LECTURE 3 - FUNCTIONS

Chapter 6 in text

Tonight's Topics

- Lab #2 Review
- Functions
  - Return Values
  - Parameter passing
  - Lab #2 Revised - *Top Down Design*
  - Pass by Reference
  - Function Prototypes

Lab #2

- The constant “e” is approximately 2.718 and has been calculated to 869,894,101 decimal places.
- e^x is approximated by the following series.

  ex = x0/0! + x1/1! + x2/2! + x3/3! + x4/4! + … + xn/n!
- Write a program that asks the user for a **positive** value for X. Display ex based on the above approximation where N is 1, 5, 25, and 125.

- e^5 (1) = 6.0000000000
- e^5 (5) = 91.4166666667
- e^5 (25) = 148.4131590981
- e^5 (125) = 148.4131591026

I used `setprecision(10)` and `fixed`

Lets start with a “bottom-up” approach

Using Functions

main.cpp

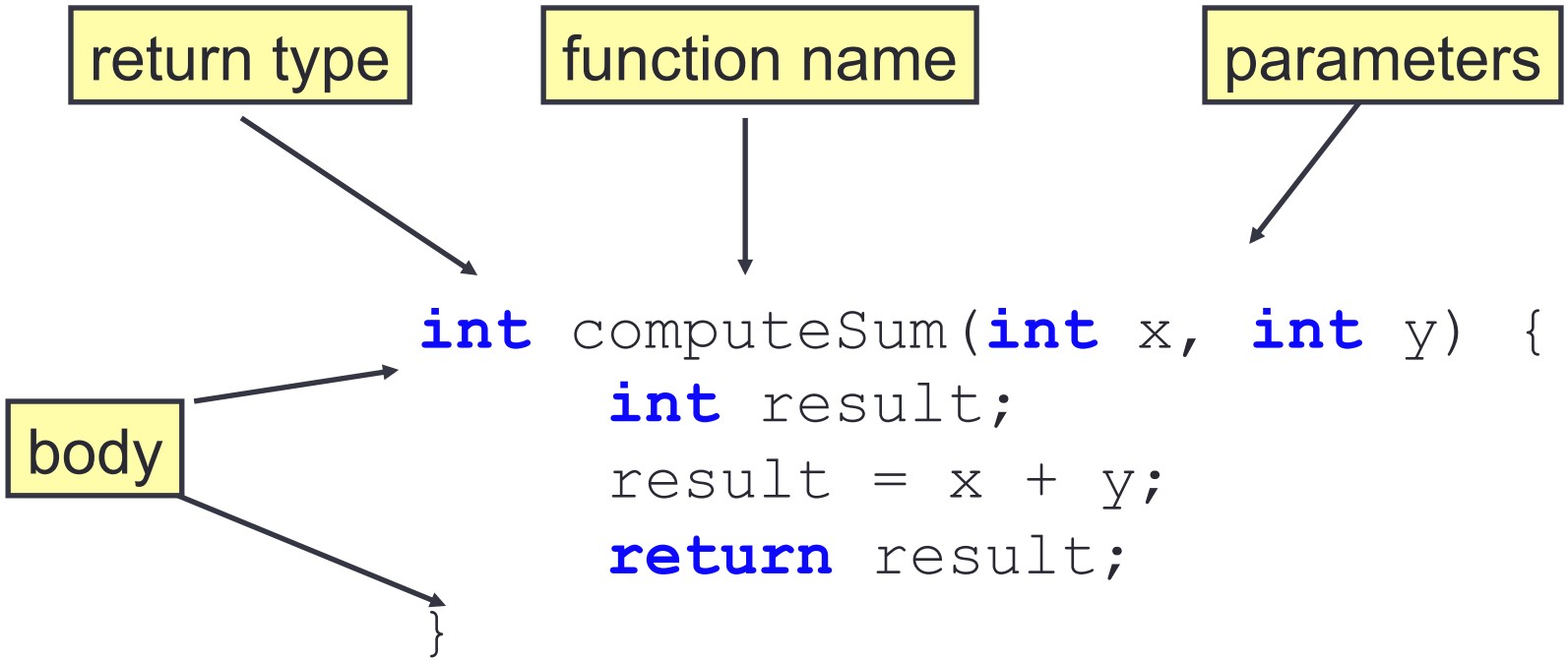
```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
  double N;
  cout << "Enter a number:
  cin   >> N;
  double S = sqrt(N);
  cout << "Square Root:    " <
  return 0;
}
```

cmath

```
double sqrt ( double x ) {
  … lots of code to calculate
    square root…
  return square_root;
}
```

Function Syntax



note the similarities with main…

Variables

Functions can have their own variables (called *local* variables)

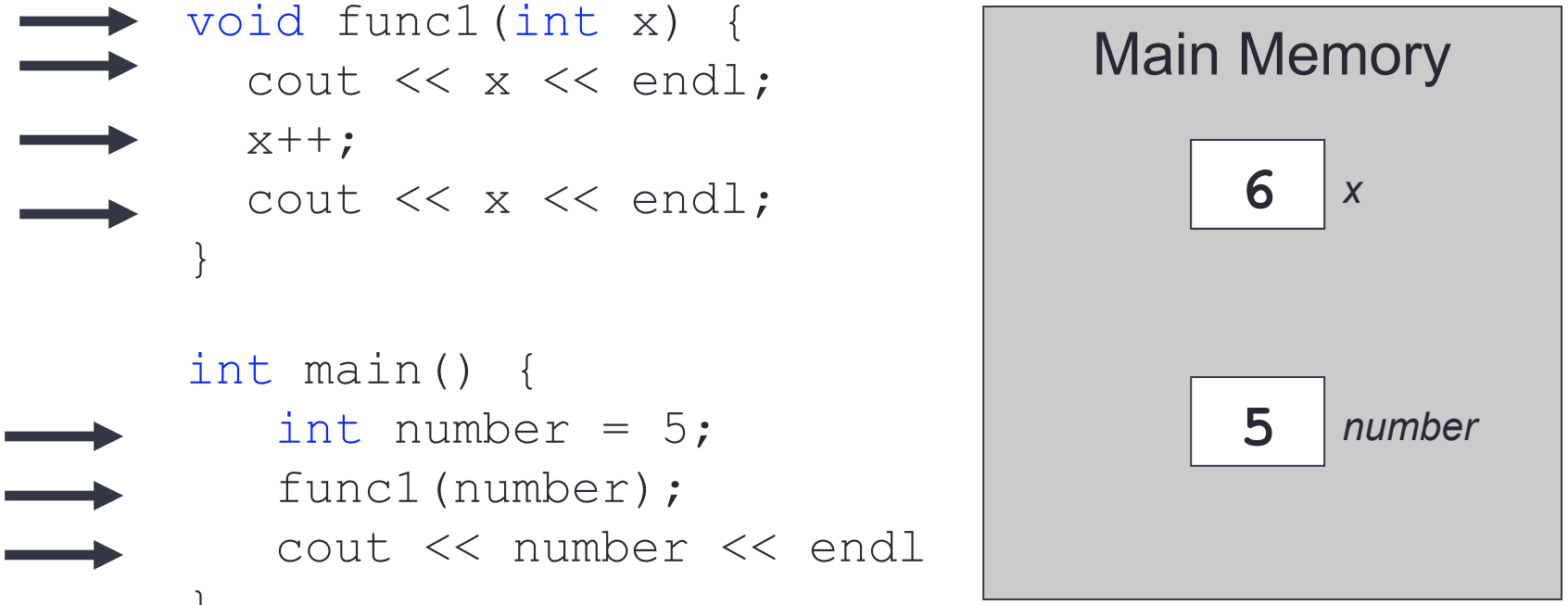
These variables are not visible to other functions
(for instance, main)

Each time the function is called, local variables are created from scratch

Parameter Passing

```
void func1(int x) {
  cout << x << endl;
  x++;
  cout << x << endl;
}

int main() {
   int number = 5;
   func1(number);
   cout << number << endl
}
```



Returning Data

Functions can return useful data:

```
double pow(double base, double exponent)
double sqrt(double value)
int abs(int value)
double product(double op1, double op2)
```

Using our product function, allow user to multiply as many numbers together as the wish until they enter 0. Printout the intermediate product after each entry.

void

void is ***not*** a data type - it denotes the *absence* of data

```
void sayHello () {
    cout << "Hello from function" << endl;
    return;  // optional
}
```

Data & Functions

Local Variables: Variable used *within a single function*. This data is only "visible" within that function!

Parameter Variables: Communication **from** *calling function* **to** the function.

Return Values: Data passed **from** the called function **back to** its caller

Top Down Design

$$e^x = x^0/0! + x^1/1! + x^2/2! + x^3/3! + x^4/4! + \ldots + x^n/n!$$

Top Down Design: Take a larger problem, and break it down into successively smaller units

- Ask the user for a positive number (input validation)
- Compute e^x
  - For I = 0…N
    - Calculate Term
      - Calculate x^i
      - Calculate i!
      - Divide the two
    - Add the term
- Print the result

Variable Scope

Every variable has a **scope**.

A *local* variable is defined within a segment of code, such as a function, loop, or `if` block.

- Local Variables are only visible within the segment they are defined (and sub-segments)

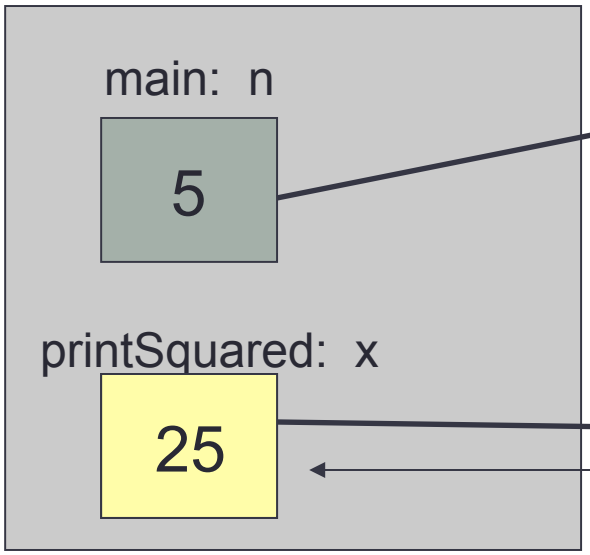
A *global variable* is defined outside *all* functions - and is visible to *all* functions.

You should never use global variables.

Exception: Constants can be defined globally

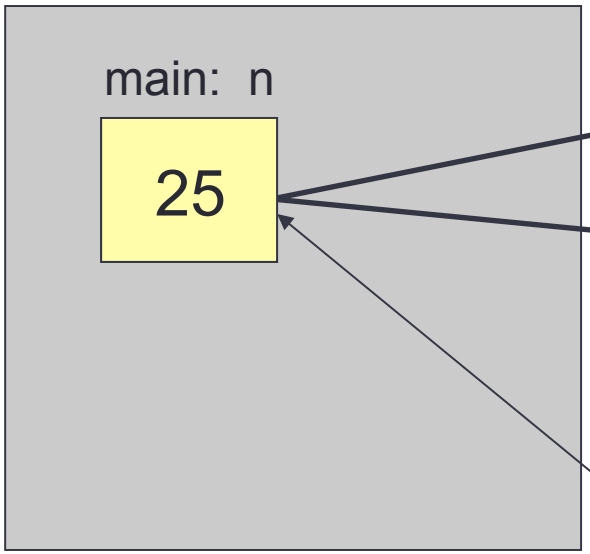
```
const double PI = 3.14159;
```

Pass by Value




```
int main() {
        int n = 5;
        printSquared(n);
        cout << n << endl;
}
void printSquared(int x) {
        x *= x;
        cout << x << endl;
}
```

Pass-by-Reference




```
int main() {
        int n = 5;
        printSquared(n);
        cout << n << endl;
}
void printSquared(int & x)
{
        x *= x;
        cout << x << endl;
}
```

Example Problem

Ask use for number between 1 and 99

calculate the most efficient coin usage:

86 cents = 3 quarters, 1 dime, 1 penny

```
int computCoin (int coinValue, int & amountLeft)
```

returns number of coins

i.e. 25, 10, 5, etc

amount of change left will be updated

Function Prototypes

Some of our programs will have many functions

- Function may call other functions…
- Functions cannot be used before they are defined.
- Can “declare” functions before actually “defining how they work” (code).

Function Prototypes

```
double myfunction1(int parameter);
double myfunction2(int parameter);
```

function **prototypes**

```
int main() {
    cout << myfunction1(5) << endl;
    cout << myfunction1(5) << endl;
}
```

```
double myfunction1(int parameter) {
    return parameter + 1;
}
double myfunction2(int parameter) {
    return parameter - 1;
}
```

function **definitions**

Example: Stats

Write a program to read a series of numbers from the user. Computer the **mean** and **standard deviation.**

- Mean = $\dfrac{\sum_{i=1}^{N} x_i}{N}$

- Standard Deviation:

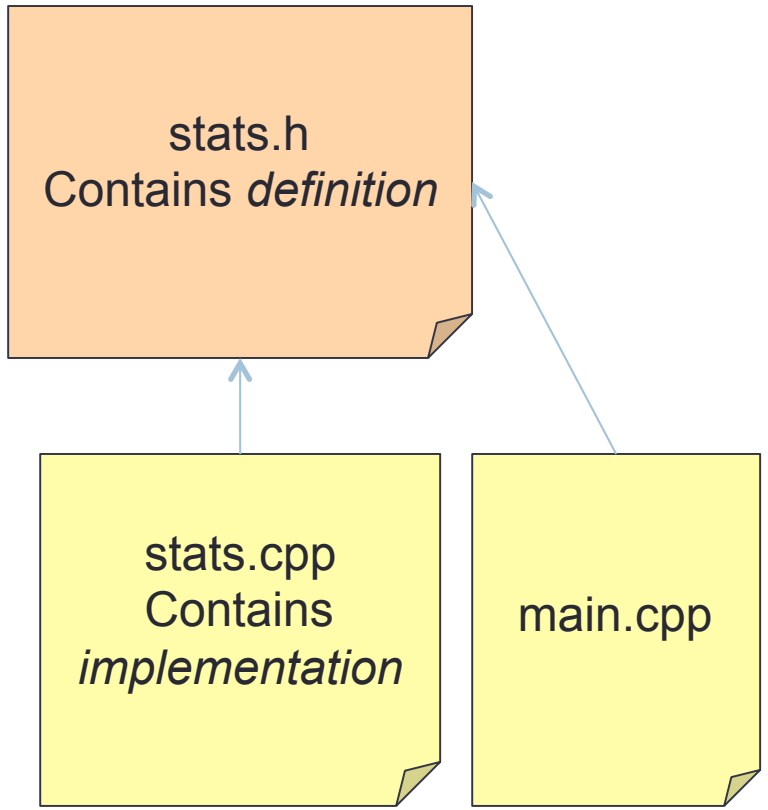
$$\sqrt{\frac{\sum_{i=1}^{N} x_i^2 - \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N}}{N-1}}$$

Header Files

Many programs contain many functions

Its often cleaner to have function definitions stored outside the main cpp file

This also allows easier code reuse and sharing



Definition / Implementation

stats.h

```
double stdev(double sumOfSquares, double mean, int n);
```

stats.cpp

```
#include “stats.h”

double stdev(double sumOfSquares, double mean, int n) {
    double t = mean * mean / n;
    double num = sumOfSquares – t;
    double r = num / ( n-1 );
    return sqrt(r);
}
```

Lab #3

Don't use arrays!

Write a program to generate the following table using two functions

```
double celsiusToFahrenheit(double c)
double fahrenheitToCelsius(double f)
```

Implement these two functions in a header/implementation file set (temp.h, temp.cpp)

Celsius	Fahrenheit	Fahrenheit	Celsius
40.0	104.0	120.0	48.89
39.0	102.2	110.0	43.33
38.0	100.4	100.0	37.78
37.0	98.6	90.0	32.22
36.0	96.8	80.0	26.67
35.0	95.0	70.0	21.11
34.0	93.2	60.0	15.56
33.0	91.4	50.0	10.00
32.0	89.6	40.0	4.44
31.0	87.8	30.0	-1.11