

CTEC I 335 Fall 2007

Lecture I: Programming in C++

Programming Fundamentals

- Computers are really very dumb machines -- they only do what they are told to do.
- In order to solve a problem using a computer, we must express the solution to that problem in a language that the computer can understand - through the **instruction set**.

- A computer **program** is just a collection of the instructions necessary to solve a specific problem.
- The approach or method that we use to solve the problem is called an **algorithm**.
- With the algorithm in hand, we can then write the instructions necessary to implement the algorithm on a particular computer system.

C++ Skeleton

```
// program comment block
```

```
#include <iostream>
```

```
// additional #includes
```

```
using namespace std ;
```

```
// named constants
```

```
int
```

```
main( )
```

```
{
```

```
// code
```

```
return 0 ;
```

```
}
```

Elements of Style

- Your code must be first and foremost **CORRECT** -- i.e., it produces accurate results and output.
- Your code must also be **MAINTAINABLE** -- that is, you or someone else should be able to repair/improve/rewrite/correct it easily.

Maintainability

- One way to achieve this is to use a **consistent programming style** in each of your programs.
- Some issues are purely “religious”, i.e., personal preference.
- Whichever style you use, the key is **consistency**! Pick your own style and stick with it!

I. Defensive Bracing

- **You must use braces for all block structures (if/else, do/while, while, for) even if the block contains only one statement!**
- The key here is maintainability.
- With a few keystrokes (i.e., adding the left brace and the right brace), you can later add as many statements as you wish to the block.

Bracing

```
if ( test == 1 ) // preferred style
{
    ...
}

if ( test == 1 ) { // K & R style
    ...
}

if ( test == 1 ) // GNU style
{
    ...
}
```

2. Naming

- Use **meaningful**, self-documenting names for all variables, constants, and functions.
- Declare one variable or constant at a time.
- Use a comment for each variable to explain its use and units, if applicable. For example,

Naming

- Sometimes, simple variable names are OK.
For example,

```
int i ; // loop counter
```

Variable Naming Conventions

```
double squareroot ; // all lower case
double square_root ; // underscore adds readability
double squareRoot ; // compound word convention
double SquareRoot ; // capitalized compound word convention

double SQUAREROOT ; // Wrong! use all caps for constants only!

// For example,

const double PI = 3.14159265 ; // constant
const int N_TEAMS = 6 ; // underscore adds readability
```

4. Spacing

- Use both horizontal (spaces, tabs) and vertical (blank lines) spacing in your code.
- Saving disk space is has not been a concern since the 1980's!
- Makes your code more readable, both on screen and on paper, for you and for others!

4. Program Comment Block

- Includes a summary of what the program does
- Contains the programmer's (or programmers') name(s)
- May contain a revision history, listing “program maintenance activities”

Program Comment Block Example

```
// vdiv.cpp:  
// Calculates a simple voltage divider circuit  
//  
// Mike Boldin, Niagara College Canada  
// <mboldin@niagarac.on.ca>  
//  
// Modification History:  
// -----  
// 2007.06.10   Created  
// 2007.06.12   Corrected number of dec. places
```

Sample Program

Comment block at top of program

```
//  
// Program:          sqroot.cpp  
//  
// Description:      This program calculates the square root for  
//                   a number that the user enters.  
//  
// Author:           Mike Boldin, <mboldin@gmail.com>  
//  
// Modification History:  
// 1999.01.12 Date of last known modification  
// 2007.06.20 Updated for CTEC1239/2007W  
//
```

Includes

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

Function declarations/prototypes

```
double squareRoot( double ) ;
```

Comment block for each function

```
//  
//  Function:      main  
//  Description:   Program entry point.  
//  Parameters:    None.  
//  Returns:       0  
//  Notes:         None.  
//
```

```
int  
main( )  
{  
    Comment for each variable  
    double number ;    // number entered by user  
  
    // Get the inputs  
  
    cout << "Enter a positive, rational number: " ;  
    cin >> number ;
```

Lots of spacing and proper indentation

```
if ( number <= 0.0 )
{
    cout << "This program can only find the square root"
        << endl
        << " of a positive rational number."
        << endl ;
}
else
{
    double root ; // square root of number

    root = square_root( number ) ;

    cout << "The square root of " << number << " is " << root
        << endl ;
}

return 0 ;
}
```

Example of a function comment block

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
//  Function:    square_root( n )
//
//  Description:    Compute the approximate value of the square root of s.
//
//  Parameters:    s    Number of which to find the square root.
//
//  Preconditions:    s is assumed to be a positive, rational number.
//
//  Returns:    Returns the square root of s.
//
//  Postconditions:    See above;
//                    square root is undefined for all other value of s.
//
//  Notes:
//
//  Use the Babylonian method, as described at:
//  http://en.wikipedia.org/wiki/Babylonian\_method#Babylonian\_method
//
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
double
square_root( double s )
{
    . . . function code goes here . . .
}
```

See the course web site
for more details . . .

<http://technology.niagarac.on.ca/courses/ctec1335/>