The Fundamentals of C++

Basic programming elements and concepts
Program Organization

- Program statement
  - Definition
  - Declaration
  - Action
- Executable unit
  - Named set of program statements
  - Different languages refer to executable units by different names
    - Subroutine: Fortran and Basic
    - Procedure: Pascal
    - Function: C++
Program Organization

● C++ program
  ■ Collection of definitions, declarations and functions
  ■ Collection can span multiple files
● Advantages
  ■ Structured into small understandable units
  ■ Complexity is reduced
  ■ Overall program size decreases
Object

- Object is a representation of some information
  - Name
  - Values or properties
    - Data members
  - Ability to react to requests (messages)!!
    - Member functions
- When an object receives a message, one of two actions are performed
  - Object is directed to perform an action
  - Object changes one of its properties
```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
    cout << "Hello world!" << endl;
    return 0;
}
```
Greeting Output

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

int main() {
    // Extract length and width
    cout << "Rectangle dimensions: ";
    float Length;
    float Width;
    cin >> Length >> Width;

    // Compute and insert the area
    float Area = Length * Width;
    cout << "Area = " << Area << " = Length "
         << Length << " * Width " << Width << endl;
    return 0;
}
```
Area.cpp Output

Rectangle dimensions: 20.5 88
Area = 1804 = Length 20.5 × Width 88
Comments

- Allow prose or commentary to be included in program
- Importance
  - Programs are read far more often than they are written
  - Programs need to be understood so that they can be maintained
- C++ has two conventions for comments
  - // single line comment (preferred)
  - /* long comment */ (save for debugging)
- Typical uses
  - Identify program and who wrote it
  - Record when program was written
  - Add descriptions of modifications
Fundamental C++ Objects

- C++ has a large number of fundamental or built-in object types
- The fundamental object types fall into one of three categories
  - Integers objects
  - Floating-point objects
  - Character objects
Integer Object Types

- The basic integer object type is `int`
  - The size of an `int` depends on the machine and the compiler
    - On PCs it is normally 16 or 32 bits
- Other integers object types
  - `short`: typically uses less bits
  - `long`: typically uses more bits
- Different types allow programmers to use resources more efficiently
- Standard arithmetic and relational operations are available for these types
Integer Constants

- Integer constants are positive or negative whole numbers
- Integer constant forms
  - Decimal
  - Octal (base 8)
    - Digits 0, 1, 2, 3, 4, 5, 6, 7
  - Hexadecimal (base 16)
    - Digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
- Consider
  - 31 oct and 25 dec
Specifying Syntax

● Need
  ■ A notation for exactly expressing a programming language element
    – Notation is describing the programming language
    – Notation is not part of the programming language

● Notation must be able to describe
  ■ Elements that have several forms
  ■ Elements that are
    – Required
    – Optional
    – Repeated
Notation Conventions

- Parentheses ():
  - Anything surrounded by parentheses must be used
- Braces []:
  - Anything surrounded by brackets is optional
- Vertical line |
  - Elements on either side of the line are acceptable
- Ellipsis ...
  - The pattern established before the ellipsis continues
-Specifier
  - Name of a language element
Notation Examples

- **NonZeroDigit**
  - 1 | 2 | ... 9
- **Digit**
  - 0 | NonZeroDigit
- **OctalDigit**
  - 0 | 1 | ... 7
- **HexDigit**
  - 0 | 1 | ... 9 | A | B | ... F | a | b | ... f
- **Digits**
  - NonZeroDigit [ Digit ... Digit ]
Decimal Constants

- Examples
  - 97
  - 40000L
  - 50000
  - 23a (illegal)
- The type of the constant depends on its size, unless the type specifier is used
Octal Constants

- **Examples**
  - 017
  - 0577777L
  - 01267333L
  - 01267335
  - 0482 (illegal)

- The type of the constant depends on its size, unless the type specifier is used.

  Sequence of one or more octal digits. First digit must be 0

  OctalDigits [L | l]

  Optional L or l
Hexadecimal Constants

- Letters represent the hex digits
  - a or A - 10
  - b or B - 11
  - c or C - 12
  - d or D - 13
  - e or E - 14
  - f or F - 15

- Examples
  - 0x2C
  - 0XAC12EL

- The type of the constant depends on its size, unless the type specifier is used
Character Object Types

- Character type `char` is related to the integer types.
- Characters are encoded using a scheme where an integer represents a particular character.
- ASCII is the dominant encoding scheme.

- **Examples**
  - `' '` encoded as 32
  - `'+'` encoded as 43
  - `'A'` encoded as 65
  - `'Z'` encoded as 90
  - `'a'` encoded as 97
  - `'z'` encoded as 122
Character Operations

- Arithmetic and relational operations are defined for character types
  - 'a' < 'b' is true
  - '4' > '3' is true
  - 'b' + 2 produces the number that represents 'd'
  - '8' − '3' produces 5

- Arithmetic with characters needs to be done with care
  - '9' + 3 produces the number that represents '<'
Character Constants

- Explicit characters within single quotes
  - 'a'
  - 'D'
  - '*'

- Special characters - delineated by a backslash \ 
  - Two character sequences (sometimes called escape codes) within single quotes
  - Important special characters
    - '\t' denotes a tab
    - '\n' denotes a new line
    - '\\' denotes a backslash
# Escape Codes

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Name</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>newline</td>
<td>NL</td>
<td>\n</td>
</tr>
<tr>
<td>horizontal tab</td>
<td>HT</td>
<td>\t</td>
</tr>
<tr>
<td>backspace</td>
<td>BS</td>
<td>\b</td>
</tr>
<tr>
<td>form feed</td>
<td>FF</td>
<td>\f</td>
</tr>
<tr>
<td>alert or bell</td>
<td>BEL</td>
<td>\a</td>
</tr>
<tr>
<td>carriage return</td>
<td>CR</td>
<td>\r</td>
</tr>
<tr>
<td>vertical tab</td>
<td>VT</td>
<td>\v</td>
</tr>
<tr>
<td>backslash</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>single quote</td>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>double quote</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>question mark</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Literal String Constants

- A literal string constant is a sequence of zero or more characters enclosed in double quotes
  - "Are you aware?\n"
- Individual characters of string are stored in consecutive memory locations
- The null character (\'\0\') is appended to strings so that the compiler knows where in memory strings ends

```
Memory

010040  010041  010042  ...
Are   you   aware?
010054  010055
\n\0
```
Floating-Point Object Types

- Floating-point object types represent real numbers
  - Integer part
  - Fractional part
- The number 108.1517 breaks down into the following parts
  - 108 - integer part
  - 1517 - fractional part
- C++ provides three floating-point object types
  - `float`
  - `double`
  - `long double`
Floating-Point Constants

- **Standard decimal notation**
  - Digits . Digits [f | F | l | L]
  - 134.123
  - 0.15F

- **Standard scientific notation**
  - Digits . Digits Exponent [f | F | l | L]
  - Where
  - Exponent is (e | E) [+ | -] Digits
  - 1.45E6
  - 0.979e-3L

- When not specified, floating-point constants are of type **double**
Names

- Used to denote program values or components
- A valid name is a sequence of
  - Letters (upper and lowercase)
  - Digits
    - A name cannot start with a digit
  - Underscores
    - A name should not normally start with an underscore
- Names are case sensitive
  - MyObject is a different name than MYOBJECT
- There are two kinds of names
  - Keywords
  - Identifiers
Keywords

- Keywords are words reserved as part of the language
  - int, return, float, double
  - They cannot be used by the programmer to name things
  - They consist of lowercase letters only
  - They have special meaning to the compiler
# Keywords

<table>
<thead>
<tr>
<th>asm</th>
<th>do</th>
<th>if</th>
<th>return</th>
<th>typedef</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>double</td>
<td>inline</td>
<td>short</td>
<td>typeid</td>
</tr>
<tr>
<td>bool</td>
<td>dynamic_cast</td>
<td>int</td>
<td>signed</td>
<td>typename</td>
</tr>
<tr>
<td>break</td>
<td>delete</td>
<td>long</td>
<td>sizeof</td>
<td>union</td>
</tr>
<tr>
<td>case</td>
<td>else</td>
<td>mutable</td>
<td>static</td>
<td>union</td>
</tr>
<tr>
<td>catch</td>
<td>enum</td>
<td>namespace</td>
<td>static_cast</td>
<td>using</td>
</tr>
<tr>
<td>char</td>
<td>explicit</td>
<td>new</td>
<td>struct</td>
<td>virtual</td>
</tr>
<tr>
<td>class</td>
<td>extern</td>
<td>operator</td>
<td>switch</td>
<td>void</td>
</tr>
<tr>
<td>const</td>
<td>false</td>
<td>private</td>
<td>template</td>
<td>volatile</td>
</tr>
<tr>
<td>const_cast</td>
<td>float</td>
<td>protected</td>
<td>this</td>
<td>wchar_t</td>
</tr>
<tr>
<td>continue</td>
<td>for</td>
<td>public</td>
<td>throw</td>
<td>while</td>
</tr>
<tr>
<td>default</td>
<td>friend</td>
<td>register</td>
<td>true</td>
<td>union</td>
</tr>
<tr>
<td>delete</td>
<td>goto</td>
<td>reinterpret_cast</td>
<td>try</td>
<td>unsigned</td>
</tr>
</tbody>
</table>
Identifiers

● Identifiers should be
  ■ Short enough to be reasonable to type (single word is norm)
    – Standard abbreviations are fine (but only standard abbreviations)
  ■ Long enough to be understandable
    – When using multiple word identifiers capitalize the first letter of each word

● Examples
  ■ Min
  ■ Temperature
  ■ CameraAngle
  ■ CurrentNbrPoints
Definitions

- All objects that are used in a program must be defined
- An object definition specifies
  - Type
  - Name
- A common definition form

\[
\text{Type } \text{Id, Id, ..., Id;}
\]

- Our convention is one definition per statement!
Examples

char Response;
int MinElement;
float Score;
float Temperature;
int i;
int n;
char c;
float x;

Objects are uninitialized with this definition form
(Value of a object is whatever is in its assigned memory location)
Arithmetic Operators

- **Common**
  - Addition +
  - Subtraction -
  - Multiplication *
  - Division /
  - Mod %

- **Note**
  - No exponentiation operator
  - Single division operator
  - Operators are overloaded to work with more than one type of object

Write \( m \times x + b \) not \( mx + b \)
Integer Division

- Integer division produces an integer result
  - Truncates the result
- Examples
  - $3 \div 2$ evaluates to 1
  - $4 \div 6$ evaluates to 0
  - $10 \div 3$ evaluates to 3
Mod

- Produces the remainder of the division
- Examples
  - \( 5 \% 2 \) evaluates to 1
  - \( 12 \% 4 \) evaluates to 0
  - \( 4 \% 5 \) evaluates to 4
Operators and Precedence

- Consider $mx + b$
- Consider $m\times x + b$ which of the following is it equivalent to
  - $(m \times x) + b$
  - $m \times (x + b)$
- Operator precedence tells how to evaluate expressions
- Standard precedence order
  - () Evaluate first, if nested innermost done first
  - * / % Evaluate second. If there are several, then evaluate from left-to-right
  - + - Evaluate third. If there are several, then evaluate from left-to-right
Operator Precedence

- Examples
  
  $1 + 2 \times 3 / 4 - 5$

  $2 \times 4 / 5 + 3 \times 5 \% 4$

  $3.0 \times 3 / 4$

  $(1 + 3) \times ((2 + 4 \times 6) \times 3) / 2 + 2$
Defining and Initializing

- When an object is defined using the basic form, the memory allotted to it contains random information.
  - Good idea to specify its desired value at the same time.
    - Exception is when the next statement is an extraction for the object.

Expressions are used to initialize corresponding objects:

- Known type
- Identifiers
- Type \( \text{Id} = \text{Exp} \), \( \text{Id} = \text{Exp} \), ..., \( \text{Id} = \text{Exp} \);

- Our convention is one definition per statement!
Examples

int FahrenheitFreezing = 32;
char LetterGrade = 'A';
cout << "Slope of line: ";
float m;
cin >> m;
cout << "Intercept: ";
float b;
cin >> b;
cout << "X value of interest: ";
float x;
cin >> x;
float y = (m * x) + b;
// Program 2.11: Compute velocity of car
#include <iostream>
#include <string>
using namespace std;

int main() {
    cout << "All inputs are integers!\n";
    cout << "Start milepost? ";
    int StartMilePost;
    cin >> StartMilePost;

    cout << "Elapsed time (hours minutes seconds)? ";
    int EndHour;
    int EndMinute;
    int EndSecond;
    cin >> EndHour >> EndMinute >> EndSecond;

    cout << "End milepost? ";
int EndMilePost;
cin >> EndMilePost;
float ElapsedTime = EndHour + (EndMinute / 60.0)
    + (EndSecond / 3600.0);
int Distance = EndMilePost - StartMilePost;
float Velocity = Distance / ElapsedTime;

cout << "\nCar traveled " << Distance
    << " miles in ";
cout << EndHour << " hrs " << EndMinute
    << " min " << EndSecond << " sec\n";
cout << "Average velocity was " << Velocity
    << " mph " << endl;

return 0;