Outline

- In this lesson, we will:
  - Review mathematical functions
  - Look at the use of functions in C++
  - Describe function declarations
    - Their relation to the domain and range
  - Parameters and arguments
  - Example: a fast sine function
  - Side effects
  - Functions with no return values
    - The `void` keyword
  - A second-last look at understanding `int main()`

Using functions

- In secondary school mathematics courses, you were introduced to numerous functions:
  - The trigonometric functions $\sin(x), \cos(x)$, etc.
    - Possibly including hyperbolic functions and the inverses of these
  - The exponential and logarithmic functions $e^x, \ln(x), \log_{10}(x)$
  - The absolute value $|x|$
  - The square root $\sqrt{x}$
  - The ceiling and floor functions $\lceil x \rceil, \lfloor x \rfloor$
  - The greatest common divisor and least common multiple functions $\gcd(m,n), \operatorname{lcm}(m,n)$
  - The maximum or minimum of two arguments $\max(m,n), \min(m,n)$

Using functions

- All of these have some properties in common:
  - Each function requires a fixed number of arguments that must be of a certain type, either integers or real numbers
  - Given the same arguments, the functions return the same value

- Many of these functions are implemented in the `cmath` library

Using functions

```
#include <iostream>
#include <cmath>
int main();

int main() {
        std::cout <<   "sin(3.2) = " <<      std::sin( 3.2 )   << std::endl;
        std::cout <<   "tan(3.2) = " <<      std::tan( 3.2 )   << std::endl;
        std::cout <<   "csc(3.2) = " << (1.0/std::sin( 3.2 )) << std::endl;
        std::cout <<   "cot(3.2) = " << (1.0/std::tan( 3.2 )) << std::endl;
        std::cout <<  "sinh(3.2) = " <<     std::sinh( 3.2 )   << std::endl;
        std::cout <<  "tanh(3.2) = " <<     std::tanh( 3.2 )   << std::endl;
        std::cout <<  "acos(0.2) = " <<     std::acos( 0.2 )   << std::endl;
        std::cout << "asech(0.2) = " <<  std::acosh( 1.0/0.2 ) << std::endl;

        return 0;
}
```

Must use `std::sin( 3.2 )`
- Mathematicians sometimes use $\sin\theta$ or $\sin 2\theta$

Output:
```
sin(3.2) = -0.0583741
tan(3.2) = 0.0584739
csc(3.2) = -17.1309
cot(3.2) = 17.1017
sinh(3.2) = 12.2459
tanh(3.2) = 0.996682
acos(0.2) = 1.36944
asech(0.2) = 2.29243
```


Using functions

- In each case, the argument is of type `double`
  - A double-precision floating-point number
  - The output is a floating-point number

  - The compiler knows this, so while you compile
    `std::cout << std::sin(32);`
    the compiler knows that:
    - The integer 32 must be converted to a floating-point number
    - The above statement must call the routines for printing a floating-point number

Function declarations

- How does the compiler know this about the sine function?
  - We must declare the sine function in a manner similar to `main()`
    `int main();`
- This says `main()` does not have any parameters and it returns an integer
- For the sine function, we know it has a domain and range:

$$\sin : \mathbf{R} \to \mathbf{R}$$

name domain range

Function declarations

- Suppose we wanted to define a polynomial p such that when it is called with an argument x, it returns the value

$$5x^2 - 3x - 9$$

- To this point, we have seen that `int` represents that the return type is an integer
  - Polynomials, however, are defined for all real numbers
    - Floating point numbers in C++
  - A type for floating-point numbers is `double`
    - Short for *double-precision floating-point numbers*

Function parameters

- Additionally, the polynomial
$$5x^2 - 3x - 9$$
requires a *variable* or *parameter* x
  - x can take on any real value, so we call it a *variable*
  - The result of the polynomial depends on x, so we say x is a parameter
- The function declaration is
```
double p( double x );
```
- This function:
  - Has the identifier p
  - Takes a variable parameter x that must be a floating-point number
  - Returns a floating-point number

Function definitions

- The function int main() simply returned 0
  - To evaluate
$$p(x) \overset{\text{def}}{=} 5x^2 - 3x - 9$$
we must perform an arithmetic calculation
- The *function definition* would be:
```
double p( double x ) {
    return 5.0*x*x - 3.0*x - 9.0;
}
```

Using `5.0` emphasizes to the reader that it is a floating-point number.

Parameters and arguments

- In the *function definition*, variable x is referred to as a *parameter*:
```
double p( double x ) {
    return 5.0*x*x - 3.0*x - 9.0;
}
```
the parameter x

- If you call this function with a value that is to be used in the function, that value is said to be the *argument*:

the argument `4.2`
```
int main() {
    std::cout << "p(4.2) = " << p( 4.2 )
              << std::endl;
    return 0;
}
```


Another example

- Suppose we wanted to define a bivariate polynomial q such that when it is called with arguments x and y, it returns the value
$$x^2 - 2xy + y^2 = (x - y)(x - y)$$
- The function declaration and definitions could be
```
double q( double x, double y );

double q( double x, double y ) {
    return x*x - 2*x*y + y*y;
}
```

Alternate function definition:
```
double q( double x, double y ) {
    return (x - y)*(x - y);
}
```

Another example

- Question: which implementation is faster?

```
double q( double x, double y ) {
    return x*x - 2*x*y + y*y;
}
```

```
double q( double x, double y ) {
    return (x - y)*(x - y);
}
```

- These require:
  - Four multiplications and two addition/subtractions
  - One multiplication and two addition/subtractions (or just one?)

The greatest common divisor

- The greatest common divisor (gcd) is another function you saw in secondary school
  - E.g., gcd(42, 70) = 14
  - It depends on two integer parameters and returns an integer

$$\mathrm{gcd} : \mathbf{Z} \times \mathbf{Z} \to \mathbf{N}$$

name

domain

Cartesian product of two sets of integers

range

natural numbers

  - Its function declaration would be

```
unsigned int gcd( int m, int n );
```


Other examples

- Some functions you saw in secondary school were represented graphically:
  - Exponents were written as superscripts: x^y
  - The square root was written with a radical symbol: $\sqrt{x}$
  - n^{th} roots had even further decorations: $\sqrt[n]{x}$
  - The absolute value was two bars: $|x|$
- In C++, we are restricted to functions and identifiers:

```
double pow( double x, double y );
double sqrt( double x );
double sqrt( double x, int n );
double abs( double x );
```


Other examples

- The standard mathematics library has some of these:

```
double std::pow( double x, double y );
double std::sqrt( double x );
double std::abs( double x );
```

- For the n^{th} root, the user is expected to use pow:

$$\sqrt[n]{x} = x^{\frac{1}{n}}$$

A fast sine function

- Engineers are always concerned with the trade-off between precision and speed (run times)
  - The `std::sin(…)` function gives 16-decimal digits of precision
  - It's also relatively slow
- Suppose we know we only need to calculate $\sin(x)$ for

$$0 \le x \le \frac{\pi}{2}$$

- Can we find a *good enough* approximation of $\sin(x)$ restricted to this interval?

A fast sine function

- Without proof, the polynomial

$$p(x) = \frac{4\pi - 16}{\pi^3} x^3 + \frac{12 - 4\pi}{\pi^2} x^2 + x$$

satisfies:

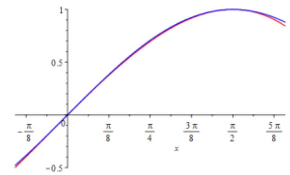
$$p(0) = 0 \qquad p\left(\frac{\pi}{2}\right) = 1$$

$$\frac{\mathrm{d}p}{\mathrm{d}x}(0) = 1 \quad \frac{\mathrm{d}p}{\mathrm{d}x}\left(\frac{\pi}{2}\right) = 0$$

- The value of these coefficients are:

$$\frac{4\pi - 16}{\pi^3} \approx -0.11073981636184074$$

$$\frac{12 - 4\pi}{\pi^2} \approx -0.057385341027109429$$



A fast sine function

- We can thus implement:

$$p(x) = \frac{4\pi - 16}{\pi^3} x^3 + \frac{12 - 4\pi}{\pi^2} x^2 + x$$

as

```
double fast_sin( double x );

double fast_sin( double x ) {
    return -0.11073981636184074*x*x*x
           - 0.057385341027109429*x*x + x;
}
```


A faster sine function

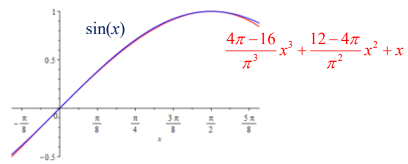
- We can even reduce the number of multiplications by one:

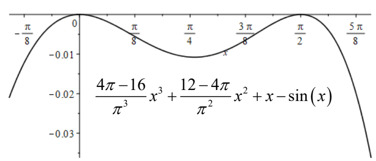
```
double fast_sin( double x );

double fast_sin( double x ) {
    return ((-0.11073981636184074*x
             - 0.057385341027109429)*x + 1.0)*x;
}
```

as $ax^3 + bx^2 + cx = ((ax + b)x + c)x$

A fast sine function

- Comparing $\sin(x)$ and our approximation:





The absolute error is less than 0.011 on $[0, \pi/2]$

Comments

- Functions must be commented to ensure other programmers know what is expected
  - Comments must be in English, not pseudocode or obvious

```
// Function declarations
double fast_sin( double x );

// Function definitions

/* double fast_sin( double x )
 *
 * A function that quickly calculates sin(x) for values
 * that satisfy 0 <= x <= pi/2
 */
double fast_sin( double x ) {
    return ((-0.11073981636184074*x
              - 0.0573853341027109429)*x + 1.0)*x;
}
```

Line comments:
– from // to the end of the line

Block comments:
– from /* to */

Other functions

- Most functions require more than simple calculations
  - Most require decision making processes:

$$|x| = \begin{cases} x & x \ge 0 \\ -x & x < 0 \end{cases} \qquad \max(x,y) = \begin{cases} x & x \ge y \\ y & x < y \end{cases} \qquad \min(x,y) = \begin{cases} x & x \le y \\ y & x > y \end{cases}$$

  - Others require a repetitive algorithm until some condition is met
    - E.g., finding the gcd, calculating the square root
  - In some cases, some functions can be defined in terms of others
    - E.g., the least common multiple:

$$\operatorname{lcm}(m,n) = \frac{mn}{\gcd(m,n)}$$

Side effects

- In mathematics, the result of a function depends entirely on the arguments
  - Anything else a function does is called a *side-effect*
  - The side-effect of the `int main()` function is to print "Hello world!" to the console output
- A side effect of this function is to record to a log file what was being calculated

```
double fast_sin( double x );

double fast_sin( double x ) {
    std::clog << "Calculating p(" << x << ")" << std::endl;
    return ((-0.11073981636184074*x
              - 0.0573853341027109429)*x + 1.0)*x;
}
```

No return value

- Some functions have no return value:
  - Such functions are identified by describing the return type as `void`:

No return value

```
void print_my_name();

void print_my_name() {
    std::cout << "Zaphod Beeblebrox" << std::endl;

    return;
}
```

Just return, don't return any value
- You can even leave this off

The keyword `void`

- The identifier `void` is a keyword in C++
  - Problem: every identifier that is used as a keyword restricts the identifiers that may be used by the programmers
  - Problem: too many keywords frustrate programmers
  - Solution: use the same keyword to mean different things in different contexts…
- In English, homographic homonyms are a source of puns:
  - "He picked up his **saw**."
  - "That was something she **saw**."
  - "Did you hear about the miracle of the carpenter who was cured of blindness? He picked up his hammer and **saw**."
- We will see `void` used in two different contexts—get used to it now

Why not `void main()` ?

- The function declaration for `main()` has it returning an `int`
  - Executing programs can cause other programs to execute
    - When a program exits, the value returned by `main()` could be used by the program that launched it
  - The value `0` is generally used to indicate "a successful execution"
  - If something went wrong, the program could return a non-zero integer that can be used to flag what the issue was
- For this course, `main()` will always return `0`

Arithmetic expression

- We have already described how an arithmetic expression can be any sum, difference, product or ratio of either integers or floating-point numbers
  - We can now also allow the operands to be functions that return either integers or floating-point numbers
- For example, this function returns a valid arithmetic expression:

```
double f( double x, double y ) {
    return -(3.0 + y)*(1.0 + 2.0*(std::sin( x ) - y));
}
```

Summary

- Following this lesson, you now:
  - Understand that functions in C++ are called like functions in math
  - Understand the purpose of the function declaration:
    - The type of any parameters and the type of the return value
  - Know the difference between parameters and arguments
  - Know how to implement a simple function
  - Can describe the difference between the return value and side effects
  - Know how to indicate a function has no return value: `void`
  - Understand the `int main()` function

References

[1] cplusplus.com
http://www.cplusplus.com/reference/cmath/

[2] Wikipedia: block comments
https://en.wikipedia.org/wiki/Comment_(computer_programming)#Block_comment

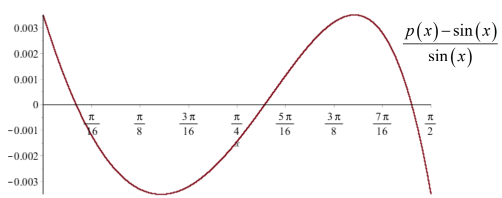
[3] Wikipedia: line comments
https://en.wikipedia.org/wiki/Comment_(computer_programming)#Line_comments

Additional note

- If your goal is different, you can get different functions:

$$p(x) \stackrel{\text{def}}{=} -0.12982726700166469x^3 - 0.031041616418863258x^2 + 1.0034924244212799x$$

- This minimizes the *relative error* on the interval $[0, \pi/2]$ to less than 0.35%
  - It will never be more than 0.35% different than the actual value



Additional note

- The implementation should reduce the number of necessary operations

```
double sin_min_rel_err( double x );

double sin_min_rel_err( double x ) {
    return ((
       -0.12982726700166469*x - 0.031041616418863258
    )*x + 1.0034924244212799)*x;
}
```

$$-0.12982726700166469x^3 - 0.031041616418863258x^2 + 1.0034924244212799x$$

Incidentally, the absolute error is less than 0.0033 on $[0, \pi/2]$, so it's also better than our first approximate with respect to absolute error…

Acknowledgments

Proof read by Dr. Thomas McConkey.

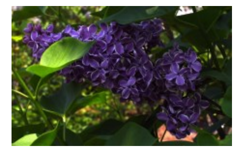
Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.



Disclaimer