IS 0020
Program Design and Software Tools
Introduction to C++ Programming

Lecture 1
Jan 6, 2004

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
- Quiz 10% (in the beginning of the class; on previous lecture)
- Homework/Programming Assignments 40% (typically every week)
- Midterm 25%
- Comprehensive Final 25%

Lecture:
- James B D Joshi
- Tuesdays: 6:00-8.50 PM
  - One (two) 15 (10) minutes break(s)
- Office Hours: Wed 3:00-5:00PM/Appointment

Pre-requisite
- IS 0015 Data Structures and Programming Techniques

Textbook
Course Policy

- Your work MUST be your own
  - Zero tolerance for cheating
  - 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

Computer Languages

Machine language
- Only language computer directly understands
- Defined by hardware design
- Machine-dependent
- Generally consist of strings of numbers
  - Ultimately 0s and 1s
- Instruct computers to perform elementary operations
  - One at a time
- Cumbersome for humans
- Example:
  - +1300042774
  - +1400593419
  - +1200274027

Assembly language
- English-like abbreviations representing elementary computer operations
- Clearer to humans
- Intelligible to computers
- Translator programs (assemblers)
  - Convert to machine language
- Example:
  - LOAD BASEPAY
  - ADD OVERPAY
  - STORE GROSSPAY
Computer Languages

- **High-level languages**
  - Similar to everyday English, use common mathematical notations
  - Single statements accomplish substantial tasks
  - Assembly language requires many instructions to accomplish simple tasks
  - Translator programs (compilers)
    - Convert to machine language
  - Interpreter programs
    - Directly execute high-level language programs
  - Example:
    - `grossPay = basePay + overtimePay`

History of C and C++

- **History of C**
  - Evolved from two other programming languages
    - BCPL and B
    - “Typeless” languages
  - Dennis Ritchie (Bell Laboratories)
    - Added data typing, other features
    - Development language of UNIX
    - Hardware independent
      - Portable programs
    - 1989: ANSI standard
    - 1990: ANSI and ISO standard published
      - ANSI/ISO 9899: 1990

- **History of C++**
  - Extension of C
    - Early 1980s: Bjarne Stroustrup (Bell Laboratories)
  - Provides capabilities for object-oriented programming
    - Objects: reusable software components
      - Model items in real world
    - Object-oriented programs
      - Easy to understand, correct and modify
    - Hybrid language
      - C-like style
      - Object-oriented style
      - Both

C++ Standard Library

- C++ programs
  - Built from pieces called classes and functions
- C++ standard library
  - Rich collections of existing classes and functions
  - “Building block approach” to creating programs
    - “Software reuse”
Java

- 1991: Sun Microsystems
  - Green project
- 1995: Sun Microsystems
  - Formally announced Java at trade show
- Web pages with dynamic and interactive content
- Develop large-scale enterprise applications
- Enhance functionality of web servers
- Provide applications for consumer devices
  - Cell phones, pagers, personal digital assistants, …

Structured Programming

- Structured programming (1960s)
  - Disciplined approach to writing programs
  - Clear, easy to test and debug, and easy to modify
- Pascal
  - 1971: Niklaus Wirth
- Ada
  - 1970s - early 1980s: US Department of Defense (DoD)
  - Multitasking
    - Programmer can specify many activities to run in parallel

The Key Software Trend: Object Technology

- Objects
  - Reusable software components that model real world items
  - Meaningful software units
    - Date objects, time objects, paycheck objects, invoice objects, audio objects, video objects, file objects, record objects, etc.
    - Any noun can be represented as an object
  - More understandable, better organized and easier to maintain than procedural programming
  - Favor modularity
    - Software reuse
      - Libraries
        - MFC (Microsoft Foundation Classes)
        - Rogue Wave

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

Program is created in the editor and stored on disk.
Preprocessor program processes the code.
Compiler creates object code and stores it on disk.
Linker links the object code with the libraries, creates an executable file and stores it on disk.
Loader puts program in memory.

Common Input/output functions
- `cin`  
  - Standard input stream
  - Normally keyboard
- `cout`  
  - Standard output stream
  - Normally computer screen
- `cerr`  
  - Standard error stream
  - Display error messages

A Simple Program: Printing a Line of Text

Before writing the programs
- Comments
  - Document programs
  - Improve program readability
  - Ignored by compiler
- Single-line comment
  - Use C’s comment /* ... */ OR Begin with //
- Preprocessor directives
  - Processed by preprocessor before compiling
  - Begin with #

```
// Fig. 1.2: fig01_02.cpp
// A first program in C++.
#include <iostream>

int main()
{
    std::cout << "Welcome to C++!
";
    return 0;   // indicate that program ended successfully
} // end function main
```

Welcome to C++!
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 using name that belongs to “namespace”
- `std`: removed through use of `using` statements

Escape characters
- \: Indicates “special” character output

--

Another Simple Program: Adding Two Integers

Variables
- Location in memory where value can be stored
- Common data types
  - `int`: integer numbers
  - `char`: characters
  - `double`: floating point numbers
- Declare variables with name and data type before use
  - `int integer1;`
  - `int integer2;`
  - `int sum;`
- Can declare several variables of same type in one declaration
  - `int integer1, integer2, sum;`

---

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; does not advance to the next line.
`\a` | Alert: Sound the system bell.
`\b` | Backspace: Used to print a backslash character.
`\"` | Double quotes: Used to print a double quote character.

---

Input stream object
- `>>(stream extraction operator)`
  - Used with `std::cin`
  - Waits for user to input value, then press Enter (Return) key
  - Stores value in variable to right of operator
    - Converts value to variable data type
- `=` (assignment operator)
  - Assigns value to variable
  - Binary operator (two operands)
  - Example:
    ```
    sum = variable1 + variable2;
    ```
1 // Fig. 1.6: fig01_06.cpp
2 // Addition program.
3
4 #include <iostream>
5
6 int main()
7 {
8     int integer1; // first number to be input by user
9     int integer2; // second number to be input by user
10    int sum;      // variable in which sum will be stored
11
12    std::cout << "Enter first integer\n"; // prompt
13    std::cin >> integer1;                 // read an integer
14
15    std::cout << "Enter second integer\n"; // prompt
16    std::cin >> integer2;                  // read an integer
17
18    sum = integer1 + integer2;  // assign result to sum
19
20    std::cout << "Sum is " << sum << std::endl; // print sum
21
22    return 0;   // indicate that program ended successfully
23
24 } // end function main

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;

Arithmetic

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
<th>What is evaluated (precedence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>1</td>
<td>Evaluated first. If the parentheses are nested, the expressions 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.</td>
</tr>
<tr>
<td>* / %</td>
<td>2</td>
<td>Evaluated second. If there are several, they are evaluated left to right.</td>
</tr>
<tr>
<td>+ -</td>
<td>3</td>
<td>Evaluated last. If there are several, they are evaluated left to right.</td>
</tr>
</tbody>
</table>

- Using statements
  - Eliminate use of std:: prefix
  - Write cout instead of std::cout

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
  - using statements
    • Eliminate use of std:: prefix
    • Write cout instead of std::cout
## Decision Making: Equality and Relational Operators

<table>
<thead>
<tr>
<th>Relational operator</th>
<th>C++ equality operator</th>
<th>Example of C++ condition</th>
<th>Meaning of C++ condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>x &gt; y</td>
<td>x is greater than y</td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td>x &lt; y</td>
<td>x is less than y</td>
<td></td>
</tr>
<tr>
<td>≥</td>
<td>x ≥ y</td>
<td>x is greater than or equal to y</td>
<td></td>
</tr>
<tr>
<td>≤</td>
<td>x ≤ y</td>
<td>x is less than or equal to y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equality operators</th>
<th>C++ equality operator</th>
<th>Example of C++ condition</th>
<th>Meaning of C++ condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>x == y</td>
<td>x is equal to y</td>
<td></td>
</tr>
<tr>
<td>≠</td>
<td>x != y</td>
<td>x is not equal to y</td>
<td></td>
</tr>
</tbody>
</table>

### Algorithms

- 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

- **Pseudocode**
  - Artificial, informal language used to develop algorithms
  - Similar to everyday English
- **Not executed on computers**
  - Used to think out program before coding
  - Easy to convert into C++ program
  - Only executable statements
  - No need to declare variables

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
    - Selection structures
      - if, if/else, switch
  - Repetition structures
    - while, do/while, for

Keywords

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

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)
- **Single-entry/single-exit control structures**
  - Connect exit point of one to entry point of the next
  - Control structure stacking
Selection structure
• Choose among alternative courses of action
• Pseudocode example:
  
  If student’s grade is greater than or equal to 60
  Print “Passed”

  If the condition is true
  • Print statement executed, program continues to next statement
  If the condition is false
  • Print statement ignored, program continues
  • Indenting makes programs easier to read
  • C++ ignores whitespace characters (tabs, spaces, etc.)

Translation into C++
If student’s grade is greater than or equal to 60
Print “Passed”

if ( grade >= 60 )
  cout << "Passed";

Diamond symbol (decision symbol)
• Indicates decision is to be made
• Contains an expression that can be true or false
  • Test condition, follow path
• if structure
  • Single-entry/single-exit

Flowchart of pseudocode statement
A decision can be made on any expression:
zero = false
nonzero = true
Example:
3 - 4 = true

if/else Selection Structure
• if
  • Performs action if condition true
• if/else
  • Different actions if conditions true or false
• Pseudocode
  if student’s grade is greater than or equal to 60
  print “Passed”
  else
  print “Failed”
• C++ code
  if ( grade >= 60 )
    cout << "Passed";
  else
    cout << "Failed";
if/else Selection Structure

- Ternary conditional operator (?:)
  - Three arguments (condition, value if true, value if false)
- Code could be written:
  ```cpp
  cout << ( grade >= 60 ? "Passed" : "Failed" );
  ```

if/else Selection Structure

- Nested if/else structures
  - One inside another, test for multiple cases
  - Once condition met, other statements skipped
- Example
  ```cpp
  if ( grade >= 90 )         // 90 and above
      cout << "A";
  else if ( grade >= 80 )   // 80-89
      cout << "B";
  else if ( grade >= 70 )   // 70-79
      cout << "C";
  else if ( grade >= 60 )   // 60-69
      cout << "D";
  else                     // less than 60
      cout << "F";
  ```

if/else Selection Structure

- Compound statement
  - Set of statements within a pair of braces
  ```cpp
  if ( grade >= 60 )
      cout << "Passed.\n";
  else
      cout << "Failed.\n"
      cout << "You must take this course again.\n";
  ```
  - Without braces,
    ```cpp
    cout << "You must take this course again.\n";
    ```
  - always executed

- Block
  - Set of statements within braces
while Repetition Structure

• Repetition structure
  – Action repeated while some condition remains true
  – Pseudocode
    while there are more items on my shopping list
      Purchase next item and cross it off my list
    – while loop repeated until condition becomes false
• Example
  ```cpp
  int product = 2;
  while ( product <= 1000 )
    product = 2 * product;
  ```

Counter-Controlled Repetition

• Counter-controlled repetition
  – Loop repeated until counter reaches certain value
• Definite repetition
  – Number of repetitions known
• Example
  A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.

```cpp
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
  int total;        // sum of grades input by user
  int gradeCounter; // number of grade to be entered next
  int grade;        // grade value
  int average;      // average of grades

  // initialization phase
  total = 0;          // initialize total
  gradeCounter = 1;  // initialize loop counter

  // function main begins program execution
  for ( ; gradeCounter <= 10; gradeCounter++ ), // number of grades to be entered
  {
    cout << "Enter grade (0-100): ";
    cin >> grade;
    total += grade; // sum grade
  }

  // compute class average
  average = total / gradeCounter;

  // display result
  cout << "Class average: ", average;
}
```
Sentinel-Controlled Repetition

Suppose problem becomes:
Develop a class-averaging program that will process an arbitrary number of grades each time the program is run
– Unknown number of students
– How will program know when to end?

• Sentinel value
– Indicates “end of data entry”
– Loop ends when sentinel input
– Sentinel chosen so it cannot be confused with regular input
  • -1 in this case

The counter gets incremented each time the loop executes. Eventually, the counter causes the loop to end.

Data type double used to represent decimal numbers.

static_cast<double>() treats total as a double temporarily (casting).

Required because dividing two integers truncates the remainder.

gradeCounter is an int, but it gets promoted to double.
```cpp
// display average with two digits of precision
cout << "Class average is " << setprecision( 2 ) << fixed << average << endl;
else // if no grades were entered, output appropriate message
    cout << "No grades were entered" << endl;
return 0;   // indicate program ended successfully
}

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;
}
```

- Example upcoming
  - Program to read grades (A-F)
  - Display number of each grade entered

- Details about characters
  - Single characters typically stored in a `char` data type
    - `char` is a 1-byte integer, so `char` can be stored as `int`
    - Can treat character as `int` or `char`
    - 97 is the numerical representation of lowercase 'a' (ASCII)
    - Use single quotes to get numerical representation of character
      ```cpp
      cout << "The character (" << 'a' << ") has the value "
           << static_cast< int > ( 'a' ) << endl;
      ```
      Prints
      `The character (a) has the value 97`
```cpp
// Fig. 2.22: fig02_22.cpp  // Counting letter grades.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

// function main begins program execution
int main()
{
    int grade;       // one grade
    int aCount = 0;  // number of As
    int bCount = 0;  // number of Bs
    int cCount = 0; // number of Cs
    int dCount = 0; // number of Ds
    int fCount = 0;  // number of Fs

    cout << "Enter the letter grades." << endl
         << "Enter the EOF character to end input." << endl;

    while ( ( grade = cin.get() ) != EOF ) {
        // determine which grade was input
        switch ( grade ) { // switch structure nested in while
            case 'A':        // grade was uppercase A
                case 'a':        // or lowercase a
                    ++aCount;    // increment aCount
                    break;        // necessary to exit switch
            case 'B':       // grade was uppercase B
                case 'b':       // or lowercase b
                    ++bCount;    // increment bCount
                    break;        // exit switch
            case 'C':       // grade was uppercase C
                case 'c':        // or lowercase c
                    ++cCount;     // increment cCount
                    break;        // exit switch
            case 'D':        // grade was uppercase D
                case 'd':       // or lowercase d
                    ++dCount;     // increment dCount
                    break;        // exit switch
            case 'F':        // grade was uppercase F
                case 'f':        // or lowercase f
                    ++fCount;   // increment fCount
                    break;        // exit switch
            case '
':       // ignore newlines,
                case '	':      // tabs,
                case ' '        // and spaces in input
                        break;        // exit switch
            default:         // catch all other characters
                cout << "Incorrect letter grade entered." << endl
                                      << "Enter a new grade." << endl;
                        break;       // optional; will exit switch anyway
        } // end switch
    } // end while

    // output summary of results
    cout << "Totals for each letter grade are:" << endl
         << "A: " << aCount // display number of A grades
         << "B: " << bCount // display number of B grades
         << "C: " << cCount // display number of C grades
         << "D: " << dCount // display number of D grades
         << "F: " << fCount // display number of F grades
         << endl;

    return 0;  // indicate successful termination
} // end function main
```

**do/while Repetition Structure**

- Similar to \texttt{while} structure
  - Makes loop continuation test at end, not beginning
  - Loop body executes at least once
- Format
  \[
  \text{do} \{ \text{statement} \} \text{ while ( condition );}
  \]

**break and continue Statements**

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

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

• Example
  if ( payCode == 4 )
  cout << "You get a bonus!" << endl;
  – If paycode is 4, bonus given
  – If == was replaced with =
  if ( payCode = 4 )
  cout << "You get a bonus!" << endl;
  – Paycode set to 4 (no matter what it was before)
  – Statement is true (since 4 is non-zero)
  – Bonus given in every case
Confusing Equality (==) and Assignment (=) Operators

- **Lvalues**
  - Expressions that can appear on left side of equation
  - Can be changed (i.e., variables)
    - \( x = 4; \)
- **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

- All programs broken down into
  - Sequence
  - Selection
    - if, if/else, or switch
    - Any selection can be rewritten as an if statement
  - Repetition
    - while, do/while, or for
    - Any repetition structure can be rewritten as a while statement