1 Review

- Data types
- Variable names
- Loops

2 Functions

- Useful analysis/computational units.
- Object and nonobject functions:
  - `cout.setf()`
  - `ShowHeader()`
  - `sin()`
- Functions improve modularity of programs.
- Functions normally have two parts:
  - Prototype (sometimes called interface)
  - Definition (sometimes called implementation)

2.1 General Function Layout

return_type **name** (*<args>*)
`{`
    function body

`}`

return_type
`void`, basic data types (int, double, etc.)
user/system defined types

name
Any valid name.

<args>
The names and types of all arguments (zero or more), separated by commas

function body
Zero or more valid statements.

2.2 Function Operation

When a function is invoked (called), the flow of control is transferred to the function. This means that the next statement to be executed is the first one in the function that was called. After the function completes its task the flow of control returns to the statement in the function that invoked the function.

2.3 Function Prototypes

- The prototype specifies the data type, name, and input (and/or output) parameter(s) of a function.
  - The type of a function is the data type of a value returned by the function.
  - The name of a function is any valid identifier. Use action words (verbs) whenever possible in naming functions.

  - Parameters define input (and/or output) to a function.

- Header files (e.g., `iostream.h`, `math.h`, `ctype.h`) usually contain prototypes.
- Examples:
  - `double sin( double );`
  - `double Avg( double x1, double x2, double x3 );`

Note: that the function parameters are separated by commas.

2.4 Function Definition

- A function definition is a sequence of statements.
- Function names should imply the operation(s) performed by the function.
- All variables must be declared.
- If the function does not return a value, the function return type must be declared as `void`.
- Any expression is allowed in the `return` statement.
  - `return 1;`
  - `return 'A';`
  - `return; // optional`
- Good programming practice to have only one return.

2.5 Using Functions

- A function's argument data types must match the parameter specified in the prototype (definition).
- If a function returns a value, the variable (object) that receives the return value must match the function data type.
- Mismatch of arguments and/or return values is the source of many compiler errors/warnings.
- Mismatch of arguments and/or return values is the source of many run-time defects/errors.

2.6 Square and Cube of 1 through 5

```
//  Powers.cpp

#include <iostream.h>
#include <iomanip.h>
#include <math.h>

int main()
{
    for( int i = 1 ; i <= 5 ; i++ )
    {
        cout << setw(6) << i ;
        cout << setw(6) << i*i;
        cout << setw(6) << i*i*i << endl;
    }

    return  0;
}
```

Output:

```
  1     1     1
  2     4     8
  3     9    27
  4    16    64
  5    25   125
```



```
//  Powers2.cpp

#include <iostream.h>
#include <iomanip.h>
#include <math.h>

    //  Prototype
void ShowPowers();


int main()
{
    ShowPowers();

    return  0;
}

/*  ShowPowers()
 *
 *  Show the square and cube of 1 through 5.
 */
void ShowPowers()
{
    for( int i = 1 ; i <= 5 ; i++ )
    {
        cout << setw(6) << i ;
        cout << setw(6) << i*i;
        cout << setw(6) << i*i*i << endl;
    }
}
```

```
//  Powers3.cpp

#include <iostream.h>
#include <iomanip.h>
#include <math.h>

    //  Prototypes
void ShowPowers();
int Square( int );
int Cube( int i );

int main()
{
    ShowPowers();

    return  0;
}

/*  ShowPowers()
 *
 *  Show the square and cube of 1 through 5.
 */
void ShowPowers()
{
    for( int i = 1 ; i <= 5 ; i++ )
    {
        cout << setw(6) << i ;
        cout << setw(6) << Square( i );
        cout << setw(6) << Cube( i ) << endl;
    }
}


/*  Square
 *
 *  Calculate the square of an integer.
 */
```



```
int Square( int i )
{
    return  i * i;
}


/*  Cube
 *
 *  Calculate the cube of an integer.
 */
int Cube( int i )
{
    return  i * i * i;
}
```

```
//  Powers4.cpp

#include <iostream.h>
#include <iomanip.h>
#include <math.h>

    //  Prototypes
void ShowPowers();
double Power( double x, int pow );


int main()
{
    ShowPowers();

    return  0;
}

/*  ShowPowers()
 *
 *  Show the square and cube of 1 through 5.
 */
void ShowPowers()
{
    for( double x = 1.0 ; x <= 5.0 ; x += 1.0 )
    {
        cout << setw(6) << x ;
        cout << setw(6) << Power(x,2);  // spaces
        cout << setw(6) << Power( x, 3 ) << endl;
    }
}


/*  Power
 *
 *  Calculate the square of an integer.
 *  Should check that iPow > 0!
 */
```



```
double Power( double x, int iPow )
{
    double  retVal = 1.0;

    for( int i = 0 ; i < iPow ; i++ )
        retVal *= x;

    return  retVal;
}
```

2.7 Averages

```
//  Average.cpp

#include <iostream.h>
#include <fstream.h>

    //  prototype
double Average( double x1, double x2, double x3 );

main()
{
    ofstream fOut( "Average.out", ios::out );

    if( !fOut )              //  verify file was opened
    {
         cerr << "Unable to open output file:  Average.out" << endl;
         exit( -1 );
    }

    double avg;

    double x1 = 2.0;
    double x2 = 3.0;
    double x3 = 4.0;

    avg = Average( x1, x2, x3 );
    fOut << "Average 1: " << avg << endl;

    avg = Average( 5.0, 10.0, 15.0 );
    fOut << "Average 2: " << avg << endl;

    avg = Average( x1, x3-6.0, x2+5);
    fOut << "Average 3: " << avg << endl;

    fOut.close();
}
```



```
double Average( double x1, double x2, double x3 )
{
     double rVal;    //  return value

     rVal = (x1 + x2 + x3) / 3.0;

     return rVal;
}
```

2.8 Minimum function

Calculate the minimum of two integers. There is a built-in function, `min()`, for performing this operation.

```
int  Minimum( int i, int j )
{
     if( i < j )
         return i;   //  else not necessary
     return j;
}
```

2.9 IsDigit function

Test if a character is a digit or not. There is a built-in function, `isdigit()`, for performing this operation.

```
int  IsDigit( char ch )
{
     int iVal = 0;

     if( ch >= '0' && ch <= '9' ) iVal = 1;

     return iVal;
}
```

2.10 Other function-related topics

- Parameter passing mechanisms
- Recursion
- Overloading