AN OVERVIEW OF C, PART 3

CSE 130: Introduction to Programming in C
Stony Brook University
Recall that you can insert a text field width value into a `printf()` format specifier:

```c
printf("%5d", number);
```

For floating-point values (floats and doubles), you can also specify the number of digits to display before/after the decimal point:

```c
printf("%5.3f", average);
```
Constants

- A **constant** is a value that cannot change
- Ex. numeric literals (42, 23, 3.14)
- Variables can be declared as constants using the keyword `const`:
  
  ```
  const double pi = 3.1415926;
  ```
- Strings (sequences of characters enclosed in double quotes) are also constants.
MORE ELABORATE LOOPS

- Recall that every loop contains a test
  - As long as the test is true (has a nonzero value), the loop will continue to execute
- Tests don’t have to be simple Boolean comparisons
  - They can involve function calls...
RETURN VALUES REVISITED

- `printf()` and `scanf()` each return an integer value when they complete.
- `printf()` returns the number of characters printed, or a negative value if an error occurred.
- `scanf()` returns the number of successful conversions or the system-defined end-of-value.
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i;
    double x, min, max, sum, avg;

    if (scanf("%lf", &x) != 1) {
        printf("No data found - bye!\n");
        exit(1);
    }
min = max = sum = avg = x;

printf("%5s%9s%9s%9s%12s%12s\n", "Count", "Item", "Min", "Max", "Sum", "Average");

printf("%5s%9s%9s%9s%12s%12s\n\n", "-----", "----", "---", "---", "---", "-------");

printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n", 1, x, min, max, sum, avg);
for (i = 2; scanf("%lf", &x) == 1; i++)
{
    if (x < min)
        min = x;
    else if (x > max)
        max = x;
    sum += x;
    avg = sum / i;

    printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n", i, x, min, max, sum, avg);
}

return 0;
} /* end of main() */
FUNCTIONS
FUNCTIONS

- A **function** is a small block of code that can be called from another point in a program.

- Functions enable reuse, and can be used to abstract out common tasks.
  - Ex. computing the factorial of a number.

- Function results can be changed by supplying different input values.
CALLING A FUNCTION

- To call a function, write its name, followed by a pair of parentheses, followed by a semicolon

  Ex. `rand()`;

- If the function takes any input, those values go inside the parentheses

  Ex. `printf("%d", value);`
FUNCTION ARGUMENTS

- **Arguments** are pieces of data that are passed into a function.
- Different input can produce different results.
- Arguments can be manipulated, like variables.
- Arguments are normally passed as copies — changes are not sent back when the function returns.
RETURN VALUES

- Some functions pass a value back to the place where they were called

- Ex. `factorial()` sends back an answer

- The return value effectively replaces the function call in the original expression

  ```
  int answer = factorial(3);
  
  becomes
  
  int answer = 6;
  ```
RETURN VALUES

- If a function returns a value, it must contain a return statement:
  
  ```
  return value;
  ```

- The return value **must** match the return type in the function header!

- A function may return any value of the specified type
FUNCTION EXECUTION

- Only one function can be active at a time.

- When a function is called, the calling function is put on hold while the called function executes.

- When the called function completes (returns), control returns to the calling function.

- Function calls can be nested (e.g., A calls B, which calls C — when C completes, B resumes execution, then returns control to A when it’s done).
DEFINING A FUNCTION

- A function definition consists of a function header and a function body.
- The function header specifies the return type, name, and arguments list.
- The function body is a brace-enclosed set of 0 or more program statements.
GENERAL FORM

```
return_type function_name ( arguments )
{
    function body
}
```
NOTES ON DEFINING FUNCTIONS

- Like variables, functions must be defined before they can be used.
- Some programming conventions state that `main()` should come before any other functions in a program.
- How can `main()` use the function if it hasn’t been defined yet?
- Answer: Precede `main()` with one or more function prototypes.
FUNCTION PROTOTYPES

- A function prototype tells the compiler:
  - the number and types of arguments the function takes in
  - the type of value that the function returns

- General form:

  return-type function-name (parameter type list) ;

  e.g., double pow (double x, double y);
EXAMPLE 1

```c
void printDashedLine ()
{
    printf("---------------------");
}
```
EXAMPLE 2

```c
void clearScreen ()
{
    int i;
    for (i = 0; i < 24; i++)
    {
        printf("\n");
    }
}
```
EXAMPLE 3

```c
void printSomeStars (int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        printf("*\n");
    }
    printf("\n");
}
```
void print1ToN (int n)
{
    int i;
    for (i = 1; i <= n; i++)
        printf("%d\n", i);
}
EXAMPLE 5

```c
int getYear ()
{
    int value;
    printf("Enter the year: ");
    scanf("%d", &value);
    return value;
}
```
EXAMPLE 6

```c
int average (int a, int b, int c)
{
    int sum = a + b + c;
    return sum/3;
}
```
EXAMPLE 7

int multiply (int first, int second)
{
    return (first * second);
}
EXAMPLE 8

```c
int factorial (int value)
{
    int fac;
    for (fac = 1; value > 1; value--)
        fac = fac * value;
    return fac;
}
/* value is unchanged in the calling ftn */
```
VARIABLE SCOPE

- **Scope** refers to the area of a program for which a variable is defined.
- Scope is restricted to the smallest set of curly braces around the variable.
- Ex. the function in which a variable is defined.
int myFunction ()
{
    ...
    int x;
    ...
    /* x is in scope here */
}

/* x is out of scope here */
GLOBAL VARIABLES

- A *global variable* is declared outside of any function
- Global variables are accessible from anywhere in a program
- Global variables are used to share data
- Constants are usually declared as globals
GLOBAL VARIABLES

const float PI = 3.1415926;
int main (void)
{
    float area = PI * 2 * 2;
    ...
}

SCOPE AND NAMING

- Several variables can have the same name, as long as they are in different scopes
- The most recently-declared variable takes precedence
- We say that it *shadows* the other variable
SAME NAMES

```c
int x = 5; /* this x is global */

void foo ()
{
    int x = 10; /* this x shadows the global one */
    printf("%d", x); /* prints 10 */
}
```