Functions

CSC100 / Introduction to programming in C/C++

Marcus Birkenkrahe

April 25, 2023

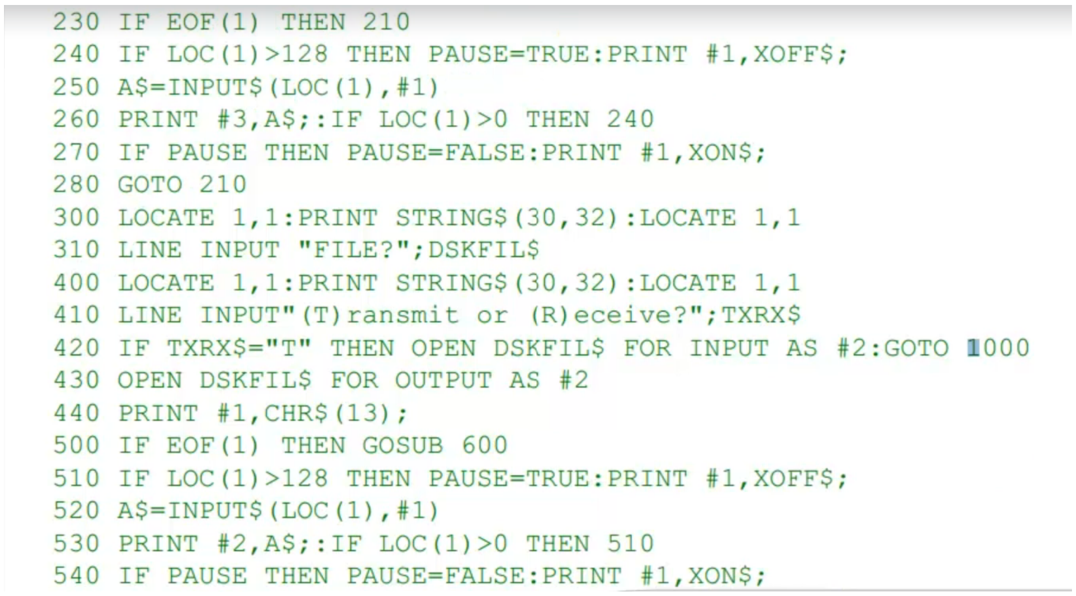
Contents

README

- This script introduces C functions.
- The PDF version of this file and of the completed practice workbook is available in GitHub.
- This section, including some sample code, is based on chapter 9 in King (2008).

Overview

- C functions do not always resemble math functions `f(x)`
- C functions don't need to have arguments (e.g. `main(void)`)
- C functions need not compute a value (e.g. `void hello()`)
- Each function is a small program with its own declarations and statements
- Functions allow us to
  1. **reuse** functions in other programs
  2. **recall** functions instead of duplicating code
  3. **modularize**, and easier understand and modify programs
- Once upon a time, programs didn't use to have functions!



```
230 IF EOF(1) THEN 210
240 IF LOC(1)>128 THEN PAUSE=TRUE:PRINT #1,XOFF$;
250 A$=INPUT$(LOC(1),#1)
260 PRINT #3,A$;:IF LOC(1)>0 THEN 240
270 IF PAUSE THEN PAUSE=FALSE:PRINT #1,XON$;
280 GOTO 210
300 LOCATE 1,1:PRINT STRING$(30,32):LOCATE 1,1
310 LINE INPUT "FILE?";DSKFIL$
400 LOCATE 1,1:PRINT STRING$(30,32):LOCATE 1,1
410 LINE INPUT"(T)ransmit or (R)eceive?";TXRX$
420 IF TXRX$="T" THEN OPEN DSKFIL$ FOR INPUT AS #2:GOTO 1000
430 OPEN DSKFIL$ FOR OUTPUT AS #2
440 PRINT #1,CHR$(13);
500 IF EOF(1) THEN GOSUB 600
510 IF LOC(1)>128 THEN PAUSE=TRUE:PRINT #1,XOFF$;
520 A$=INPUT$(LOC(1),#1)
530 PRINT #2,A$;:IF LOC(1)>0 THEN 510
540 IF PAUSE THEN PAUSE=FALSE:PRINT #1,XON$;
```

Figure 1: BASIC program snippet (Source: Collingbourne, 2022).

Example: `hello_world` function: mostly `void`

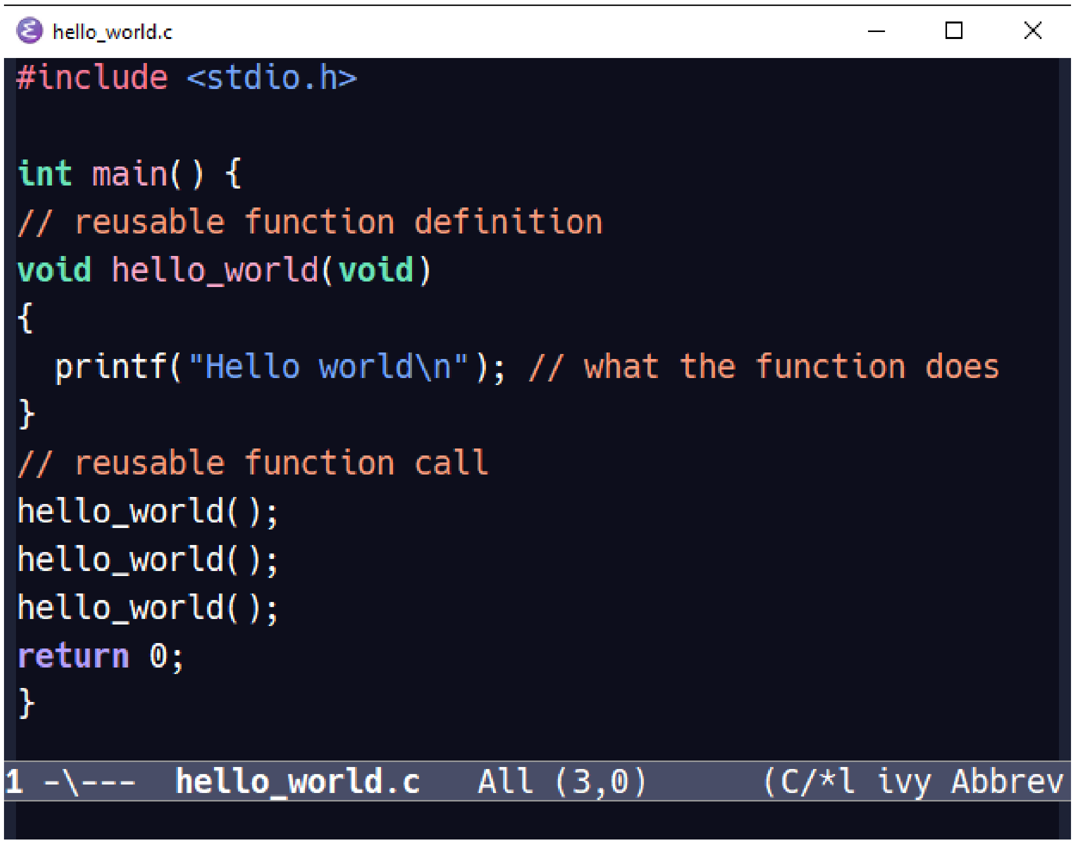
- Can you guess what the output of this code block will be?

  ```
  // reusable function definition
  void hello_world(void)
  {
    printf("Hello world\n"); // what the function does
  }
  // reusable function call
  hello_world();
  hello_world();
  hello_world();
  ```

  ```
  Hello world
  Hello world
  Hello world
  ```

- function is doubly `void`: no return value, no argument
- function code can be reused elsewhere

- function can be recalled at will
- Remember that the C compiler really sees this source file:
  1. `#include` header file for Input/output
  2. `main` function definition {...}
  3. `hello_world` function definition {...}
  4. three `hello_world` function calls



```
#include <stdio.h>

int main() {
// reusable function definition
void hello_world(void)
{
  printf("Hello world\n"); // what the function does
}
// reusable function call
hello_world();
hello_world();
hello_world();
return 0;
}
```

```
1 -\---  hello_world.c   All (3,0)      (C/*l ivy Abbrev
```

Functions are everywhere in C!

- How many functions do you see in this code block?

```
#include <stdio.h>
```

```
int main(void)
{
  const double E = 2.7182818;

  printf("%g\n", log(E));
  return 0;
}
```

```
1
```

Answer:

FUNCTION	DEFINITION	INPUT	OUTPUT
`main()`	main function	None (`void`)	`return 0`
`printf()`	printing function	Arithmetic	Formatted
`log()`	logarithmic function	Constant	Log of e

Example: computing averages

Function definition

- We want to compute the average of two `double` values, we can define a function to do it:

```
double average ( double a, double b)
{
  return (a + b) / 2.;
}
```

- Here, `double` is *return type* and *argument data type.*
- `a` and `b` are *function parameters* - their values are supplied when the function is called
- The *function body* is the executable part, enclosed in `{...}`
- What's being executed by the body of the function `average`?
    1. computing the average of two `double` numbers
    2. returning the result as a `double` number

Function calls

Overview

- To call a function, write the *function name* followed by a list of *function arguments*.
- The arguments are assigned to the function parameters.
- The argument can be any *expression*.

Simple call with numbers

```
// function definition
double average ( double a, double b)
{
  return (a + b) / 2.;
}

// function call - result assigned to avg
double avg = average(5.1, 8.9); // compute average of two numbers

// function call inside function
printf("Average of %g and %g: %g\n", 5.1, 8.9, avg);
```

```
Average of 5.1 and 8.9: 7
```

Call with expressions

- Functions can have expressions as arguments.

  ```
  // function definition
  double average ( double a, double b)
  {
    return (a + b) / 2.;
  }

  // declarations
  double x=5.1, y=8.9, avg2;

  // function call with expression
  avg2 = average(x/2., y/2.);
  ```

```
// function call inside function
printf("Average of %g/2 and %g/2: %g\n", x, y, avg2);

Average of 5.1/2 and 8.9/2: 3.5
```

Call by other functions

- Functions can be called by other functions.

```
// function definition
double average ( double a, double b)
{
  return (a + b) / 2.;
}

  // declarations
double x=5.1, y=8.9;

// function call inside function
printf("Average of %g and %g: %g\n", x, y, average(x,y));

Average of 5.1 and 8.9: 7
```

- What's happening in the last line exactly? Describe it!

    1. The `average` function is called with `x` and `y` as arguments.
    2. `average` executes its `return` statement, returning `(a+b)/2`.
    3. `printf` prints the value that `average` returns.
    4. The `return` value of `average` becomes an argument of `printf`.

What happens to function results?

- The value of `average` is not saved anywhere. It is printed and then discarded.
- If we had needed to keep the value, we'd have to capture it in a variable (like `avg` in **??**, and `avg2` in **??**).

Using a function in a program

- The following program reads three numbers and computes their averages, one pair at a time.

  Sample input:

  ```
  echo "3.5 9.6 10.2" > ./src/input
  cat ./src/input
  ```

  Sample output:

  ```
  : Enter three numbers: 3.5 9.6 10.2
  : Average of 3.5 and 9.6: 6.55
  : Average of 9.6 and 10.2: 9.9
  : Average of 3.5 and 10.2: 6.85
  ```

  Code:

  ```
  // function definition
  double average(double a,double b) {
    return (a+b)/2.;
  }

  int main (void)
  {
    float x, y, z;
    printf("Enter three numbers: ");
    scanf("%f%f%f", &x, &y, &z);  // input
    printf("%g %g %g\n", x, y, z); // input check

    // print averages
    printf("Average of %g and %g: %g\n", x, y, average(x,y));
    printf("Average of %g and %g: %g\n", y, z, average(y,z));
    printf("Average of %g and %g: %g\n", x, z, average(x,z));

    return 0;
  }
  ```

- Important: the definition of `average` needs to be put **before** `main` - otherwise the function needs to be declared.

Let's practice!

- The practice file is in GitHub. Remember to download the **raw** Org-mode file and open it in Emacs.

References

- Kernighan/Ritchie (1978). The C Programming Language (1st). Prentice Hall.
- King (2008). C Programming - A modern approach (2e). W A Norton.
- Orgmode.org (n.d.). 16 Working with Source Code [website]. URL: orgmode.org