6 Overloading Functions

- C++ allows more than one definition for the same function name
  - Very convenient for situations in which the "same"function is needed for different numbers or types of arguments
- Overloading a function name means providing more than one declaration and definition using the same function name

```
double ave(double n1, double n2)
{
        return ((n1 + n2) / 2);
}

double ave(double n1, double n2, double n3)
{
        return (( n1 + n2 + n3) / 3);
}
```

Compiler checks the number and types of arguments in the function call to decide which function to use

```
cout << ave( 10, 20, 30);
```

uses the second definition

6.1 Overloading Details

- Overloaded functions
  - Must have different numbers of formal parameters
    AND / OR
  - Must have at least one different type of parameter
- Return types are not considered in overload resolution.

6.2 Overloading Example

- Revising the Pizza Buying program
  - Rectangular pizzas are now offered!
  - Change the input and add a function to compute the unit price of a rectangular pizza
  - The new function could be named **unitprice_rectangular**
  - Or, the new function could be a new (overloaded) version of the **unitprice** function that is already used
    - Example:

```
double unitprice(int length, int width, double price)
{
        double area = length * width;
        return (price / area);
}
```

sample08.cpp

```cpp
#include <iostream>
#include <cmath>

using namespace std;
double distance(double x1,double x2){
  return abs(x1 - x2);
}

double distance(double x1,double y1,double x2,double y2){
  return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}

int main (int argc, char *argv[]) {
  double x1 = 1.2;
  double x2 = 0.8;
  double y1 = 3.2;
  double y2 = 3.0;

  cout << "1d distance=" << distance(x1,x2) << endl;
  cout << "2d distance=" << distance(x1,y1,x2,y2) << endl;

  return 0;
}
```

sample09.cpp

```cpp
#include <iostream>

using namespace std;

void show_me(int a){
  cout << "Integer value is " << a << endl;
}

void show_me(double a){
  cout << "Double value is " << a << endl;
}

int main (int argc, char *argv[]) {
  int a = 1;
  double x = 2.5;

  show_me(a);
  show_me(x);

  return 0;
}
```

6.3 Automatic Type Conversion

Given the definition

```
double mpg(double miles, double gallons)
{
        return (miles / gallons);
}
```

what will happen if **mpg** is called in this way?

```
cout << mpg(45, 2) << " miles per gallon";
```

The values of the arguments will automatically be converted to type **double (45.0 and 2.0)**

6.3.1 Type Conversion Problem

Given the previous mpg definition and the following definition in the same program

```
int mpg(int goals, int misses)
// returns the Measure of Perfect Goals
{
        return (goals - misses);
}
```

what happens if mpg is called this way now?

```
cout << mpg(45, 2) << " miles per gallon";
```

The compiler chooses the function that matches parameter types so the Measure of Perfect Goals will be calculated

Do not use the same function name for unrelated functions