HEADERS, THE MATH LIBRARY AND FORMATTED OUTPUT

Today

- Today we will cover some final things on functions:
  - Headers
  - The math library
- and some things that aren't directly to do with functions but which we need to cover at this point.
  - Scope
  - Constants
  - Formatted output

Headers

- These were mentioned in the notes, but we didn't explicitly cover them in class.
- We have been writing functions with the function definition before `main()`.
- I said (and it is true) that we have to tell the compiler about the function before we use it.
- But it turns out that we don't have to tell the compiler *everything*.
- We can just give it the information about the name of the function, what arguments the function takes, and what value it returns,

- Instead of:

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

void sayHello() // define function
{
 cout << "hello" << endl;
 return 0;
}

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

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

void sayHello(); // function header only

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

void sayHello() // define function
{
  cout << "hello" << endl;
  return 0;
}
```


Math library

- In the roomba example, let's imagine we want to see how far the roomba is from some piece of dirt.
- We have `x` and `y` which give us the roomba's position.
- We have `dirtX` and `dirtY` which give us the position of the dirt.
- The distance between them is:

$$distance = \sqrt{(x - dirtX)^2 + (y - dirtY)^2}$$

- How can we compute this?

- The squares are easy enough to compute.

  `(x - dirtX) * (x - dirtX)`

  and

  `(y - dirtY) * (y - dirtY)`

- For the square root we can use the *math library function* `sqrt`.

```
  distance = sqrt(((x - dirtX) * (x - dirtX))
                 + ((y - dirtY) * (y - dirtY)));
```

- To use the math library, we need to add in

  `#include<cmath>`

  at the start of the program.

- The math library contains a bunch of other functions:
- `double pow( double x, double y )`
  raises `x` to the power of `y`. This works for fractional `y`.
- `double sin( double x )`
  which does the usual trig function. The other trig functions are also in the library:
- `double cos( double x )`
- `double tan( double x )`
- `double asin( double x )`
- `double acos( double x )`
- `double atan( double x )`

Scope

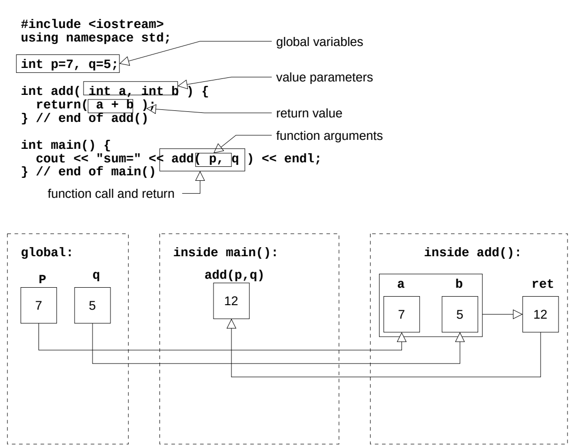
- Variables are defined within either a *global* or a *local* scope
- *Local* variables are defined inside a function and these “go away” when the function exits.
- *Global* variables are defined outside of any function, and these do not go away (as long as the program is running)
- Global variables should be used with care.


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

int p = 7, q = 5; // declare global variables

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

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

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;
} // 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


```
#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
```

- Each `cout` prints to a new 10 character wide field.
- By default the text is aligned at the righthand side of the field.
- Note that you have to repeat the `cout.width( 10 );`

```
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;
} // 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
        C=-0.31
```


Summary

- This lecture has finished what we need to know of fucntions for now, dealing with headers and the math library.
- We also talked about scope and constants.
- These often come together since constants are often set up as with global scope.
- (If a global value is a `const`, it is hard for it to cause side-effects).
- Finally we talked about formatted output.