Functions
## Moon Landings

<table>
<thead>
<tr>
<th>Name</th>
<th>Time</th>
<th>Fuel</th>
<th>Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Will Cyr</td>
<td>13</td>
<td>86.20</td>
<td>-0.49</td>
</tr>
<tr>
<td>Yongjiao Yu</td>
<td>13</td>
<td>86.00</td>
<td>-1.82</td>
</tr>
<tr>
<td>Bin Tian</td>
<td>13</td>
<td>87.00</td>
<td>-1.69</td>
</tr>
<tr>
<td>Nan Xia</td>
<td>13</td>
<td>87.00</td>
<td>-1.69</td>
</tr>
<tr>
<td>Chenli Yuan</td>
<td>13</td>
<td>87.00</td>
<td>-1.69</td>
</tr>
<tr>
<td>Scott Oliver</td>
<td>13</td>
<td>87.70</td>
<td>-2.74</td>
</tr>
<tr>
<td>Mike Robell</td>
<td>13</td>
<td>87.88</td>
<td>-3.00</td>
</tr>
</tbody>
</table>
Triangle Area Computation

\[ a = |p_2 - p_1| \]
\[ b = |p_3 - p_1| \]
\[ c = |p_3 - p_2| \]
\[ a = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \]
\[ \text{area} = \sqrt{p(p-a)(p-b)(p-c)} \]
\[ p = \frac{a + b + c}{2} \]

How would you write this program?
int main()
{
    double x1=0, y1=0;
    double x2=17, y2=10.3;
    double x3=-5.2, y3=5.1;

    double a, b, c;  /* Triangle side lengths */
    double p;  /* For Heron's formula */
    double area;

    a = \left| p_2 - p_1 \right|
    b = \left| p_3 - p_1 \right|
    c = \left| p_3 - p_2 \right|

    a = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}
Lengths of Edges

\[ a = \sqrt{((x1 - x2) \cdot (x1 - x2)) + ((y1 - y2) \cdot (y1 - y2))}; \]
\[ b = \sqrt{((x1 - x3) \cdot (x1 - x3)) + ((y1 - y3) \cdot (y1 - y3))}; \]
\[ c = \sqrt{((x2 - x3) \cdot (x2 - x3)) + ((y2 - y3) \cdot (y2 - y3))}; \]

\[ a = |p2 - p1| \]
\[ b = |p3 - p1| \]
\[ c = |p3 - p2| \]

\[ a = \sqrt{(x2 - x1)^2 + (y2 - y1)^2} \]

area = \sqrt{p(p-a)(p-b)(p-c)}

\[ p = \frac{a + b + c}{2} \]
\[
p = \frac{(a + b + c)}{2}; \\
\text{area} = \sqrt{p \times (p - a) \times (p - b) \times (p - c)};
\]

\[
\text{printf}("%f\n", \text{area});
\]
int main()
{
    double x1=0, y1=0;
    double x2=17, y2=10.3;
    double x3=-5.2, y3=5.1;

double a, b, c; /* Triangle side lengths */
double p; /* For Heron's formula */
double area;

    a = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    b = sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
    c = sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3));

    p = (a + b + c) / 2;
    area = sqrt(p * (p - a) * (p - b) * (p - c));

    printf("%f\n", area);
}

What if I made a mistake on the edge length equation?
Functions

• Functions are subprograms that perform some operation and return one value
• They “encapsulate” some particular operation, so it can be re-used by others (for example, the abs() or sqrt() function)
Characteristics

• Reusable code
  – code in sqrt() is reused often
• Encapsulated code
  – implementation of sqrt() is hidden
• Can be stored in libraries
  – sqrt() is a built-in function found in the math library
Writing Your Own Functions

• Consider a function that converts temperatures in Celsius to temperatures in Fahrenheit.
  – Mathematical Formula:

    \[ F = C \times 1.8 + 32.0 \]

  – We want to write a C function called CtoF
Convert Function in C

double CtoF ( double paramCel )
{
    return paramCel*1.8 + 32.0;
}

• This function takes an input parameter called paramCel (temp in degree Celsius) and returns a value that corresponds to the temp in degree Fahrenheit
#include <stdio.h>

double CtoF( double );

/************************************************************************
* Purpose: to convert temperature from Celsius to Fahrenheit
************************************************************************/
int main()
{
    double c, f;
    printf("Enter the degree (in Celsius): ");
    scanf("%lf", &c);

    f = CtoF(c);
    printf("Temperature (in Fahrenheit) is %lf\n", f);
}

double CtoF( double paramCel)
{
    return paramCel * 1.8 + 32.0;
}
Terminology

• **Declaration**: double CtoF( double );

• **Invocation (Call)**: Fahr = CtoF(Cel);

• **Definition**:  
  double CtoF( double paramCel )  
  {  
    return paramCel*1.8 + 32.0;  
  }
Function Declaration

• Also called function prototype:

\[
\text{return\_type \ function\_name (parameter\_list)}
\]

\[
\text{double CtoF(double)}
\]

• Declarations describe the function:
  – the return type and function name
  – the type and number of parameters
Function Definition

```
return_type function_name (parameter_list) 
{
    ....
    function body
    ....
}

double CtoF(double paramCel)
{
    return paramCel*1.8 + 32.0;
}
```
1. Call copies argument c to parameter paramCel

2. Control transfers to function “CtoF”

```c
int main()
{
    f = CtoF(c);
}

double CtoF ( double paramCel )
{
    return paramCel*1.8 + 32.0;
}
```
int main()
{
  ... 
  f = CtoF(c);
}

3. Expression in "CtoF" is evaluated

double CtoF ( double paramCel )
{
  return paramCel*1.8 + 32.0;
}

4. Value of expression is returned to "main"
Local Objects

• The parameter “paramCel” is a local object which is defined only while the function is executing. Any attempt to use “paramCel” outside the function is an error.

• The name of the parameter need not be the same as the name of the argument. Types must agree.
int main()
{
    double x1=0, y1=0;
    double x2=17, y2=10.3;
    double x3=-5.2, y3=5.1;

    double a, b, c;   /* Triangle side lengths */
    double p;         /* For Heron's formula */
    double area;

    a = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    b = sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
    c = sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3));

    p = (a + b + c) / 2;
    area = sqrt(p * (p - a) * (p - b) * (p - c));

    printf("%f\n", area);
}

Can we do better than this?
What should we name our function?

```
a = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
```

“Length” sounds like a good idea.

```
??? Length( ???  )
{
}
```
What does our function need to know?

\[
a = \sqrt{(x1 - x2)^2 + (y1 - y2)^2};
\]

(x, y) for two different points:

```c
???
Length(double x1, double y1,
        double x2, double y2)
{
}
```
What does our function return?

\[ a = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}; \]

A computed value which is of type \textit{double}

```c
double Length(double x1, double y1, double x2, double y2) {
}
```
How does it compute it?

\[
a = \sqrt{(x1 - x2) \times (x1 - x2) + (y1 - y2) \times (y1 - y2)};
\]

A computed value which is of type \textit{double}

define \textbf{double} Length(double x1, double y1, double x2, double y2)  
{  
  double len;  
  len = \sqrt{(x1 - x2) \times (x1 - x2) + (y1 - y2) \times (y1 - y2)};  
  return(len);  
}
```c
#include <stdio.h>
#include <math.h>

double Length(double x1, double y1, double x2, double y2);

int main()
{
    double x1=0, y1=0;
    double x2=17, y2=10.3;
    double x3=-5.2, y3=5.1;

    double a, b, c;  /* Triangle side lengths */
    double p;       /* For Heron's formula */
    double area;

    a = Length(x1, y1, x2, y2);
    b = Length(x1, y1, x3, y3);
    c = Length(x2, y2, x3, y3);

    p = (a + b + c) / 2;
    area = sqrt(p * (p - a) * (p - b) * (p - c));

    printf("%f\n", area);
}

double Length(double x1, double y1, double x2, double y2)
{
    double len;
    len = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    return(len);
}
```

**Declaration**

/* Declaration */
double Length(double x1, double y1, double x2, double y2);

/*
 * Program to determine the area of a triangle
 */

int main()
{
    double x1=0, y1=0;
    double x2=17, y2=10.3;
    double x3=-5.2, y3=5.1;

    double a, b, c;  /* Triangle side lengths */
    double p;       /* For Heron's formula */
    double area;

    a = Length(x1, y1, x2, y2);
    b = Length(x1, y1, x3, y3);
    c = Length(x2, y2, x3, y3);

    p = (a + b + c) / 2;
    area = sqrt(p * (p - a) * (p - b) * (p - c));

    printf("%f\n", area);
}

**Definition**

/* Definition */
double Length(double x1, double y1, double x2, double y2)
{
    double len;
    len = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    return(len);
}
#include <stdio.h>
double convert( double );

int main()
{
    double c, f;
    printf(“Enter the degree (in Celsius): “);
    scanf(“%lf”, &c);
    f = convert(c);
    printf(“Temp (in Fahrenheit) for %lf Celsius is %lf”, paramCel, f);
}

double CtoF( double paramCel)
{
    return c * 1.8 + 32.0;
}

\textbf{Potential Errors}

\textbf{Scope} – Where a variable is known to exist.

\textbf{Error! paramCel is not defined}

\textbf{No variable is known outside of the curly braces that contain it, even if the same name is used!}
```c
#include <stdio.h>

double GetTemperature();
double CelsiusToFahrenheit( double );
void DisplayResult( double, double );

int main()
{
    double TempC, TempF;
        // Temperature in degrees Celsius
    TempF = CelsiusToFahrenheit(TempC);
        // Temperature in degrees Fahrenheit
    TempC = GetTemperature();
    TempF = CelsiusToFahrenheit(TempC);
    DisplayResult(TempC, TempF);

    return 0;
}
```
double GetTemperature()
{
    double Temp;

    printf("\nPlease enter a temperature in degrees Celsius: ");
    scanf("%lf", &Temp);
    return Temp;
}
Function: CelsiusToFahrenheit

double CelsiusToFahrenheit(double Temp) {
    return (Temp * 1.8 + 32.0);
}
Function: DisplayResult

```c
void DisplayResult(double CTemp, double FTemp)
{
    printf("Original: %5.2f C \n", CTemp);
    printf("Equivalent: %5.2f F \n", FTemp);

    return;
}
```
Declarations (Prototypes)

double GetTemp( );
double CelsiusToFahrenheit( double );
void Display( double, double );

• *void* means “nothing”. If a function doesn’t return a value, its return type is *void*
Abstraction

int main()
{
    double TempC, // Temperature in degrees Celsius
         TempF; // Temperature in degrees Fahrenheit

    TempC = GetTemperature();
    TempF = CelsiusToFahrenheit(TempC);
    DisplayResult(TempC, TempF);

    return 0;
}

We are hiding details on how something is done in the function implementation.
Another Way to Compute Factorial

**Pseudocode for factorial(n)**

```plaintext
if n == 0 then
   result = 1
else
   result = n * factorial(n - 1)
```

After all, \(5! = 5 \times 4 \times 3 \times 2 \times 1 = 5 \times 4!\)
Recursive Functions

Factorial function contains an invocation of itself.

We call this a: *recursive call.*

```c
int Factorial(int n)
{
    if(n == 0) return 1;
    else return n * Factorial(n-1);
}
```

This works much like proof by induction.

Recursive functions must have a *base case* (if n == 0): why?

```c
if n == 0 then
    result = 1
else
    result = n * factorial(n - 1)
```
Infinite Recursion

What if I omit the “base case”?

This leads to infinite recursion!

```c
int Factorial(int n)
{
    return n * Factorial(n-1);
}
```

Factorial(3) =
3 * Factorial(2) =
3 * 2 * Factorial(1) =
3 * 2 * 1 * Factorial(0) =
3 * 2 * 1 * 0 * Factorial(-1) =
...

cbowen@ubuntu:~/cse251$ ./combi1
Input n: 5
Input k: 3
Segmentation fault
Psuedocode and Function

```c
int Factorial(int n)
{
    if(n == 0)
        return 1;
    else
        return n * Factorial(n-1);
}
```

**Declaration:**
```
int Factorial(int n);
```

**Invocation:**
```
f = Factorial(7);
```

**Definition:**
```
int Factorial(int n)
{
    if(n == 0)
        return 1;
    else
        return n * Factorial(n-1);
}
```