CS1B: Introduction to C++

Nadia & Domingos Begalli
Based on Work of Davender S. Malik and Douglas Schmidt

Saddleback College, Computer Science

CS1B, Fall 2017
C++ Introduction

Design

C with classes

• designed at AT&T Bell Labs by Bjarne Stroustrup (early 80s)
• the original cfront translated C++ into C for portability
• many native host machine compilers now exist
  ○ GNU g++
  ○ LLVM clang++
  ○ Intel C++ Compiler
  ○ Microsoft Visual C++
  ○ Oracle Studio (formerly SUN Studio)
  ○ C++ Builder
  ○ Comeau C/C++
  ○ etc...
C++ Introduction

Design

Characteristics

- stronger typechecking
- support for data abstraction
- support for object-oriented programming (OOP)
- support for generic programming (parameterized types)
C++ Introduction

Design

characteristics

- as with C, run-time efficiency is important
- no need for complicated run-time libraries (ADA, Java, C#)
- no need for virtual machines (Java, C#, Swift)
- no language-specific support for persistence or distribution
- concurrency support was added in C++11

C++ is a systems programming language
**C++ Introduction**

**Design**

**goals**

- compatibility with C libraries & traditional development tools
  - object code reuse
  - same storage layout of structures, which enables...
  - support for the standard ANSI C library
  - support for UNIX/Windows system calls
  - compatibility with the `make` recompilation utility

- however, “as close to C as possible, but no closer”
  - C++ is not a proper superset of C since...
  - C compatibility is not entirely maintained
C++ Introduction
Overview of the Language
C++ Introduction

Overview of the Library

- std::iostreams
- std::string
- std::complex
- std::vallarray
- std::locales
- std::allocators
- std::STL
- std::standard library
- C standard library

C++ Introduction

Features

key features

- data abstraction & encapsulation
  - class mechanism
  - name spaces
- object-oriented programming features
  - abstract classes
  - inheritance
  - virtual methods
- generic programming
  - parameterized types
- sophisticated error handling
  - exceptions
- run-time identification of object type (RTTI)
C++ Introduction

Learning C++

strategies

• focus on concepts, design, and programming techniques
  ○ don’t get lost in language features

• learn and apply patterns and idioms

• learn C++ gradually
  ○ no need to know every detail of C++ to write good code
standard & consistent style obviates code complexity

- https://google.github.io/styleguide/cppguide.html
- http://geosoft.no/development/cppstyle.html
Let's convert $209_{10}$ to base 16:

$209 / 16 = 13 R 1$

So that,

$209_{10} == D1$

What about $978_{10}$ two ways:

$978 / 16 = 61 R 2$

$61 / 16 = 3 R 13$

So that,

$978_{10} == 0x3D2 ... or$

$978 / 2 = 489 R 0; 489 / 2 = 244 R 1$

$244 / 2 = 122 R 0; 122 / 2 = 61 R 0$

$61 / 2 = 30 R 1; 30 / 2 = 15 R 0$

$15 / 2 = 7 R 1; 7 / 2 = 3 R 1$

$3 / 2 = 1 R 1$

$0011 1101 0010 == 3D2$
# C++ Introduction

## Style Matters

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

int main() {
    cout << "
    ***************
    * Your Name * CS1B TTh 2–5:30pm * Assig 1 *
    * Assig title * Due: Tuesday, August 30, 2016 *
    ***************
    
    #include <iostream>
    using namespace std;

    ****************************
    * Program name and description (vars, i/o), etc *
    ****************************

    int main() {
        cout << "
        ****************************
        * Your Name * CS1B MTWTh ... *
        ... // use proper indentation (no tabs, two spaces)
        ... // comments where necessary throughout code
        return 0;
    }

Nadia & Domingos Begalli CS1B Fall 2017
```
C++ Introduction

C++ special symbols, keywords

- arithmetic
  
  + - * / %

- punctuation
  
  . ; ? : ,

- two characters regarded as a single symbol
  
  <= != == >=

- keywords
  
  int, float, double, char, const, void, return

- identifiers are variables, constants, function names, etc composed of alphanumeric characters and the underscore
data type: a set of values and a set of operations

- simple data types; types that can hold only one value:
  
  ```
  int, char, bool, long, ... // integral
  float, double, long double // floating-point
  enum // enumeration; user-defined
  ```

- structured data types can store a collection of items

- `pointers` are types that hold memory addresses
# C++ Introduction

## Data Types; Memory Allocation

<table>
<thead>
<tr>
<th>type</th>
<th>size</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>32 bits</td>
<td>$-2147483648(-2^{31})$ to $2147483647(2^{31} - 1)$</td>
</tr>
<tr>
<td>unsigned int</td>
<td>32 bits</td>
<td>0 to $4294967295(2^{32})$</td>
</tr>
<tr>
<td>long</td>
<td>32 bits</td>
<td>$-2147483648(-2^{31})$ to $2147483647(2^{31} - 1)$</td>
</tr>
<tr>
<td>unsigned long</td>
<td>32 bits</td>
<td>0 to $4294967295(2^{32})$</td>
</tr>
<tr>
<td>long long</td>
<td>64 bits</td>
<td>$-2^{63}$ to $2^{63} - 1$</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>$-32768(-2^{15})$ to $32767(2^{15} - 1)$</td>
</tr>
<tr>
<td>unsigned short</td>
<td>16 bits</td>
<td>0 to $65535(2^{16})$</td>
</tr>
<tr>
<td>char</td>
<td>8 bits</td>
<td>$-128(-2^{7})$ to $127(2^{7} - 1)$</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>$+-3.4 \times 10^{38}$</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>$+-1.7 \times 10^{308}$</td>
</tr>
<tr>
<td>bool</td>
<td>1 bit</td>
<td>0 or 1 true or false</td>
</tr>
</tbody>
</table>
C++ Introduction

Arithmetic

arithmetic operators and precedence

- operators
  
  + // addition
  - // subtraction
  * // multiplication
  / // division
  % // modulus

- order of precedence

  * / %
  + -

  use parentheses to group expressions
Arithmetic

expressions

- integral expressions - operands are integers
- floating point expressions - operands are decimals
- mixed expressions - operands contain both
  - use rules of precedence
  - convert integers to floating points by adding a decimal part of 0


C++ Introduction

Type Conversion

type casting

- implicit type coercion; integers in mixed expression

- explicit type conversion; type casting

\[
\text{static\_cast}<\text{dataType}>(\text{expression})
\]

example:

```
static_cast<int>(5.8)  // evaluates to 5
```
Strings

the string type

- it is a programmer-defined data type
- it is a feature of ANSI/ISO standard C++ library
- it is not simple (eg, primitive, built-in) data type
- a sequence of zero or more characters
memory allocation

- allocation can be...
  - `static`: allocated on the stack at compile-time
  - `dynamic`: allocated in the heap at run-time

- mechanism: through variables and constants
  - `declaration` - allocates static memory (on the stack)
  - `malloc()`, `new[]` - allocates dynamic memory (in the heap)
  - `initialization` - populates memory
C++ Introduction
Memory Matters

declaring and initializing variables

- constants must be declared/initialized in the same statement

```cpp
const Type variable = value;
```

```cpp
const int CS1B_STUDENTS = 38;
const char SPACE = ' ';  
```

- variables can be declared/initialized in separate statements

```cpp
Type var1, var2,...; // declaration

double perimeter, area; // declares two vars
area = 12.5; // initializes (type double)
int depth = 10; // declares and initializes
```
input

- **standard input device** (UNIX CLI stdin; usually keyboard)

  ```
  cin >> variable >> variable ... ;
  ```

  where:

  - `cin` is a predefined object representing `stdin`
  - `>>` is the *stream extraction operator*

  example:

  ```
  // initialize variable age from keyboard input
  cin >> age;
  ```
**C++ Introduction**

**Input / Output**

- **standard output device** (UNIX CLI `stdout`; usually screen)
  
  ```
  cout << variable << variable ... ;
  ```

  where:
  
  - `cout` is a predefined object representing `stdout`
  - `<<` is the *stream insertion operator*

  example:

  ```
  cout << "Print on the screen.\n";
  cout << "Your age: " << age << endl;
  ```
Increment / Decrement Operators

**increment**

- increment value of `int` variable by one:
  ```cpp
  int x = 0;
x = x + 1;  // x is now 1
  ```

- using pre-increment or post-increment
  ```cpp
  x++;       // x is now 2
  ++x;       // x is now 3
  ```

- pre-increment vs post-increment
  ```cpp
  int y = ++x;  // x and y are now 4
  int z = x++;  // z is 4, x is 5
  ```
C++ Introduction

Increment / Decrement Operators

decrement

- decrement value of `int` variable by one:

  ```cpp
  int x = 10;
  x = x - 1;  // x is now 9
  ```

- using pre-increment or post-increment

  ```cpp
  x--;    // x is now 8
  --x;    // x is now 7
  ```

- pre-increment vs post-increment

  ```cpp
  int y = --x;  // x and y are now 6
  int z = x--;  // z is 6, x is 5
  ```
The Preprocessor

- **conditional directives**
  - 
  ```
  #if, #ifdef, #ifndef, #else, #elif, #endif
  ```

- **text macros**
  - 
  ```
  #define, #undef
  ```

- **include files**
  - 
  ```
  #include <iostream>
  #include <string>
  ```
**C++ Introduction**

**Program Structure**

```cpp
#include <iostream>

using namespace std;

int main() {
    statement_1;   // comment where
    statement_2;   // necessary
        .             // throughout code
        .
        .
    statement_n;
    return 0;
}
```
Commenting

• consider this block of code

```cpp
const double Z = 3.1514;
double x; // radius
double y; // perimeter
x = 3.0; // initialize radius
y = 2 * x * Z; // calculate perimeter
```

• versus this:

```cpp
const double PI = 3.1514;
double radius;
double circum;
radius = 3.0;
circum = 2 * PI * radius;
```