Introduction To C Programming

Lesson 04

Functions, Arguments, Lifetime of Variables

Objectives

- Understand C functions, arguments, and return values
- Understand lifetime of variables inside functions and out

Functions

- C is designed to use many functions.
  - Each function should do one task
- Functions are the building blocks of programs
- Are self-contained units of code designed to accomplish some task
- Referred to as ‘subroutines’ or ‘procedures’ in other languages
- main() is a user written function; printf() is a standard library function

Functions

- When a function is called, control is transferred to the first executable statement in that function
- When the function returns, control is transferred to the first executable statement after the one that called the function

Example

```
#include <stdio.h>
int main() {
    printf(“I’m inside main().\n”);
    hello();
    hi_again();
    bye();
}
void hello() {
    printf(“Hi from hello() function\n”);
}
```

Example

```
void hi_again()
{
    printf(“Hi again, from hi_again()\n”);
}
void bye()
{
  printf(“bye from the bye() function\n”);
}
```

Output

```
I’m inside main().
Hi from the hello() function
Hi again, from hi_again()
Goodbye from the bye() function
```

Functions

- There are four kinds of function calls:
  1. No input and no output
  2. Input and no output
  3. Input and output
  4. No input but output

```
1. hello();
2. printf(“Hello C students\n”);
```

Functions

(Technically, printf() does return a value, which is the number of characters printed)

```
3. x = printf("Hello, world.\n");
```

Functions

```
/* example of function call returning a
** value to the caller */
#include <stdio.h>

int main() {
    int    x;
    x = printf(“Hello, world.\n”);
    printf(“printed char count %d“, x);
}
```

Output

```
Hello, world.
The number of characters printed was 14
```

Functions

```
x = printf(“Hello\n”) + printf(“there\n”);
```

```
4. status = init();
```

Returning a status value is a common practice.

Functions

- No function may be defined within another function.
- Function definitions may appear before main
  - Function return type declared just like a variable
- Format:

```
[<return type>] fname (argument list)
{
/* function statements */
}
```

Functions

- The required parts of every function are:
  - *name*
  - *open & close parenthesis*
  - *open and close curly braces*
- Invoke a function by giving its name and parenthesis (don’t need to say “call func()”):
  - *name();*
- Empty parentheses means that no parameters are passed to the function

Functions With Arguments

- Format:

```
name(type arg1, type arg2)
{
        /* body of function */
}
```

- Alternate format (old style - don't use for new code):

```
name(arg1, arg2)
type arg1;
type arg2;
{ ...
```

Function With Arguments

```
sum(int num1, int num2)
{
    printf("The sum is %d\n", num1 + num2);
}
main()
{
    . . .
    sum(2,3);
}
```

Functions With Return Values

- Format:

```
[return type] name(type arg1, type arg2) {
  /* body of function */
  return value
}
```

- The [return type] specifies the type of data value that is returned. If no return type is specified, the function returns an int
  - Good style - always explicitly type function return values.

Functions With Return Values

```
int three()
{
    return 3;
}

int sum(int num1, int num2)
{
    return num1 + num2;
}
```

Functions With Return Values

```
/* example of sum function */
#include <stdio.h>
int sum(int num1, int num2) {
    return num1 + num2;
}

int main() {
    int    x = 10, y = 15;
    printf("x + y = %d\n", sum(x,y));
}
```

Functions With Return Values

```
/* example of square function */
#include <stdio.h>
long int square(int);
int main() {
    int x = 2500;
    printf("%d2 is %ld\n", x, square(x));
}
long int square(int num1) {
    return num1 * num1;
}
```

Functions With No Return Value

- To specify that a function returns no value at all, use **`void`** as the return type:

```
void name(type arg1, type arg2)
```

- Functions that do not return a value are not required to have a **`return`** statement.

Functions With No Return Value

```
/* functions with no return value */
#include <stdio.h>
void fun1();
int main() {
    fun1();
}
void fun1() {
    printf("Hello from fun1\n");
    return; /* the return is optional */
}
```

Function Prototyping

- Prototypes allow function declaration before use
  - Similar to how variables are declared before use (not mandatory for functions, but improves code)
  - Includes name, return type, and parameter types
- A function that is called before it is declared will be assumed by C to return an int, and have any number of arguments.

Function Prototyping

- If the declaration of the function conflicts with this assumption, the compiler will generate errors.
- To prevent this the function must be prototyped first.

```
type name(type name, type name, …);
```

Function Prototyping

```
int mult(int num1, int num2)
{
    /* body of function */
}
```

- Prototype: **`int mult(int, int);`**
- A function prototype can be declared inside the function, or outside of the function before the call is made.

Use the standard library

- Common standard C documentation practice - provide prototype when defining use of library function - #include <math.h> to use
- double pow(double x, double y);
  - returns x^y
  - /* use */ double x = 3.0, y = 4.0, result;
  - result = pow(x, y); /* result is 81 */

Prototypes and the standard library

- double sqrt(double x);
  - returns square root of its argument
- Lesson: look to C standard library before writing your own function

Lifetime of Variables

- Arguments received by a function are received *by value*, not *by reference*. This means that the variables sent to a function are copies of the original variables.
  - Will explain details when we cover pointers
  - For now, note that changes made to variables in function are not seen by caller.
- When the program returns from the function, those parameter variables are unavailable just as any function local variables are unavailable.

```
/* example of function pass by value */
#include <stdio.h>
void fun1(int a) {
    a += 15;
}
int main() {
    int   x = 0;
    fun1(x);
    printf("x is: %d\n", x);
}
```

Lifetime of Variables

- Variables declared outside of any function are “global” variables, and exist for the life of the program.

```
void fun1(void);

int x;

int main()
{
    . . .
```

```
/* example of global variables */
#include <stdio.h>
int   x = 0;
int main() {
    int   y = 0;
    x += 15;
    y += 10;
    printf("x is: %d\n", x);
    printf("y is: %d\n", y);
}
```

File1.c

```
int x;
int main() {
    . . .
}
void fun1()
{
    . . .
}
```

File2.c

```
extern int x;
void fun2() {
    ++x;
    . . .
}
```

Static

- A variable can have global lifetime and local access by being declared static
- A static variable does not change its value from call to call of the function it exists in

```
void func1()
{
    static int x = 0;
    x += 5;
    . . .
}
```