CISC 1110 (CIS 1.5)
Introduction to Programming Using C++

Spring 2012
Instructor : K. Auyeung

Email Address: kenny@sci.brooklyn.cuny.edu
Course Page: http://www.sci.brooklyn.cuny.edu/~kenny/cisc1110

Class Hours: MW 2:40 – 4:45PM 4428N

Agenda

- functions

- built-in/library functions:
  – cmath
  – cctype

- User defined functions review

- Call by Value

- Call by Reference

- Summary

Library Functions

- we have already used some library functions

- iostream C++ library:
– iostream.cout – iostream.cin

- string C++ library:
– string.length()
– string.find()
– string.replace()
– string.insert()

- stdlib C library: – srand()
– rand()

cmath library

• there is a standard library of useful math functions defined in C that is commonly used in C++

• the header file: #include <cmath>
using namespace std;

• these include, for example:
– double sqrt(double x)
– double pow(double x, double y)
– double sin(double x)

• these take arguments and return values, e.g.: double f = sqrt(4.0);

• many other functions are defined in math.h, including trigonometry functions (like sin, cos, tan), and constants like MATH_PI

• on-line reference for cmath: http://www.cplusplus.com/reference/clibrary/cmath/

- example:

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

int main() {
    int x1 = 7, y1 = 5;
    int x2 = 1, y2 = 3;
    double dist;

    dist = sqrt ( pow((double)(x2-x1),2) + pow((double)(y2-y1),2) );
    cout << "the distance from point (" << x1 << "," << y1 << ") "
         << "to point (" << x2 << "," << y2 << ") is " << dist << endl;

    return 0;
} // end of main()
```

• there is a standard library of useful character functions defined in C that is commonly used in C++

• the header file:

```
#include <cctype>
using namespace std;
int isalnum( int c )
int isalpha( int c )
int isdigit( int c )
int islower( int c )
int ispunct( int c )
int isupper( int c )
int tolower( int c )
int toupper( int c )
```

•checks if character argument is alphanumeric
•checks if character argument is alphabetic
•checks if character argument is a decimal digit
•checks if character argument is a lowercase letter
•checks if character argument is a punctuation character
•checks if character argument is an uppercase letter
•converts uppercase letter argument to lowercase
•converts lowercase letter argument to uppercase

• on-line reference for cctype: http://www.cplusplus.com/reference/clibrary/cctype/

• example:

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

int main() {
    bool q = false;
    char c;

    while ( ! q ) {
          cout << "enter a character (q to quit): ";
          cin >> c;
          cout << "you entered: " << c << endl;

          if ( islower( c )) {
                c = toupper( c );
          }

          cout << "upper case = " << c << endl;
          q = ( c == 'Q' );

    } // end while

    return 0;
} // end of main()
```

Writing Your Own Functions

- modularity

– we can divide up a program into small, understandable pieces (kind of like steps in a recipe)
– this makes the program easier to read
– and easier to debug (i.e., check and see if it works, and fix it if it doesn't work)

- write once, use many times

– if we have a task that will be performed many times, we only have to define a function once; then we can call (or invoke) the function as many times as we need it
– also, we can use parameters (or arguments) to use the function to perform the same task on or with different data values

How Functions Work

how functions work

• functions must be declared and defined before they can be called (or "invoked"); first you can declare a function prototype (or "header") and then later list the function definition; you can invoke the function anywhere in the code after you have listed the prototype

• example:

```
#include <iostream>
using namespace std;
void sayHello(); // declare function (prototype or header)

int main() {
  sayHello(); // call function return 0;

  return 0;
} // end of main()

void sayHello() { // define function
  cout << "hello\n";
} // end of sayHello()
```

User Defined Functions Review

- Two components to user defined functions
  - Function declaration (prototype)
  - Function definition

Function Prototype

- The function heading without the body of the function

```
[return type] functionName (parameter list);
```

- Parameters are variables which provide input to a function

```
[data type] identifier, [data type] identifier, …
```

Function Definition

- **Header**
  - Data type return or void
  - Function name
  - Parameter (formal parameter) list
- **Body**
  - The set of statements that constitutes the function
  - The body is written within braces `{…}`
  - Uses a `return` statement to return a value to the calling function

User Defined Example

```
#include <iostream>

using namespace std;

// Declares function
void whoAmI();      //function prototype

int main()  {

    whoAmI();

    return 0;
}

void whoAmI()  {  // defines function whoAmI

  cout << endl << "My name is John" << endl;
}
```

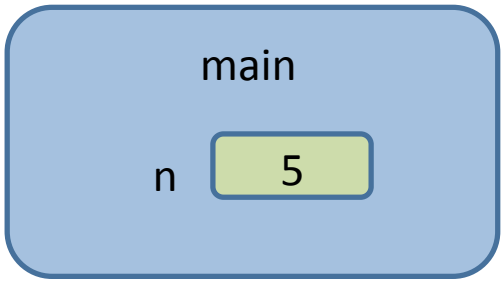
“Call by Value”

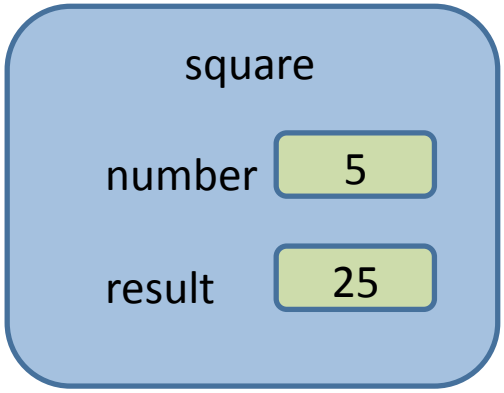
- Each argument in a function call is evaluated and **copied** to the variables specified in the formal parameters list

```
#include <iostream>

using namespace std;

// Declares function
int square(int);      //function prototype

int main()  {
     int n = 5;
     int answer = square(n);
     cout << “The answer is “ << answer;
     return 0;
}

int square(int number)  {  // defines function square

  int result = number * number;
  return result;
}
```

“Call by Value” Continued…

```
#include <iostream>

using namespace std;

// Declares function
int square(int);

int main()  {
     int n = 5;
     int answer = square(n);
     cout << “The answer is “ << answer;
     return 0;
}

int square(int number)  {

  int result = number * number;
  return result;
}
```





“Call by Reference”

- The address of the memory location of an argument (actual parameter) is passed to the function

- This allows a function to change the values of arguments

- Why?!

    - If the value of the argument needs to be changed
    - If you want to return more than one value from the function
    - Passing a memory address can save time and memory

“Call by Reference” Continued…

- **Reference parameter** – a formal parameter that receives a memory address
- The function prototype

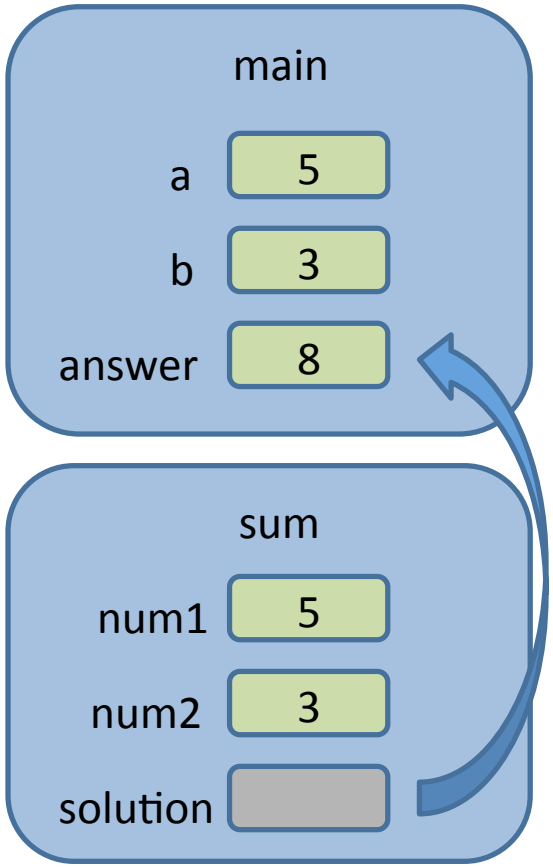
```
[return type] functionName (parameter list);
```

- Parameters are variables which provide input to a function

```
[data type] identifier[&], [data type] identifier[&], …
```

“Call by Reference” Example

```
// see reference.cpp
#include <iostream>

using namespace std;

//Function Prototypes
void sum(int, int, int&);

int main()  {

    int a = 5, b = 3, answer;

    sum(a, b, answer);

    cout << a << " + " << b << " = " << answer;

    return 0;
}

void sum(int num1, int num2, int& solution)  {

    solution = num1 + num2;
}
```



Summary

- **"Call by Value"** - Each argument in a function call is evaluated and **copied** to the variables specified in the formal parameters list

- **"Call by Reference"** - The address of the memory location of an argument (actual parameter) is passed to the function