IS 0020

Program Design and Software Tools

Introduction to C++ Programming
Spring 2005

Lecture 1
Jan 6, 2005

Course Information

- Lecture:
  - James B D Joshi
  - Tuesdays/Thursdays: 1:00-2:15 PM
    - Office Hours: Wed 3:00-5:00PM/Appointment
  - GSA: TBA
- Pre-requisite
  - IS 0015 Data Structures and Programming Techniques
- Textbook
  - ***C++ How to Program***- Fourth Edition, by H. M. Deitel, P. J. Deitel, Prentice Hall, New Jersey, 2003, ISBN: 0-13-038474.

Course Information

- Course Description
  - An introduction to the development of programs using C++.
  - Emphasis is given to the development of program modules that can function independently.
    - Object-oriented design
  - The theory of data structures and programming language design is continued.
- Grading
  - 2 Exams 30%
  - Assignments/quizzes 70%

Course Policy

- Your work MUST be your own
  - Zero tolerance for cheating
  - Discussing problems is encouraged, but each must present his own answers
  - You get an F for the course if you cheat in anything however small – NO DISCUSSION
- Homework
  - There will be penalty for late assignments (15% each day)
  - Ensure clarity in your answers – no credit will be given for vague answers
  - Homework is primarily the GSA's responsibility
- Check webpage for everything!
  - You are responsible for checking the webpage for updates

5

Computer Languages

- Machine language
  - Generally consist of strings of numbers - Ultimately 0s and 1s - Machine-dependent
  - Example: `+1300042774`
    `+1400593419`
- Assembly language
  - English-like abbreviations for elementary operations
  - Incomprehensible to computers - Convert to machine language
  - Example:
```
LOAD   BASEPAY
ADD    OVERPAY
STORE  GROSSPAY
```
- High-level languages
  - Similar to everyday English, use common mathematical notations
  - Compiler/Interpreter
  - Example:
```
grossPay = basePay + overTimePay
```


6

History of C and C++

- History of C
  - Evolved from two other programming languages
    - BCPL and B: “Typeless” languages
  - Dennis Ritchie (Bell Lab): Added typing, other features
  - 1989: ANSI standard/ ANSI/ISO 9899: 1990
- History of C++
  - Early 1980s: Bjarne Stroustrup (Bell Lab)
  - Provides capabilities for object-oriented programming
    - Objects: reusable software components
    - Object-oriented programs
- Building block approach” to creating programs
  - C++ programs are built from pieces called classes and functions
  - C++ standard library: Rich collections of existing classes and functions

Structured/OO Programming

- Structured programming (1960s)
  - Disciplined approach to writing programs
  - Clear, easy to test and debug, and easy to modify
  - E.g.Pascal:1971: Niklaus Wirth
- OOP
  - "Software reuse"
  - "Modularity"
  - "Extensible"
  - More understandable, better organized and easier to maintain than procedural programming

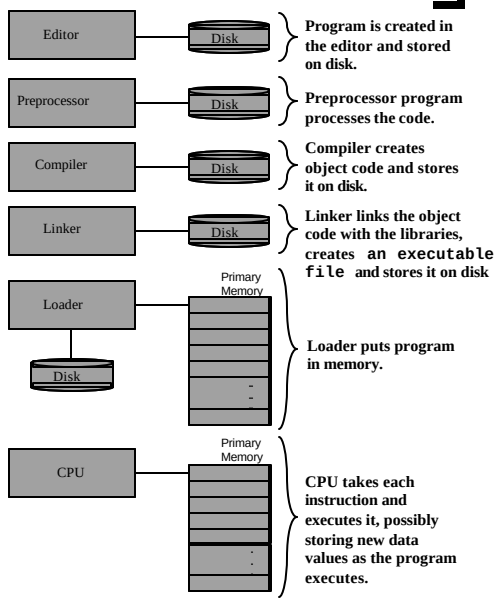
Basics of a Typical C++ Environment

- C++ systems
  - Program-development environment
  - Language
  - C++ Standard Library
- C++ program names extensions
  - .cpp
  - .cxx
  - .cc
  - .C

Basics of a Typical C++ Environment

Phases of C++ Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



Basics of a Typical C++ Environment

- Common Input/output functions
  - **`cin`**
    - Standard input stream
    - Normally keyboard
  - **`cout`**
    - Standard output stream
    - Normally computer screen
  - **`cerr`**
    - Standard error stream
    - Display error messages
- Comments: C's comment /* .. */ OR Begin with **`// or`**
- Preprocessor directives: Begin with **`#`**
  - Processed before compiling

A Simple Program: Printing a Line of Text

- Standard output stream object
  - **`std::cout`**
  - "Connected" to screen
  - **`<<`**
    - Stream insertion operator
    - Value to right (right operand) inserted into output stream
- Namespace
  - **`std::`** specifies that entity belongs to "namespace" **`std`**
  - **`std::`** removed through use of **`using`** statements
- Escape characters: **`\`**
  - Indicates "special" character output

Outline

fig01_02.cpp
(1 of 1)

fig01_02.cpp
output (1 of 1)

```
1  // Fig. 1.2: fig01_02.cpp
2  // A first program in C++.
3  #include <iostream>
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n";
9
10    return 0;   // indicate that program ended successfully
11
12 } // end function main
```

```
Welcome to C++!
```

A Simple Program: Printing a Line of Text

Escape Sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double quote character.

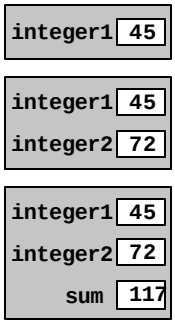
Memory Concepts

- Variable names
  - Correspond to actual locations in computer's memory
  - Every variable has name, type, size and value
  - When new value placed into variable, overwrites previous value

  - std::cin >> integer1;
  - Assume user entered 45

  - std::cin >> integer2;
  - Assume user entered 72

  - sum = integer1 + integer2;



15

Arithmetic

- Arithmetic calculations
  - `*` : Multiplication
  - `/` : Division
    - Integer division truncates remainder
      - `7 / 5` evaluates to 1
  - `%` : Modulus operator returns remainder
      - `7 % 5` evaluates to 2

Operator(s)	Operation(s)	Order of evaluation (precedence)
`()`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*`, `/`, or `%`	Multiplication Division Modulus	Evaluated second. If there are several, they re evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

16

Decision Making: Equality and Relational Operators

- `if` structure
  - Make decision based on truth or falsity of condition
    - If condition met, body executed
    - Else, body not executed
- Equality and relational operators
  - Equality operators
    - Same level of precedence
  - Relational operators
    - Same level of precedence
  - Associate left to right

Decision Making: Equality and Relational Operators

17

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	`>`	`x > y`	**x** is greater than **y**
<	`<`	`x < y`	**x** is less than **y**
≥	`>=`	`x >= y`	**x** is greater than or equal to **y**
≤	`<=`	`x <= y`	**x** is less than or equal to **y**
Equality operators			
=	`==`	`x == y`	**x** is equal to **y**
≠	`!=`	`x != y`	**x** is not equal to **y**

Algorithms /pseudocode

18

- Computing problems
  - Solved by executing a series of actions in a specific order
- Algorithm: a procedure determining
  - Actions to be executed
  - Order to be executed
  - Example: recipe
- Program control
  - Specifies the order in which statements are executed
- Pseudocode
  - Artificial, informal language used to develop algorithms
  - Similar to everyday English

Control Structures

- Sequential execution
  - Statements executed in order
- Transfer of control
  - Next statement executed *not* next one in sequence
  - Structured programming – "goto"-less programming
- 3 control structures to build any program
  - Sequence structure
    - Programs executed sequentially by default
  - Selection structures
    - **`if`**, **`if/else`**, **`switch`**
  - Repetition structures
    - **`while`**, **`do/while`**, **`for`**

Keywords

- C++ keywords
  - Cannot be used as identifiers or variable names

C++ Keywords

Keywords common to the C and C++ programming languages

`auto`	`break`	`case`	`char`	`const`
`continue`	`default`	`do`	`double`	`else`
`enum`	`extern`	`float`	`for`	`goto`
`if`	`int`	`long`	`register`	`return`
`short`	`signed`	`sizeof`	`static`	`struct`
`switch`	`typedef`	`union`	`unsigned`	`void`
`volatile`	`while`			

C++ only keywords

`asm`	`bool`	`catch`	`class`	`const_cast`
`delete`	`dynamic_cast`	`explicit`	`false`	`friend`
`inline`	`mutable`	`namespace`	`new`	`operator`
`private`	`protected`	`public`	`reinterpret_cast`	
`static_cast`	`template`	`this`	`throw`	`true`
`try`	`typeid`	`typename`	`using`	`virtual`
`wchar_t`				

Control Structures

- Flowchart
  - Graphical representation of an algorithm
  - Special-purpose symbols connected by arrows (flowlines)
  - Rectangle symbol (action symbol)
    - Any type of action
  - Oval symbol
    - Beginning or end of a program, or a section of code (circles)

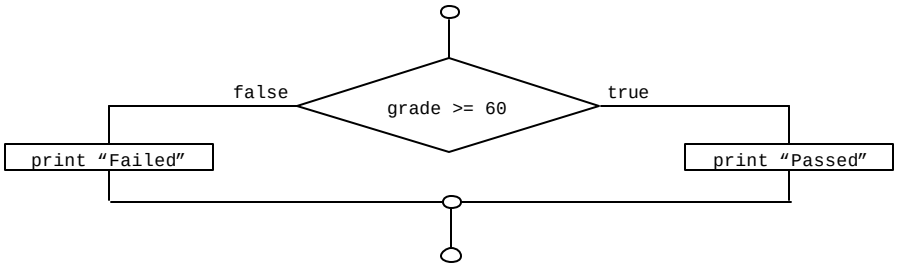
Exercise: Find greater of three numbers

`if/else` Selection Structure

- Ternary conditional operator (**`?:`**)
  - Three arguments (condition, value if **`true`**, value if **`false`**)
- Code could be written:

```
cout << ( grade >= 60 ? "Passed" : "Failed" );
```

Condition **Value if true** **Value if false**



while Repetition Structure

- Repetition structure
  - **`Counter-controlled`**
    - **`While/do while`** loop: repeated until condition becomes false
    - **`For`**: loop repeated until counter reaches certain value Flowchart representation?
  - **`Sentinel value`**
    - Indicates "end of data entry"
    - Sentinel chosen so it cannot be confused with regular input
- Example

```
int product = 2;
while ( product <= 1000 ) {
   product = 2 * product;
    cout << product;
}
```

Flowchart representation?
What is the output?

switch Multiple-Selection Structure

- **`switch`**
  - Test variable for multiple values
  - Series of **`case`** labels and optional **`default`** case

```
switch ( variable ) {
     case value1:       // taken if variable == value1
            statements
            break;      // necessary to exit switch

     case value2:
     case value3:       // taken if variable == value2 or == value3
            statements
            break;

     default:           // taken if none matches
            statements
            break;
}
```

25

break and **continue** Statements

- **`break`** statement
  - Immediate exit from **`while`**, **`for`**, **`do/while`**, **`switch`**
  - Program continues with first statement after structure
- Common uses
  - Escape early from a loop
  - Skip the remainder of **`switch`**

26

Logical Operators

- Used as conditions in loops, **if** statements
- **`&&`** (logical **`AND`**)
  - **`true`** if both conditions are **`true`**

```
if ( gender == 1 && age >= 65 )
   ++seniorFemales;
```

- **`||`** (logical **`OR`**)
  - **`true`** if either of condition is **`true`**

```
if ( semesterAverage >= 90 || finalExam >= 90 )
   cout << "Student grade is A" << endl;
```

Logical Operators

- **!** (logical **NOT**, logical negation)
  - Returns **true** when its condition is **false**, & vice versa

```
if ( !( grade == sentinelValue ) )
   cout << "The next grade is " << grade << endl;
```

Alternative:

```
if ( grade != sentinelValue )
   cout << "The next grade is " << grade << endl;
```


Confusing Equality (==) and Assignment (=) Operators

- Common error
  - Does not typically cause syntax errors
- Aspects of problem
  - Expressions that have a value can be used for decision
    - Zero = false, nonzero = true
  - Assignment statements produce a value (the value to be assigned)

```
if == was replaced with =
   if ( payCode = 4 )
      cout << "You get a bonus!" << endl;
```

What happens?

Confusing Equality (==) and Assignment (=) Operators

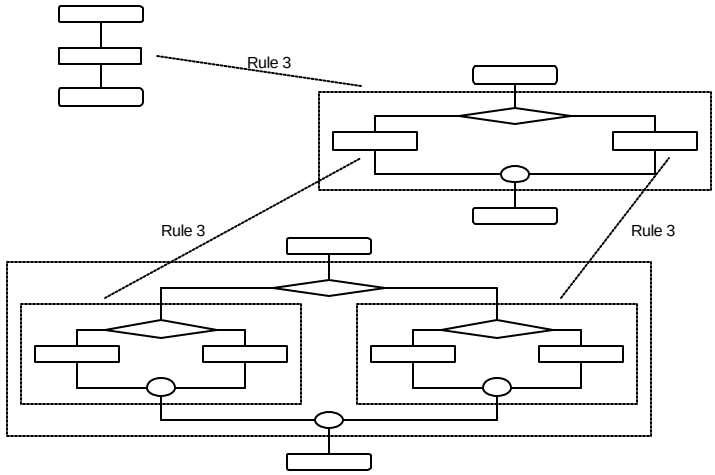
- *Lvalues*
  - Expressions that can appear on left side of equation
  - Can be changed
    ```
    x = 4;
    ```
- *Rvalues*
  - Only appear on right side of equation
  - Constants, such as numbers (i.e. cannot write `4 = x;`)
- *Lvalues* can be used as *rvalues*, but not vice versa

Structured-Programming Summary

- Structured programming
  - Programs easier to understand, test, debug and modify
- Rules for structured programming
  - Only use single-entry/single-exit control structures
  - Rules
    1) Begin with the "simplest flowchart"
    2) Any rectangle (action) can be replaced by two rectangles (actions) in sequence
    3) Any rectangle (action) can be replaced by any control structure (sequence, if, if/else, switch, while, do/while or for)
    4) Rules 2 and 3 can be applied in any order and multiple times

Structured-Programming Summary

Representation of Rule 3 (replacing any rectangle with a control structure)



Program Components in C++

- Modules: *functions* and *classes*
- Programs use new and "prepackaged" modules
  - New: programmer-defined functions, classes
  - Prepackaged: from the standard library
- Functions invoked by function call
  - Function name and information (arguments) it needs
- Function definitions
  - Only written once
  - Hidden from other functions