

CSC 222: Computer Programming II

Spring 2004

- Review of C++ basics
 - program structure, input, output
 - variables, expressions, functions, parameters
 - control: if, if-else, while, for
 - predefined classes: string
 - arrays

1

Program structure & output

```
// hello.cpp
////////////////////////////////////
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello and welcome to CSC222." << endl << endl;

    cout << "Programming is both intellectually challenging" << endl
         << "and artistically rewarding. Enjoy!" << endl;
    cout << "                                     -- Dr. Reed" << endl;

    return 0;
}
```

- `#include` to load libraries (new standard requires specify namespace)
- every program must have function named `main` (return type `int`)
- C++ comments specified using `//` or `/* ... */`
- can output text using standard output stream `cout` and `<<` operator

2

Variables & input

```
// ftoc.cpp
////////////////////////////////////

#include <iostream>
using namespace std;

int main()
{
    double tempInFahr;
    cout << "Enter the temperature (in degrees Fahrenheit): ";
    cin >> tempInFahr;

    cout << "You entered " << tempInFahr
         << " degrees Fahrenheit." << endl
         << "That's equivalent to " << (5.0/9.0 * (tempInFahr - 32))
         << " degrees Celsius" << endl;

    return 0;
}
```

- declare a variable by specifying type, followed by variable name
 - types include `int`, `double`, `char`, `bool`, `string*`
 - can assign a value to a variable using `=`
 - can read a value into a variable using standard input stream `cin` and `>>` operator

3

Strings

```
// greet.cpp
////////////////////////////////////

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

int main()
{
    string firstName, lastName;
    cout << "Enter your name (first then last): ";
    cin >> firstName >> lastName;

    cout << "Nice to meet you, " << firstName << " " << lastName
         << ". May I just call you " << firstName
         << "?" << endl;

    return 0;
}
```

- `string` type is defined in the library file `<string>`
 - note: if you used `char*` in 221, this is MUCH better!!!!
 - can read and write strings just like any other type
 - when reading a string value, delimited by whitespace

4

Expressions

- C++ provides various operators for constructing expressions

+ - * / %

(+ can be applied to strings to concatenate)

- `cmath` library contains many useful routines

`pow fabs sqrt
floor ceil`

```
// change.cpp
/////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;

int main()
{
    int amount;
    cout << "Enter an amount (in cents) to make change: ";
    cin >> amount;

    int quarters = amount/25;
    amount = amount%25;

    int dimes = amount/10;
    amount = amount%10;

    int nickels = amount/5;
    amount = amount%5;

    int pennies = amount;

    cout << "Optimal change:" << endl;
    cout << "    # of quarters =\t" << quarters << endl;
    cout << "    # of dimes =\t" << dimes << endl;
    cout << "    # of nickels =\t" << nickels << endl;
    cout << "    # of pennies =\t" << pennies << endl;

    return 0;
}
```

5

Constants & formatted I/O

- constants define values that will not change
safer (compiler enforces)
easier to manage (global OK)

- `iomanip` library contains routines for formatting output

`setiosflags
setprecision
setw`

```
// pizza.cpp          Dave Reed          9/6/01
//
// This program determines the cost per sq. inch of a pizza.
/////////////////////////////////////////////////////////////////

#include <iostream>
#include <iomanip>
using namespace std;

const double PI = 3.14159;

int main()
{
    double pizzaDiameter, pizzaCost;
    cout << "Enter the diameter of the pizza (in inches): ";
    cin >> pizzaDiameter;
    cout << "Enter the price of the pizza (in dollars): ";
    cin >> pizzaCost;

    double pizzaArea = PI*(pizzaDiameter*pizzaDiameter)/4.0;
    double costPerSqInch = pizzaCost/pizzaArea;

    cout << setiosflags(ios::fixed);

    cout << "Total area of the pizza: "
         << setprecision(4) << pizzaArea << " square inches."
         << endl;
    cout << "    price per square inch = $"
         << setprecision(2) << costPerSqInch << "." << endl;

    return 0;
}
```

6

Functions

```
// lowhigh.cpp
////////////////////////////////////

#include <iostream>
using namespace std;

double FahrToCelsius(double tempInFahr)
// Assumes: tempInFahr is a temperature in Fahrenheit
// Returns: equivalent temperature in Celsius
{
    return (5.0/9.0 * (tempInFahr - 32));
}

int main()
{
    double lowTemp, highTemp;
    cout << "Enter the forecasted low (in degrees Fahrenheit): ";
    cin >> lowTemp;
    cout << "Enter the forecasted high (in degrees Fahrenheit): ";
    cin >> highTemp;

    cout << endl
         <<"The forecasted low (in Celsius): " << FahrToCelsius(lowTemp) << endl
         <<"The forecasted high (in Celsius): " << FahrToCelsius(highTemp) << endl;

    return 0;
}
```

functions encapsulate
computations

- define once, call many times
- abstract away details in main
- parameters in function store values passed in as arguments
- params & args match by position
- should always document any assumptions, return value

7

Functions calling functions

```
// convert.cpp
////////////////////////////////////

#include <iostream>
using namespace std;

double centimetersToInches(double cm);
double metersToFeet(double m);
double kilometersToMiles(double km);

int main()
{
    double distanceInKM;
    cout << "Enter a distance in kilometers: ";
    cin >> distanceInKM;

    cout << "That's equivalent to "
         << kilometersToMiles(distanceInKM)
         << " miles." << endl;

    return 0;
}

////////////////////////////////////
```

can place function prototypes above main, then
function definitions in any order after main

```
double centimetersToInches(double cm)
// Assumes: cm is a length in centimeters
// Returns: equivalent length in inches
{
    return cm/2.54;
}

double metersToFeet(double m)
// Assumes: m is a length in meters
// Returns: equivalent length in feet
{
    double cm = 100*m;
    double in = centimetersToInches(cm);
    return in/12;
}

double kilometersToMiles(double km)
// Assumes: km is a length in meters
// Returns: equivalent length in miles
{
    double m = 1000*km;
    double ft = metersToFeet(m);
    return ft/5280;
}
```

8

value vs. reference parameters

by default, parameters are passed *by-value*

- a copy of the input is stored in the parameter (a local variable)
- result: value passed in, no changes are passed out

& implies *by-reference*

- the parameter does not refer to a new piece of memory – it is an alias for the argument
- result: changes to the parameter simultaneously change the input

```
void foo(int x)
{
    x = 5;
    cout << x << endl;
}

int a = 3;
foo(a);

cout << a << endl;
foo(3);
```

note: input can be any value

```
void foo(int &x)
{
    x = 5;
    cout << x << endl;
}

int a = 3;
foo(a);

cout << a << endl;
```

note: input must be a variable

9

Advantages of functions

computational abstraction

- define & reuse
- ignore details

simplify repeated tasks

- avoids repeated code
- can generalize using params

can place useful functions in library

- reuse using `#include`

```
// oldmac.cpp
////////////////////////////////////
#include <iostream>
#include <string>
using namespace std;

void Verse(string animal, string noise);

int main()
{
    Verse("cow", "moo");
    Verse("horse", "neigh");
    Verse("duck", "quack");

    return 0;
}

////////////////////////////////////
void Verse(string animal, string noise)
// Assumes: animal is an animal name, and noise is the noise it makes
// Results: displays the corresponding OldMacDonald verse
{
    cout << "Old MacDonald had a farm, E-I-E-I-O." << endl;
    cout << "And on that farm he had a " << animal << ", E-I-E-I-O." << endl;
    cout << "With a " << noise << "-" << noise << " here, and a "
        << noise << "-" << noise << " there," << endl;
    cout << " here a " << noise << ", there a " << noise << ", everywhere a "
        << noise << "-" << noise << "." << endl;
    cout << "Old MacDonald had a farm, E-I-E-I-O." << endl << endl;
}
```

10

If statements

- if statements provide for conditional execution
- simple if specifies code to be executed or not (depending on Boolean condition)

comparison operators

== != > >= < <=

logical connectives

&& || !

```
//change.cpp
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

void DisplayCoins(int & amount, int coinValue, string coinType);

int main()
{
    int amount;
    cout << "Enter an amount (in cents) to make change: ";
    cin >> amount;

    cout << "Optimal change:" << endl;
    DisplayCoins(amount, 25, "quarters");
    DisplayCoins(amount, 10, "dimes");
    DisplayCoins(amount, 5, "nickels");
    DisplayCoins(amount, 1, "pennies");

    return 0;
}

/////////////////////////////////////////////////////////////////

void DisplayCoins(int & amount, int coinValue, string coinType)
// Assumes: amount >= 0, coinValue >= 1, coinType is a name
// Results: displays a message if number of coinTypes > 0
{
    int numCoins = amount / coinValue;
    amount = amount % coinValue;

    if (numCoins > 0) {
        cout << setw(4) << numCoins << " " << coinType << endl;
    }
}
```

11

Cascading if-else

- if-else provides for 2-way conditional
- can be chained together

```
// grades.cpp
/////////////////////////////////////////////////////////////////

#include <iostream>
using namespace std;

const double A_CUTOFF = 90.0;
const double B_PLUS_CUTOFF = 87.0;
const double B_CUTOFF = 80.0;
const double C_PLUS_CUTOFF = 77.0;
const double C_CUTOFF = 70.0;
const double D_CUTOFF = 60.0;

int main()
{
    double grade;
    cout << "Enter your grade: ";
    cin >> grade;
```

```
string letterGrade;
if (grade >= A_CUTOFF) {
    letterGrade = "A";
}
else if (grade >= B_PLUS_CUTOFF) {
    letterGrade = "B+";
}
else if (grade >= B_CUTOFF) {
    letterGrade = "B";
}
else if (grade >= C_PLUS_CUTOFF) {
    letterGrade = "C+";
}
else if (grade >= C_CUTOFF) {
    letterGrade = "C";
}
else if (grade >= D_CUTOFF) {
    letterGrade = "D";
}
else {
    letterGrade = "F";
}

cout << "You are guaranteed at least a "
    << letterGrade << "." << endl;

return 0;
}
```

12

Local vs. global scope

scope = portion of program where variable is accessible

if declared in function or block { ... }, then scope is limited to that function/block (i.e., *local*)

variables declared outside of functions are accessible to all (i.e., *global*)

✓ variables should be as local as possible

✓ global constants are OK

```
// chill.cpp
///////////////////////////////////////////////////////////////////
#include <iostream>
#include <cmath>
using namespace std;

double WindChill(double temp, double wind);

int main()
{
    double temperature;
    cout << "Enter the temperature (in degrees Fahrenheit): ";
    cin >> temperature;

    if (temperature < -35 || temperature > 45) {
        cout << "That temperature is too extreme. "
             << "Wind-chill is not defined." << endl;
    }
    else {
        int windSpeed;
        cout << "Enter the wind speed (in miles per hour): ";
        cin >> windSpeed;

        cout << endl << "The wind-chill factor is "
             << WindChill(temperature, windSpeed) << endl;
    }

    return 0;
}

double WindChill(double temp, double wind)
// Assumes: -35 <= temp <= 45, wind >= 0
// Returns: wind-chill factor given temperature and wind speed
{
    if (wind < 4) {
        return temp;
    }
    else {
        return 35.74 + 0.6215*temp +
            (0.4274*temp - 35.75)*pow(wind, 0.16);
    }
}
```

13

C++ libraries

C++ provides numerous libraries of useful code

- **cmath** functions for manipulating numbers, including:

```
double fabs(double x);
double sqrt(double x);
double ceil(double x);
double floor(double x);
double pow(double x, double y);
```

```
int numBits;
cin >> numBits;

cout << "With " << numBits << "bits, "
     << "you can represent "
     << pow(2,numBits) << "patterns.";
```

- **cctype** functions for testing and manipulating characters, including:

```
bool isalpha(char ch);
bool islower(char ch);
bool isupper(char ch);
bool isdigit(char ch);
bool isspace(char ch);

char tolower(char ch);
char toupper(char ch);
```

```
char response;
cout << "Do you want to play again? (y/n) ";
cin >> response;

if (tolower(response) == 'y') {
    PlayGame();
}
else {
    cout << "Thanks for playing." << endl;
}
```

14

Pig Latin (v. 2)

```
// piglatin.cpp      Dave Reed      9/30/01
//
// Third version of Pig Latin translator.
///////////////////////////////////////////////////////////////////

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

string PigLatin(string word);
bool IsVowel(char ch);

int main()
{
    string word;
    cout << "Enter a word: ";
    cin >> word;

    cout << "That translates to: "
         << PigLatin(word) << endl;

    return 0;
}

/////////////////////////////////////////////////////////////////
```

better way uses string search

- search for ch in a string of vowels

```
string PigLatin(string word)
// Assumes: word is a single word (no spaces)
// Returns: the Pig Latin translation of the word
{
    if (IsVowel(word.at(0))) {
        return word + "way";
    }
    else {
        return word.substr(1, word.length()-1) +
            word.at(0) + "ay";
    }
}

bool IsVowel(char ch)
// Assumes: ch is a letter
// Returns: true if ch is a vowel ("aeiouAEIOU")
{
    const string VOWELS = "aeiouAEIOU";

    return (VOWELS.find(ch) != string::npos);
}
```

17

While loops

provide for conditional repetition

pseudo-code:

```
ROLL DICE;
DISPLAY RESULTS;
INITIALIZE ROLL COUNT;

while (DICE ARE DIFFERENT){
    ROLL DICE AGAIN;
    DISPLAY RESULTS AGAIN;
    INCREMENT ROLL COUNT;
};

DISPLAY NUMBER OF ROLLS;
```

```
// doubles.cpp
//
// Simulates rolling two dice until doubles.
///////////////////////////////////////////////////////////////////

#include <iostream>
#include <ctime>
using namespace std;

int DieRoll(int numSides = 6);

int main()
{
    srand(unsigned(time(0)));

    int roll1 = DieRoll();
    int roll2 = DieRoll();
    cout << "You rolled " << roll1
         << " and " << roll2 << endl;

    int rollCount = 1;
    while (roll1 != roll2) {
        roll1 = DieRoll();
        roll2 = DieRoll();
        cout << "You rolled " << roll1
             << " and " << roll2 << endl;

        rollCount++;
    }

    cout << "It took " << rollCount << " roll(s)."
         << endl;

    return 0;
}

int DieRoll(int numSides)
{
    return static_cast<int>(rand()) % numSides + 1;
}
```

18

Priming a loop

can avoid redundancy with a KLUDEGE (a quick-and-dirty trick for making code work)

- only roll the dice inside the loop
- initialize the roll variables so that the loop test succeeds the first time
- after the first kludgy time, the loop behaves as before

```
// doubles.cpp
//
// Simulates rolling two dice until doubles.
//
#include <iostream>
#include <ctime>
using namespace std;

int DieRoll(int numSides = 6);

int main()
{
    srand(unsigned(time(0)));

    int roll1 = -1;
    int roll2 = -2;

    int rollCount = 0;
    while (roll1 != roll2) {
        roll1 = DieRoll();
        roll2 = DieRoll();
        cout << "You rolled " << roll1
              << " and " << roll2 << endl;

        rollCount++;
    }

    cout << "It took " << rollCount << " roll(s)."
          << endl;

    return 0;
}

int DieRoll(int numSides)
{
    return static_cast<int>(rand()) % numSides + 1;
}
```

19

Paper folding puzzle

if you started with a regular sheet of paper and repeatedly fold it in half, how many folds would it take for the thickness of the paper to reach the sun?

pseudo-code:

```
INITIALIZE PAPER THICKNESS;
INITIALIZE FOLD COUNT TO 0;

while (THICKNESS < SUN DISTANCE) {
    DOUBLE THICKNESS;
    INCREMENT FOLD COUNT;
}

DISPLAY FOLD COUNT;
```

```
// fold.cpp
//
// Simulates the paper folding puzzle.
//
#include <iostream>
using namespace std;

const double PAPER_THICKNESS = 0.002;
const double DISTANCE_TO_SUN = 93.3e6*5280*12;

int main()
{
    double thick = PAPER_THICKNESS;
    int foldCount = 0;

    while (thick < DISTANCE_TO_SUN) {
        thick *= 2;
        foldCount++;
    }

    cout << "It took " << foldCount << " folds."
          << endl;

    return 0;
}
```

20

Sums & averages

can use a variable to keep a running sum (init & add to)

a *sentinel value* is a special value that marks the end of input

```
// avg.cpp
//
// Computes and displays average of grades.
//
#include <iostream>
using namespace std;

int main()
{
    int numGrades = 0, gradeSum = 0;

    int grade;
    cout << "Enter grades (terminate with -1):" << endl;
    cin >> grade;

    while (grade != -1) {
        gradeSum += grade;
        numGrades++;

        cin >> grade;
    }

    double avg = static_cast<double>(gradeSum)/numGrades;
    cout << "Your average is " << avg << endl;

    return 0;
}
```

21

While loops vs. for loops

use for loop when you know the number of repetitions ahead of time
use while loop when the number of repetitions is unpredictable

```
int i = 0;
while (i < MAX) {
    DO SOMETHING;
    i++;
}
```

```
for (int i = 0; i < MAX; i++) {
    DO SOMETHING;
}
```

while loop version:

```
int numTimes = 0;
while (numTimes < 10) {
    cout << "Howdy" << endl;
    numTimes++;
}
```

for loop version:

```
for (int numTimes = 0; numTimes < 10; numTimes++) {
    cout << "Howdy" << endl;
}
```

```
int i = 1, sum = 0;
while (i <= 100) {
    sum += i;
    i++;
}
```

```
int sum = 0;
for (i = 1; i <= 100; i++) {
    sum += i;
}
```

22

Pig Latin (general version)

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

string PigLatin(string word);
bool IsVowel(char ch);
int FindVowel(string str);

int main()
{
    string word;
    cout << "Enter a word: ";
    cin >> word;

    cout << "That translates to: "
         << PigLatin(word) << endl;

    return 0;
}

////////////////////////////////////
bool IsVowel(char ch)
// Assumes: ch is a letter
// Returns: true if ch is a vowel
// ("aeiouAEIOU")
{
    const string VOWELS = "aeiouAEIOU";
    return VOWELS.find(ch) != string::npos;
}

string PigLatin(string word)
// Assumes: word is a single word (no spaces)
// Returns: the Pig Latin translation of the word
{
    int vowelIndex = FindVowel(word);

    if (vowelIndex == 0 || vowelIndex == string::npos) {
        return word + "way";
    }
    else {
        return word.substr(vowelIndex,
                          word.length()-vowelIndex) +
               word.substr(0, vowelIndex) + "ay";
    }
}

int FindVowel(string str)
// Assumes: str is a single word (no spaces)
// Returns: the index of the first vowel in str, or
// string::npos if no vowel is found
{
    for (int index = 0; index < str.length(); index++) {
        if (IsVowel(str.at(index))) {
            return index;
        }
    }

    return string::npos;
}
```

23

Arrays

in C++, an *array* allows for related values to be

- stored under a single name, accessed via an index
- traversed & operated on systematically using a loop
- passed as a single entity to and from functions

an array is a homogeneous collection of values, accessible via an index

example: suppose want to simulate dice rolls, maintain statistics

- could have 11 distinct variables, each corresponding to a roll count tedious, can't generalize to N-SIDED dice
- better solution: have an array of counters, indexed from 0 to 2*DIE_SIDES initialize all array elements to 0 (although 0 & 1 indices are wasted) for each dice roll, increment the corresponding index

24

