Lecture 2

- Functions
  - Declaration (prototype)
  - Definition (implementation)
  - Function calls
  - Parameters
    - Call by reference
    - Call by value
  - Return value
- Function overloading
- Header files
- Standard library: `cmath`, `cstdlib`, `iomanip`
- Variables of reference type

Functions in C++

- Structured programming
- Programs constructed from functions
- Performing **one task**
  - E.g. Compute the n!, sort a sequence of names
- Divide and conquer
  - Construct a program from smaller pieces or components
  - Each piece more manageable than the original program
Program Components in C++

- C++ programs composed of
  - Programmer-defined functions
  - Prepackaged: from the C++ Standard Library

See Fig03_03.cpp

Functions

```c
int a = 6, b = 6;
cout << fun(a,b);
```

Function definition

```c
int fun(int x, int y)
{
    return x / (y%6 + 1);
}
```
// fig03_04.cpp
// Finding the maximum of three floating-point numbers.
#include <iostream>
using namespace std;

double maximum( double, double, double ); // function prototype

int main()
{
    double number1;
    double number2;
    double number3;

    cout << "Enter three floating-point numbers: ";
cin >> number1 >> number2 >> number3;

    // number1, number2 and number3 are arguments to
    // the maximum function
    cout << "Maximum is: 
    << maximum( number1, number2, number3 ) << endl;
    return 0; // indicates successful termination
}

// function maximum definition;
// x, y and z are parameters
double maximum( double x, double y, double z )
{
    double max = x;  // assume x is largest
    // max is a local variable

    if ( y > max ) // if y is larger,
        max = y;   // assign y to max

    if ( z > max ) // if z is larger,
        max = z;   // assign z to max

    return max;    // max is largest value
} // end function maximum
Functions

- Local variables
  - Variables declared in the function
  - Known only in the function in which they are defined

- Parameters (arguments)
  - Local variables
    - Initialized with the arguments of the function call
  - Provide outside information

Important Concepts about Functions

- Function prototype
  - Function header
  - Function declaration

- Function call

- Function definition
  - Function implementation
Function Prototype

- Function prototype
  - Tells compiler argument type and return type of function
    
    ```
    int square(int);
    ```
  - Function takes an `int` and returns an `int`
  - Function prototypes should appear before the function is called

Function Prototypes

- Function prototype contains
  - Function name
  - Parameters (number and data type)
  - Return type (void if returns nothing)
  - Only needed if function definition after function call

- Prototype must match function definition
  - Function prototype
    ```
    double maximum(double, double, double);
    ```
  - Definition
    ```
    double maximum(double x, double y, double z)
    {
    ...
    }
    ```

Function Definitions

- Format for function definition

```
return-value-type function-name( parameter-list )
{
  declarations and statements
}
```

- Parameter list
  - Comma separated list of arguments
    - Data type needed for each argument

- Return-value type
  - Data type of result returned (use `void` if nothing returned)

```cpp
int square1( int y )
{
  return y * y;
}
```

- `return` keyword
  - Returns data, and control goes to function’s caller
    - If no data to return, use `return`;
  - Function ends when reaches right brace or return
    - Control goes to caller

- Functions cannot be defined inside other functions

```cpp
void square2( int y )
{
  cout << y * y ;
}
```
Function Arguments

• Argument Coercion
  – Force arguments to be of proper type
    • Converting int (4) to double (4.0)
      \[
      \text{cout} \ll \text{sqrt}(4);
      \]
  – Conversion rules
    • Arguments usually converted automatically
    • Changing from double to int can truncate data
      E.g. int k = square1(3.4);  //3.4 to 3

Function Arguments

• Can be
  – Constants
    \[
    \text{square1}( 4 );
    \]
  – Variables
    \[
    \text{square1}( x );
    \]
  – Expressions
    \[
    \text{square1}( \text{factorial}( x ) );
    \text{square1}( 3 - 6*x );
    \]
Function Overloading

```cpp
int max( int x, int y )
{
    return x > y ? x : y;
}
```

```cpp
double max(double x, double y);
char max(char x, char y);
```

See overload.cpp

Math Library Functions

- Perform common mathematical calculations
  - Include the header file
    ```cpp
    #include <cmath>
    ```
  - Trigonometric functions (e.g. `cos`, `sin`, `tan`)
  - Exponential and logarithmic functions (e.g. `exp`, `log`)
  - Power functions (e.g. `pow`, `sqrt`)
  - Rounding, absolute value and remainder functions (e.g. `ceil`, `floor`)
  - All functions in math library return a `double`

- Example
  ```cpp
cout << sqrt( 900 );
```

- See
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceil( ( x ) )</td>
<td>rounds ( x ) to the smallest integer not less than ( x )</td>
<td>ceil( 3.2 ) is 4.0, ceil( -9.8 ) is -9.0</td>
</tr>
<tr>
<td>cos( ( x ) )</td>
<td>trigonometric cosine of ( x ) (( x ) in radians)</td>
<td>cos( 0.0 ) is 1.0</td>
</tr>
<tr>
<td>exp( ( x ) )</td>
<td>exponential function ( e^x )</td>
<td>exp( 1.0 ) is 2.71828, exp( 2.0 ) is 7.38906</td>
</tr>
<tr>
<td>fabs( ( x ) )</td>
<td>absolute value of ( x )</td>
<td>fabs( 5.1 ) is 5.1, fabs( 0.0 ) is 0.0, fabs( -8.76 ) is 8.76</td>
</tr>
<tr>
<td>floor( ( x ) )</td>
<td>rounds ( x ) to the largest integer not greater than ( x )</td>
<td>floor( 9.2 ) is 9.0, floor( -9.8 ) is -10.0</td>
</tr>
<tr>
<td>fmod( ( x, y ) )</td>
<td>remainder of ( x/y ) as a floating-point number</td>
<td>fmod( 13.657, 2.333 ) is 1.992</td>
</tr>
<tr>
<td>log( ( x ) )</td>
<td>natural logarithm of ( x ) (base ( e ))</td>
<td>log( 2.71828 ) is 1.0, log( 7.38906 ) is 2.0</td>
</tr>
<tr>
<td>log10( ( x ) )</td>
<td>logarithm of ( x ) (base 10)</td>
<td>log10( 10.0 ) is 1.0, log10( 100.0 ) is 2.0</td>
</tr>
<tr>
<td>pow( ( x, y ) )</td>
<td>( x ) raised to power ( y ) (( x^y ))</td>
<td>pow( 2, 7 ) is 128, pow( 2, 3 ) is 8</td>
</tr>
<tr>
<td>sin( ( x ) )</td>
<td>trigonometric sine of ( x ) (( x ) in radians)</td>
<td>sin( 0.0 ) is 0</td>
</tr>
<tr>
<td>sqrt( ( x ) )</td>
<td>square root of ( x )</td>
<td>sqrt( 900.0 ) is 30.0, sqrt( 9.0 ) is 3.0</td>
</tr>
<tr>
<td>tan( ( x ) )</td>
<td>trigonometric tangent of ( x ) (( x ) in radians)</td>
<td>tan( 0.0 ) is 0</td>
</tr>
</tbody>
</table>

**Header Files**

- Header files contain
  - Function prototypes
  - Definitions of data types and constants

- Header files end with `.h`
  - Programmer-defined header files
    ```
    #include "myheader.h"
    ```

- Library header files
  ```
  #include <cmath>
  ```
Random Numbers

• **rand** function (**include <cstdlib>**)
  - `i = rand();`
  - Generates unsigned integer between 0 and `RAND_MAX` (at least 32767)

• Scaling and shifting
  - Modulus (remainder) operator: `%`
    - `x % y` is between 0 and `y - 1`
  - Example
    - `i = rand() % 6 + 1;
    - "rand() % 6" generates a number between 0 and 5 (scaling)
    - "+ 1" makes the range 1 to 6 (shift)
  - See program to roll dice `dice.cpp`

References

• A **reference** is an alias for a variable

• Mostly used for function arguments
  ```cpp
  int x = 1;
  int& xref = x;  // xref alias for x
  xref = 4;  // x changed via xref
  ```

• References must be initialized at declaration (compile error)

• References cannot be reassigned as aliases to other variable
Example

```cpp
int squareByValue(int a)
{
    return a *= a; //a = a * a;
}

int squareByRef(int& b )
{
    return b *= b; //b = b * b;
}

int x = 4;
cout << squareByValue( x ) << x << endl;

int z = 4;
cout << squareByRef( z ) << z << endl;
```

- What do you think is the output?

Call by Value versus Call by Reference

- **Call by value** (used by default)
  - Copy of data passed to function
  - Changes to copy do not change original
  - Prevent unwanted side effects

- **Call by reference**
  - Function can directly access data
  - Changes affect original
  - See Fig03_20.cpp
Call by Value

• Why should I need call by value?
  – It is safer
    • Arguments protected from side effects
    • It works with a copy of the arguments passed in the function call
  – But, it implies overhead
    • Memory space and time needed to make the copies
    • What if an argument is many bytes long?

Call by Reference

• Why should I need call by reference?
  – Efficiency
  – But, there’s more
    • Next example should help to answer this question
Call by Value

//swap the value of two variables
void swap1(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}

Call by Reference

//swap the value of two variables
void swap2(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
Calling `swap1` and `swap2`

- **Call by value** `swap1(a, b);`
  - x and y in swap gets copies of a and b
  - Local copies of the arguments are changed

- **Call by reference** `swap2(a, b);`
  - Reference (address) to a and b is copied to x and y
  - Arguments are changed

Further reading …

- About *recursive* functions