Functions

Functions

- A function is like a class method, but without any class*
  - *so to speak
- A function receives arguments, and uses their values to calculate and return a new value or do a task
- A good program divides its work into smaller, self-contained tasks
  - Tasks may be divided into smaller subtasks
- Individual functions do each self-contained task

Example Program with Functions

- Program: Calculate a user's Body Mass Index (BMI) from their weight and height
- Tasks:
  - Input height as feet and inches, weight as pounds
  - Convert to meters, and kilograms
  - Calculate BMI
    - » trivial calculation
  - Output the result

Functions for Tasks

- Input task – handled by scanf(), or fgets() and atoi(), or similar predefined functions
- Output task – handled by printf()
- Conversion tasks – simple functions accept imperial units, return metric units
  - feet and inches → meters
  - pounds → kilograms
- BMI-calculation task – one-liner
  - but do it as a function anyway
  - Okay, the conversion functions could also be one-liners…

The BMI main() program

```c
/* BMI calculator example */
#include <stdio.h>

int main(int argc, char **argv)
{
    int feet;
    float inches;
    float pounds;

    printf("Enter height (ft, in) and weight (lbs): ");
    scanf(" %d %f %f", &feet, &inches, &pounds);

    float meters = fi2m(feet, inches);
    float kilos = lbs2k(pounds);

    float bmi = calc_bmi(meters, kilos);

    printf("BMI: %.1f\n", bmi);
    return 0;
}
//--------------------------------------------------------------
```

Making the BMI calculator

- Enter the main program, try to compile it
  - What does the compiler say?
- Functions must be written, and saved somewhere
  - in the same file as main()

    or
  - in one or more separate files
- The main() function needs to know what the functions need, and what they return
  - Use a **prototype**

Function Prototypes

- Define the function's calling "signature"
  - Function name
  - Number of, and types of, arguments
  - Type of returned value
- Must go in main()'s file, before main() itself
- Prototypes look like the first line of the function itself
- Functions that don't return a value:
  - Type *void*
- Functions that don't take any arguments:
  - Argument list is "type" *void*

The BMI main() program, with prototypes

```c
/* BMI calculator example */
#include <stdio.h>

float fi2m(int, double);
float lbs2k(float lb);
float calc_bmi(float m, float k);

int main(int argc, char **argv)
{
    int feet;
    float inches;
    float pounds;

    printf("Enter height (ft, in) and weight (lbs): ");
    scanf(" %d %f %f", &feet, &inches, &pounds);

    float meters = fi2m(feet, inches);
    float kilos = lbs2k(pounds);

    float bmi = calc_bmi(meters, kilos);

    printf("BMI: %.1f\n", bmi);
    return 0;
}
//------------------------------------------------------------
```

The Functions

```c
float fi2m(int f, double i)
{
    i += 12 * f;
    return 0.0254 * i;
}


float lbs2k(float lb)
{
    return lb / 2.205;
}


float calc_bmi(float m, float k)
{
    return k / (m*m);
}
```

Another simple function, without and with ***side effects***

- counting function, without side effects:

```c
unsigned numVowels(char *s)
{
   unsigned count = 0;
   unsigned i;
   for (i = 0 ; s[i] != '\0' ; i++) {
      switch (s[i]) {
         case 'a':
         case 'e':
         case 'i':
         case 'o':
         case 'u':
            count++;
      }
   }
   return count;
}
```

- counting function with side effects:

```c
unsigned numVowels(char *s)
{
   unsigned count = 0;
   unsigned i;
   for (i = 0 ; s[i] != '\0' ; i++) {
      switch (s[i]) {
         case 'a':
         case 'e':
         case 'i':
         case 'o':
         case 'u':
            count++;
            s[i] = 'X';   // side-effect!
      }
   }
   return count;
}
```

side effect changes the argument!

main() function calls numVowels() – try it out:

```
// use numVowels
// 2020-09-10
#include <stdio.h>

unsigned numVowels(char *s);

int main(int argc, char **argv)
{
    printf("program: %s\n", argv[0]);

    for (int i = 1; i < argc; i++) {
        unsigned n = numVowels( argv[i] );
        printf("%u vowels in %s\n", n, argv[i]);
    }
    return 0;
}
```

Some uses of argc and argv

One more simple function

- scanf() uses conversion specifiers like printf() does.
- The comments show an alternative way to get input, using fgets() and strtof().

```
1  /* F to C converter */
2  #include<stdio.h>
3  #include<stdlib.h>      // strtof()
4
5  float FtoC(float f)
6  {
7      return (f - 32) * 5.0 / 9.0;
8  }
9
10 int main(int argc, char **argv)
11 {
12     //char fstr[200];
13     float fahrenheit;
14     float celsius;
15
16     printf("Enter temperature in F: ");
17     scanf("%f", &fahrenheit);   // address of fahrenheit
18     //fgets( fstr, 200, stdin );
19     //fahrenheit = strtof(fstr, NULL);
20
21     celsius = FtoC( fahrenheit );
22     printf( "%f F equals %f C\n", fahrenheit, celsius );
23
24     return 0;
25 }
26
```

Function locations

- The functions can be in any order, but at least a *prototype* must appear before the first use.

```
1  /* F to C converter */
2  #include<stdio.h>
3  #include<stdlib.h>      // strtof()
4
5  float FtoC(float f);   // This is a prototype
6
7  int main(int argc, char **argv)
8  {
9     //char fstr[200];
10    float fahrenheit;
11    float celsius;
12
13    printf("Enter temperature in F: ");
14    scanf("%f", &fahrenheit);   // address of fahrenheit
15    //fgets( fstr, 200, stdin );
16    //fahrenheit = strtof(fstr, NULL);
17
18    celsius = FtoC( fahrenheit );
19    printf( "%f F equals %f C\n", fahrenheit, celsius );
20
21    return 0;
22 }
23
24 // This function could be in another file
25 float FtoC(float f)
26 {
27    return (f - 32) * 5.0 / 9.0;
28 }
```