3. Intro to C++

**Topic: Introduction to C++**
- Higher-level languages
- Programming environments
- A minimal C/C++ program
- Variables, assignments, and initializations
- Input/output streams
- Named constants
- Comments
- The grammar of C++

**Higher-level languages**
- Support control structures and modular decomposition
- Shield programmer from hardware and operating-system details
- Are portable
- Are translated to machine language by compilers or interpreters
- Examples: COBOL, Fortran, Pascal, C, C++, Java

**Skeleton of a C++ program**
```cpp
// null.cpp:
// Does nothing

void main()
{
}
```

• Every C or C++ program defines a function named `main`

**Software development**

**C/C++ compiler environment**

**Using an IDE**
- **Steps**: Code or edit; Build; Execute
- MSVC generates a “console application”
- To correct a syntax error: double-click on error message
- Tip: resist temptation to follow advice of compiler
- For help: press F1 on error message, or highlight a word and press F1
- Remember, one error can trigger many error messages
- Compile on network, not on floppy disk

**An integrated development environment (IDE)**
- Editor
- Error diagnostics
- Warnings
- Compiler
- Linker
- Debugger
- Loader
- Help system
“Work spaces” in MSVC
- The IDE creates a work area for files that are part of a “project” (application)
- Intermediate files and .exe files generated by the compiler and linker go into a subdirectory, Debug
- To test a new program, close the work space of the old one
- Later delete any files except .cpp

A program with output
```cpp
// hello.cpp:
// Says hello
#include <iostream>

void main()
{
    cout << "Hello";
}
```

A program to add numbers
1. Prompt for integers input1, input2
2. sum ← input1 + input2
3. Display sum
```cpp
// add.cpp
// Prints sum of 2 input integers.
#include <iostream>

void main()
{
    cout << "Enter 2 integers to add: ";
    cin >> input1 >> input2;
    sum = input1 + input2;
    cout << input1 << " + " << input2
    << " = " << sum << endl;
}
```

Variables and assignments
- A variable is a named data address that occupies space and may be given a value
- Programmer must declare a variable before using it
- A variable has a data type
- A variable gets its value by initialization, assignment, or input
- The assignment operator has a variable to its left and an expression to its right

C/C++ statements
- Kinds:
  - variable declarations
  - assignment
  - input/output
  - compound
- Simple statements end with a semicolon (;)
- Compound statements are enclosed by braces

Elements of the C and C++ languages
C++ (mid-80s) is an extension of C (1971).
Assignments and initializations

• An assignment expression has a value:
  ```
  int a, b;
  a = b = 2;
  ```

• An initialization statement is a declaration, and the initialization operator (=) produces no expression:
  ```
  int n = 4;
  ```

Input/output

• Input/output (I/O) uses streams in C++
• Stream: a sequence of characters going to or from a device
• `cout` is the standard output stream object
• `cin` is the standard input stream object
• The inserter operator `<<` inserts data into an output stream
• The extractor operator `>>` extracts data from an input stream and assigns it to one or more variables

Standard C and C++ libraries

• Many basic tools, such as for input/output, are in standard libraries, not in the language itself
• Standard C++ stream library: `iostream.h`, which declares stream objects like `cin`, `cout`
• To use a standard library, write `#include` and library file name in angle brackets
• Named file is treated as if it were part of your program source file

A binary operator yields a value

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Associativity</th>
<th>Example</th>
<th>Value</th>
<th>Side effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -</td>
<td>left</td>
<td>3 − 1 + 4</td>
<td>6</td>
<td>none</td>
</tr>
<tr>
<td>=</td>
<td>right</td>
<td>a = b = 1</td>
<td>1</td>
<td>assign</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>left</td>
<td>cout &lt;&lt; a &lt;&lt; b cout</td>
<td>I/O</td>
<td></td>
</tr>
</tbody>
</table>

• A binary expression may have both a value and a side effect
• The value may be used in chaining
• Associativity affects result value in chaining

Named constants

```
// yearly.cpp
// Tells yearly cost of cable service.
// Uses named constant to store
// the number of months in a year.
#include <iostream.h>
const int MONTHS_IN_YR = 12;
void main()
{
 int per_month = 26; // monthly cost
cout << "Yearly rate at $" << per_month
 << " per month is $" << MONTHS_IN_YR * per_month << endl;
}
```

• Use the `const` keyword when appropriate
• Named constants are reusable, updatable

Ways to give a value to a variable

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Who chooses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>int n = 3;</td>
<td>programmer</td>
</tr>
<tr>
<td>Assignment</td>
<td>n = 3;</td>
<td>programmer</td>
</tr>
<tr>
<td>Input</td>
<td>cin &gt;&gt; n;</td>
<td>user</td>
</tr>
</tbody>
</table>

Giving a value to a constant

• Named constants: one way
  ```
  Example: const float price = 3.95;
  ```
• Literals cannot take new value
  ```
  Invalid: 3.95 = price;
  ```
3. Intro to C++

Comments

- // starts a comment that lasts to the end of the source line
- /* and */ delimit a comment
- Comments help make a program readable and understandable
- To debug or maintain a program, one must understand it
- Guideline: write a comment wherever necessary to make programmer’s intention clear

Writing clear comments

- Not clear:
  // Computes result from input.
- Not clear:
  /* Displays the absolute value of the difference. */
- Clear:
  /* This program prompts for 2 integers and displays the absolute value of the difference between them. */
- A comment is usually a narrative that tells the story of what a program does, or is an assertion about values present as the program runs.

Guidelines for formatting source code

- Example:
  #include <iostream.h>
  void main() {
    cout << "Hello";
  }
- Leave an empty line before a function definition such as main
- Align pairs of braces vertically
- Indent statements 2-3 spaces

Syntax and semantics

- Syntax is the set of grammar rules that define a language formally
- Semantics is the set of meanings of each of the syntax elements
- The compiler handles a syntax error by halting and displaying a message (usually misleading)
- The compiler follows semantics by generating the appropriate machine code for statements

Kinds of tokens (lexical elements)

- keyword (void, main, int, …)
- identifier (letter or '_' followed by a series of letters, digits, '_')
- constant literal (numeral, double-quoted string, single quoted character)
- operator (=, +, *, −)
- punctuator (semicolon, comma, paren, brace)

Not tokens:

- The compiler ignores white space (space characters, tabs, newlines)
- Compiler ignores comments (//…, /*…*/)
- Comments and readability are a major factor in effective programming

The compiler is case sensitive

- Total is a different identifier from total
- Int is not a keyword

Identifiers and operators

- May not contain spaces,
- Are different from literals:
  “+” is not same as +
  "input" is not same as ID input
  "2" is not same as numeral 2
3. Intro to C++

Specifying grammar rules

- A language is a set of strings, e.g., the set of all possible C++ programs
- A grammar is a set of rules for what is permitted in a language
- C/C++ tokens are formed by simple rules; e.g., an integer literal is a series of digits
- Higher-level (nonterminal) components (program, statement, expression, etc.) are built from tokens or other nonterminals

Ways to specify syntax

- Plain English (e.g., “A compound statement is a series of statements, in braces”)
- List of alternatives; e.g.:
  - `statement-list:
    - statement
    - statement-list`
- Diagram; e.g., `sign:
  +
  -`

Syntax rules and diagrams

- `compound-statement:
  - { statement-list }
- statement-list:
  - statement
  - statement statement-list
  - statement:
    - nothing
    - declaration
    - assignment
    - IO-statement
    - compound-statement`

Diagram for `statement-list`

- Each syntax rule is applied by putting a defined element’s components under the name of the element

The parser generates a parse tree of a program

- Virtual machines and applets
  - Java compilers may generate applets (web applications) that consist of “byte code” rather than machine code
  - The “byte code” runs on any browser that supports Java
  - The browser contains a virtual machine that interprets the byte code
  - Virtual machine enables distribution of processor independent compiled code on the Web

Statements, data, functions, objects

How the concepts fit together
3. Intro to C++

Discussion problems
Write a program, with documentation to:
1. Display the words “This is a program” on four lines
2. Initialize and display four integer variables, with values 1, 2, 3, 4
3. As #2, but label output with variable names
4. Input and display four integers
5. Correct this code:
   ```cpp
   int id_num;
   cin >> "Enter your ID#: " >> id_num;
   ```

More discussion problems
1. Display
   ```cpp
   i
   ii
   iii
   ```
   in one statement.
2. Declare named constants for tax as a percentage (5) and discount (33). Display amount due on 4 inputs, consisting of a price and quantity for each of two items. Use no fractions, use integer division if necessary.

1. Find syntax errors;
2. Find other errors
   ```cpp
   #include <iostream>
   int m;
   void()
   }
   cout >> "Enter 2 #s: ";
   cin >> m, n;
   int m,n,s,tl,4f,g.0,n,v+2;
   s == m + n, m;
   cout << "sum=", s;
   ```