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
The constant “e” is approximately 2.718 and has been calculated to 869,894,101 decimal places.

\[ e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots + \frac{x^n}{n!} \]

Write a program that asks the user for a positive value for X. Display \( e^x \) based on the above approximation where N is 1, 5, 25, and 125.

\begin{itemize}
  \item \( e^5 \) (1) = 6.0000000000
  \item \( e^5 \) (5) = 91.4166666667
  \item \( e^5 \) (25) = 148.4131590981
  \item \( e^5 \) (125) = 148.4131591026
\end{itemize}

I used \texttt{setprecision(10)} and fixed

\textbf{Lab #2}

Lets start with a “bottom-up” approach
Using Functions

main.cpp

```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:  " << S << end;
    return 0;
}
```

cmath

```cpp
double sqrt ( double x ) {
    ... lots of code to calculate square root...
    return square_root;
}
```
Function Syntax

```
int computeSum(int x, int y) {
    int result;
    result = x + y;
    return result;
}
```

*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

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

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

Main Memory

- `number`: 5
- `x`: 6
Returning Data

Functions can return useful data:

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

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

\[ e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots + \frac{x^n}{n!} \]

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 \( I = 0 \ldots N \)
    - 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;
  ```
int main() {
    int n = 5;
    printSquared(n);
    cout << n << endl;
}

void printSquared(int x) {
    x *= x;
    cout << x << endl;
}

**Pass by Value**

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

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

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

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

double myfunction2(int parameter) { 
    return parameter - 1; 
}
Write a program to read a series of numbers from the user. Compute the mean and standard deviation.

• Mean = \[ \frac{\sum_{i=1}^{N} x_i}{N} \]

• Standard Deviation:

\[ \sqrt{\frac{\sum_{i=1}^{N} x_i^2 - \left(\frac{\sum_{i=1}^{N} x_i}{N}\right)^2}{N-1}} \]
Many programs contain many functions

It's often cleaner to have function definitions stored outside the main .cpp file

This also allows easier code reuse and sharing
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

Write a program to generate the following table using two functions

\[
\begin{align*}
\text{double} & \quad \text{celsiusToFahrenheit(\text{double} c)} \\
\text{double} & \quad \text{fahrenheitToCelsius(\text{double} f)}
\end{align*}
\]

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

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