# Functions

- A block of code that performs a specific task
- Modularize your code for efficiency and reuse
- Already using functions : `printf(), scanf()`
- User defined functions can be created
- Allows for a divide and conquer approach
- User functions can lead to user defined libraries

# Functions

- First function is always main
- Functions are called from within other functions
- Defined by a type and name
  - `char, int, float, double, void`
- Each returns a value (except `void`)
- Arguments passed by value or by reference
- Pointer functions return pointers

# Function Structure and Syntax

- Prototype : All functions at the start of the program.

```
returntype functionname(argtype1, argtype2, ...);
```

  - C must have knowledge about the function type it returns and the parameter types the function expects.
- Arguments
  - Provide input but input is not always needed
  - Can be constant values, variables, or expressions
  - No limit on the number of arguments

# Function Structure and Syntax

- Calling
  - Specified as a statement
  - Required data type arguments included
  - Returned values can be stored in a variable, used as an expression within another function, or ignored

```
returntype fn_name(1, parameterdef2,...); // prototype

int main() {
    fn_name(a,b,...); //call
}

returntype fn_name(1, parameterdef2,...) // definition
{
   localvariables
   functioncode
}
```

```c
#include <stdio.h>

float findaverage(float, float); // function prototype

int main() {
    float a=5.0,b=15.0,result;
    result=findaverage(a,b); // function call
    printf("average=%f\n",result);
    return 0;
}

float findaverage(float a, float b) // function definition
{
    float average;
    average=(a+b)/2.0;
    return(average);
}
```

# void functions

- Use the return type void when nothing is being returned:

```
#include <stdio.h>

void squares();

void main() {
    squares();
}

void squares() {
    int loop;
    for (loop=1;loop<10;loop++)
    { printf("%d\n",loop*loop); }
    return;
}
```

# Static Variables

- Local to particular function
- Initialized once (on the first call to function)
- Value of static variable remains intact
- Next function call static variable has the same value
- To define a static variable simply prefix the variable declaration with the static keyword.

```
static int count = 0;
```

# Passing Arguments to Functions

- Function variables are "pass by value" *except for arrays*

```
void demonstrate_pass_by_value(float a, float b) {
   float temp;
   temp = a;
   a = b;
   b = temp;
   printf("demo value function: Swapping values,
    a = %2.2f, b = %2.2f\n", a, b);
}

void demonstrate_pass_by_reference(float *a, float *b) {
   float temp;
   temp = *a;
   *a = *b;
   *b = temp;
   printf("demo reference: Swapping values,
    a = %2.2f, b = %2.2f\n", *a, *b);
}
```

# Recursive Function

- A function that calls itself
- Useful for repetitive tasks

Simple factorial example:

```
factorial = 1;
for (counter = number; counter >= 1; counter --) {
    factorial = factorial * counter;
}
```

## Add GCC Options

- Go to the Explorer
- Open .vscode folder
- Open tasks.json
- Add `-lm` option to gcc command

```
args": [
    "-fdiagnostics-color=always",
    "-g",
    "${file}",
    "-o",
    "${fileDirname}/${fileBasenameNoExtension}",
    "-lm"
]
```

# Questions