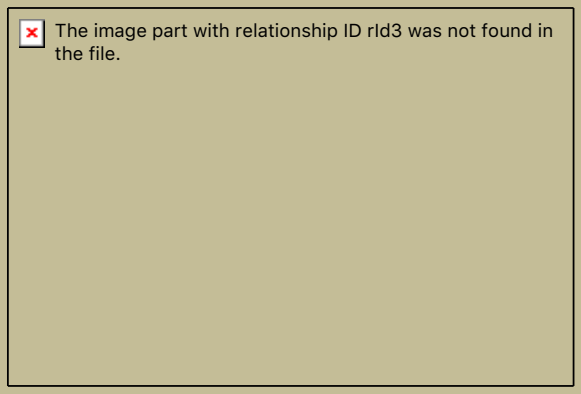
COMPUTER SCIENCE 1: STARTING COMPUTING
CSCI 1300



Ioana Fleming / Vipra Gupta
Spring 2018
Lecture 5

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

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

- Cloud9 example 3:
  - cin example with integers
  - cin example with strings

Learning Objectives

- Programmer-defined Functions
  - Defining, Declaring, Calling

Programmer-Defined Functions

- Write your own functions!
- Building blocks of programs
  - Divide & Conquer
  - Readability
  - Re-use
- Your "definition" can go in either:
  - Same file as main()
  - Separate file so others can use it, too

Components of Function Use

- 3 Pieces to using functions:
  - Function Declaration/prototype
    - Information for compiler
    - To properly interpret calls
  - Function Definition
    - Actual implementation/code for what function does
  - Function Call
    - Transfer control to function

Function Declaration

- Also called function prototoype
- An "informational" declaration for compiler
- Tells compiler how to interpret calls
  - Syntax:
    <return_type> FnName(<formal-parameter-list>);
  - Example:

```
double totalCost(int numberParameter,
                                 double priceParameter);
```

- Placed before any calls

Function Definition

- Implementation of function
- Just like implementing function main()
- Example:

```
double totalCost(  int numberParameter,
          double priceParameter)
{
 const double TAXRATE = 0.05;
 double subTotal;
 subtotal = priceParameter * numberParameter;
 return (subtotal + subtotal * TAXRATE);
}
```

- Notice proper indenting

Function Definition Placement

- Placed after function main()
  - **NOT "inside" function main()!**
- Functions are "equals"; no function is ever "part" of another
- Formal parameters in definition
  - "Placeholders" for data sent in
    - "Variable name" used to refer to data in definition
- return statement
  - Sends data back to caller

Function Call

- Just like calling predefined function
  `bill = totalCost(number, price);`
- Recall: totalCost returns double value
  - Assigned to variable named "bill"
- Arguments here: number, price
  - Recall arguments can be literals, variables, expressions, or combination
  - In function call, arguments often called "actual arguments"
    - Because they contain the "actual data" being sent

Function Example: Calculate Total Cost (1 of 2)

Display 3.5

```
1   #include <iostream>
2   using namespace std;

3   double totalCost(int numberParameter, double priceParameter);
4   //Computes the total cost, including 5% sales tax,
5   //on numberParameter items at a cost of priceParameter each.

6   int main( )
7   {
8       double price, bill;
9       int number;

10      cout << "Enter the number of items purchased: ";
11      cin >> number;
12      cout << "Enter the price per item $";
13      cin >> price;

14      bill = totalCost(number, price);
```

Function declaration; also called the function prototype

Function call

Function Example: Calculate Total Cost (2 of 2)

```
15      cout.setf(ios::fixed);
16      cout.setf(ios::showpoint);
17      cout.precision(2);
18      cout << number << " items at "
19           << "$" << price << " each.\n"
20           << "Final bill, including tax, is $" << bill
21           << endl;

22      return 0;
23  }

24  double totalCost(int numberParameter, double priceParameter)
25  {
26      const double TAXRATE = 0.05; //5% sales tax
27      double subtotal;

28      subtotal = priceParameter * numberParameter;
29      return (subtotal + subtotal*TAXRATE);
30  }
```

Function head

Function body

Function definition

SAMPLE DIALOGUE

Enter the number of items purchased: **2**
Enter the price per item: $**10.10**
2 items at $10.10 each.
Final bill, including tax, is $21.21

- Cloud9 example 5: the *totalCost* function
  - creating functions:
    - to get values from user
    - to perform computation
    - to display result

Alternative Function Declaration

- Recall: Function declaration is "information" for compiler
- Compiler only needs to know:
  - Return type
  - Function name
  - Parameter list
- Formal parameter names not needed:
  `double totalCost(int, double);`
  - Still "should" put in formal parameter names
    - Improves readability

Parameter vs. Argument

- Terms often used interchangeably
- Formal parameters/arguments
  - In function declaration
  - In function definition's header
- Actual parameters/arguments
  - In function call
- Technically parameter is "formal" piece while argument is "actual" piece*
  - *Terms not always used this way

Functions Calling Functions

- We're already doing this!
  - main() IS a function!
- Only requirement:
  - Function's declaration must appear first
- Function's definition typically elsewhere
  - After main()'s definition
  - Or in separate file
- Common for functions to call many other functions
- Function can even call itself → "Recursion"

Boolean Return-Type Functions

- Return-type can be any valid type
  - Given function declaration/prototype:
    ```
    bool appropriate(int rate);
    ```
  - And function's definition:
    ```
    bool appropriate (int rate)
    {
        return (((rate>=10)&&(rate<20))||(rate==0);
    }
    ```
  - Returns "true" or "false"
  - Function call, from some other function:
    ```
    if (appropriate(entered_rate))
         cout << "Rate is valid\n";
    ```

Declaring Void Functions

- Similar to functions returning a value
- Return type specified as "void"
- Example:
  - Function declaration/prototype:

```
void showResults( double fDegrees, double cDegrees);
```

    - Return-type is "void"
    - Nothing is returned

Declaring Void Functions

- Function definition:

```
void showResults(double fDegrees, double cDegrees)
{
      cout  << fDegrees
      << " degrees fahrenheit equals \n"
      << cDegrees << " degrees celsius.\n";
}
```

- Notice: no return statement
  - Optional for void functions

Calling Void Functions

- Same as calling predefined void functions
- From some other function, like main():

```
showResults(degreesF, degreesC);
showResults(32.5, 0.3);
```

- Notice no assignment, since no value returned
- Actual arguments (degreesF, degreesC)
  - –Passed to function
  - –Function is called to "do it's job" with the data passed in

More on Return Statements

- Transfers control back to "calling" function
  - For return type other than void, MUST have return statement
  - Typically the LAST statement in function definition
- return statement optional for void functions
  - Closing } would implicitly return control from void function