FUNCTIONS

Remember: Math Functions

- A function has three parts:
  - Name
  - Input parameters
  - Evaluates to an output
- Remember `y = f(x)`
  - `f` is the name
  - `x` is input parameter
  - `y` is the output

sqrt(*a*)

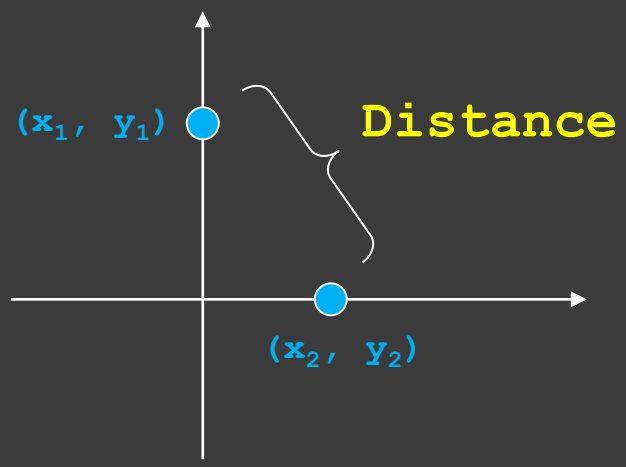
exp(*a*)

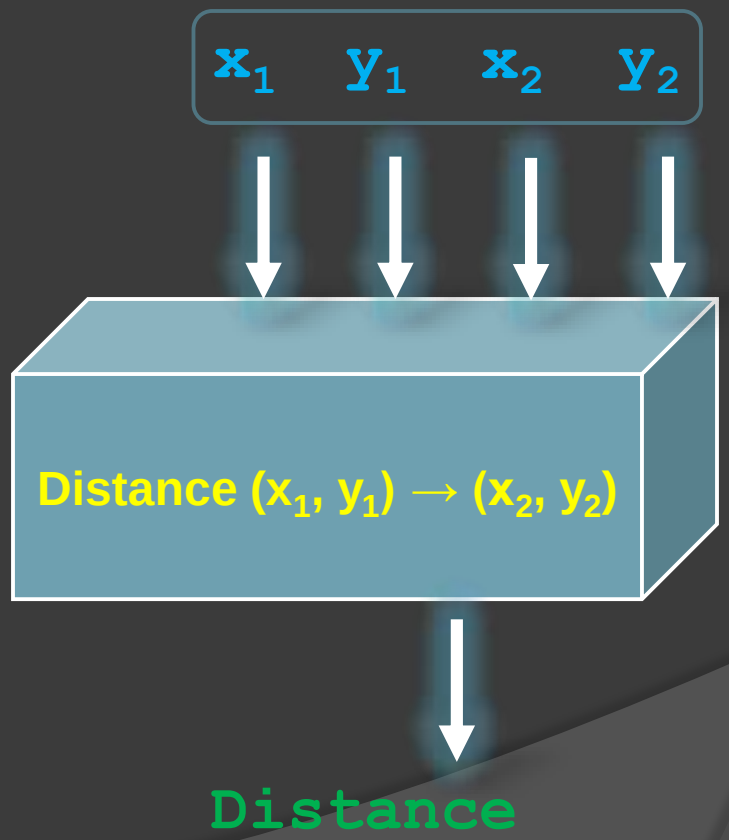
cos(*a*)

pow(*a*, *b*)

Function

- Computes a ***task*** given ***arguments*** (input parameters) and may return an ***answer***





Compute distance again and again

```
int main()
{
   . . .

   dx = x1 - x2;
   dy = y1 - y2;
   distance = sqrt(dx * dx + dy * dy);

   . . .

   dx = x1 - x2;
   dy = y1 - y2;
   distance = sqrt(dx * dx + dy * dy);

   . . .

   dx = x1 - x2;
   dy = y1 - y2;
   distance = sqrt(dx * dx + dy * dy);

   return 0;
}
```

Define a function

- **Name?**
- **Arguments?**
- **Return?**

Function dist

```
double dist(double x1, double y1, double x2, double y2);

int main()
{
  . . .
  distance = dist(x1, y1, x2, y2);
  . . .
  distance = dist(x1, y1, x2, y2);
  . . .
  distance = dist(x1, y1, x2, y2);

  return 0;
}

double dist(double x1, double y1, double x2, double y2)
{
  double dx, dy, d;

  dx = x1 - x2;
  dy = y1 - y2;
  d = sqrt(dx * dx + dy * dy);

  return d;
}
```

Header

Calls

Definition

Function Header

```
double dist(double x1, double y1, double x2, double y2);
```

Name

Notice semicolon

Return Type

Arguments

Also called *prototype*

Function Definition

```
double dist(double x1, double y1,
            double x2, double y2)
{
  double dx, dy, d;

  dx = x1 - x2;
  dy = y1 - y2;
  d = sqrt(dx * dx + dy * dy);

  return d;
}
```

Formal Parameters

Task

Answer

Function Call

- The value returned (*answer*) replaces the syntax of the function call
  - Remember we saw this with the math functions

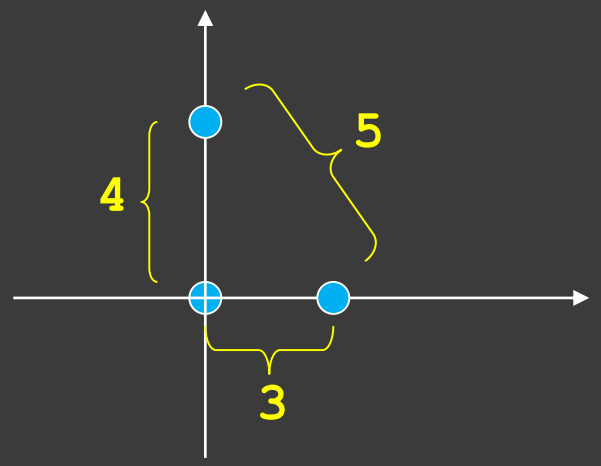
```
x1 = 0; y1 = 5; x2 = 3; y2 = 0;
distance = dist(x1, y1, x2, y2);
```

- *Evaluates* as follows when run (executed):

Actual Parameters

```
distance = dist(0, 5, 3, 0); // Eval args
distance = 5.83;     // 1) Execute function.
                     // 2) Call syntax replaced with answer.
distance = 5.83;    // Finish assignment.
```

Problem: Compute distances between three points

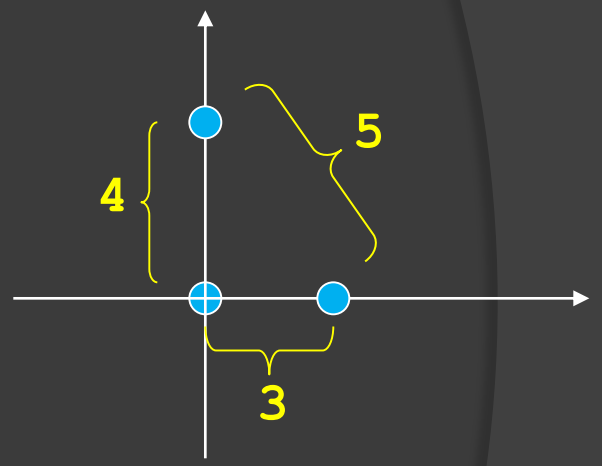



```
> pointDist1.exe
Enter point 1 (2 floats): 0 0
Enter point 2 (2 floats): 3 0
Enter point 3 (2 floats): 0 4
Distance between (0,0) and (3,0) = 3
Distance between (0,0) and (0,4) = 4
Distance between (3,0) and (0,4) = 5
```

We need to define the steps to solve the problem: Algorithm

Algorithm

1. *Input point p*
2. *Input point q*
3. *Input point r*
4. *Compute distance* $p \rightarrow q$
5. *Compute distance* $p \rightarrow r$
6. *Compute distance* $q \rightarrow r$
7. *Output distance* $p \rightarrow q$
8. *Output distance* $p \rightarrow r$
9. *Output distance* $q \rightarrow r$



pointDist1.cpp (1)

```
// compute the distances between three points
...
int main()
{
  double px, py, qx, qy, rx, ry;
  double dx, dy, dist_pq, dist_pr, dist_qr;

  // read input
  cout << "Enter point 1 (2 floats): ";
  cin >> px >> py;
  cout << "Enter point 2 (2 floats): ";
  cin >> qx >> qy;
  cout << "Enter point 3 (2 floats): ";
  cin >> rx >> ry;
```

pointDist1.cpp (2)

```
...
  // calculate distances
  dx = px - qx;
  dy = py - qy;
  dist_pq = sqrt(dx*dx + dy*dy);
  dx = px - rx;
  dy = py - ry;
  dist_pr = sqrt(dx*dx + dy*dy);
  dx = qx - rx;
  dy = qy - ry;
  dist_qr = sqrt(dx*dx + dy*dy);
...
```

pointDist1.cpp (3)

```
...
  // output distances
  cout << "Distance between (" << px << "," << py << ") "
       << "and (" << qx <<"," << qy << ") = "
       << dist_pq << endl;
  cout << "Distance between (" << px << "," << py << ") "
       << "and (" << rx <<"," << ry << ") = "
       << dist_pr << endl;
  cout << "Distance between (" << qx << "," << qy << ") "
       << "and (" << rx <<"," << ry << ") = "
       << dist_qr << endl;

  return 0;
}
```

pointDist1.cpp (2)

```
...
  // calculate distances
  dx = px - qx;
  dy = py - qy;
  dist_pq = sqrt(dx*dx + dy*dy);
  dx = px - rx;
  dy = py - ry;
  dist_pr = sqrt(dx*dx + dy*dy);
  dx = qx - rx;
  dy = qy - ry;
  dist_qr = sqrt(dx*dx + dy*dy);
...
```

We already defined a function for this task (earlier)

pointDist2.cpp

```
  // calculate distances
  dist_pq = dist(px, py, qx, qy);
  dist_pr = dist(px, py, rx, ry);
  dist_qr = dist(qx, qy, rx, ry);

  // output distances
  cout << "Distance between (" << px << "," << py << ") "
       << "and (" << qx <<"," << qy << ") = "
       << dist_pq << endl;
  cout << "Distance between (" << px << "," << py << ") "
       << "and (" << rx <<"," << ry << ") = "
       << dist_pr << endl;
  cout << "Distance between (" << qx << "," << qy << ") "
       << "and (" << rx <<"," << ry << ") = "
       << dist_qr << endl;

  return 0;
}
```

Can we define a function for this repetitive task?

pointDist2.cpp

```
  // output distances
  // Note redundancy in these statements
  cout << "Distance between (" << px << "," << py << ") "
       << "and (" << qx <<"," << qy << ") = "
       << dist_pq << endl;
  cout << "Distance between (" << px << "," << py << ") "
       << "and (" << rx <<"," << ry << ") = "
       << dist_pr << endl;
  cout << "Distance between (" << qx << "," << qy << ") "
       << "and (" << rx <<"," << ry << ") = "
       << dist_qr << endl;

  return 0;
}
```

Define a function

- **Name?**
- **Arguments?**
- **Return?**

Function: Output distance

```
cout << "Distance between (" << px << "," << py << ") "
     << "and (" << qx <<"," << qy << ") = "
     << dist_pq << endl;
```

- Name?
  - `output_distance`
- Arguments?
  - 5 values, type double
- Answer (return value)?
  - None. Use `void`.

Function: output_distance

```
void output_distance(double x1, double y1,
                     double x2, double y2,
                     double distance)
{
  cout << "Distance between (" << x1 << "," << y1 << ") "
       << "and (" << x2 <<"," << y2 << ") = "
       << distance << endl;
}
```

Formal Parameters

Task

No return

Function Header

```
void output_distance(double x1, double y1,
                     double x2, double y2,
                     double distance);
```

- The **`void`** data type indicates a function ***returns no value***
- Defines a ***procedure***
  - A function that returns no value

pointDist3.cpp

```
. . .
  // calculate distances
  dist_pq = dist(px, py, qx, qy);
  dist_pr = dist(px, py, rx, ry);
  dist_qr = dist(qx, qy, rx, ry);

  // output distances
  output_distance(px, py, qx, qy, dist_pq);
  output_distance(px, py, rx, ry, dist_pr);
  output_distance(qx, qy, rx, ry, dist_qr);

  return 0;
}
```

Functions

```
double dist(double x1, double y1, double x2, double y2);
void output_distance(double x1, double y1, double x2, double y2,
                     double distance);

int main()
{
  . . .
  return 0;
}

double dist(double x1, double y1, double x2, double y2)
{
  double dx, dy, d;

  dx = x1 - x2;
  dy = y1 - y2;
  d = sqrt(dx * dx + dy * dy);

  return d;
}

void output_distance(double x1, double y1,
                     double x2, double y2,
                     double distance)
{
  cout << "Distance between (" << x1 << "," << y1 << ") "
       << "and (" << x2 <<"," << y2 << ") = "
       << distance << endl;
}
```

Headers

Calls

Definitions

Function: No Arguments

- Define a function that prompts the user for an age and returns the age when valid

```
int age(0);

cout << "Please enter your age: ";
cin >> age;
while (age < 0 || age > 120) {
    cout << "Age must be between 0 and 120."
         << endl;
    cout << "Please enter your age: ";
    cin >> age;
 }
 return age;
```

Function: read_age

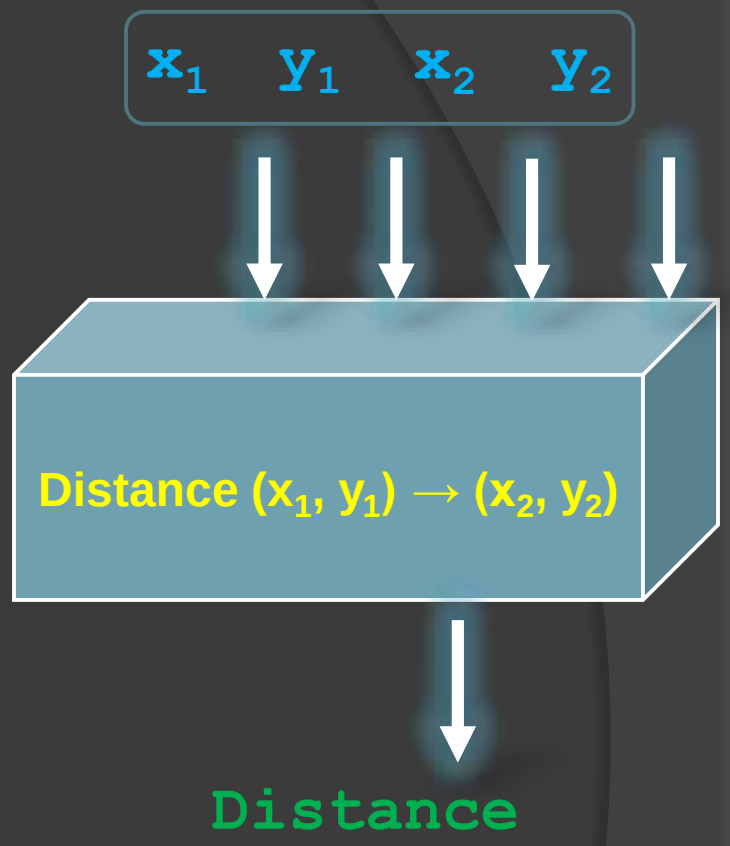
```
int read_age()
{
   int age(0);

   cout << "Please enter your age: ";
   cin >> age;
   while (age < 0 || age > 120) {
       cout << "Age must be between 0 and 120."
   << endl;
       cout << "Please enter your age: ";
       cin >> age;
    }

    return age;
}
```

- A function may not need any **arguments**
- For example, read in data

Why Functions?

- Reduce duplicate code
- Enables *code reusability*
- Complex programs can be decomposed into *simpler parts*
- Hide (*abstract*) parts of a program

Information Hiding



Abstraction: Function implementation hidden in the "box"

- I showed the function headers of the **`cmath`** functions (**`pow`**, **`sqrt`**, etc), but their implementations are hidden from us
  - Yet you use them in your program anyway
  - You are not concerned with their implementation
  - If someone changed the implementation of the function (inside the "box"), we would not have to change our program calling it

Example: Circle Area

- Define a function to compute the area of a circle
- Name?
  - `circle_area`
- Arguments?
  - radius (type?)
- Answer?
  - Return a value of type double

Function Prototype

```
double circle_area(double radius);
```

- Name?
  - `circle_area`
- Arguments?
  - radius (type?)
- Answer?
  - Return a value of type double

Function Definition

```
double circle_area(double radius)
{
  double area;

  area = M_PI * radius * radius;
  return area;
}
```

circleArea.cpp

```
int main()
{
  double area(0.0);                          // area of circle
  double radius(0.0);                        // radius of circle
  for (int i = 1; i <= 5; i++)
  {
    radius = i;                              // radius of circle
    area = circle_area(radius);              // call function
    cout << "Radius: " << radius
         << "   Area: " << area << endl;
  }
  return 0;
}
```

Use return value in the call line

```
...
int main()
{
  double area(0.0);                          // area of circle
  double radius(0.0);                        // radius of circle
  for (int i = 1; i <= 5; i++)
  {
    radius = i;                              // radius of circle
    area = circle_area(radius);              // call function
    cout << "Radius: " << radius
         << "   Area: " << area << endl;
  }
...
```

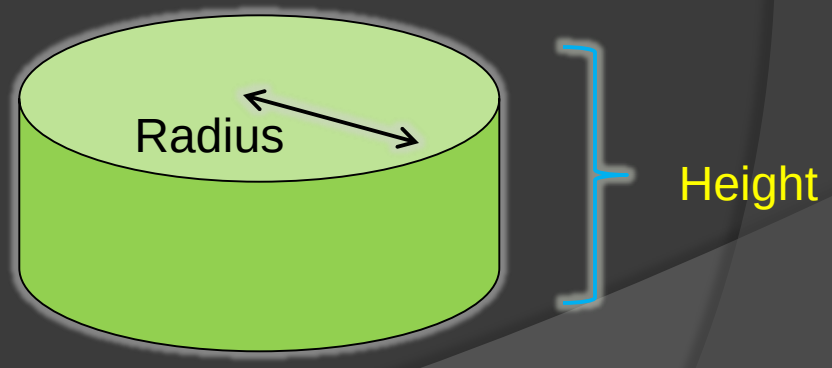
```
> circleArea.exe
Radius: 1   Area: 3.14159
Radius: 2   Area: 12.5664
Radius: 3   Area: 28.2743
Radius: 4   Area: 50.2655
Radius: 5   Area: 78.5398
```

Example: Cylinder Volume

- Define a function to compute the volume of a cylinder

- Name?
  - **`cylinder_volume`**
  - *cylinder volume = circular base area × height*
  - **Note**: *cylinder volume = circle_area(radius) × height*

- Arguments?
  - Radius, height



- Answer?
  - Return a value of type double

Function Prototype

```
double cylinder_volume(double radius,
                       double height);
```

- Name?
  - `cylinder_volume`
- Arguments?
  - Radius, height
- Answer?
  - Return a value of type double

Function Definition

```
double cylinder_volume(double radius,
                       double height)
{
  double volume;
  double base_area;

  base_area = circle_area(radius);
  volume = base_area * height;

  return volume;
}
```

Call another function

Better Function Definition

```
double cylinder_volume(double radius,
                       double height)
{
  return circle_area(radius) * height;
}
```

More efficient implementation

circleAreaError.cpp

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

int main()
{ . . .
  for (int i = 1; i <= 5; i++)
    {
      radius = i;
      area = circle_area(radius); // call function
      . . .
    }
  return 0;
}

// function circle_area definition
double circle_area(double radius)
{
  . . .
  return(area);
}
```

Oops! Forgot function header

```
14.   for (int i = 1; i <= 5; i++)
15.     {
16.         radius = i;
17.         area = circle_area(radius); // call function
             ...
24.     }
25.
26.   // function circle_area definition
27.   double circle_area(double radius)
28.   {
        ...
31.     return(area);
32.   }
```

```
> g++ circleAreaError.cpp
Compiling circleAreaError.cpp into circleAreaError.exe.
circleAreaError.cpp: In function 'int main()':
circleAreaError.cpp:17: error: 'computeCircleArea' was not
   declared in this scope
>
```

Layout of Your Program

```
preprocessor directives

function headers (prototypes)

int main()
{
   constant definitions (ALL CAPS)
   variable declarations

   other statements

   return 0;
}

function definitions
```

Function Prototype

- Function prototypes (headers) can be placed:
  - Anywhere ***above*** where the function will be called (just like variables)
  - In general, at least for this course, just place prototypes above **`int main()`**

Function Prototype

```
double dist(double x1, double y1, double x2, double y2);
double life_expectancy(int age, double height, double weight);
```

- Alternative function prototype format (used by text):

```
// Note: Variable names omitted
double dist(double, double, double, double);

double life_expectancy(int, double, double);
```

Function Definition

- After declaring the function prototype, we have to define what it does
- In this course, you should place all function definitions ***after*** **`main()`**

What's the error?

```
. . .
void computeCircleArea(double radius);
. . .
int main()
{ . . .
  for (int i = 1; i <= 5; i++)
    {
      radius = i;                          // radius of circle
      area = computeCircleArea(radius); // call function
      . . .
    }
  . . .
}

double computeCircleArea(double radius)
{
  . . .
  return area;
}
```

```
8.   void computeCircleArea(double radius);
     . . .
14.    for (int i = 1; i <= 5; i++)
15.      {
16.        radius = i;                          // radius of circle
17.        area = computeCircleArea(radius); // call function
           . . .
23.    }
         . . .
26.  double computeCircleArea(double radius)
27.  {
       . . .
31.    return area;
32.  }
```

```
> g++ circleAreaError2.cpp
Compiling circleAreaError2.cpp into circleAreaError2.exe.
circleAreaError2.cpp: In function 'int main()':
circleAreaError2.cpp:17: error: void value not ignored as it ought to be
circleAreaError2.cpp: In function 'double computeCircleArea(double)':
circleAreaError2.cpp:26: error: new declaration 'double
    computeCircleArea(double)'
circleAreaError2.cpp:8: error: ambiguates old declaration 'void
    computeCircleArea(double)'
>
```

What's the error?

```
. . .
double computeCircleArea(int radius);
. . .
int main()
{ . . .
  for (int i = 1; i <= 5; i++)
    {
      radius = i;                          // radius of circle
      area = computeCircleArea(radius); // call function
      . . .
    }
  . . .
}

double computeCircleArea(double radius)
{
  . . .
  return area;
}
```

```
8.   double computeCircleArea(int radius);
     . . .
14.    for (int i = 1; i <= 5; i++)
15.      {
16.        radius = i;                          // radius of circle
17.        area = computeCircleArea(radius); // call function
           . . .
23.    }
         . . .
26.  double computeCircleArea(double radius)
27.  {
       . . .
31.    return area;
32.  }
```

```
> g++ circleAreaError3.cpp
Compiling circleAreaError3.cpp into circleAreaError3.exe.
/tmp/ccV2a0SZ.o: In function `main':
circleAreaError3.cpp:(.text+0x30): undefined reference to
   `computeCircleArea(int)'
collect2: ld returned 1 exit status
>
```

distance.cpp

```
. . .
// function prototype
double computeDistance(double x, double y);

int main()
{
  double x(0.0), y(0.0);

  cout << "Enter x, y: ";
  cin >> x >> y;

  cout << "Distance to origin = " << computeDistance(x, y)
       << endl;

  return 0;
}
```

distance.cpp

```
. . .
// function to compute distance to origin
double computeDistance(double x, double y)
{
  double d(0.0);

  d = sqrt(x*x + y*y);

  return d;
}
```

What's the error?

```
. . .
double computeDistance(double x, double y);

int main()
{
  . . .
  cout << "Distance to origin = " << computeDistance(x,y)
       << endl;
  . . .
}

double computeDistance(double x, double y, double z)
{
  double d(0.0);

  d = sqrt(x*x + y*y + z*z);
  return d;
}
```

```
double computeDistance(double x, double y);
  . . .
  cout << "Distance to origin = " << computeDistance(x,y)
       << endl;
  . . .
double computeDistance(double x, double y, double z)
{
  . . .
  d = sqrt(x*x + y*y + z*z);
  return d;
}
```

```
> g++ distanceError.cpp
Compiling distanceError.cpp into distanceError.exe.
/tmp/cccKGvqW.o: In function `main':
distanceError.cpp:(.text+0x54): undefined reference to
   `computeDistance(double, double)'
collect2: ld returned 1 exit status
>
```

Function Prototype

- The function prototype should look **EXACTLY LIKE** the function definition (except that the prototype ends with a ';').

- Protoype:

```
double computeDistance(double x, double y);
```

- Definition:

```
double computeDistance(double x, double y)
```

Functions in Expressions

```
double dist(double x1, double y1, double x2, double y2);
```

- Functions can appear in expressions:

```
double z = dist(1.0, 0.0, 0.0, 1.0) + 2;
```

- Expressions can be arguments to functions:

```
double x1 = 6.0, x2 = 3.0;
double z = dist(x2 - x1, 2.0 * 3, x1 * 2, 2.0 + 4.0);
```

Invalid Function Calls

```
double dist(double x1, double y1, double x2, double y2);
double average(int i, int j);
double life_expectancy(int age, double height,
                       double weight);
```

- Why are these function calls ***invalid***?

```
double z = dist(2.0, 0.0, 0.0);
double z = average(3, 5, 9);
double life = life_expectancy(22, false, 5.5);
```

More Invalid Function Calls

```
void output_distance(double x1, double y1,
                     double x2, double y2,
                     double distance);
```

- Why is this function call *invalid*?

```
int k = output_distance(3.0, 0.0, 0.0, 4.0, 5.0);
```

What is wrong with this program?

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

int maxInt(int a, int b);

int main()
{
   cout << maxInt(88, 102) << endl;

   return 0;
}

int maxInt(int a, int b);
{
   if (a >= b)
       { return a; }
   else
       { return b; }
}
```

What is wrong with this program?

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

int maxInt(int a, int b)

int main()
{
   cout << maxInt(88, 102) << endl;

   return 0;
}

int maxInt(int a, int b)
{
   if (a >= b)
       { return a; }
   else
       { return b; }
}
```

RETURN

The `return` Statement

```
void output_distance(double x1, double y1,
                     double x2, double y2,
                     double distance)
{
  cout << "Distance between (" << x1 << "," << y1 << ") "
       << "and (" << x2 <<"," << y2 << ") = "
       << distance << endl;
}
```

No return

- All functions that have a ***void return type*** do not need a return statement

The `return` Statement

```
double computeDistance(double x, double y)
{
  double d(0.0);

  d = sqrt(x*x + y*y);
  return d;
}
```

Return

- All functions that have a ***non-void return type*** must finish execution with `return` statement

The `return` Statement

```
int maxInt(int a, int b)
{
   if (a >= b) {
       return a;
       // Don't put code here!
   }

   return b;
   // Don't put code here!
}
```

- A function may have more than one return statement
  - ***One and only one*** return statement is executed
- Function execution ***immediately exits*** and returns to the function call when a `return` statement is executed

The `return` Statement

```
int maxInt(int a, int b)
{
   if (a >= b) {
       return a;
   }
   else {
       return b;
   }
   // Will never reach here!
}
```

```
int maxInt(int a, int b)

{
   if (a >= b) {
       return a;
   }

   return b;
}
```

- Why is the `else` unnecessary?

VARIABLE SCOPE IN FUNCTIONS

Variable Scoping

```
...
double computeCircleArea(double radius);

int main()
{
  double r(3.0);
  double a(0.0);

  a = computeCircleArea(r);
  cout << "Radius: " << r
       << "   Area: " << a << endl;
...
}
```

Scope of "r" and "a" is function main()

```
double computeCircleArea(double radius)
{
  double area(0.0);

  area = M_PI * radius * radius;
  return area;
}
```

Scope of "radius" and "area" is function computeCircleArea()

Variable Scoping Error

```
...
double computeCircleArea(double radius);

int main()
{
  double r(3.0); // radius of circle
  double a(0.0);

  a = computeCircleArea(r);
  cout << "Radius: " << r
       << "   Area: " << a << endl;
...
}

double computeCircleArea(double radius)
{
  double area(0.0);

  area = M_PI * r * r;
  return area;
}
```

ERROR: Not IN SCOPE

Scope of "r" and "a" is function main()

Scope of "radius" and "area" is function computeCircleArea()

```
10.int main()
11.{
12.   double r(3.0);      // radius of circle
...
19.}
...
22.double computeCircleArea(double radius)
23.{
24.   double area(0.0);
25.
26.   area = M_PI * r * r;    // *** ERROR: Not in scope
27.   return area;
28.}
```

```
> g++ scopeError.exe
scopeError.cpp: In function 'double
   computeCircleArea(double)':
scopeError.cpp:26: error: 'r' was not declared in this scope
```

scopeExample.cpp()

```
...
void f(int x);
void g(int y);

int main()
{
  int a(5);

  f(5);
  g(5);
  cout << "main: a = " << a
       << endl;

  return 0;
}
```

```
void f(int x)
{
  int a(0);

  a = 2 * x;
  cout << "f: a = " << a
       << endl;

}

void g(int x)
{
  int a(0);

  a = x * x;
  cout << "g: a = " << a
       << endl;

}
```

```
...
void f(int x);
void g(int y);

int main()
{
  int a(5);

  f(5);
  g(5);
  cout << "main: a = " << a
       << endl;
...
```

```
void f(int x)
{
  int a(0);

  a = 2 * x;
  cout << "f: a = " << a << endl;
...

void g(int x)
{
  int a(0);

  a = x * x;
  cout << "g: a = " << a << endl;
...
```

```
> scopeExample.exe
f: a = 10
g: a = 25
main: a = 5
```

GLOBAL VARIABLES

globalExample.cpp()

```
...
void read_inputs();

// global variables
int age(0);
double height(0.0), weight(0.0);

int main()
{
  read_inputs();

  cout << "age = " << age
       << endl;
  cout << "height = "
       << height << endl;
  cout << "weight = "
       << weight << endl;

  return 0;
}
```

```
void read_inputs()
{
  cout << "Enter age: ";
  cin >> age;
  cout << "Enter height: ";
  cin >> height;
  cout << "Enter weight: ";
  cin >> weight;
}
```

Global Variables: Variables defined outside of all functions have scope over the entire program

```
...
// global variables
int age(0);
double height(0.0), weight(0.0);

int main()
{
  read_inputs();

  cout << "age = " << age <<
    endl;
  cout << "height = " << height
    << endl;
  cout << "weight = " << weight
    << endl;
...
```

```
void read_inputs()
{
  cout << "Enter age: ";
  cin >> age;
  cout << "Enter height: ";
  cin >> height;
  cout << "Enter weight: ";
  cin >> weight;
}
```

```
> globalExample.exe
Enter age: 35
Enter height: 5.8
Enter weight: 155
age = 35
height = 5.8
weight = 155
```

globalExample2.cpp()

```
...
// global variables
const double TOLERANCE = 0.001;

int main()
{
  double x(0.0), y(0.0);

  cout << "Enter x, y: ";
  cin >> x >> y;

  cout << "1/x = " << reciprocal(x)
       << endl;
  if (equals(x,y))
    { cout << "x equals y."
           << endl; }
  else
    { cout << "x does not equal y."
           << endl; }

  return 0;
}
```

```
double reciprocal(double x)
{
  if (abs(x) >= TOLERANCE)
    { return 1 / x; }
  else
    { return 1 / TOLERANCE; }
}

bool equals(double x, double y)
{
  return abs(x-y) < TOLERANCE;
}
```

```
...
// global variables
const double TOLERANCE = 0.001;
...
int main()
{
...
  cout << "1/x = " << reciprocal(x)
    << endl;
  if (equals(x,y))
    { cout << "x equals y." << endl;
    }
  else
    { cout << "x does not equal y."
    << endl; }

  return 0;
}
```

```
double reciprocal(double x)
{
  if (abs(x) >= TOLERANCE)
    { return(1/x); }
  else
    { return(1/TOLERANCE); }
}

bool equals(double x, double y)
{
  if (abs(x-y) < TOLERANCE)
    { return(true); }
  else
    { return(false); }
}
```

```
> globalExample2.exe
Enter x, y: 0.00004 0.00002
1/x = 1000
x equals y.
```

globalError.cpp()

```
...
void read_inputs();

// global variables
int age(0);
double height(0.0), weight(0.0);

int main()
{
  read_inputs();

  cout << "age = " << age
       << endl;
  cout << "height = " << height
       << endl;
  cout << "weight = " << weight
       << endl;

  return 0;
}
```

```
void read_inputs()
{
  // These mask the globals!!!
  int age(0);
  double height(0.0), weight(0.0);

  cout << "Enter age: ";
  cin >> age;
  cout << "Enter height: ";
  cin >> height;
  cout << "Enter weight: ";
  cin >> weight;
}
```

```
...
// global variables
int age(0);
double height(0.0), weight(0.0);

int main()
{
  read_inputs();

  cout << "age = " << age
       << endl;
  cout << "height = " << height
       << endl;
  cout << "weight = " << weight
       << endl;
...
```

```
void read_inputs()
{
  // These mask the globals!!!
  int age(0);
  double height(0.0), weight(0.0);

  cout << "Enter age: ";
  cin >> age;
  cout << "Enter height: ";
  cin >> height;
  cout << "Enter weight: ";
  cin >> weight;
}
```

```
> globalExample.exe
Enter age: 35
Enter height: 5.8
Enter weight: 155
age = 0
height = 0
weight = 0
```

WHY?

Documenting Functions

- EVERY function should have a comment explaining what it does;
- EVERY function parameter should have a comment explaining the parameter.
- Example:

```
// Return the distance from (x1,y1) to (x2,y2).
// x1 = x-coordinate of first point.
// y1 = y-coordinate of first point.
// x2 = x-coordinate of second point.
// y2 = y-coordinate of second point.
double dist(double x1, double y1, double x2, double y2)
{
...
}
```