MECH215
Lecture Notes 3

Library Functions

In this section we turn our attention to using Standard Library functions. We have already seen a limited use of library functions (recall the use of the fabs function).

C++ contains many different libraries. To access a function within a particular library it is necessary to #include <library_name> in your C++ program. Before discussing the different libraries available in C++, we will give an example of the sqrt function which is found in the cmath library (or the old C math.h library).

sqrt Function

The two real roots of the general quadratic equation:

$a x^2 + b x + c = 0$

are given by the formula : $x = \frac{-b \pm \sqrt{(b - 4ac)}}{2a}$

We will omit complex roots for the time being.

The following program reads in three doubles for the values of a,b,c. It checks that $b - 4ac > 0$ and if this is true, then it computes the two real roots by PASSING a value to the sqrt library function:

```
// Author: Ted Obuchowicz and John Peach
// sqrt.cpp
// Example program illustrating use of square root library function
// found in the cmath library

#include <iostream>
#include <string>
#include <cmath>  // needed for the definition of the sqrt function

using namespace std;

int main() {

    double a,b,c;       // Coefficients for sqrt formula
    double radical;     // The radical inside the sqrt
    double root1;       // first root
    double root2;      // second root

    // Get the coefficients from the user
    cout << "Enter the coefficients of the equation: a, b, c: ";
    cin >> a >> b >> c;
```

```
    // Compute the radical
    radical = (b*b) - (4*a*c);

    // Make sure the radical is positive and a is not zero
    if ( (a != 0) && (radical > 0)  ) {
        // Calculate the roots
        root1 = (-b + sqrt(radical)) / (2*a);
        root2 = (-b - sqrt(radical)) / (2*a);

        // Report the values of the roots
        cout << "The roots are " << root1 << "  and "
             << root2 << endl;
    } else {

        // inform the user that there are no real roots
        cout << "The equation does not have two real roots!" << endl;
    }

    return 0;
}
```

When the program gets to the two lines:

```
root1 = (-b + sqrt(radical)) / (2*a);
root2 = (-b - sqrt(radical)) / (2*a);
```

a sequence of events occurs:

1. The appearance of the expression sqrt(radical) in the two lines will cause a FUNCTION INVOCATION to occur. We say that the variable radical is a FUNCTION ARGUMENT. The sqrt function function receives the value of this variable, it then calculates the value of the square root of whatever value was passed to it.
2. When the function has computed the calculation of the square root, it returns the value back to the program. In programming terminology, we say that there is an "invocation" of the square root function which "returns" a value back to the "calling program".

In order to make use of library functions we must first know several things concerning the function:

1. The name of the function
2. The number and type of arguments that the function expects
3. The return type

All this information is contained in what is known as the function prototype. For example, the prototype for the sqrt function is:

```
double sqrt(double);
```

The above prototype (which is found in the math.h library file, which is located in /usr/include/math.h

on our UNIX systems). We can interpret this command a meaning: the function is called sqrt; it expects a single formal parameter of type double; and the function return a double. The parameter list is the part enclosed in () after the function name. Note that there are no variable names just the data types. We can add variable names, and it is best to do so, but they are ignored by the compiler. Also notice that it ends with a semi-colon. A function definition will have a { instead of a ; after the parameter list.

The type of the actual parameter which is passed to a function should MATCH the type of the formal parameter given in the function prototype. If there is mismatch between the types of the formal parameter and the actual arguments passed to the function, the compiler will attempt to convert the types, if possible. For example, promoting an int to a double. If the usual conversions cannot convert the actual argument into the type specified by the formal parameter, the compiler reports an error.

Here are some examples of legal invocations of sqrt:

```
// Author: Ted Obuchowicz and John Peach
// sqrtSample.cpp
// example program illustrating use of square root invocation calls

#include <iostream>
#include <string>
#include <cmath>  // needed for the definition of the sqrt function

using namespace std;

int main() {

    cout << sqrt(2) << endl;    // integer 2 will be promoted to a
                                // double and a double is returned
    cout << sqrt(4) << endl;    // legal
    cout << sqrt(-2.0) << endl; // Not an error, it returns NaN
    return 0;
}
```

The program output is:

```
1.41421
2
-nan
```

One can even do something of the sort illustrated in the next program:

```
// Author: Ted Obuchowicz and John Peach
// quadraticRoot.cpp
// example program illustrating use of square root library function
// found in the cmath library
```

```
#include <iostream>
#include <string>
#include <cmath>  // needed for the definition of the sqrt function

using namespace std;

int main() {

    double num;              // Number to calculate the quadratic root
    double quadraticRoot;     // the quadratic root of num

    // Get a number from the user
    cout << "Enter a number: ";
    cin >> num;

    // Calculate the quadratic root and report
    quadraticRoot = sqrt(sqrt(num));  // use the concept that to
                                      // square root of a square root
                                      // is a quadratic root
    cout << "The quadratic root is " << quadraticRoot << endl;

    return 0;
}
```

An example of an illegal invocation (where the usual conversions are not able to convert the actual argument into the type defined by the formal parameter) are:

```
// Author: Ted Obuchowicz and John Peach
// sqrtTooManyParameters.cpp
// example program illustrating use of illegal square root invocation
calls

#include <iostream>
#include <string>
#include <cmath>  // needed for the definition of the sqrt function

using namespace std;

int main() {

    // This is illegal as there are two many parameters.
    // We are only allowed 1
    cout << sqrt(3.4, 5.6) << endl;

    return 0;
}
```

The errors reported by the compiler are:

```
#>g++ -o sqrtTooManyParameters sqrtTooManyParameters.cpp
sqrtTooManyParameters.cpp: In function 'int main()':
sqrtTooManyParameters.cpp:14:26: error: no matching function for call to
'sqrt(double, double)'
sqrtTooManyParameters.cpp:14:26: note: candidates are:
/usr/include/x86_64-linux-gnu/bits/mathcalls.h:158:1: note: double sqrt(double)
/usr/include/x86_64-linux-gnu/bits/mathcalls.h:158:1: note:   candidate expects 1
argument, 2 provided
/usr/include/c++/4.6/cmath:426:5: note: template<class _Tp> typename
__gnu_cxx::__enable_if<std::__is_integer<_Tp>::__value, double>::__type
std::sqrt(_Tp)
/usr/include/c++/4.6/cmath:420:3: note: long double std::sqrt(long double)
/usr/include/c++/4.6/cmath:420:3: note:   candidate expects 1 argument, 2 provided
/usr/include/c++/4.6/cmath:416:3: note: float std::sqrt(float)
/usr/include/c++/4.6/cmath:416:3: note:   candidate expects 1 argument, 2 provided
```

Note that the compiler reports the line number in the include file which contains the prototype

```
/usr/include/x86_64-linux-gnu/bits/mathcalls.h:158:1: note: double sqrt(double)
/usr/include/x86_64-linux-gnu/bits/mathcalls.h:158:1: note:   candidate expects 1
argument, 2 provided
```

and it also indicates the line number in the source code where the illegal invocation appears.

```
sqrtTooManyParameters.cpp:14:26: error: no matching function for call to
'sqrt(double, double)'
```

In C++ it is possible to have multiple possible candidates. This is called function overloading. That is what is being reported on the rest of the lines.

A function can return a single value, or it can return no value. For functions which do not return a value, the return type in the function prototype must be the built-in type called void. The following function prototype uses void to indicate that the function thisFunctionReturnsNoValue does not return a value;

```
void thisFunctionReturnsNoValue(int, int);
```

The above is a prototype of a function which takes two formal arguments of type int and does not return a value. You may think it strange that we would declare and define function with no return types, but they are going some work such as displaying a message, changing a file etc. Some programming languages make a distinction between a function that does not return a value and one that does. A function that does not return a value is called a subprogram. However, C++ makes no such distinction.

The following program uses what is known as a user-defined function, these will be explained in greater detail in a later section:

```
// Author: Ted Obuchowicz and John Peach
// void.cpp
// example program illustrating use of a function with no return
// value

#include <iostream>
#include <string>

using namespace std;

// Function prototypes
void thisFunctionReturnsNoValue(int, int);

int main() {

    // Repeated calls to a subprogram
    thisFunctionReturnsNoValue(1,2);
    thisFunctionReturnsNoValue(3,4);
    thisFunctionReturnsNoValue(34,7869);

    return 0;
}

// define the function first

void thisFunctionReturnsNoValue(int first, int second) {
    cout << "first is " << first << endl;
    cout << "second is " << second << endl;
}
```

The output is:

```
first is 1
second is 2
first is 3
second is 4
first is 34
second is 7869
```

Some Common C++ Libraries

We will now examine some of the common C++ libraries and the functions they contain. We first discuss various methods of including libraries. The standard C libraries are those with a .h suffix in their library name. The "h" stands for header and the contain function prototypes, struct declarations and compiler directives. They do not, or should not, contain any definitions of functions, just their declarations. They are included as:

```
#include <stdlib.h>
```

C++ libraries do not require the .h suffix and are included by simply specifying the name of the library within the <> brackets as in:

```
#include <iostream>
```

Some C++ libraries are derived from the standard C libraries and have a 'c' as the prefix as in:

```
#include <cassert>
#include <cmath>
```

Again, for the derived libraries, there is no need for the .h suffix in the library name.

The best source of information for access to standard libraries is your compiler documentation and a language reference manual.

The stdlib Library

This is a C-based library called "standard lib". It contains a collection of miscellaneous functions and type definitions. Some of the commonly used functions found in stdlib are:

```
int abs(int)
```
Returns the absolute value of the integer argument passed

```
void exit(int)
```
Causes the calling program to terminate with a return value equal to the parameter:
```
exit(0); // terminate program with a return value of 0
exit(1); // terminate program with a return value of 1
```

In linux it is possible to see the return value from a program by issuing the following command after the program has ended.
```
#> echo $?
```

```
#include <iostream>
#include <string>
#include <stdlib.h>

using namespace std;

int main() {

    exit(1);
    cout << "This will never be printed " << endl;
}
```

This program when compiled and run will do nothing and control will be passed back to the operating system.

Here is an interesting experiment to perform on a UNIX system. From a X-window, type the following from the UNIX prompt:

```
exit(0)
```

What happens to your window? It has disappeared. exit is actually a system call. C was originally developed to write the UNIX operating system. There is a strong coupling between UNIX and the C-language.

```
int rand()
```

Returns a pseudo-random number between 0 and RAND_MAX, where RAND_MAX is an operating system defined value. (On our UNIX system RAND_MAX is defined in /usr/include/stdlib.h as #define RAND_MAX 2147483647) The default seed value is set to 1.

```
void srand(unsigned int val)
```

Sets the seed value used by the rand function to the value given in the formal parameter.

Example programs making use of rand() and srand():

```
// Author: Ted Obuchowicz and John Peach
// randomNotSo.cpp
// example program illustrating use of random number generator

#include <iostream>
#include <string>
#include <stdlib.h>

using namespace std;

int main() {

    for(int i = 0 ; i <= 9 ; i++) {
        cout << rand() << endl;  // print out 10 random numbers
                                 // between 0 and RAND_MAX
    }

    return 0;
}
```

EVERY time that we run the above program it will produce the same values (not very random). These values will vary from machine to machine but not on the same machine:

```
1804289383
846930886
```

```
1681692777
1714636915
1957747793
424238335
719885386
1649760492
596516649
1189641421
```

If we wish to produce different sequences every time we call the rand() function, we must ensure that the starting seed value is different:

```
// Author: Ted Obuchowicz and John Peach
// randomSeeded.cpp
// example program illustrating use of random seed function

#include <iostream>
#include <string>
#include <stdlib.h>

using namespace std;

int main() {
    int seedValue;   // Seed value for pseudo random number generator

    // Get the seed value
    cout << "Please enter a seed value: ";
    cin >> seedValue;

    // Seed the pseudo random number generator
    srand(seedValue);

    // Display some values
    for(int i = 0 ; i <= 9 ; i++) {
        cout << rand() << endl;  // print out 10 random numbers
                                 // between 0 and RAND_MAX
    }

    return 0;
}
```

The program's output when we run it two times with different initial seed values is:
First Run:

```
Please enter a seed value: 5
590011675
```

```
99788765
2131925610
171864072
317159276
171035632
602511920
963050649
1069979073
1919854381
```

Second Run:

```
Please enter a seed value: 6
290852541
2066988985
1717401112
865455665
182811308
1730087285
1385463286
531287043
665477002
2111229778
```

We can modify the above program to display random numbers between 0 and 6 by changing this line:

```
cout << rand() % 7 << endl;
```

The output is now:

```
0
1
3
0
3
6
4
6
6
5
0
2
0
1
2
4
5
```

```
4
6
6
2
```

An "upside-down" histogram which plots the frequency of occurrence of each number is:

```
0     1     2     3     4     5     6
-----------------------------------
*     *     *     *     *     *     *
*     *     *     *     *     *     *
*           *           *           *
*                                   *
```

You can see that even for a very small sample space of 20 values, the distribution is quite uniform. Each of the 7 values between 0 and 6 is equally likely to occur.