Creating our own functions

- so far we have just used existing library functions
- we can create our own functions for use within a program
- we'll need to provide definitions that include
- the name of the function
- it's return type (what kind of data does it return when done)
- the types and order of expected parameters
- the code the function will run when called
- the value the function will return

Declaration syntax

- C++ has a fixed syntax for defining a function

```
retType funcName ( paramtype paramName, ptype2 pName2 )
{
    // code statements function will execute
    return value;
}
```

Example: average of three params

- here is a sample function to compute the average of three floats passed to it

```
float average(float num1, float num2, float num3)
{
    // we can refer to the params by name
    float total = num1 + num2 + num3;
    return (total/3); // the result to be sent back
}
```

- and here is a sample call to the function

```
x = average(10, 3.7, 204);
```

Parameters

- the average function knew it expected three parameters, and that within the function it would refer to the first one as num1, the second as num2, the third as num3
- the caller passes any three values - the first gets stored in the function's num1, the second in num2, the third in num3

```
float a = 10, b = 20, c = 30;
float result;
result = average(a, b, c);
```

- a's value gets copied to num1, b's to num2, c's to num3

Return values

- the function specifies what type of data it will return (e.g. float) and anywhere in the function it can have a return statement that sends back a value of that type, e.g.

```
return 217.5 * x + 7;
```

- when a return statement runs, the function *immediately* ends and sends back the value
- the value returned must be ok for the return type specified by the function

Example: multiple calls

```
#include <iostream>
using namespace std;
int sub1(int x)
{
    int result = x - 1;
    return x
}
```

```
int main()
{
    int a = 10;
    int b = 200;
    int c = sub1(a); // returns 9 to c
    c = sub1(b);     // returns 199 to c
    a = sub1(c);     // returns 198 to c
}
```

Code standards

- to help keep our code readable and maintainable (by ourselves and others), we adhere to certain standards in the layout of our code and naming of code elements
  - use meaningful names for variables, functions, constants, etc to clarify their purpose
  - consistently indent each statement (code and comment) inside a function or the main routine (e.g. 4 spaces)
  - use blank lines to clearly divide the code into sections
  - add comments to clarify named elements and code sections
- for the code standards for this course, see csci160/courses/csci160/labs/standards.html

Prototypes/forward declaration

- prototypes are 1-line versions of our functions that tell the compiler the function return type, name, and parameter list but don't include the function body, e.g.

```
int somefunction(float a, int b, char c);
// note the presence of the semicolon in the prototype
```

- our standards will require we put forward declarations for all functions before the main routine, and the full implementations of the functions afterward

Example with prototypes

```
#include <cstdio>
float miles2kms(float miles); // calculate #kms matching specified miles
int main()
{
   float milesDist = 4.5; // starting distance in miles
   float kmDistance = miles2kms(milesDist); // converted distance
}
float miles2kms(float miles)
{
   const float mperkm = 0.623; // number of miles in one km
   float kms = miles * mperkm; // km distance corresponding to miles
   return kms;
}
```

Functions with no parameters

- some functions do not require parameters, e.g. consider the welcome function below that simply gets (and returns) a number from the user

```
float getNumber()
{
        printf("enter a number: ");
        float userEntry;
        scanf("%f", &userEntry);
        return userEntry;
}
```

- a call could look like

```
x = getNumber();
```

Functions with no return value

- not all functions have to return a value, consider the welcome function below that simply displays information

```
void welcome( int visits )
{
      printf("Welcome back!\n");
      printf("This is your %dth visit!\n", visits);
}
```

- such functions use a return type of *void*, and cannot return data