Chapter 2
Simple C++ Programs

The C++ Programming Language

- A superset of C
  - C++ compilers can be used to compile C programs
- Supports
  - Programmer Defined Types
  - Templates
  - Overloading of functions and operators
- Best to think of C++ as its own language

Dev-C++

- Bloodshed Software
  - [http://www.bloodshed.net/dev/devcpp.htm](http://www.bloodshed.net/dev/devcpp.htm)
  - Dev-C++ 5.0 beta 9.2 (4.9.9.2) (9.0 MB) with Mingw/GCC 3.4.2
- Download from:
  - SourceForge

Windows Friendly Exit

```
// Windows friendly exit
system("PAUSE");
return 0;
```
Program structure

Comments
preprocessor directives
using directives
Block of code
{
  comments
  statements
}

C++ Program

/* Hello World program */
#include <iostream>
int main()
{
  char ch;
  cout << "Hello World" << endl;
  cin >> ch;
  return(0);
}

5 Step Problem Solving Methodology

1. State the problem clearly.
2. Describe the input and output.
3. Work a hand example.
4. Develop a solution.
5. Test your solution.

Example

Problem Statement
- Compute the straight-line distance between two points.

Input/Output Description
- Point 1 (x1, y1)
- Point 2 (x2, y2)
- Distance (d)

Algorithm Development (outline)
- Give values to the two points
- Compute lengths of two sides of right triangle
- Compute distance between points (hypotenuse)
- Print distance between points
C++ Program

```cpp
/*-----------------------------------------------*/
/* Program chapter1_1 */
/* This program computes the distance between */
/* two points */
#include <iostream>
#include <cmath>
using namespace std;
```

Comments

- Comments help people read programs, but are ignored by the compiler.
- In C++ there are two types of comments.
  - Line comments begin with // and continue for the rest of the line.
  - Delimited comments begin with /* and end with */

Preprocessor Directives

- Provide instructions to the compiler that are performed before the program is compiled.
- Begin with a #
- Example:
  ```cpp
  #include <iostream>
  The #include directive instructs the compiler to include statements from the file iostream.
  ```

using Directive

- The using directive instructs the compiler to use files defined a specified namespace.
- Example:
  ```cpp
  using namespace std;
  std is the name of the Standard C++ namespace.
  ```

Block of Code

- A block of code is defined by a set of curly braces {...}.
- Example:
  ```cpp
  int main()
  { //Block defines body of main function
double x1(1), x2(4), side1;
side1 = x2 - x1;
cout << side1 << endl; //main() returns int value of 0
} //end definition of main
``` 

- Every C++ problem solution contains exactly one function named main()
- C++ program solutions always begin execution in main()
C++ Program

```cpp
// Print distance
cout << "The distance between the two points is " << distance << endl;

// Windows friendly exit
system("PAUSE");
return 0;
```

Constants and Variables

Constants and variables represent memory locations that we reference in our program solutions.

- Constants are objects that store specific data that cannot be modified.
  - `10` is an integer constant
  - `4.5` is a floating point constant
  - "Sid" is a string constant
  - `'a'` is a character constant

- Variables are memory locations that store values that can be modified.
  - `double x1(1.0), x2(4.5), side1;`  
  - `side1 = x2-x1;`  
  - `x1, x2` and `side1` are examples of variables that can be modified.

Memory Snapshot

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

int main()
{
  double x1(1.0), x2(4.5), side1;
  side1 = x2-x1;
  cout << "side 1 has length: " side1;
}
```

Common C++ Data Types

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Example of a constant</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>char</code></td>
<td>&quot;5&quot;</td>
</tr>
<tr>
<td><code>int</code></td>
<td>25</td>
</tr>
<tr>
<td><code>double</code></td>
<td>25.0</td>
</tr>
<tr>
<td><code>string</code></td>
<td>&quot;hello&quot;</td>
</tr>
</tbody>
</table>

C++ Identifiers

- Identifiers are used to name objects in C++.
- Rules for selecting valid identifiers:
  - Identifier must begin with a letter or underscore `_`
  - Identifiers consist of alphanumeric characters and underscore character only.
  - An identifier cannot be a reserved word.
  - Only the first 31 characters of an identifier are used to distinguish it from other identifiers.
- **C++ is case sensitive.** Thus, `Sid` and `side1` are unique identifiers that represent different objects.

Declarations

- A type declaration statement defines new identifiers and allocates memory.
- An initial value may be assigned to a memory location at the time an identifier is defined.

Syntax:

```
[modifier] data_type identifier_list;
```

Examples:

```
double length(20.75), width(11.5), volume;
int numberOfFeetInYard(3);
const int MIN_SIZE = 0;
```
Symbolic Constants

- A symbolic constant is defined in a declaration statement using the modifier `const`.
- A symbolic constant allocates memory for an object that can not be modified during execution of the program. Any attempt to modify a constant will be flagged as a syntax error by the compiler.
- A symbolic constant must be initialized in the declaration statement.

Assignment Operator

The assignment operator (=) is used in C++ to assign a value to a memory location.

- The assignment statement:
  
  ```
  x1 = 1.0;
  ```

- assigns the value 1.0 to the variable x1.
- Thus, the value 1.0 is stored in the memory location associated with the identifier x1.

Assignment Operators - Examples

- Example 1 - initialization
  
  ```
  double sum = 0;
  ```

- Example 2
  
  ```
  int x;
  x=5;
  ```

- Example 3
  
  ```
  char ch;
  ch = 'a';
  ```

Assignment Statements - continued

- Example 3
  
  ```
  int x, y, z;
  x=y=0;
  z=2;
  ```

- How is the memory map affected by the following statement?
  
  ```
  y = z;
  ```

Arithmetic Operators

- Addition `+`
- Subtraction `-`
- Multiplication `*`
- Division `/`
- Modulus `%`
  
  - Modulus returns remainder of division between two integers
  - Example 5 % 2 returns a value of 1
Integer Division

- Division between two integers results in an integer.
- The result is truncated, not rounded.
- Example:
  - The expression \(5/3\) evaluates to 1
  - The expression \(3/6\) evaluates to 0

C++ Operators

- The cast operator.
  - The cast operator is a unary operator that requests that the value of the operand be cast, or changed, to a new type for the next computation. **The type of the operand is not affected.**
  - Example:
    ```c++
    int count(10), sum(55);
    double average;
    average = (double)sum/count;
    ```

Memory snapshot:

```
int count 10
int sum 55
double average
```

Priority of Operators

1. Parentheses Inner most first
2. Unary operators Right to left
   (+ -)
3. Binary operators Left to right
   (*/ %)
4. Binary operators Left to right
   (+ -)

Practice! – Evaluate Expressions

- \(7 + 3 * 5 - 2\)
- \(4 + 7 / 3\)
- \(8 \% 3 * 6\)
- \((7 + 3) * 5 - 2\)

Overflow and Underflow

- Overflow
  - answer too large to store
  - Example: using 16 bits for integers
    ```c++
    result = 32000 + 532;
    ```
  - Exponent overflow
    - answer's exponent is too large
      - Example: using float, with exponent range -38 to 38
        ```c++
        result = 3.25e28 * 1.0e15;
        ```

- Exponent underflow
  - answer's exponent too small
    - Example: using float, with exponent range -38 to 38
      ```c++
      result = 3.25e-38 * 1.0e-15;
      ```

Numeric Data Types

- Integers
  - short
  - int
  - long
- Floating-Point
  - float
  - double
  - long double
Boolean Data Type

Example

```cpp
bool error(false), status(true);
cout << error << endl << status;
```

Program Output

```
0
1
```

Increment and Decrement Operators

- **Increment Operator** `++`
  - post increment `x++;`
  - pre increment `++x;`
- **Decrement Operator** `--`
  - post decrement `x--;`
  - pre decrement `--x;`
- For examples assume `k=5` prior to executing the statement.
  - `m= ++k;` both `m` and `k` become 6
  - `n = k--;` `n` becomes 5 and `k` becomes 4

Abbreviated Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+=</code></td>
<td><code>x=x+2;</code></td>
<td><code>x=x+2;</code></td>
</tr>
<tr>
<td><code>-=</code></td>
<td><code>x=x-2;</code></td>
<td><code>x=x-2;</code></td>
</tr>
<tr>
<td><code>*=</code></td>
<td><code>x*y;</code></td>
<td><code>x=x*y;</code></td>
</tr>
<tr>
<td><code>/=</code></td>
<td><code>x/y;</code></td>
<td><code>x=x/y;</code></td>
</tr>
<tr>
<td><code>%=</code></td>
<td><code>x%y;</code></td>
<td><code>x=x%y;</code></td>
</tr>
</tbody>
</table>

Precedence of Arithmetic and Assignment Operators

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parentheses <code>()</code></td>
<td>Innermost first</td>
</tr>
<tr>
<td>2</td>
<td>Unary operators <code>+-++--</code></td>
<td>(type) Right to left</td>
</tr>
<tr>
<td>3</td>
<td>Binary operators <code>*/%</code></td>
<td>Left to right</td>
</tr>
<tr>
<td>4</td>
<td>Binary operators <code>+-</code></td>
<td>Left to right</td>
</tr>
<tr>
<td>5</td>
<td>Assignment operators <code>+= -= *= /= %=</code></td>
<td>Right to left</td>
</tr>
</tbody>
</table>

STANDARD INPUT AND OUTPUT

Simple Input - `cin`

- `cin` is an `istream` object defined in the header file `iostream`
- `cin` is defined to stream data from standard input (the keyboard)
- We use the input operator `>>` with `cin` to assign values to variables

General Form:

```
cin >> identifier >> identifier;
```

Note: Data entered from the keyboard must be compatible with the data type of the variable.
Simple Output - `cout`

- `cout` is an `ostream` object, defined in the header file `iostream`.
- `cout` is defined to stream data to standard output (the display).
- We use the output operator `<<` with `cout` to output the value of an expression.

General Form: `cout << expression << expression;`

Note: An expression is a C++ constant, identifier, formula, or function call.

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

int main()
{
    int i, j;
    double x;
    string units = " cm";
    cin >> i >> j;
    cin >> x;
    cout << "output 
";
    cout << i << ',' << j << endl;
    cout << x << units << endl;
    return 0;
}
```

`//Example 1: Determine the output`

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

int main()
{
    int i, j;
    double x, y;
    cin >> i >> j >> x >> y;
    cout << "First output " << endl;
    cout << i << ',' << j << ',' << x << ',' << y << endl;
    cin >> x >> y >> i >> j;
    cout << "Second output" << endl;
    cout << i << ',' << j << ',' << x << ',' << y << endl;
    return 0;
}
```

`//Input stream:
1 2 4.5 cm`

`First output 1, 2, 4.5 cm
Second output 4.5 cm`

```cpp
#include <iostream>
#include <iomanip>

int main()
{
    int i, j;
    double x, y;
    cin >> i >> j >> x >> y;
    cout << scientific
    cout << "The area of the circle is: " << setw(12) << area << " square centimeters" << endl;
}
```

Characters and input

- The `>>` operator discards leading whitespace.
- `get()` method gets the next character even if it is a whitespace character.

```cpp
int x;
char ch;
 cin >> x >> ch;
 cin >> x;
 cin.get(ch);
```

Standard Output

- Standard Output
  - `#include <iostream>`
  - `cout << "Hello " << name;`
- Stream Manipulators
  - `#include <iomanip>`
  - `setprecision(n), fixed, scientific`
  - `setw(n), left, right, setfill(ch)`
  - `dec, oct, hex`
  - `endl`

```cpp
cout << setprecision(4)
    << "The radius of the circle is: "
    << setw(10) << radius << " centimeters"
    << endl;
```

```
1 2
3 4
3.4 5
2 3 3.4 7
```

```
//  Output results
cout << setprecision(4)
    << "The radius of the circle is: "
    << setw(10) << radius << " centimeters"
    << endl;
```
Common Functions in `<cmath>`

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(x)</td>
<td>computes absolute value of x</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>computes square root of x, where x ≥ 0</td>
</tr>
<tr>
<td>pow(x, y)</td>
<td>computes x^y</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>nearest integer larger than x</td>
</tr>
<tr>
<td>floor(x)</td>
<td>nearest integer smaller than x</td>
</tr>
<tr>
<td>exp(x)</td>
<td>computes e^x</td>
</tr>
<tr>
<td>log(x)</td>
<td>computes ln x, where x &gt; 0</td>
</tr>
<tr>
<td>log10(x)</td>
<td>computes log_{10}x, where x &gt; 0</td>
</tr>
<tr>
<td>sin(x)</td>
<td>sine of x, where x is in radians</td>
</tr>
<tr>
<td>cos(x)</td>
<td>cosine of x, where x is in radians</td>
</tr>
<tr>
<td>tan(x)</td>
<td>tangent of x, where x is in radians</td>
</tr>
</tbody>
</table>

Common Functions in `<cctype>`

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalpha(ch)</td>
<td>Returns true if ch is an upper or lower case letter.</td>
</tr>
<tr>
<td>isdigit(ch)</td>
<td>Returns true if ch is a decimal digit.</td>
</tr>
<tr>
<td>isgraphic(ch)</td>
<td>Returns true if ch is a whitespace character.</td>
</tr>
<tr>
<td>islower(ch)</td>
<td>Returns true if ch is a lower case letter.</td>
</tr>
<tr>
<td>isupper(ch)</td>
<td>Returns true if ch is an uppercase letter.</td>
</tr>
<tr>
<td>tolower(ch)</td>
<td>Returns the lowercase version of ch if ch is an uppercase character, returns ch otherwise.</td>
</tr>
<tr>
<td>toupper(ch)</td>
<td>Returns the uppercase version of ch if ch is a lowercase character, returns ch otherwise.</td>
</tr>
</tbody>
</table>