Programming

A program is a set of instructions that the computer follows to perform a task.

It must be translated from a programming language (C++) to machine code in order to run on the machine.

Structure of a C++ Program

Hello world:

```cpp
//This program outputs a message to the screen
#include <iostream>
using namespace std;

int main() {
    cout << "Hello world!" << endl;
}
```

In general:

```cpp
//This is a comment
#include <includefile> ...
using namespace std;

int main() {
    statements ...
}
```

Variables, Data Types

- **Variable**: portion of memory that stores a value
- **Identifier**: name of a program element
- **Fundamental data types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>short</code></td>
<td><code>short</code></td>
</tr>
<tr>
<td><code>int</code></td>
<td><code>int</code></td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>long</code></td>
</tr>
<tr>
<td><code>float</code></td>
<td><code>float</code></td>
</tr>
<tr>
<td><code>double</code></td>
<td><code>double</code></td>
</tr>
<tr>
<td><code>long double</code></td>
<td><code>long double</code></td>
</tr>
<tr>
<td><code>bool</code></td>
<td><code>bool</code></td>
</tr>
<tr>
<td><code>char</code></td>
<td><code>char</code></td>
</tr>
</tbody>
</table>

- **Variable Declaration** statement

  ```cpp
datatype identifier;  
``` 

- **Variable Initialization** statement

  ```cpp
datatype identifier = constant; 
int count = 0; 
```
Integer types

- Integers are whole numbers such as 12, 7, and -99

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>int</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
</tbody>
</table>

- char type stores characters such as 'A', '@', and '9'
  - The ascii code value (an integer) of the character is stored in memory.

Floating-point types (and bool)

- Floating point types store real numbers such as 12.45 and -3.8

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>±3.4E-38 to ±3.4E38</td>
</tr>
<tr>
<td>double</td>
<td>±1.7E-308 to ±1.7E308</td>
</tr>
<tr>
<td>long double</td>
<td>±1.7E-308 to ±1.7E308</td>
</tr>
</tbody>
</table>

- bool type stores values that are true or false
  - false is 0, true is 1.

Constants

- Literals (specific value of a given type)

- Named Constants: variable whose value cannot be changed

```c
const datatype identifier = constant;
const double TAX_RATE = 0.0675;
```

Assignment statement, expressions

- To change the value of a variable:

```c
variable = expression;
```

- The lefthand side must be a variable
- The righthand side is an expression of the right type

- What is an expression?
  - an expression has a type and evaluates to a value
    - literal
    - named constant
    - variable
    - arithmetic expression
    - etc.
Arithmetic Operations

- arithmetic operators:
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - % modulo (remainder)

- Integer division:

  - $14 \div 3 = 4 \text{ r. } 2$ (because $3 \times 4 + 2 = 14$)
  - $14/3 \Rightarrow 4 \text{ in C++}$
  - $14\%3 \Rightarrow 2 \text{ in C++}$
  - $14.0/3.0 \Rightarrow 4.6666667 \text{ in C++}$

Operator precedence

- In an expression with multiple operators, which one happens first?
- Use this order for different operators:
  - $+$ (unary)
  - $*$, $/$, $\%$
  - $+$ (binary)
  - $<, >$, $<=$, $>=$, $==$, $!=$, $&&$, $||$

  We will study relational and logical operators next week.

- Use this order for multiple occurrences of the same operator
  - $-$ (unary negation) associates right to left
  - $+, /, \%, +, -$ associate left to right

Basic Input/Output

- Output (cout and <<)
  - sends data to the screen (console)

```
cout << expression;
cout << expr1 << expr2;
cout << "hello";
cout << "Count is: " << count << endl;
```

- Input (cin and >>)
  - receives data typed in from the keyboard (stops at space)

```
cin >> variable;
cin >> var1 >> var2;
cout << "Enter the height and width: ";
cin >> height >> width;
cout << "The height is " << height << endl;
```

Formatting output

- Goal: control how output displays for numeric data
- these require \#include<iomanip>

```
cout << setw(x) << 1234 << setw(y) << 5 << endl;
cout << fixed << setprecision(x);
cout << 3.14159 << endl;
float x = 20;
cout << x << endl;
```

<table>
<thead>
<tr>
<th>1234</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14</td>
<td>20.00</td>
</tr>
</tbody>
</table>
The string class

- **string literals**: represent sequences of chars, inside of double quotes:

```
cout << "Hello";
```

- To define string variables:

```
string firstName, lastName;
```

- Operations include:
  - `=` for assignment
  - `.size()` function for length
  - `[n]` to access one character in the nth position.

```
cout << name.size() << " ";
cout << name[2] << endl;
```

Type conversions

- **Implicit**
  - Assignment:
    ```
    int x;
    double d = 3.1415;
    x = d;
    cout << x << endl;
    ```
  - Binary operations:
    ```
    int x = 10;
    double d = 2.3;
    cout << x + d << endl;
    ```

- **Explicit**

```
int x, y;
...
float avg = static_cast<float>(x)/y;
```

```
or
float avg = x/(float)y; //c-style notation
```

Math Library functions

- These require cmath header file
- These take double argument, return a double
- Commonly used functions:

```
pow y = pow(x,d);  returns x raised to the power d
abs y = abs(x);    returns absolute value of x
sort y = sq_root(x); returns square root of x
ceil y = ceil(x);  returns the smallest integer >= x
sin y = sin(x);    returns the sine of x (in radians)
```

Comments

- **Single-Line Comments**

```
// this text is ignored, to end of line
```

- **Multi-Line Comments**

```
/* Anything occurring between a slash star and a star slash is ignored. Even when spanning multiple lines. */
```

- **Use comments to explain your code to a human reader who knows C++.**
Programming Style

- The visual organization of the source code
- Purpose: improve the readability of the source code
- Includes the use of spaces, tabs, and blank lines
- Includes naming of variables, constants.
- Includes where to use comments.
- Common elements to improve readability:
  - Braces {} aligned vertically
  - Indentation of statements within a set of braces
  - Lines shorter than 80 characters.