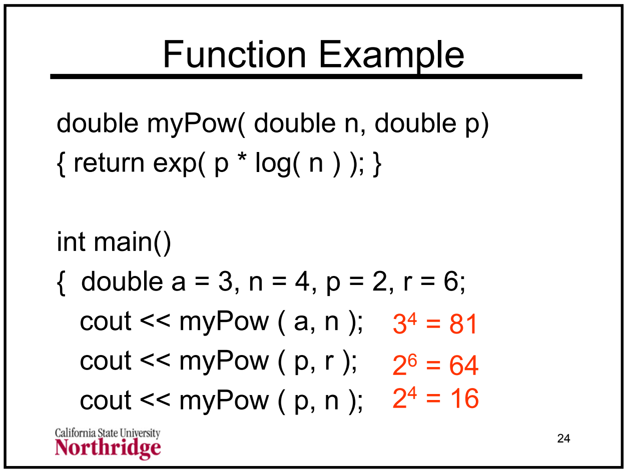
Use of bool leap(int year)

```
bool leap ( year ); // prototype
int main()  // examples of use
{
      cout << “Enter a year: “;
      int y; cin >> y;
      bool cond = leap( y );
      if ( leap( y ) )
      if ( leap( y ) && month == 2 )
      ……………………………
```



Use of Functions

- Data is transmitted to a function based on the order of the arguments in the function header
- The order of the arguments in the function call gives the correspondence between the call and the function
  - Names do not matter, it is only the order of the arguments in the function header and the call statement that count



Function Example

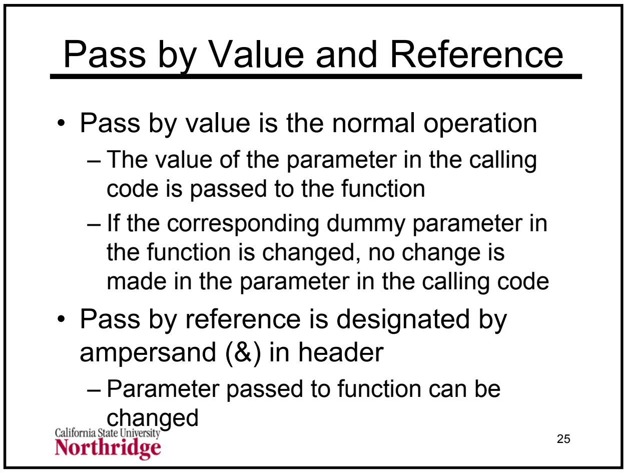
```
double myPow( double n, double p)
{ return exp( p * log( n ) ); }

int main()
{  double a = 3, n = 4, p = 2, r = 6;
   cout << myPow ( a, n );
   cout << myPow ( p, r );
   cout << myPow ( p, n );
```

$3^4 = 81$

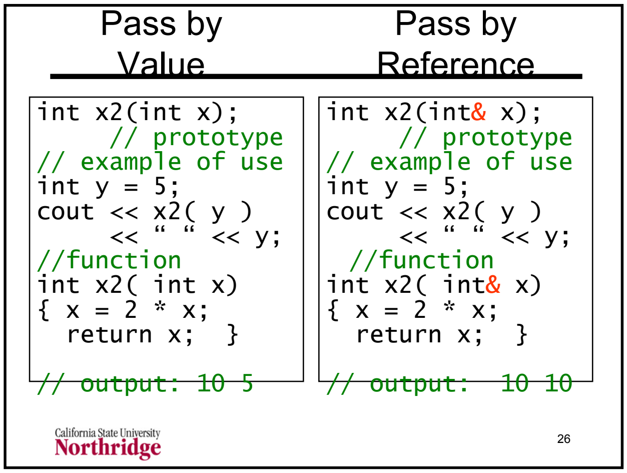
$2^6 = 64$

$2^4 = 16$



Pass by Value and Reference

- Pass by value is the normal operation
  - The value of the parameter in the calling code is passed to the function
  - If the corresponding dummy parameter in the function is changed, no change is made in the parameter in the calling code
- Pass by reference is designated by ampersand (&) in header
  - Parameter passed to function can be changed



Pass by Value

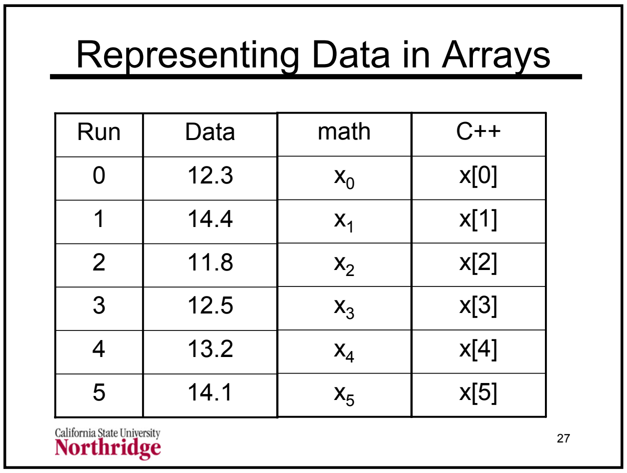
```
int x2(int x);
     // prototype
// example of use
int y = 5;
cout << x2( y )
      << “ “ << y;
//function
int x2( int x)
{ x = 2 * x;
  return x;  }

// output: 10 5
```

Pass by Reference

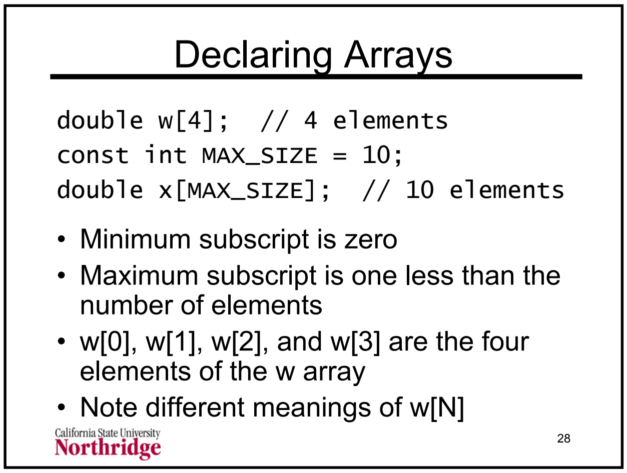
```
int x2(int& x);
     // prototype
// example of use
int y = 5;
cout << x2( y )
      << “ “ << y;
  //function
int x2( int& x)
{ x = 2 * x;
  return x;  }

// output:  10 10
```



Representing Data in Arrays

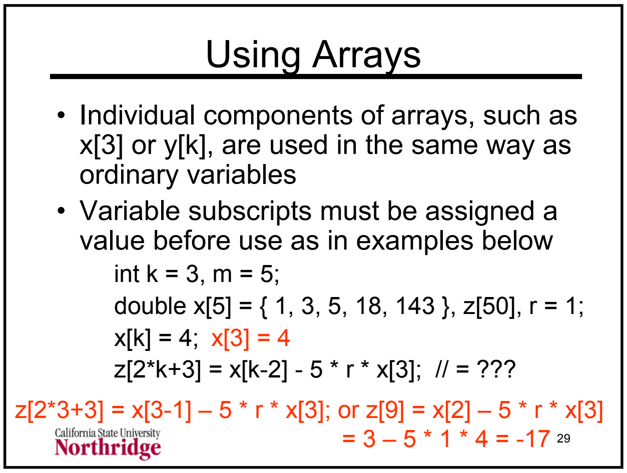
Run	Data	math	C++
0	12.3	x_0	x[0]
1	14.4	x_1	x[1]
2	11.8	x_2	x[2]
3	12.5	x_3	x[3]
4	13.2	x_4	x[4]
5	14.1	x_5	x[5]



Declaring Arrays

```
double w[4];  // 4 elements
const int MAX_SIZE = 10;
double x[MAX_SIZE];  // 10 elements
```

- Minimum subscript is zero
- Maximum subscript is one less than the number of elements
- w[0], w[1], w[2], and w[3] are the four elements of the w array
- Note different meanings of w[N]

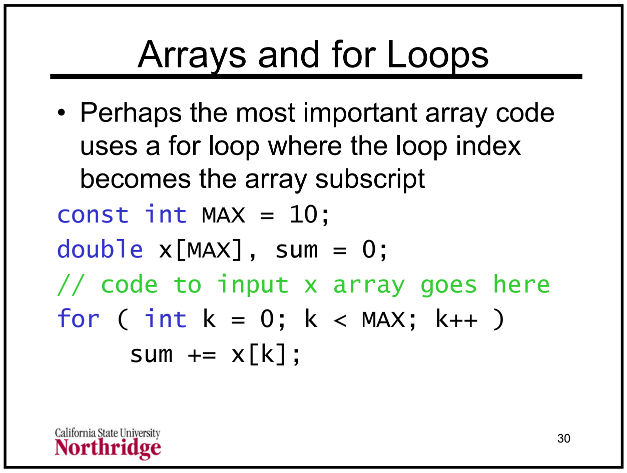


Using Arrays

- Individual components of arrays, such as x[3] or y[k], are used in the same way as ordinary variables
- Variable subscripts must be assigned a value before use as in examples below

```
int k = 3, m = 5;
double x[5] = { 1, 3, 5, 18, 143 }, z[50], r = 1;
x[k] = 4;  x[3] = 4
z[2*k+3] = x[k-2] - 5 * r * x[3];  // = ???
```

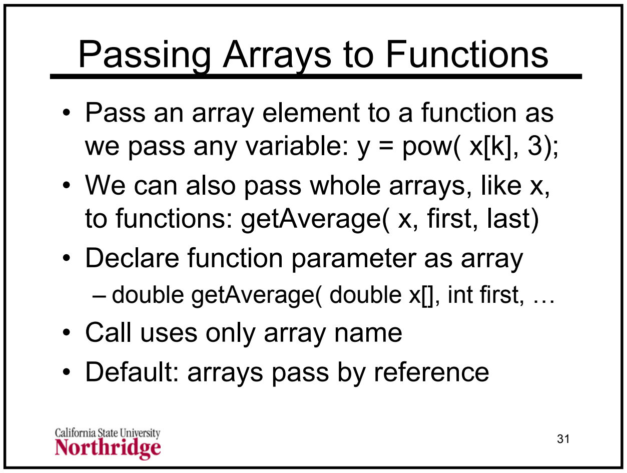
z[2*3+3] = x[3-1] – 5 * r * x[3]; or z[9] = x[2] – 5 * r * x[3]
= 3 – 5 * 1 * 4 = -17



Arrays and for Loops

- Perhaps the most important array code uses a for loop where the loop index becomes the array subscript

```
const int MAX = 10;
double x[MAX], sum = 0;
// code to input x array goes here
for ( int k = 0; k < MAX; k++ )
     sum += x[k];
```

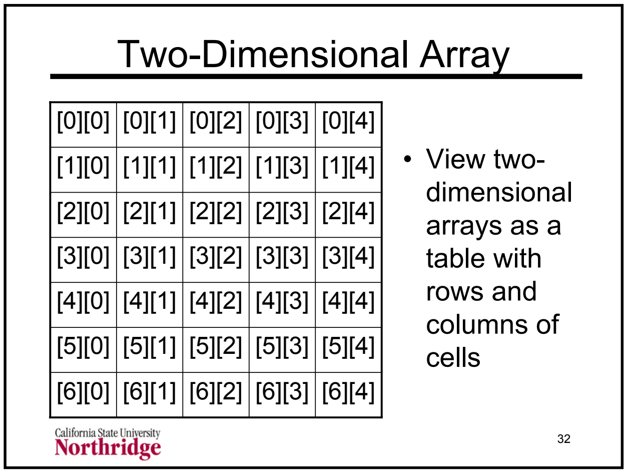


Passing Arrays to Functions

- Pass an array element to a function as we pass any variable: y = pow(x[k], 3);
- We can also pass whole arrays, like x, to functions: getAverage(x, first, last)
- Declare function parameter as array
  - double getAverage(double x[], int first, …
- Call uses only array name
- Default: arrays pass by reference

California State University Northridge

31



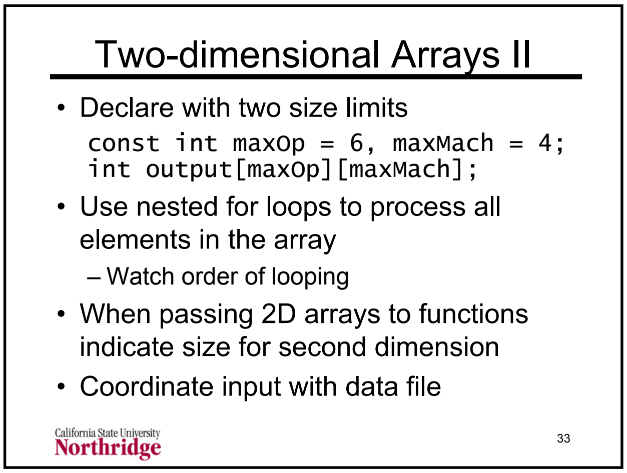
Two-Dimensional Array

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]
[5][0]	[5][1]	[5][2]	[5][3]	[5][4]
[6][0]	[6][1]	[6][2]	[6][3]	[6][4]

- View two-dimensional arrays as a table with rows and columns of cells

California State University Northridge

32



Two-dimensional Arrays II

- Declare with two size limits

```
const int maxOp = 6, maxMach = 4;
int output[maxOp][maxMach];
```

- Use nested for loops to process all elements in the array
  - Watch order of looping
- When passing 2D arrays to functions indicate size for second dimension
- Coordinate input with data file

California State University Northridge

33