cis1.5 spring2009 lecture III.1

today we are going to talk about...

- what are functions and why to use them
- built-in/library functions:
  - cmath
  - cctype
  - formatted output
- constants

advantages of 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
- some functions are *standard* and come with the language; these are referred to as *library* functions
- in order to use these, you have to tell the compiler how they are defined, and you do this by using a `#include` statement at the beginning of your code

- for example:

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

- the file that is being "included" is called a *header* file
- the `using namespace std` part tells the compiler that the header file is part of the standard ("std") C++ "namespace"
- if the header file ends in **.h**, then it is part of C (instead of C++), but it can be used with C++ just the same;
- for example:

```
#include <stdlib.h>
#include <sys/time.h>
#include <iostream>
using namespace std;
```


library functions

- we have already used some *library* functions
- *iostream* C++ library:
  - `iostream.cout`
  - `iostream.cin`
- *stdlib* C library:
  - `srand`
  - `rand`

constants

- **constants** are types of data values that are defined in programs and do NOT change while the program runs
- these are similar to **variables** because they have a *name*, *data type* and *value*
- BUT they are DIFFERENT from variables because the value DOES NOT CHANGE
- some libraries define constants as well as functions
- you can also define your own constants

- to define a constant, use the keyword `const`
- for example:

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

int main() {
  const int NORTH = 0;
  const int WEST  = 1;
  const int SOUTH = 2;
  const int EAST  = 3;
  cout << "the robot is moving " << EAST << "\n";
} // end of main()
```


cmath library

- there is a standard library of *math functions* defined in C that is typically 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 trig functions (like sin, cos, tan), and constants like `MATH_PI`
- on-line reference for **math.h**:
  `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;
} // end of main()
```

cctype

- the header file:

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

- `int isalnum( int c )` checks if character argument is alphanumeric
- `int isalpha( int c )` checks if character argument is alphabetic
- `int isdigit( int c )` checks if character argument is a decimal digit
- `int islower( int c )` checks if character argument is a lowercase letter
- `int ispunct( int c )` checks if character argument is a punctuation character
- `int isupper( int c )` checks if character argument is an uppercase letter
- `int tolower( int c )` converts uppercase letter argument to lowercase
- `int toupper( int c )` converts lowercase letter argument to uppercase
- on-line reference for **cctype.h**:
  `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
} // end of main()
```


formatted output

- part of `iostream` library
- `cout.setf()` defines the type of output field
- `cout.precision()` sets the decimal precision (for real numbers)
- `cout.width()` sets the width of the output field
- example:

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

int main() {
  cout << "here's a table with lined-up columns:\n";
  cout.width( 10 );
  cout << "monday";
  cout.width( 10 );
  cout << "tuesday";
  cout.width( 10 );
```



```
  cout << "wednesday";
  cout << endl;
  cout.width( 10 );
  cout << "1";
  cout.width( 10 );
  cout << "2";
  cout.width( 10 );
  cout << "3";
  cout << endl;
} // end of main()
```

- output:

```
here's a table with lined-up columns:
    monday   tuesday wednesday
         1         2         3
```

- another example:

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

int main() {
  const int A = 5;
  const double B = 3.4568;
  double C;
  cout << "output using fixed precision, 2 decimal places:\n";
  cout.setf( ios::fixed );
  cout.precision( 2 );
  cout << "B=" << B << endl;
  cout << "output using width=10, left justified:\n";
  cout.setf( ios::left );
  cout.width( 10 );
  cout << "B=" << B << endl;
  cout << "output using width=10, right justified:\n";
  cout.setf( ios::right );
```



```
  cout.width( 10 );
  cout << "B=" << B << endl;
  cout << "you have to repeat the formatting if you want the same thing again:\n"
  C = sin( B );
  cout.setf( ios::right );
  cout.width( 10 );
  cout << "C=" << C << endl;
} // end of main()
```

- output:

```
output using fixed precision, 2 decimal places:
B=3.46
output using width=10, left justified:
B=        3.46
output using width=10, right justified:
        B=3.46
you have to repeat the formatting if you want the same thing again:
        C=-0.31
```