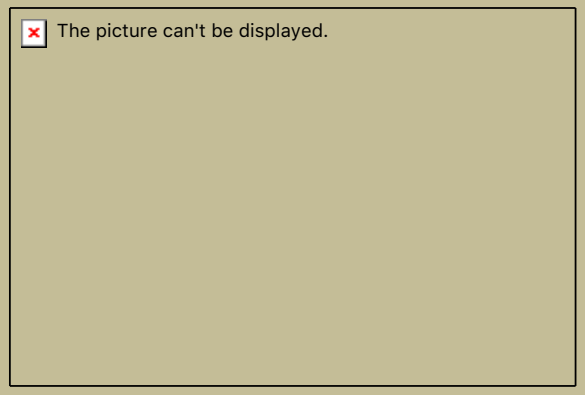
COMPUTER SCIENCE 1: STARTING COMPUTING
CSCI 1300



Ioana Fleming / Vipra Gupta
Spring 2018
Lecture 4

University of Colorado
Boulder

Agenda

- Representations → Code
- C++
  - Declaring variables
  - Data Types
  - Rules for variable names
  - Console Input / Output
    - cin “>>”
    - cout “<<“
  - Defining Functions
  - Operators / Operator precedence

Syntax of a Computer Language

- **Syntax *refers to the spelling and grammar of a programming language***.
- Computers are inflexible machines that understand what you type only if you type it in the exact form expected.
- The expected form is referred to as the correct syntax.
- For example: C++ expects a semi-colon at the end of each statement

Syntax Example

- **Syntax *refers to the spelling and grammar of a programming language*.**

- Syntax for defining a function

```
type functionName ( [type argname [, type ...] ] )
{
   statement;
   [ statement; …]
}
```

Example of Simple Program

```
// Author: CS1300 Fall 2017
// Recitation: 123 - Favorite TA
//
// First programs in C++

#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello CS1300 World!";
    return 0;
}
```

Example of Simple Program

```
// Author: CS1300 Fall 2017
// Recitation: 123 - Favorite TA
//
// First programs in C++

#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello CS1300 World!";
    return 0;
}
```

- Cloud9 example 1:
  - create your first program
  - preprocessor directives
  - main()

5 Building Blocks for Computational Representations

1. **Create a variable to store a value for later use**
2. **Modify the value of a variable**
3. **Get input or generate output**
4. Check if a statement is True or False
5. *Repeat a statement or collection of statements*
6. *Encapsulating a collection of statements*

- Cloud9 example 2:
  - creating variables of different types
  - following naming rules
  - variables must be declared before being used
  - declaration, initialization, reassignment

C++ Data Types

- int
  - Integer
- float
  - Floating point number
- char
  - Single character
- string
  - Character string
- bool
  - Boolean value which can only be TRUE or FALSE

C++ requires all variables be declared

- C++ wants to know how you will use that variable

- Knowing the type of data will allow C++ to create code that executes faster
  - important in higher level courses that process or perform analysis of large data sets

Rules for Variable Names

- Uppercase characters are distinct from lowercase characters.
- All variable names must begin with a letter of the alphabet or an underscore(_).
- After the first initial letter, variable names can also contain letters, numbers, and underscore characters.
- Special Characters are not allowed
- You cannot use a C++ keyword (reserved word) as a variable name.

Keyword Reserved Names

- alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t, char32_t, class, compl, const, constexpr, const_cast, continue, decltype, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, noexcept, not, not_eq, nullptr, operator, or, or_eq, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_assert, static_cast, struct, switch, template, this, thread_local, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while, xor, xor_eq

Formatted Output Operations

- The ***cout*** object is used together with the *insertion operator*, which is written as << (i.e., two "less than" signs).

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120; // prints number 120 on screen
cout << x; // prints the value of x on screen
```

The << operator inserts the data that follows it into the output.

- Cloud9 example 2:
  - cout examples

Examples of *cout* statement

```
cout << "Output sentence";
// prints Output sentence on screen

cout << 120 << endl;
// prints number 120 on screen

int x;
x = 100
cout << x << endl;
// prints the value of x on screen
```

```
Output sentence
```

Examples of *cout* statement

```
cout << "Output sentence";
// prints Output sentence on screen

cout << 120 << endl;
// prints number 120 on screen

int x;
x = 100
cout << x << endl;
// prints the value of x on screen
```

```
Output sentence120
```

Examples of *cout* statement

```
cout << "Output sentence";
// prints Output sentence on screen

cout << 120 << endl;
// prints number 120 on screen

int x;
x = 100
cout << x << endl;
// prints the value of x on screen
```

```
Output sentence120
100
```

Examples of *cout* statement

```
string Hello;
Hello = "good bye";

// If we want the text to show up
cout << "Hello" << endl;
// prints Hello

cout << Hello << endl;
// prints the content of variable Hello
```

```
Hello
```

Examples of *cout* statement

```
string Hello;
Hello = "good bye";

// If we want the text to show up
cout << "Hello" << endl;
// prints Hello

cout << Hello << endl;
// prints the content of variable Hello
```

```
Hello
good bye
```

Examples of *cout* statement

```
// Multiple insertion operations
// (<<) may be chained in a
// single statement:

  cout << "This " << " is a "
       << "single C++ statement";
```

```
This is a single C++ statement
```

Examples of *cout* statement

```
// Multiple insertion operations
// (<<) may be chained in a
// single statement:

  cout << "This " << " is a "
       << "single C++ statement";

// Chaining insertions is especially
// useful to mix literals and
// variables in a single statement:

  cout << "I am " << age
       << " years old and "
       << "my zipcode is "
       << zipcode;

// if age variable has value 24 and
// the zipcode variable had 80309
```

```
This is a single C++ statement
```

```
I am 24 years old and my zipcode is 80309
```

Operators

operator	description
+	addition
-	subtraction
*	multiplication
/	division
%	modulo
( )	ordering

5 * 3 + 1 = ?

(5*3) + 1
or
5 * (3 + 1)

1 + 5 * 3 = ?

Precedence	
Unary	+ -
Multiplicative	* / %
Additive	+ -

5 Building Blocks for Computational Representations

1. Create a variable to store a value for later use
2. Modify the value of a variable
3. Get input or generate output
4. Check if a statement is True or False
5. *Repeat a statement or collection of statements*
6. ***Encapsulating a collection of statements***

- Cloud9 example 3:
  - cin examples
- Cloud9 example 4:
  - creating functions:
    - to get values from user
    - to perform computation
    - to display result

```
. . .

int main (void){

  // Get user's favorite number
  int favorite;
  cout << "What is your favorite number?";
  cin >> favorite;
  cout << "your favorite number is"
       << favorite << endl;

  // Calculate one half of that number
  float half;
  half = favorite / 2.0;
  cout << "Half your favorite is"
       << half << endl;

  // Calculate twice that number
  int twice;
  twice = favorite * 2;
  cout << "Twice your favorite is"
       << Twice << endl;

    return 0;
}
```

Example of Simple Program

This code is procedural.

It performs a set of individual procedures.

We can abstract the individual procedures.

```
. . .

int main (void){

  // Get user's favorite number
  int favorite;
  favorite = GetUserFavorite();



  // Calculate one half of that number
  float half;
  half = favorite / 2.0;
  cout << "Half your favorite is"
       << half << endl;

  // Calculate twice that number
  int twice;
  twice = favorite * 2;
  cout << "Twice your favorite is"
       << Twice << endl;

    return 0;
}
```

Example of Simple Program

This code is procedural.

It performs a set of individual procedures.

We can abstract the individual procedures.

```
. . .
type name( ??? ) {

}
```

```
int main (void){

  // Get user's favorite number
  int favorite;
  favorite = GetUserFavorite()

  // Calculate one half of that number
  float half;
  half = favorite / 2.0;
  cout << "Half your favorite is"
       << half << endl;
    .
    .
    .
```

Example of Simple Program

Just like variables, functions must be defined before we use them.

```
. . .
int GetUserFavorite(void) {




}

int main (void){

  // Get user's favorite number
  int favorite;
  favorite = GetUserFavorite()

  // Calculate one half of that number
  float half;
  half = favorite / 2.0;
  cout << "Half your favorite is"
       << half << endl;
    .
    .
    .
```

Example of Simple Program

Just like variables, functions must be defined before we use them.

```
. . .
int GetUserFavorite(void) {
  int value;
  cout << ”What is your favorite number?";
  cin >> value;
  cout << “your favorite number is”
       << value << endl;
  return value;
}

int main (void){

  // Get user’s favorite number
  int favorite;
  favorite = GetUserFavorite()

  // Calculate one half of that number
  float half;
  half = favorite / 2.0;
  cout << “Half your favorite is”
       << half << endl;
    .
    .
    .
```

Example of Simple Program

Just like variables, functions must be defined before we use them.

```
. . .
int GetUserFavorite(void) {
  int value;
  cout << "What is your favorite number?";
  cin >> value;
  cout << "your favorite number is"
       << value << endl;
  return value;
}

int main (void){

  // Get user's favorite number
  int favorite;
  favorite = GetUserFavorite();

  // Calculate one half of that number
  float half;
  half = favorite / 2.0;
  cout << "Half your favorite is"
       << half << endl;
    .
    .
    .
```

Example of Simple Program

Similarly, we can abstract the other individual procedures.

```
. . .
int GetUserFavorite(void) {
  int value;
  cout << ”What is your favorite number?";
  cin >> value;
  cout << “your favorite number is”
       << value << endl;
  return value;
}

int main (void){

  // Get user’s favorite number
  int favorite;
  favorite = GetUserFavorite();

  // Calculate one half of that number
   PrintHalf(favorite);

  // Calculate twice that number
  PrintTwice(favorite);
    .
    .
    .
```

Example of Simple Program

Similarly, we can abstract the other individual procedures.

```
. . .
int GetUserFavorite(void) {
  int value;
  cout << ”What is your favorite number?";
  cin >> value;
  cout << “your favorite number is”
       << value << endl;
  return value;
}

void PrintHalf(int value) {
  float half;
  half = value / 2.0;
  cout << “Half your favorite is” << half << endl;
}

void PrintTwice(int value) {
  int twice;
  twice = value * 2;
  cout << “Twice your favorite is” << twice << endl;
}

int main (void){

  // Get user’s favorite number
  int favorite;
  favorite = GetUserFavorite();

  // Calculate one half of that number
   PrintHalf(favorite);

  // Calculate twice that number
  PrintTwice(favorite);
. . .
```

Example of Simple Program

We have created functions in the program for each of the abstract procedures.