1

# C++ Basics - Part 1 (I/O, Flow Control)

CSC 340 - Appendices A, G, H, I, Credit to Hui Yang

*February 3, 2016*

2

# Overview

- **Comments, keywords, variables, data types, typedef**
- I/O
- Flow Control
- Functions
- Arrays
- Structures
- Strings
- File I/O
- Program Style and Documentation

3

# Keywords

- Part of the language
- Reserved for the language
- Examples
  - Data Types: `int, double, char`, etc.
  - Flow Control: `switch, for, else, while`

## Identifiers

4

- Named data items used in a program
- Use meaningful names that represent data you are storing
- Identifier rules:
  - First character must be a letter or an underscore
  - Remaining characters must be a sequence of letters, digits, or underscores

## Variables

5

- A named C++ identifier
- Represents a memory location that contains a value of a particular data type
- Examples:

```
int age = 0;
double radius = 2.5;
```

## Named Constants

6

- Variables whose value can not change during program execution
- Once initialized, cannot be changed
- Examples:

```
const double PI = 3.14;
```

# Data Types (and Assignment)

7

- Simple data types
  `int, double, bool, char, long, short, long double, long long`
- Declarator operators-based data types (T represents a type)

```
T a[n];       //array
T* p = null;  // pointer
T& r = p;     // reference variable
T f(A);       // function w/ arg of type A returning type T
```

- User defined data types
  `struct, class, union, enumeration`

# Type Ranges

8

- Machine dependent!!
- You can use `sizeof()` function to find out how much internal space a data object uses
- https://gist.github.com/jrob8577/5c22e85451283ef29524

# Type Ranges

9

| Name | Description | Size* | Range* | Precision |
|---|---|---|---|---|
| `char` | Character or small integer. | 1byte | signed: -128 to 127<br>unsigned: 0 to 255 | |
| `short int(short)` | Short Integer. | 2bytes | signed: -32768 to 32767<br>unsigned: 0 to 65535 | |
| `int` | Integer. | 4bytes | signed: -2147483648 to 2147483647<br>unsigned: 0 to 4294967295 | |
| `long int (long)` | Long integer. | 4bytes | signed: -2147483648 to 2147483647<br>unsigned: 0 to 4294967295 | |
| `bool` | Boolean value. It can take one of two values: true or false. | 1byte | `true` or `false` | |
| `float` | Floating point number. | 4bytes | +/- 3.4e +/- 38 (~7 digits) | 7 digits |
| `double` | Double precision floating point number. | 8bytes | +/- 1.7e +/- 308 (~15 digits) | 15 digits |
| `long double` | Long double precision floating point number. | 8bytes | +/- 1.7e +/- 308 (~15 digits) | 19 digits |
| `wchar_t` | Wide character. | 2 *or* 4 bytes | 1 wide character | |

# Type Compatibilities

10

- In general, store values in variables of the same data type
  - This is a type mismatch
    ```
    int variable;
    variable = 2.99;
    ```
  - If your compiler allows this, `variable` will most likely contain the value 2 (truncated)

# Type Compatibilities

11

- Implicit type conversion
  - Occurs during both assignment and expression evaluation
  - Order:
    - Promotions:
      ```
      char -> int -> unsigned -> long ->
      unsigned long -> float -> double -> long
      double
      bool <-> int
      ```

# Operators

12

- Same as in Java (except C++ allows operator overloading)
- Arithmetic operators
  +, -, *, /, %
- Assignment operators
  =, +=, -=, *=, /=
- Increment operators
  ++x, x++
- Precedence rules for expression evaluation
- Comparison operators
  >, <, >=, <=, ==, !=
- Logical operators
  &&, ||, ~, &, |, ^

# Typedef

13

- Create an alias of an existing data type
- Can make your program easier to modify and read
- Examples:

```
typedef int age;
age array[10];
typedef int * intptr;
```

# Overview

14

- Comments, keywords, variables, data types, typedef
- **I/O**
- Flow Control
- Functions
- Arrays
- Structures
- Strings
- File I/O
- Program Style and Documentation

# Basic I/O

15

- The `#include` directive adds library files to our programs
  - The iostream library provides basic input and output functionality with stream `cin` and `cout`

```
#include <iostream>
```

- The using directive includes a collection of defined names
  - To make `cin` and `cout` available to our program (without an explicit namespace)

```
using namespace std;
```

Use angle brackets for including built in libraries

16

## Input using cin

- `cin` is an input stream exposing data entered from the keyboard (standard input)
- The extraction operator (>>) removes data from the stream
- Example:

```
int counter, other;
cin >> counter >> other;
```

- This code reads two data items from `cin`
  - First value read is stored in `counter`
  - Second value read is stored in `other`

17

## Input using cin

- Multiple data items are separated by spaces
- Data is not read until the enter key is pressed
- Example:

```
cin >> v1 >> v2 >> v3;
```

  - Requires three space separated values

```
34 45 12 <enter>
```

18

## Output using cout

- `cout` is the standard output stream that can be written to with the insertion operator (<<)
- Example:

```
cout << "Please enter a number why not: ";
cin >> some_number;
```

- Does not insert new lines at end of string (unless explicitly included with an escape sequence)

19

## Character I/O

- The extraction operator skips any whitespace

```
char c1, c2;
in >> c1 >> c2;
// Input "a b" yields c1='a', c2='b'
```

- cin.get() reads individual characters including whitespace and special characters

```
cin.get(c1);
c1 = cin.get();
```

- We can output individual characters as well:

```
cout << c1;
cout.put(c1);
```

20

## I/O with Class string

- The insertion operator works for strings, as we've seen:

```
string s = "Hello, World!";
cout << s;
```

- The extraction operator can be used to input data for objects of type string:

```
string s1;
cin >> s1; // stops at whitespace!
```

21

## I/O with Class string

- A `getline` function used to read entire lines into a string variable (including whitespace)
  - This version is not a member of the `istream` class (it is a non member function, more on this later)
- Syntax:

```
getline( istream_object, string_object);
```

- Example:

```
string line;
cout << "Enter a line of input:\n";
getline( cin, line );
cout << line << "\n";
```

## Overview

22

- Comments, keywords, variables, data types, typedef
- I/O
- **Flow Control**
- Functions
- Arrays
- Structures
- Strings
- File I/O
- Program Style and Documentation

## Flow Control

23

- Same as Java (mostly)
- Decision: `if/else, switch`
- Loops: `for, while, do/while`

## Branching/Decisions If/Else

24

- Formal syntax

```
if ( <boolean_expression> )
  <yes_statement>
else
  <no_statement>
```

- Can also be used for multiple branches

```
else if ( <boolean_expression> )
```

- Caution (what's wrong with this):

```
if( x = 6 ) y++;
```

Angle brackets indicate a non-terminal in the language specification

x = 6 is always true (any non-zero value is true)

## Branching/Decisions
## Switch

25

- Formal syntax:

```
switch( <controlling_expression> ) {
  case <constant_1>:
    <statement_sequence_1>;
    break;
  case <constant_n>:
    <statement_sequence_n>;
    break;
  default:
    <default_statement_sequence>;
}
```

## Branching/Decisions
## Switch

26

- `break` is optional
  - If `break` is omitted from a case, case evaluation continues until a `break` is encountered, or the end of the `switch` statement is reached
- `default` is executed when no cases matche the controlling expression

## Loops

27

- Three types of loops in C++
  - While
    - Most flexible
    - No restrictions
  - Do while
    - Least flexible
    - Always executes loop body at least once
  - For
    - Natural “counting” loop

## Loops
## While

28

- Formal syntax:

```
while ( <boolean_expression> )
{
  <statement_sequence>
}
```

- Condition must be initialized before entering the loop
- Condition must be updated within the body of the loop

## Loops
## Do/While

29

- Formal syntax:

```
do
{
  <statement_sequence>
} while ( <boolean_expression> );
```

- Condition must be initialized before entering the loop
- Condition must be updated within the body of the loop
- Loop executed at least once
- Do not forget that final semicolon!

## Loops
## For

30

- Formal syntax:

```
for( <initializeCondition>; <boolean_exp>;
<update_condition> )
{
  <statement_sequence>
}
```

- Initialization, conditional check, and update all contained within the syntax of the loop