Introduction to C++
Brief History of C++

In 1972 the C language was developed by Dennis Ritchie of AT&T. (Used in UNIX)

Circa 1979, Bjarne Stroustrup (also of AT&T) tried using both C and Simula to write operating system simulations. Frustrated, (or Motivated :-) he invented C with Classes using the C pre macros on top of C.

C with Classes is used internally by several people at AT&T and grows in popularity. Additional capabilities are added. This was a research project with almost no budget.

Interest in this object-oriented C grows and C++ is born (say 1984). Initially implemented as a translator called Cfront to C code. Growth is exponential:

**TABLE 1. From Stroustrup, DEOC++, pg 164**

<table>
<thead>
<tr>
<th>Date</th>
<th>Estimated C++ Users</th>
<th>Date</th>
<th>Estimated C++ Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>1979</td>
<td>1</td>
<td>1986</td>
<td>2,000</td>
</tr>
<tr>
<td>1980</td>
<td>16</td>
<td>1987</td>
<td>4,000</td>
</tr>
<tr>
<td>1981</td>
<td>38</td>
<td>1988</td>
<td>15,000</td>
</tr>
<tr>
<td>1982</td>
<td>85</td>
<td>1989</td>
<td>50,000</td>
</tr>
<tr>
<td>1983</td>
<td>??+2</td>
<td>1990</td>
<td>150,000</td>
</tr>
<tr>
<td>1984</td>
<td>??+50</td>
<td>1991</td>
<td>400,000</td>
</tr>
<tr>
<td>1985</td>
<td>500</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1989 ANSI C is now a standard, but C++ has already taken off. C++ tries to conform to the ANSI C changes where possible.

1990,1991 The C++ Programming Language (Stroustrup) and the Annotated Reference Manual (Ellis and Stroustrup) are published.

1994 Standard Template Library is solicited and accepted.

1996 ANSI Standard was expected, still pending.
Language Features

Design Goals of ANSI C
- Efficiency
- Portability
- Expressiveness

Design Goals for C++
- Compatibility
- Expressiveness
- Not Inefficient
- Language “Pureness”

Object-Oriented Features
- Classes and Inheritance
- Multiple Inheritance
- Polymorphism
- Access Control (Security)
- Run Time Type Inference

Other
- Extensive libraries, especially I/O and memory
- Standard Template Library
- Exception Handling

Missing Features
- Standard Graphics Library
- Garbage Collection
- Threads
- Polymorphic Containers
```cpp
// -----------------------------
// hello.C is a demonstration program based on the original
// Hello World program of Kernighan and Ritchie, “The C

#include<iostream.h>
#include<conio.h>
void main()
{
    cout << "My very first program again"
    getch();
}

Programs typically contain the following elements:

- Descriptive comments
- Include files
- Declarations: function prototypes and global variables
- Functions including exactly one main function

The main function
- Should have it’s type and arguments (if any) declared
- Should return a value of the correct type
- Controls the execution of the program, even if it is not first in the file
```
**Declarations / Header Files**

Headers files make your easier to write and clearer to read by grouping commonly used function prototypes into a few special files. E.g. `iostream.h`

Frequently used include files¹:

- **stdlib.h**  
  Standard functions, abs(), malloc()

- **iostream.h**  
  Basic I/O and cin, cout, cerr

- **string.h**  
  String manipulation functions, e.g. strcmp(), strcat(), strcpy()  

- **ctype.h**  
  Character manipulation

- **math.h**  
  Mathematical functions, e.g. sqrt(), sin(), cos(), tan(), ceil(), pow(), exp(), etc...
The Pre-Processor and Directives

The Pre-Processors main job is to alter the actual code seen by the main compiler. The most common example is with directives such as #include. Note that all directives must start at the beginning of the line, and shouldn’t have anything on the rest of the line, including comments. The pre-processor also removes comments from the code.

#include directives

Tell the compiler’s pre-processor to include the declarations specified by a filename. The standard library functions are kept in a special directory, e.g. /usr/include, and this is the directory used with angled brackets. Users may also define header files, which are specified with a filename enclosed with quotes.

    #include <foo.h>    // System header file
    #include "bar.h"    // User header file "in local directory"

#define directives

Typically used to define constants which are textually substituted into the code.

    #define PI 3.14159
    #define SQUARE(X) ((X)*(X))

A better C++-like way to define constants is with the const keyword:

    const double PI = 3.14159;
Comments, Indentation, and WhiteSpace

The goal is to improve the readability of the code using careful use of these elements. Comments serve as a description and reminder of what the code is intended to do. Good use of indentation helps visually distinguish different control blocks of code and proper use of whitespace makes reading lines and expressions easier.

There are two styles of comments in C++, block-style and line-style:

    /* Old C block style comments. Can span multiple lines.
       May not be nested */

    double pi_root = sqrt(PI);   // Computes square root of PI.

Most editors support a smart indentation mode. The best rules are:

- Be consistent with the style you chose (same number of spaces, etc...)
- Indent the body of a function.
- Use spaces after comma separated arguments: (3.7,4.9,5.3) vs. (3.7, 4.9, 5.3)
- Break control blocks up consistently (e.g. where you put the braces)

The compiler will ignore whitespace, which includes blanks, tabs, and carriage returns between tokens, but not for example, inside of strings.
Identifiers

- Used to name functions and variables (and classes and types)
- Must start with a letter and consist only of alpha-numeric characters and the underscore character '_'. It is suggested that names starting with an underscore be avoided, since compilers often use special variables starting with underscores.
- Identifiers are case-sensitive. Pick a naming style. I use Mixed_Case separated with underscores for function names, Leading Capitals for Classes, and lowercase for built-in keywords and variable names.
- Choose names that are descriptive.
- Eschew abbreviations.

Keywords

<table>
<thead>
<tr>
<th>asin</th>
<th>continue</th>
<th>float</th>
<th>new</th>
<th>signed</th>
<th>try</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>default</td>
<td>for</td>
<td>operator</td>
<td>sizeof</td>
<td>typedef</td>
</tr>
<tr>
<td>break*</td>
<td>delete</td>
<td>friend</td>
<td>private</td>
<td>static</td>
<td>union</td>
</tr>
<tr>
<td>case</td>
<td>do</td>
<td>goto</td>
<td>protected</td>
<td>struct</td>
<td>unsigned</td>
</tr>
<tr>
<td>catch</td>
<td>double</td>
<td>if</td>
<td>public</td>
<td>switch</td>
<td>virtual</td>
</tr>
<tr>
<td>char</td>
<td>else</td>
<td>inline</td>
<td>register</td>
<td>template</td>
<td>void</td>
</tr>
<tr>
<td>class</td>
<td>enum</td>
<td>int</td>
<td>return</td>
<td>this</td>
<td>volatile</td>
</tr>
<tr>
<td>const</td>
<td>extern</td>
<td>long</td>
<td>short</td>
<td>throw</td>
<td>while</td>
</tr>
</tbody>
</table>

a. There are some new ones: bool, using, namespace, typename, explicit, mutable
Literals:
String literals are sequences of characters enclosed in double quotes, e.g. "str"

Character literals are single characters enclosed in single quotes, e.g. 'A'

<table>
<thead>
<tr>
<th>TABLE 3. Special Characters (The C++ Programming Language, pg 480)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>new-line</strong></td>
</tr>
<tr>
<td><strong>horizontal tab</strong></td>
</tr>
<tr>
<td><strong>vertical tab</strong></td>
</tr>
<tr>
<td><strong>backspace</strong></td>
</tr>
<tr>
<td><strong>carriage return</strong></td>
</tr>
<tr>
<td><strong>form feed</strong></td>
</tr>
<tr>
<td><strong>alert</strong></td>
</tr>
<tr>
<td><strong>backslash</strong></td>
</tr>
<tr>
<td><strong>question mark</strong></td>
</tr>
<tr>
<td><strong>single quote</strong></td>
</tr>
<tr>
<td><strong>double quote</strong></td>
</tr>
<tr>
<td><strong>octal number</strong></td>
</tr>
<tr>
<td><strong>hex number</strong></td>
</tr>
</tbody>
</table>

Integer and Floating point constants are numeric literals. By default a floating point constant is of type double, and an integer constant is of type int, although these can be explicitly set, using numeric suffixes of “F, F, E, e, L, l” and “L, l, UL, ul, Lu, lu, etc...”. Note integers starting with a leading zero, ‘0’, are presumed to be octal!

Operators:

<table>
<thead>
<tr>
<th>TABLE 4. (C++ Primer, Lippman, pg 305). Far column is non-overloadable</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
</tr>
</tbody>
</table>
| ~ | ! | , | = | < | > | <= | >= | . | *
| ++ | -- | << | >> | === | |= | &= | || | |
| += | -= | /= | %= | ^= | &= | |= | *= | ? |
| <<= | >>= | [ ] | ( ) | -> | -* | new | delete |

Separators:
Comments, whitespace, braces, etc...
**Primative Data Types**

Integral Types: (may also be unsigned)
- **char**
  
  
  ```
  char first = 'A', second = 66, third = 'c', newline = '
', zero = 0;
  ```
- **short int**
  
  ```
  short int = 100;
  short foo = -5;
  unsigned short bar = 3;
  ```
- **int**
  
  ```
  int x(5), y = 5, z;
  ```
- **long int**
  
  ```
  long huge = 2000000L;
  ```

Floating Point Types (inexact representation)
- **float**
  
  ```
  float a=6.0, b = 2.0;
  cout << "A / B is " << (a / b) << endl;
  2.99999976
  ```
- **double**
  
  ```
  double pi = 3.14159;
  double foo = 1.0049e5; // 100490.0
  ```

Other types
- **bool**
  
  ```
  bool flag = false; // old way works, is int flag = 0;
  ```
- Disregard F&K’s use of a string type. If your compiler supports it, fine.
**Arithmetic Expressions**

Integer arithmetic truncates:

```cpp
int a=3, b=4, c=8;
cout << "a / c = " << a / c << endl;
a / c = 0  // 15 / 0 crashes
// -19 / 5 is -3 or -4, implementation dependant.
```

Implicit type conversions

```cpp
int a = 5, c;
float b = 4.0;
c = b * a;  // same as c = (int) (b * (float) a)
```

The modulus operator

```cpp
int a=3, b=4, c=8;
cout << a % c << endl;  // prints 3
cout << c % a << endl;  // prints 2
cout << a % b << endl;  // prints 3
cout << b % a << endl;  // prints 1
cout << b % 0 << endl;  // Crashes?
```
#include <iostream>

int main() {
    int num;
    cin >> num; // num % 2 computes the remainder when num is divided by 2
    if (num % 2 == 0) {
        cout << num << " is even ";
    }
    return 0;
}
#include <iostream.h>

int main ( ) {
    // Local data
    char letter1, letter2, letter3; // input: letters to display
    // Enter letters and print message.
    cout << "Enter a three letter nickname and press return: ";
    cin >> letter1 >> letter2 >> letter3;
    cout << "Hello " << letter1 << letter2 << letter3 << ". ";
    cout << "Enjoy studying C++! " << endl;
    return 0;
}

// Converts some numbers from Celsius to Farenheit

#include <iostream.h>

int main ( ) {
    float a = -40.0, b = 0.0F, c(37.0);
    const float ratio = 9.0 / 5.0;
    float tmp;
    tmp = a * ratio + 32.0;
    cout << ", -40.0, and 37 degrees Celsius are: ";
    cout << tmp << ". ";
    tmp = b * ratio + 32.0;
    cout << tmp << ". ";
    tmp = c * ratio + 32.0;
    cout << tmp << " in degrees Farenheit. " << endl;
    return 0;
}