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

Fall 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

- more on functions
- more on “call by value” and “call by reference” • passing strings to functions
- returning strings from functions
- variable scope
- global variables

“call by value” function parameters

- another call by value example:

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

int add( int, int ); // prototype

int main() {
   int p = 7, q = 5, sum;
   sum = add( p, q );
   cout << "sum=" << sum << endl;
} // end of main()


int add( int a, int b ) {
   int ret;
   ret = a + b;

   return( ret );
} // end of add()
```

• call by value parameters: same example as above, but using shorthand

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

int add( int, int ); // prototype

int main() {
   cout << "sum=" << add( 7, 5 ) << endl;
} // end of main()

int add( int a, int b ) {
   return( a + b );
} // end of add()
```

• the arguments in main() are constants (7, 5), which are used to initialize the variables
(a and b) inside the function add()

“call by reference” function parameters

- another call by reference example:

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

int add( int, int, int & ); // prototype
     void add( int a, int b, int &sum ) { sum = a + b;
} // end of add()

int main() {
    int p = 7, q = 5, sum;
    add( p, q, sum );
    cout << "sum=" << sum << endl;
} // end of main()
```

multiple function parameters

- you can write functions that have more than one parameter
- the parameters can be of any data type; they can even be different data types • example:

```
int doMath( int A, int B, char op ) {
  int result;
  if ( op=='+' ) {
    result = A + B;
  } else if ( op=='-' ) {
    result = A – B;
  } else if ( op=='*' ) {
    result = A * B;
  } else {
    result = -999;
  }
  return result;
} // end of doMath()
```

classic example, ”swap”, which uses reference parameters

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

void swap( int &, int & ); // prototype

int main() {
   int p = 7, q = 5;
   cout << "before: swap( p, q ); cout << " after:
   p=" << p << " q=" << q << endl; p=" << p << " q=" << q << endl;
} // end of main()

void swap( int &a, int &b ) {
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
   return;
} // end of swap()
```

compare with “noswap”, which uses value parameters

compare with “noswap”, which uses value parameters

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

void noswap( int, int ); // prototype

int main() {
   int p = 7, q = 5;
   cout << "before: p=" << p << " q=" << q << endl; noswap( p, q );
   cout << " after: p=" << p << " q=" << q << endl;
} // end of main()

void noswap( int a, int b ) {
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
   return;
} // end of noswap()
```

passing C++ strings to functions

- you pass C++ strings to functions in the same way that you pass primitive variables
- example “call by value string parameters:

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

void noswap( string, string ); // prototype

int main() {
   string p = "hello", q = "goodbye";
   cout << "before: p=" << p << " q=" << q << endl; noswap( p, q );
   cout << " after: p=" << p << " q=" << q << endl;
} // end of main()

void noswap( string a, string b ) {
   string tmp;
   tmp = a;
   a = b;
   b = tmp;
   return;
} // end of noswap()
```

- example “call by reference” string parameters:

```
#include <iostream>
#include <string>

using namespace std;
void swap( string &, string & ); // prototype

int main() {
   string p = "hello", q = "goodbye";
   cout << "before: p=" << p << " q=" << q << endl;
   swap( p, q );
   cout << " after: p=" << p << " q=" << q << endl;
} // end of main()

void swap( string &a, string &b ) { string tmp;
   tmp = a;
   a = b;
   b = tmp;
   return;
} // end of swap()
```

returning strings from functions

- you return C++ strings from functions in the same way that you return primitive variables
- example:

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

string getMove( char ); // function prototype

int main() {
   cout << "C = " << getMove( 'C' ) << endl;
   cout << "D = " << getMove( 'D' ) << endl;
} // end of main()

string getMove( char move ) {
   switch( move ) {
   case 'C':
      return "cooperate";
      break;
   case 'D':
      return "defect";
      break;
   }
   return " ";
} // end of getMove()
```

passing C strings to functions

- passing C strings is more complicated
- this is because C strings are always reference parameters (has to do with how they are stored)
- so just be careful if you pass C strings as parameters and know that if their value will changes inside the function, the new value will be retained outside
- here's an example:

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

int countVowels( char [] );
void transform( char [] );

int main() {
   char mystring[] = "hello";
   cout << "mystring = [" << mystring << "] ";
   cout << "number of vowels = " << countVowels( mystring ) << endl;
```

```
   cout << "before transform: [" << mystring << "]\n"; transform( mystring );
   cout << "after transform: [" << mystring << "]\n";
} // end of main()

int countVowels( char a[] ) {
   int sum = 0;
   for ( int i=0; i<strlen(a); i++ ) {
      if (( a[i] == 'A' ) || ( a[i] == 'a' ) ||
         (( a[i] == 'E' ) || ( a[i] == 'e' ) ||
         (( a[i] == 'I' ) || ( a[i] == 'i' ) ||
         (( a[i] == 'O' ) || ( a[i] == 'o' ) ||
         (( a[i] == 'U' ) || ( a[i] == 'u' ))       {
      sum++;
      }
   }
   return( sum );
} // end of countVowels()

void transform( char a[] ) {
   for ( int i=0; i<strlen(a); i++ ) {
   a[i] = toupper( a[i] ); }
} // end of transform()
```

• Note that you can return C strings from functions, but it is more complicated and involves a syntax and concepts that are beyond the scope of this class

• You will likely cover that in the next course (CISC 3110)

• So, if you want a function to return a string, then use C++ strings

variable scope

variables are defined within either a global or a local scope

• local variables are defined inside a function and these “go away” when the function exits

• global variables are defined outside of any function, and these do not go away (as long as the program is running)

• in the example below:
– a and b are local variables declared inside add(); their scope is the function add(); when add() exits, a and b no longer exist
– p and q are local variables declared inside main();
their scope is the function main();
they also go away when main() exits, which is the same thing as when the program exits, because main() is the special function that controls the program

global variables example

- example similar to those above, except using global variables

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

int add( int, int ); // prototype
int p = 7, q = 5; // declare global variables

int main() {
   cout << "sum=" << add( p, q ) << endl;
} // end of main()

int add( int a, int b ) {
   int ret;
   ret = a + b;
   return( ret );
} // end of add()
```