

A Review of CSCI 1170 C++ Topics

Goal: Establish a core understanding
for everyone taking 2170

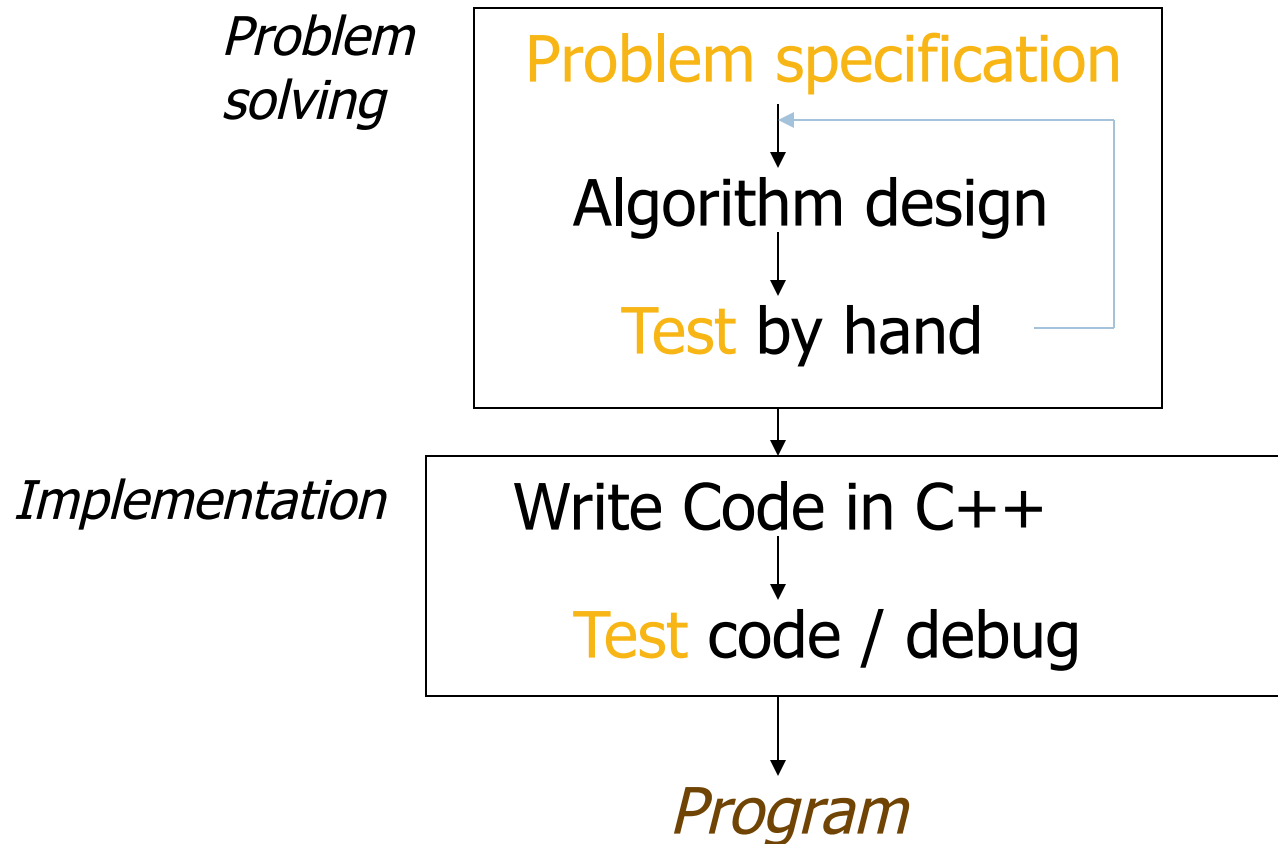
Overview

2

- Overview of program development
- Review of C++
 - ▣ Basics
 - ▣ Data types and conversion
 - ▣ Functions
 - ▣ I/O streams
 - ▣ Loops
 - ▣ 1-D and 2-D Arrays

Program Development

3



The compilation Process

4

Preprocessor

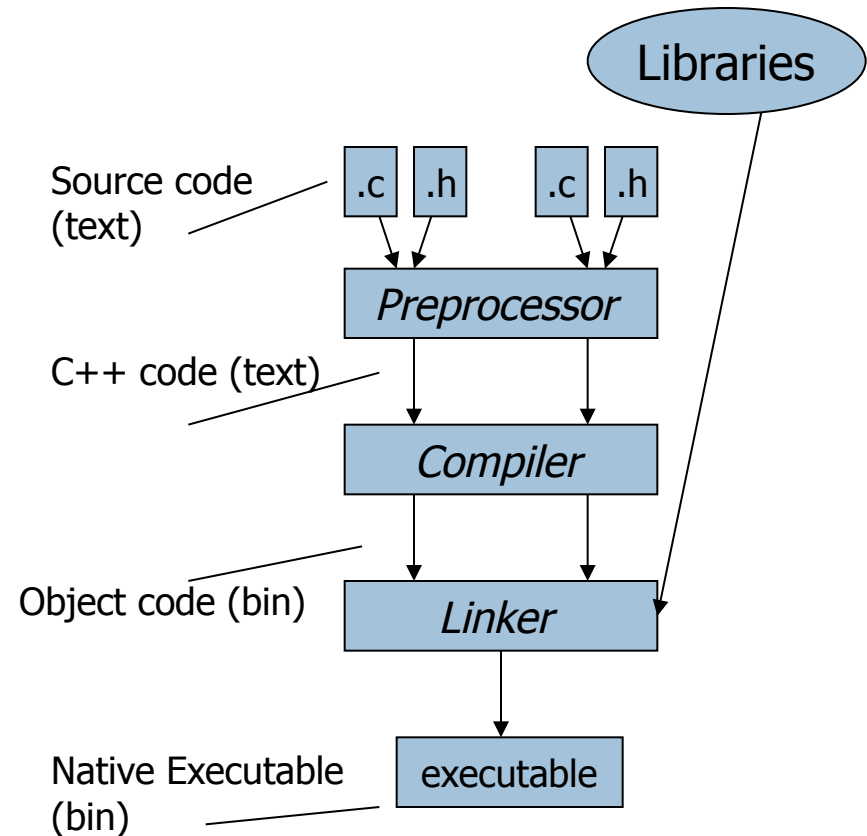
Resolves #define, #include

Compiler

Translates code to Machine Language. It outputs an "object file." This code is not executable

Linker

Takes object files and resolves references to functions and variables in different files



Preprocessor and Compiler

5

Preprocessor

- ▣ Does not understand C++
- ▣ Produces text that gets read by the C++ compiler

Compiler

- ▣ Expects C++ source code (no preprocessor stuff)
- ▣ Checks syntax
- ▣ Typechecks your code
- ▣ Generates, and optimizes, machine code
- ▣ Produces object files for the linker

Linker

6

- ▣ Resolves dependencies between object files. Later this semester, we will have multiple object files.
- ▣ Places functions, variables, in final memory location
- ▣ Knows nothing about source files (filenames or line numbers or data types)
- ▣ In C++, linker symbols have been mangled by the compiler (name mangling) to include the return type and argument types. This allows for name overloading.
- ▣ Name mangling is compiler dependent. The linker may not be able to link objects from different compilers.

C++ Basics - Comments

7

□ C++

// line comment

/* multi-line comment */

□ Program heading comments

// Name: your name goes here

// Class: 2170-xxx ← your section

// Assigement#:

// Due Date:

// Instructor:

Preprocessor Directives

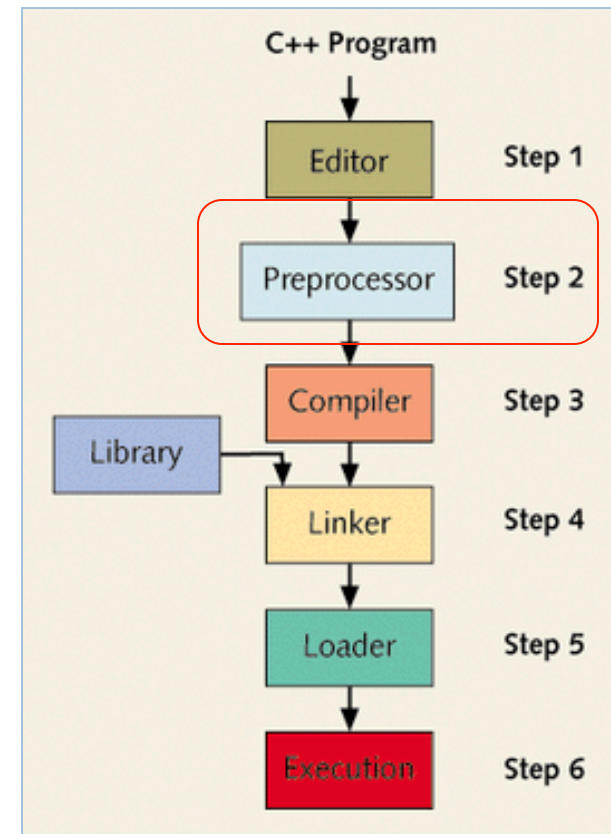
8

- Commands supplied to the preprocessor
 - ▣ Runs before the compiler
 - ▣ Modifies the text of the source code before the compiler starts
- Start with **#** symbol
- Example **#include <iostream>**

Preprocessor Directives

9

- Note the preprocessor step in the sequence



Directives

10

- **Include** Directives add library files to our programs

- To make the definitions of the `cin` and `cout` available to the program

```
#include <iostream>
```

- **Using** Directives include a collection of defined names

- To make the names `cin` and `cout` available to our program:

```
using namespace std;
```

Variables and Assignments

11

- Variables are like small storage boxes.
- We can store different things in them.
- We can change their contents.
- We can not change the type of contents.
- In C++ variables are names for memory locations

Identifiers

12

- Variables names are called identifiers
- Choosing variable names
 - ▣ Use meaningful names that represent the data to be stored
 - ▣ First character must be a letter or the underscore character
 - ▣ Remaining characters must be letters, numbers, or underscore character

Keywords

13

- Keywords (also called reserved words)
 - ▣ Are used by the C++ language
 - ▣ Must be used as they are defined in the programming language
 - ▣ Cannot be used as identifiers
 - ▣ They are case sensitive

Declaring and Defining things.....

14

- ❑ *Declarations and definitions are separate concepts*

int x; // declares x

x = 4; // defines x

Declarations

What is wrong with the following code?

```
# include iostream;
Using namespace std
int main(void)
{
    int result;
    result = add(20, 3);
    return 1.2;
}

int add(int a, int b)
{
    return a + b;
}
```

Declaration Regions

Will this work?

```
int main(void) {  
    int result;  
    int add(int,int);
```

```
    result = add(20, 3);  
    return 0;  
}
```

```
int add(int a, int b)  
{  
    return a + b;  
}
```

add is
declared in
the shaded
regions



Will this work?

```
int add(int,int);
```

```
int main(void) {  
    int result;  
  
    result = add(20, 3);  
  
    return 0;  
}
```

```
int add(int a, int b)  
{  
    return a + b;  
}
```


Declarations - Summary

17

- Declarations can be separate from definitions
- Function declarations are called prototypes
- Declarations should be given before usage
- In C++, usage of an undeclared function or variable is an error

Declaring Identifiers

18

- Before using an identifier, it must be declared
 - ▣ Declaring an identifier tells the compiler, among other things, the type association
 - ▣ Examples:

```
int    number_of_bars;  
int    funct(int, float);
```

Declaring Variables

19

Two locations for variable declarations

- Immediately prior to use

```
int main()
{
    ...
    int sum;
    sum = s1 + s2;
    ...
    return 0;
}
```

- At the beginning

```
int main()
{
    int sum;
    ...
    sum = s1 + s2;
    ...
    return 0;
}
```

C++ Basics - Variables

20

- Variables have a data type and name
- Use the following naming conventions:
 - ▣ All uppercase letters are used for constants
 - ▣ Variable names should be meaningful – for multi-words use
 - Convention 1: `alignment_sequence`
 - Convention 2: `AlignmentSequence`

C++ Basics – Data Types

21

- We will concentrate on 5 basic data types: int, float, bool, char, string
- Integer (**int**) – represents whole numbers, i.e., numbers with no decimal point
 - Ex 1: define an integer variable value1
`int value1; // not initialized to anything special`
 - Ex 2: define an integer variable value2 and initialize
`int value2=6; // initialized with value 6`

C++ Basic – Data Types (2)

22

- Floating point – represents real numbers, i.e., numbers with decimal points
 - ▣ Ex 1: define a single-precision floating-point variable named `rate` and initialize to 3.5
`float rate = 3.5;`
 - ▣ Ex 2: define variable named `OverValue` and initialize to 3
`float OverValue = 3;`

What's the value stored in `OverValue`?

C++ Basic – Data Types (3)

23

- Boolean – represents a true or false value. Also false is defined to be 0 and true is defined to be !0
 - ▣ Ex 1: define a boolean variable named `Done`
`bool Done;`
 - ▣ Ex 2: define a boolean variable named `Done` and initialize to true
`bool Done = true;`

What's the value stored by the following?

`bool Done = 3 > 4;`

C++ Basic – Data Types (4)

24

- ▣ Character – represents a single text value
 - ASCII – American Standard Code for Information Interchange ---Represents characters, numbers, punctuation, spacing and special non-printable control characters
 - Example ASCII codes: 'A' = 65, 'B' = 66, ... 'a' = 97, 'b' = 98, '\n' = 10
 - Ex 1: define a character named **Letter** and initialize it to 'C'
`char Letter = 'C';`

C++ Basic – Data Types (5)

25

- ▣ strings – represents one or more text values
 - In C++ there are two types of strings
 - C++ `string` type
 - C style string type, an array of `char`
 - Ex 1: define a C++ string named `Letters` and initialize it to “abc”
`string Letters = “abc”;`
 - Ex 2: define a C string named `Letters` and initialize it to “abc”
`char Letters[] = “abc”;`

C style strings

26

□ Ex 3. `char Letters[]="abc";`

```
char Items[10];
```

```
Items[0]='a'; Items[1]='b';
```

```
Items[2]='c'; Items[3]='\0';
```

<http://www.cppreference.com/wiki/string/start>

<http://www.cppreference.com/wiki/c/string/start>

C Strings

27

- A C style string in C++ is just an array of characters that is terminated by the null character `'\0'`
- A string literal is automatically terminated by it, `"abc"`
- Note that you can have an array of characters that is not null terminated, and thus is not really a string.
- What is the difference between:

```
char a1[] = "abcd",  
      a2[4] = { 'a', 'b', 'c', 'd' };  
sizeof a1 == ?  
sizeof a2 == ?  
a2[0] = 'z'; // Okay?  
a1 = "a"; // Okay?  
a2 = "b"; // Okay?
```

C-Strings

28

- Link to C-String functions

<http://www.cppreference.com/wiki/c/string/start>

Overflow and Underflow

29

- Overflow is when a variable is assigned a value that is too big for the data type.
 - ▣ Example: `int Total = 9999999999999999;`
 - ▣ Example `char Letter = 'ab';`
- Underflow is when a variable is assigned a value that is too small for the data type.
 - ▣ Example: `int Total = -9999999999999999;`

What's stored? `char Letter ='';`

Carefully choose your data type

30

- Don't choose a float data type for a variable when an integer will work.
 - ▣ Example, if you want to find the average of a set of integers, don't define Total as a float.
- Don't choose a char data type when a boolean variable will work.
- Don't choose a string data type when a char will work.

Choose your data type for

31

- ☐ the number of students in class?
- ☐ your final grade in this class?
- ☐ your final average in this class?
- ☐ for your grade on closed lab 1A?
- ☐ for the day of the month?
- ☐ for the distance to Knoxville?
- ☐ for your name?
- ☐ for your height?

Escape Sequences

32

- Escape sequences tell the compiler to treat characters in a special way
- `'\'` is the escape character
 - ▣ To create a newline in output use `\n`
`cout << "\n";`
or the newer alternative
`cout << endl;`
 - ▣ Other escape sequences:
 - `\t` -- a tab
 - `\\` -- a backslash character
 - `\"` -- a quote character

C++ Basics – Arithmetic Operators

33

Operators

+	add
-	subtract
*	multiply
/	divide
%	modulus

Example

```
int x, y=5, z=3;
```

```
x = y + z;  x = 8
```

```
x = y - z;  x = 2
```

```
x = y * z;  x = 15
```

```
x = y / z;  x = 1
```

```
x = y % z;  x = 2
```

C++ Basics – Auto Increment and Decrement

34

$x = 3$

□ Pre-increment/decrement

▣ $y = ++x$; same as $x = x + 1$;

$y = x$;

$x = 4$

$y = 4$

▣ $y = --x$; same as $x = x - 1$;

$y = x$;

$x = 2$

$y = 2$

□ Post-increment/decrement

▣ $y = x++$; same as $y = x$;

$x = x + 1$;

$y = 3$

$x = 4$

▣ $y = x--$; same as $y = x$;

$x = x - 1$;

$y = 3$

$x = 2$

Operator Shorthand

35

- C++ contains shorthand operators for frequently used operations

`+=` `count = count + 2;` becomes `count += 2;`

`*=` `bonus = bonus * 2;` becomes `bonus *= 2;`

`/=` `time = time / rush_factor;`
becomes `time /= rush_factor;`

`%=` `remainder = remainder % cnt1;`
becomes `remainder %= cnt1;`

C++ Basics – Relational and Logical Operators

36

□ Relational operators

`==` equal

`!=` not equal

`>` greater than

`>=` greater than or equal

`<` less than

`<=` less than or equal

□ Logical operators

`&&` and

`||` or

`!` not

The assignment operator '=' and comma operator ','

37

- What is the output of following?

```
int x=4, y=6;
```

```
cout << (y = x + 1) << endl;
```

```
cout << (x,y) << endl;
```

```
cout << ++x,x << endl;
```

```
cout << x++,x << endl;
```

Precedence of C++ operators

38

- For a complete listing of C++ operators and precedence there of, see <http://www.difranco.net/cop2220/op-prec.htm>

C++ Basics – Relational Operators

39

□ Assume x is 1, y is 4, $z = 14$

<i>Expression</i>	<i>Answer</i>
$x < y + z$	
$y == 2 * x + 3$	
$z <= x + y$	
$z > x$	
$x != y$	

C++ Basics – Logical Operators

40

□ Assume x is 1, y is 4, z = 14

<i>Expression</i>	<i>Answer</i>
<code>x <= 1 && y == 3</code>	
<code>x <= 1 y == 3</code>	
<code>!(x > 1)</code>	
<code>!x > 1</code>	
<code>!(x <= 1 y == 3)</code>	

Literals and named constants

41

- bool literals: true, false
- char literals: 'A', 'C', 'b'
- integer literals: 120, 110
- floating-point literals: 11.00, 1.23
- By default, floating-point literals are double.
- `const float PI=3.14;`
- `const int MAX=100;`

Type Casting

42

- Type casting (Type conversion) is the converting of an expression from one type to another type. Does NOT change the variable's data type.
- Different ways of type-casting
 - ▣ Implicit conversion (type coercion)
 - ▣ Explicit conversion

Type Casting (2)

43

- Implicit conversion (type coercion) is the automatic conversion of a value from one data type to another
- Explicit conversion requires the programmer to write code to initiate the conversion from one type to another type.

Explicit Type Casting (4)

44

- c-like cast notation

- syntax: `(NewType) expression`

- example: `float x = (float) 123;`

- functional notation

- syntax: `NewType (expression)`

- example: `float x = float (123);`

Skipping Enumerated Types

45

Lvalues and Rvalues

46

- An lvalue is the memory location/address of the item being referenced
- An rvalue is the value stored at the location of the item being referenced.

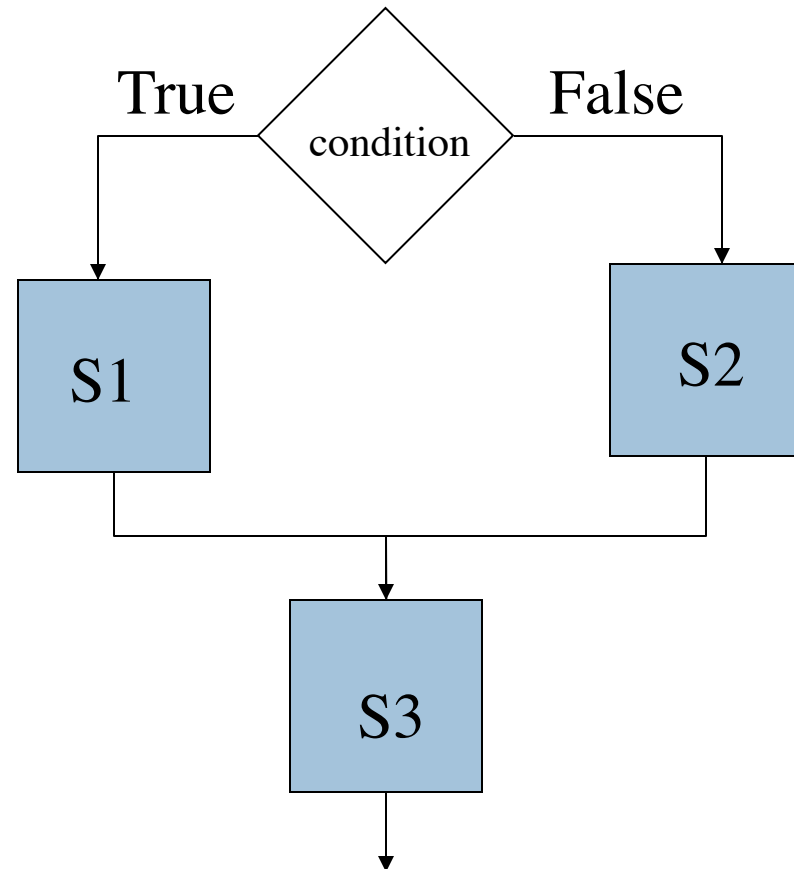
□ Ex. `x = x + 1;`

The 'x' on the left side of the assignment operator is referring to the location of 'x' while the 'x' on the right side is referring the value stored at location 'x'.

If statements

47

```
if (condition)
{
    S1;
}
else
{
    S2;
}
S3;
```



if Statement

48

□ **if** (*expression*)
 action

Example:

```
char a1 = 'A', a2 = 'C';  
int match = 0;  
if (a1 == a2)  
{  
    match++;  
}
```


if-else Statement

49

□ **if** (*expression*)

action 1

else

action 2

Example:

```
char a1 = 'A', a2 = 'C';  
int match = 0, gap = 0;  
if (a1 == a2)  
{  
    match++;  
}  
else  
{  
    gap++;  
}
```

Inequalities

50

- Be careful translating inequalities to C++
- if x is 10, y is 5, and z is 2, then

if $(x < y < z)$

evaluates as true as follows:

$x < y$ is false and has value zero while $0 < z$ is true

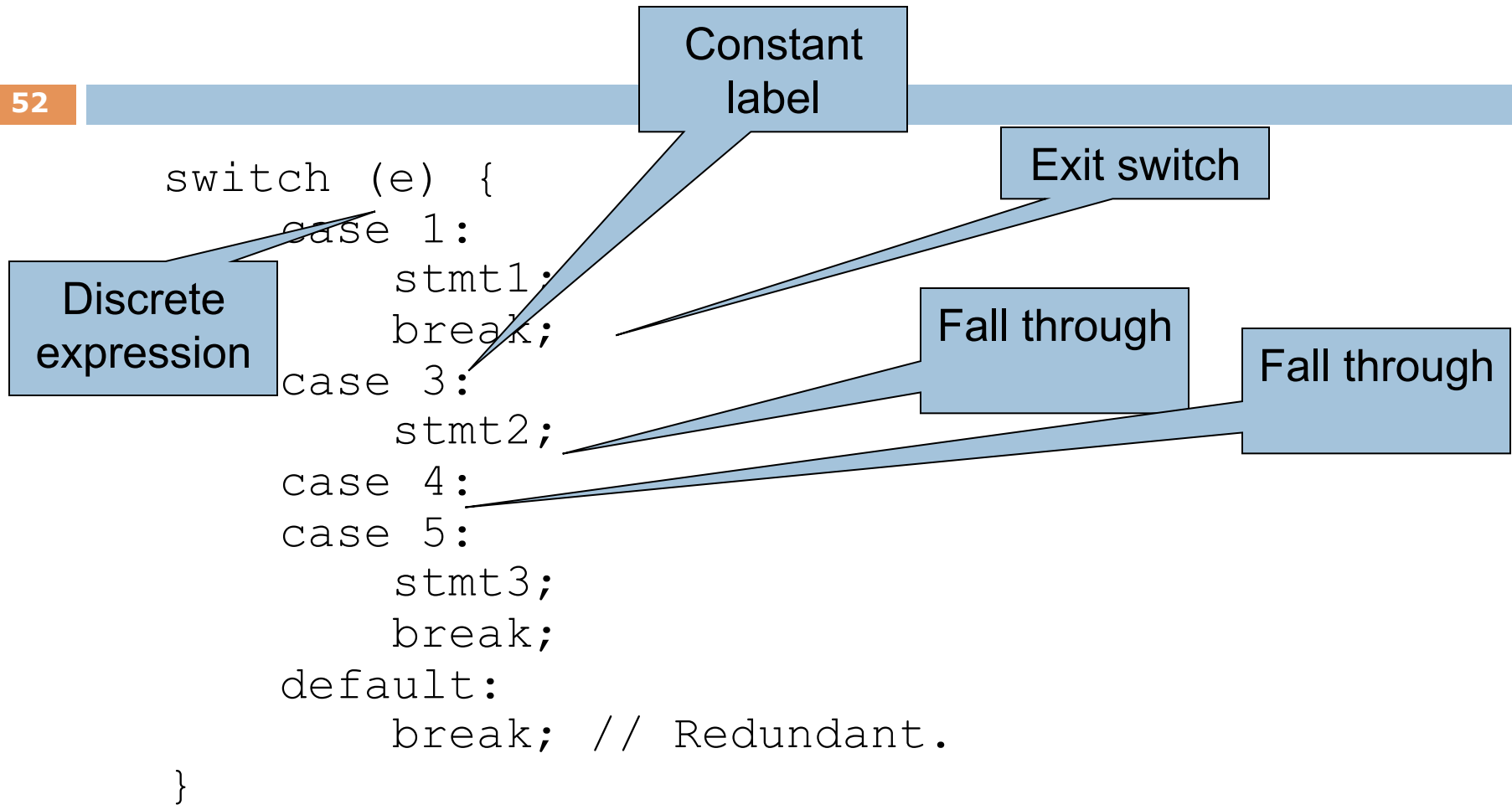
Pitfall: Using `=` or `==`

51

- `' = '` is the assignment operator
 - ▣ Used to assign values to variables
 - ▣ Example: `x = 3;`
- `' == '` is the equality operator
 - ▣ Used to compare values
 - ▣ Example: `if (x == 3)`
- The following is not reported as an error:
`if (x = 3)`
but stores 3 in x and since the result, 3, is non-zero, the expression is true

switch statement

52



Questions

53

- How do you round a float number?
- How do you round-up a number to a multiple of another number?
 - ▣ Example: 23 rounded up to a multiple of 7?
 - ▣ How do you round to a power of 2?
- How do you find the integral part of a float number?
- How do you swap the byte order of an integer?

for Statement

54

```
for( expr1; expr2; expr3 )  
    statement
```

- *Expr1* – defines initial conditions
- *Expr2* – tests for continued looping
- *Expr3* – updates loop

for Statement

55

Example

```
for(i = 1, sum = 0; i <= 4; i++)  
    sum = sum+i;
```

Iteration 1: $\text{sum}=0+1=1$

Iteration 2: $\text{sum}=1+2=3$

Iteration 3: $\text{sum}=3+3=6$

Iteration 4: $\text{sum}=6+4=10$

While statements

56

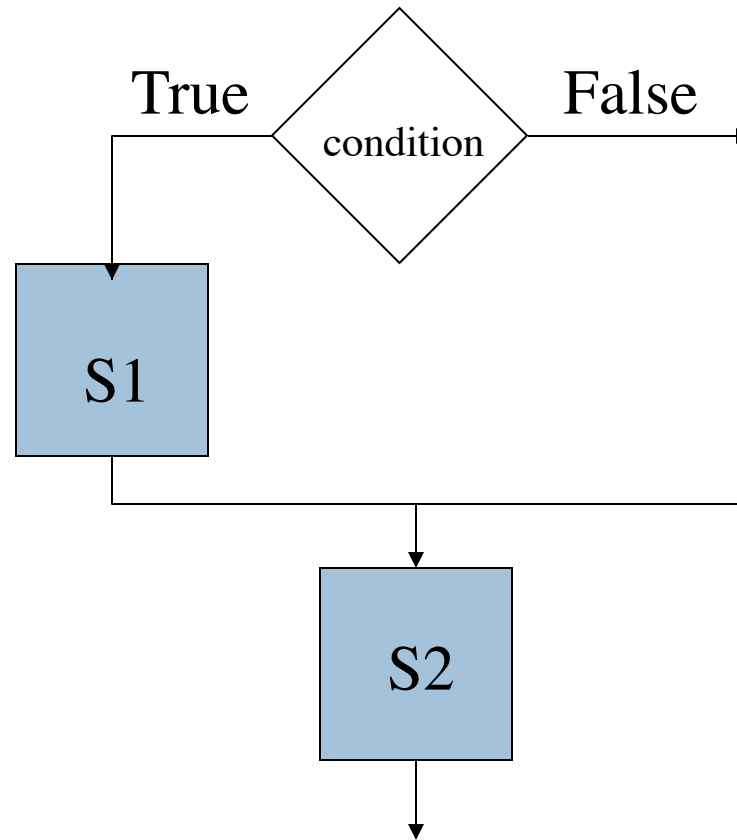
```
while (condition)
```

```
{
```

```
    S1;
```

```
}
```

```
S2;
```



while Statement

57

Example

```
int x = 0;  
while(x != 3)  
{  
    x = x + 2;  
}
```

Infinite loop!

Iteration 1: $x=0+1=1$

Iteration 2: $x=1+1=2$

Iteration 3: $x=2+1=3$

Iteration 4: don't exec

break and continue Statements

58

- break will immediately exit the **nearest** enclosing loop.
- continue will **immediately re-test** and possibly continue the nearest enclosing loop.

```
int x = 0;
while(x < 13)
{
    if (x%2)
    {
        x = x + 2;
        break;
    }
    x = x + 1;
}
```

```
int x = 0;
while(x < 13)
{
    if (x%2)
    {
        x = x + 2;
        continue;
    }
    x = x + 1;
}
```

C++ I/O streams - input

59

- Standard I/O input stream: `cin`

Ex: `int x;`

`char c1, c2, c3;`

`cin >> x >> c1 >> c2 >> c3;`

If the following input is typed: `23 a b c`

Then, `x = 23, c1 = 'a', c2 = 'b', c3 = 'c'`

(will ignore white spaces)

C++ I/O streams - output

60

- Standard I/O output stream: `cout`

Ex:

```
int x = 1;  
char c1 = '#';  
cout << "MTSU is " << c1 << x << '!' << endl;
```

The following output is displayed: **MTSU is #1!**

I/O Streams Usage

61

- Must include iostream header file
`#include <iostream>`
- There are ways to format the output to specify parameters such as the width of a field, the precision, and the output data type. We will skip this.

File Input & Output Stream

62

- File Input Stream (ifstream)
- ifstream data type represents a stream of characters coming from an input file
- ifstream is a special type of istream. In other words, all of the istream operations are also valid for the ifstream type.
- File Output Stream (ofstream)
- ofstream represents a stream of characters going to an output file.
- ofstream is a special type of ostream. In other words, all of the ostream operations are also valid for the ofstream type.

File I/O

63

- To read from a file, you must follow these steps

- Include the header file `fstream`

`#include <fstream>`

- Declare a file stream variable:

`ifstream inFile;` // Declare an input file stream variable

- Prepare the file for reading by using a method named `open`:

Syntax: `FileStreamVariable.open(a C string file name);`

Example: `inFile.open("data.txt");`

- Read data from the file by using all of the operations related with `cin`

`inFile >> i >> str1;` // Read an integer and a C++ string from

`inFile`

- After finishing all reads, close the file by using a method named `close`:

Syntax: `FileStreamVariable.close();`

Example: `inFile.close();`

File Input

64

```
#include <ifstream>
using namespace std;
int main()
{
    int value1, value2;
    ifstream InFile;
    InFile.open("ex.data");

    InFile >> value1;
    InFile >> value2;
```

Data File: ex.data

InFile

15

Values in data
file such as:

17

29 14 12

File I/O

65

- To write to a file, you must follow these steps

- Include the header file fstream

`#include <fstream>`

- Declare a file stream variable:

`ofstream outFile;` //Declare an output file stream variable

- Prepare the file for writing by using a method named open:

Syntax: `FileStreamVariable.open(a C string file name);`

Example: `outFile.open("data.txt");`

- Write the file by using all of the operations related with cout.

`outFile << "This is an example!" << endl;` // Write the string to

`outFile`

- Close the file by using a method named close:

Syntax: `FileStreamVariable.close();`

Example: `outFile.close();`

Success/Failure of File operations

66

- How does one check if the end of an input file is reached?

Answer: check the stream name

`while (InFile)`

- Meaning: Returns true if the last operation on the file succeeded, otherwise false.
- Or, combining the reading and testing as:

`while (InFile >> value)`

File Input Example: How to check if a file opening or file reading operation succeeds?

67

```
#include <ifstream>
using namespace std;
int main()
{
    int value1, value2;
    ifstream InFile;
    InFile.open("ex.data");
    if (!InFile) {
        cerr<<"Error"<<endl;
        return -1;
    }
}
```

Data File: ex.data

InFile

Values in data
file such as:

10 15 17

29 14 12

File Input Example: How to check if a file opening or file reading operation succeeds?

68

```
#include <ifstream>
#include <cassert>
using namespace std;

int main()
{
    int value1, value2;
    ifstream InFile;
    InFile.open("ex.data");
    assert(InFile);
```

Data File: ex.data

InFile

Values in data
file such as:

10 15 17

29 14 12

Arrays

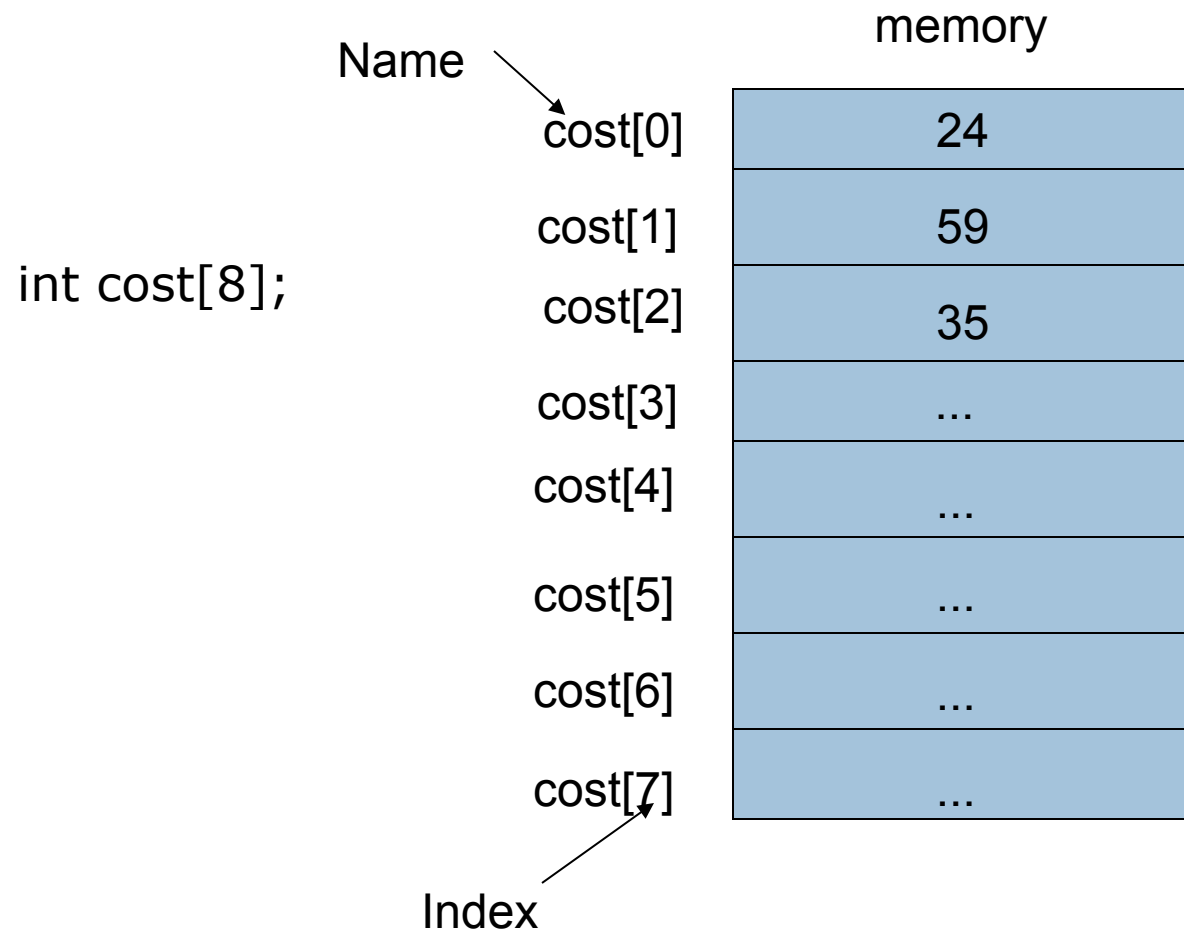
Arrays

- What is an Array?
- Declaration of an array
- Accessing values in an array
- Initializing array
- Loops and arrays
- Copying arrays
- Passing an array to function as argument
- Array of structures
- Multidimensional Arrays
- Character Arrays

Arrays

- They are special kind of data type
- An array is a consecutive group of memory locations that all have the same **name** and the same **type**
- In C++ each array has
 - ▣ name
 - ▣ data type
 - ▣ size

Storage of an array



Declaration of Arrays

Data_Type arrayName[numberOfElements] ;

For example ,

```
int age[ 10 ] ;
```

```
float cost[30];
```

- The size of the array must be a constant.
- The array index always starts at zero.
- The programmer must make sure the value of the array subscript used is never larger than the size the of the array.
- What happens here:

```
cout << age[-1] << endl;
```

```
cout << age[10] << endl;
```

```
age[10] = 4;
```

Referring to array elements

□ Examples:

```
cout<<age[4]; //print an array element
```

```
age[4]=55; //assign value to an array element
```

```
cin>>age[4]; //input element 4
```

```
for(int i=0; i < 10; ++i)
```

```
    cin >> age[i];
```

Loops and arrays : initializing

Method 1:

```
int age [ 10 ] ;
```

```
for ( int i = 0; i < 10 ; i++ )  
{  
    age [ i ] = 0 ;  
}
```

Method 2:

```
int age [ 10 ] ={0};
```

How about this?

```
int age [ 10 ] ={3};
```

Simple Array example

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

int main()
{
    int grades[10], count, sum;
    for (count=0; count < 10; ++count)
    {
        cin >> grades[count];
        sum += grades[count];
    };
    cout << "The average is " << sum / 10.0 << endl;
    return 0;
}
```

Exercise 1

- Modify the previous example to read until a negative one is read or until ten items are read. Continue to use the for loop.

Exercise 2

- Modify the example 1 to find the variance of the numbers read. The variance is found by the formula:
$$\frac{(\text{score squared minus the average score squared})}{\text{number of scores}}$$

Exercise 3

- Modify the example 1 to print the scores read in reverse order.

Multidimensional Arrays

- Multidimensional arrays can be described as "arrays of arrays". For example, a two dimensional array can be imagined as a table with rows and columns, all of them of a same uniform data type.


Columns			Rows

Matrix Example

81


```
int matrix [ 8 ] [ 10 ];  
for ( int row = 0 ; row < 8 ; ++row )  
    for ( int col = 0 ; col < 10 ; ++col )  
        cin >> matrix [ row ] [ col ] ;
```

After outer loop of input



[0]	5	2	...

After second outer loop of input



[0]	5	2	...
[1]	7	0	...

Functions

Functions: Overview

83

- Why use functions?
 - ▣ It's a way of parameterizing a piece of code.
 - ▣ It's a way of not repeating code.
- What happens when you call a function?
 - ▣ Pushes parameters on the stack (or in registers).
 - ▣ Pushes return location.
 - ▣ Makes a jump to the function code.
 - ▣ Returns a value somewhere.
 - ▣ Pops stack, jumps back.

Two ways to pass parameters to a function:

84

- Call by “**value**”
 - ▣ Useful for “protecting” the values of the variables passed to a function.
- Call by “**reference**”
 - ▣ Useful for “changing” multiple values.
- Define “**formal**” and “**actual**” function parameters

“FindMax” Example

85

```
int FindMax(int x, int y)
{
    int maximum;

    if(x>=y)
        maximum = x;
    else
        maximum = y;

    return maximum;
}
```

“FindMax” Example (cont.)

86

```
#include <iostream>
using namespace std;
int FindMax(int, int);

int main()
{
    int firstnum, secnum, max;

    cout << "\nEnter two numbers: ";
    cin >> firstnum >> secnum;

    max=FindMax( firstnum, secnum);

    cout << "The maximum is " << max
         << endl;

    return 0;
}
```

```
int FindMax(int x, int y)
{
    int maximum;

    if(x>=y)
        maximum = x;
    else
        maximum = y;

    return maximum;
}
```

Where is the function
prototype? Call?
actual and formal
parameters?

Calling a function by “value”

87

Calling a function by “value”

- The function receives a **copy** of the actual parameter values.
- The function **cannot** change the values of the actual parameters.

Calling a function by “reference”

88

- The formal parameter becomes an **alias** for the actual parameter.
- The function **can** change the values of the actual parameters.
- Formal parameters must be declared as **reference** variables!

Calling a function by reference (cont.)

89

```
#include <iostream>
void NewVal(float&, float&);

int main()
{
    float firstNum, secNum;
    cout << "Enter two numbers: ";
    cin >> firstNum >> secNum;

    NewVal(firstnum, secnum);
    cout << firstNum << secNum << endl;
    return 0;
}

void NewVal(float& xNum, float& yNum)
{
    xNum = 89.5;
    yNum = 99.5;
}
```

The "const" modifier

90

- Call by reference is the *preferred* way to pass a large structure or class instances to functions, since the entire structure need not be copied each time it is used!!

The "const" modifier

- C++ provides us with protection against accidentally changing the values of variables passed by reference with the *const* operator

```
int FindMax(const int& x, const int& y);
```

Passing by value or Passing by Reference?

92

- How to decide whether a parameter should be passed by value or reference?
- Do you want to return a new value through the parameter?
 - ▣ If Yes: pass by reference
 - ▣ If No: pass by value
- Is the size of the parameter very large (i.e. a structure, an object)?
 - ▣ If Yes: pass by reference
 - ▣ If No: pass by value
 - ▣ Reason: pass-by-reference is more efficient since only the address is copied and not the parameter value.

Passing 1-dimensional Arrays as Parameters

93

- Arrays are always passed by reference.

- Cannot append & to the array type

Example:

```
int sum(int oneArray[], int arraySize); // correct
```

```
int sum(int& oneArray[], int arraySize); // wrong no &
```

- When an array is passed as an argument, only its base address is copied.
 - Base address: The memory address of the first element of an array.
- When an array is passed as a parameter, another parameter is needed to indicate its size.

Passing 2-dimensional Arrays as Arguments?

94

- When a 2-dimensional array is passed as an argument, only its base address is copied.
- Additionally, we must specify the size of 2nd dimension – the number of columns (the size of 1st dimensional is omitted)

Example:

```
void IdentityMatrix(float matrix[][20], int numRows, int numColumns)
```

- Why? Two dimensional arrays are stored by row major order. In order to locate an array element, we have to know the number of columns.
- When a 2-dimensional array is passed as a parameter, parameters are needed to indicate the size of the two dimensions.

Function sumarray()

95

```
/*Input: numbers - an array of size number of
  integers
* Process: finds the sum of the first n elements in
  the numbers array.
. Output: returns the sum */
```

```
int SumArray(int numbers[ ], int size)
{
    int sum=0;
    for (int count = 0; count < size; count++)
        sum += numbers[count];
    return(sum);
}
```

Finding the Largest Element of an Array

96

```
/*Input:numbers - an array of size number of integers
* Process:finds the largest value in the array
Output:returns the maximum value within the array */
int FindMax(int numbers[], int size)
{
    int largestSoFar;
    largestSoFar = numbers[0];

    for (int count = 1; count < size; count++)
        if (largestSoFar < numbers[count])
            largestSoFar = numbers[count];
    return(largestSoFar);
}
```


Scope

97



Scope

98

- Scope is the context where a name is “visible”.
- C++ uses **Static (lexical)** scope:

```
void foo() {  
    int i, j;  
    ...  
    goo();    // Okay?  
}  
void goo() {  
    int j;  
    i;        // Okay?  
    j;  
}
```

C++ Scopes

99

- Local (block) uses curly braces
- Namespace 'using' statement
- Class to be covered later

Global Objects

100

- How do you share an object between files?

Automatic Objects

101

□ Block

```
void foo()  
{  
    int i;  
    ...  
    {  
        int j;  
        ...  
    }  
}
```

Namespaces

102

- Namespaces group functions and variables under a prefix
- Used to avoid name collisions.
- Declared by:

```
namespace <name>
{
    ...
}
```

- Symbols accessed by `<name>::symbol`
- Namespaces can be nested.
- There are 2 forms:
 - `using <name>::symbol;`
 - `using namespace <name>;`

```
#include <iostream>

namespace Albert {
    void greet() { std::cout << "Hi from Albert!"; }
}
namespace John {
    void greet() { std::cout << "Hello from John"; }
    void squeak() { std::cout << "squeak! Squeak!"; }
}
```

```
Albert::greet(); // Hi from Albert!
greet(); // Compile error. No symbol "greet"
```

```
Using John::greet;
greet(); // Hello from John!
Albert::greet(); // Hi From Albert!
squeak(); // Compile error. No symbol "squeak"
```

```
using namespace John;
greet(); // Hello From John!
Albert::greet(); // Hi From Albert!
squeak(); // squeak! squeak!
```

Program Style

104

- A program written with attention to style
 - ▣ is easier to read
 - ▣ easier to correct
 - ▣ easier to change

Program Style - Indenting

105

- *Items* considered a group should look like a group

- Skip lines between logical groups of statements
- Indent statements within statements

```
    if (x == 0)
        statement;
```

- Braces `{}` create groups

- Indent within braces to make the group clear
- Braces placed on separate lines are easier to locate

- Follow the Programming Style suggestion given in Chapter 1 of the text book