Lab 9
More on Functions
Dr. Abraham

In this chapter we will review functions, follow an example program from planning to completion, and learn two new concepts, namely inline functions and function overloading. A function is a subprogram that specializes in doing one specific task. A function by definition may receive none, one, or many input parameters and output none or one parameter.
None or One output argument ← | **Function** | ←None, one or many input argument(s)

```
float   findAverage (int grade1, int grade2, int grade3)
```

In this example the function **findAverage** receives 3 input arguments and returns one argument. The input arguments here are **passed by value**, meaning only copies of the three original variables appearing in the calling function are passed to the function. Even if the values of these variables are changed by the function, the original variables will remain unchanged. If a function does not return any result back just indicate so by stating **void**, example: void print (int grade1, int grade2, int grade3, float average).

Remember that the function only exists during its execution. Upon return from the function nothing remains of the function. A function is made upon calling it and destroyed upon returning. In order to call the above function, we can write the following statement:

```
average = findAverage(grade1, grade2, grade3);
```

Since we know that this function returns a float we must put this returned value somewhere; here we receive the result into **average**. We could have displayed the result directly on the screen by the statement:

```
cout << findAverage(grade1, grade2, grade3);
```

It is often necessary to use functions to read multiple variables and return them to the calling module. Based on the above definition it is impossible to send back more than one value. Here is where argument **passing by reference** comes in handy. Instead of returning the result, the function writes directly to the original memory locations of the reference variables. In the following is example, even though nothing is returned by the function, the calling function's three variables were changed with the read values.

```
void getGrades (int &grade1, int &grade2, int &grade3)
```

Suppose we want to write a program to find the average of three grades. We can generate a structure chart with three modules, getGrades, findAverage, and print. The Module getGrades receives three arguments by reference and returns none. The findAverage module receives one argument by value and returns a float. The last module receives four arguments by value and returns none. Draw a structure chart using these directions. The next step is to develop the prototypes. Translating a properly designed structure chart to prototypes is rather easy. The prototype for these three functions would be:

```
void getGrades (int&, int&, int&);
float findAverage (int, int, int);
void print (int, int, int, float);
```

Finally, let us write the whole program. The main should only declare variables that it uses. Do not declare variable a function might use as local variables. Convert the prototypes to functions and function calls.

Program 9-1

```
/************************************
Program to find average of 3 grades
objective: Review functions
Program by Dr. Abraham
*************************************/



#include <iostream>
#include <fstream>
using namespace std;

void getGrades (int&, int&,int&);   //function prototypes
float findAverage (int, int, int );
void print (int, int, int, float);
ofstream outfile;                          //outfile is declared globally
int main ()
{
        outfile.open ("a:grade.dat");
        float average;
        int grade1, grade2, grade3;
        getGrades(grade1, grade2, grade3);
        average = findAverage(grade1, grade2, grade3);

        print(grade1, grade2, grade3, average);
        outfile.close();
        return(0);
}

void getGrades (int &c, int &b, int &a) // variable names changed purposely
{
        cout << "Enter first grade   ----------> ";
        cin >> c;
        cout << "Enter second grade -----------> ";
        cin >> b;
        cout << "Enter third grade  ---------- > ";
        cin >> a;

        outfile << "\nEnter first grade   ----------> " << c;
        outfile << "\nEnter second grade -----------> " << b;
        outfile << "\nEnter third grade  ---------- > " << a;

}

float findAverage (int one, int two, int three) //variable names changed purposely
{
  int total;
  float a;
  total = one+two+three;
  a = (total)/3.0;
  return a;
}



void print (int grade1, int grade2, int grade3, float average)

{
  cout << "\nThree grades entered are: " << grade1 << " " <<grade2 << " " << grade3;
  cout << "\nAverage is " << average <<endl;
  outfile << "\nThree grades entered are: " << grade1 << " " <<grade2 << " " << grade3;
  outfile << "\nAverage is " << average <<endl;
}
```

Program Run 9-1

```
Enter first grade   ----------> 92
Enter second grade -----------> 88
Enter third grade  ---------- > 86

Three grades entered are: 92 88 86
Average is 88.6667
Press any key to continue
```

Inline Functions:

We can write a function that is so small that it will fit in one line. For example the findAverage function can be written like this:

```
inline float findAverage (int a, int b, int c) {return (a+b+c)/3.0;};
```

Program 9-2

```
/************************************
Program to find average of 3 grades
objective: Review functions
Program by Dr. Abraham
************************************/



#include <iostream>
#include <fstream>
using namespace std;


void getGrades (int&, int&,int&);   //function prototypes
inline float findAverage (int a, int b, int c) {return (a+b+c)/3.0;};
void print (int, int, int, float);
ofstream outfile;                         //outfile is declared globally
int main ()
{
        outfile.open ("grade.dat");
        float average;
        int grade1, grade2, grade3;
        getGrades(grade1, grade2, grade3);
        average = findAverage(grade1, grade2, grade3);

        print(grade1, grade2, grade3, average);
        outfile.close();
        return(0);
}

void getGrades (int &c, int &b, int &a) // variable names changed purposely
{
        cout << "Enter first grade   ----------> ";
        cin >> c;
        cout << "Enter second grade -----------> ";
        cin >> b;
        cout << "Enter third grade  ---------- > ";
        cin >> a;

        outfile << "\nEnter first grade   ----------> " << c;
        outfile << "\nEnter second grade -----------> " << b;
        outfile << "\nEnter third grade  ---------- > " << a;

}



void print (int grade1, int grade2, int grade3, float average)

{
 cout << "\nThree grades entered are: " << grade1 << " " <<grade2 << " " << grade3;
 cout << "\nAverage is " << average <<endl;
 outfile << "\nThree grades entered are: " << grade1 << " " <<grade2 << " " << grade3;
 outfile << "\nAverage is " << average <<endl;
}
```

Function Overloading:

It is possible for a function to calculate average of real numbers yielding a real number or to calculate integer average yielding an integer. To do this we have to write two functions with different set of parameters but using the same function name, **average**. Such functions are called overload functions. When an overloaded function is called, C++ compiler chooses appropriate function based on parameters passed. Here is an example of an overloaded function called findAv.

Program 9-3

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

double findAv(int, int, int);
double findAv(double,double,double);
double findAv(double,double,double,double);

int main ()
{
double average;
  //call findAv with three integer parameters
  average = findAv(80,82,95);
  cout << "\n Called overloaded function with three integer parameters.
Average is:  " <<average;
 //call findAv with three double parameters
  average = findAv(82.5,89.4,95.88);
  cout << "\n Called overloaded function with three double parameters.
Average is:  " <<average;
//call findAv with four double parameters
  average = findAv(82.5,89.4,95.88, 99.89);
  cout << "\n Called overloaded function with four double parameters.
Average is:  " <<average;
```

```
	return 0;
}

double findAv(int a, int b, int c)
{
	return (a+b+c)/3.0;
}

double findAv(double a, double b, double c)
{
	return (a+b+c)/3.0;
}

double findAv(double a, double b, double c, double d)
{
	return (a+b+c+d)/4.0;
}
```

Program Run 9-3

```
Called overloaded function with three integer parameters.  Average is:  85.6667
Called overloaded function with three double parameters.  Average is:  89.26
Called overloaded function with four double parameters.  Average is:  91.9175
```

Assignment.

When a function calls itself it is known as a recursive call. While this assignment is an easy one, it requires you to read about recursion. Just as for looping, you need to provide for stopping the recursion, otherwise an endless calling will occur.

Write a recursive program to find the factorial of a number.