The C Programming Language
Chapter 2

(material from Dr. Michael Lewis, William & Mary Computer Science)

Overview

- Functions
- Call by Value vs. Call by Reference
- Function Prototypes
- Returning Multiple Values
- The void Type
- Implicit Returns
- The static Storage Class

Python vs. C vs. C++ vs. Java

	Python	C	C++	Java
return a value from a function	`return`	same	same	same
type of functions not returning a value		`void`	same as C	same as C
value from implicit returns	`None`			

Functions

- similar to those in Python
- delimited with { } instead of indentation
- needs a return type and types of parameters

```
int square(int n)  /* Our first function. */
{
  return n * n;
}
```

Full Program

- C code in file named square.c

```
cat ch02/square.c
```

```
#include <stdio.h>

int square(int n)  /* Our first function. */
{
  return n * n;
}

int main(int argc, char **argv)
{
  int n = 42;
  printf("%d**2 = %d\n", n, square(n));

  return 0;
}
```

Full Program

- to run

```
gcc ch02/square.c
```

```
./a.out
```

```
42**2 = 1764
```

Return Type

- omitting the return type results in warning

```
cat -n ch02/untyped_fun.c
```

```
     1  foo(int n)  /* No type declared for foo(). */
     2  {
     3    return n * n;
     4  }
```

```
gcc -c ch02/untyped_fun.c
```

```
ch02/untyped_fun.c:1:1: warning: type specifier missing, defaults to 'int'; ISO C99 and later do not support implicit int [-Wim
plicit-int]
foo(int n)  /* No type declared for foo(). */
^
int
1 warning generated.
```

Parameter Type

- omitting a parameter type also results in warning

```
cat -n ch02/untyped_arg.c
```

```
     1  #include <stdio.h>
     2
     3  int bar(n)  /* No type declared for argument n. */
     4  {
     5    printf("%lu\n", sizeof(n));
     6      return n * n;
     7  }
     8
     9  int main(void)
    10  {
    11    printf("%d\n", bar(42.0));
    12
    13    return 0;
    14  }
```

```
clang -Weverything ch02/untyped_arg.c
```

```
warning: include location '/usr/local/include' is unsafe for cross-compilation [-Wpoison-system-directories]
ch02/untyped_arg.c:3:9: warning: parameter 'n' was not declared, defaults to 'int'; ISO C99 and later do not support implicit i
nt [-Wimplicit-int]
int bar(n)  /* No type declared for argument n. */
        ^
ch02/untyped_arg.c:3:5: warning: no previous prototype for function 'bar' [-Wmissing-prototypes]
int bar(n)  /* No type declared for argument n. */
    ^
ch02/untyped_arg.c:3:1: note: declare 'static' if the function is not intended to be used outside of this translation unit
int bar(n)  /* No type declared for argument n. */
^
static
ch02/untyped_arg.c:3:5: warning: a function definition without a prototype is deprecated in all versions of C and is not suppor
ted in C2x [-Wdeprecated-non-prototype]
int bar(n)  /* No type declared for argument n. */
    ^
4 warnings generated.
```

Swapping Values

- does the following Python function work?

```
def swap (x, y) :
    x = y
    y = x
```

- how about this one?

```
def swap (x, y) :
    tmp = x
    x = y
    y = tmp
```

Call by Value vs. Call by Reference

- **call by value**
  - function gets a copy of the variable's value
- **call by reference**
  - function gets a pointer to the variable or object
  - allows changes to be permanent
  - in Python, depends on mutability of object

Call by Value vs. Call by Reference

```
cat -n ch02/call_by.py
```

```
     1  # Illustration of call by value and call by reference in Python.
     2
     3  def foo(a, b):
     4      a = 54
     5      b[0] = 42
     6
     7  a = 42
     8  b = [1, 2, 3, 4]
     9
    10  print(f"before call to foo(): {a = }, {b = }")
    11
    12  # a will be passed as call by value (a copy of a), while
    13  # b will be passed as call by reference (b itself).
    14
    15  foo(a, b)
    16
    17  print(f"after call to foo():  {a = }, {b = }")
```

```
python3 ch02/call_by.py
```

```
before call to foo(): a = 42, b = [1, 2, 3, 4]
after call to foo():  a = 42, b = [42, 2, 3, 4]
```

Call by Value vs. Call by Reference

- in C, default is call by value; pointers used for call by reference

```
cat -n ch02/bad_swap.c
```

```
     1  #include <stdio.h>
     2
     3  void swap(int m, int n)
     4  {
     5      int temp;
     6
     7      temp = m;
     8      m = n;
     9      n = temp;
    10  }
    11
    12  int main(void)
    13  {
    14      int m = 42, n = 54;
    15
    16      printf("before the call to swap(): m = %d, n = %d\n", m, n);
    17
    18      swap(m, n);  /* Only copies of m, n are passed. */
    19
    20      printf("after the call to swap():  m = %d, n = %d\n", m, n);
    21
    22      return 0;
    23  }
```

```
gcc ch02/bad_swap.c
```

```
./a.out
```

```
before the call to swap(): m = 42, n = 54
after the call to swap():  m = 42, n = 54
```

Function Prototypes

- **like a function declaration or signature showing**
  - function return type
  - function name
  - function parameter types
- **used for detecting mismatches in calls to function**
- **often seen in header files**
  - e.g., **`stdio.h`** contains a function prototype for **`printf`**

Function Prototypes

```
cat -n ch02/proto1.c
```

```
     1  #include <stdio.h>
     2
     3  int foo(int n);  /* Function prototype. */
     4
     5  int main(int argc, char **argv)
     6  {
     7    printf("42**2 = %d\n", foo(42));
     8    return 0;
     9  }
    10
    11  int foo(int n)
    12  {
    13      if (n > 42) return n * n;
    14
    15      /* If n <= 42 we hit the end of the function and return nothing. */
    16  }
```

```
gcc ch02/proto1.c
```

```
ch02/proto1.c:16:1: warning: non-void function does not return a value in all control paths [-Wreturn-type]
}
^
1 warning generated.
```

Function Prototypes

- here, function prototype is in different file from where it is called

```
cat -n ch02/proto2a.c
```

```
     1  #include <stdio.h>
     2
     3  int foo(int n);  /* Function prototype. */
     4
     5  int main(int argc, char **argv)
     6  {
     7    printf("2 + 2 = %d\n", foo(2));
     8    return 0;
     9  }
```

```
cat -n ch02/proto2b.c
```

```
     1  int foo(float n)
     2  {
     3      return n + n;
     4  }
```

```
gcc ch02/proto2a.c ch02/proto2b.c
```

```
./a.out
```

```
2 + 2 = 0
```

Function Prototypes

- even better to place it in a header file, which is included wherever needed

```
cat -n ch02/proto3.h
```

```
     1  int foo(int n);  /* Function prototype. */
```

```
cat -n ch02/proto3a.c
```

```
     1  #include <stdio.h>
     2
     3  #include "proto3.h"
     4
     5  int main(int argc, char **argv)
     6  {
     7    printf("2 + 2 = %d\n", foo(2));
     8    return 0;
     9  }
```

```
cat -n ch02/proto3b.c
```

```
     1  #include "proto3.h"
     2
     3  int foo(int n)
     4  {
     5      return n + n;
     6  }
```

```
gcc ch02/proto3a.c ch02/proto3b.c
```

```
./a.out
```

```
2 + 2 = 4
```

Function Prototypes

- if not included in file where function is defined (implemented), the following may occur

```
cat -n ch02/proto4.h
```

```
     1  int foo(int n);  /* Function prototype. */
```

```
cat -n ch02/proto4a.c
```

```
     1  #include <stdio.h>
     2
     3  #include "proto3.h"
     4
     5  int main(int argc, char **argv)
     6  {
     7    printf("2 + 2 = %d\n", foo(2));
     8    return 0;
     9  }
```

```
cat -n ch02/proto4b.c
```

```
     1  /* This function does not match the prototype used in proto4a.c !! */
     2
     3  double foo(double n)
     4  {
     5      return n + n;
     6  }
```

```
gcc ch02/proto4a.c ch02/proto4b.c
```

```
./a.out
```

```
2 + 2 = 2
```

Function Prototypes

- including the header file exposes the error

```
cat -n ch02/proto5.h
```

```
     1  int foo(int n);  /* Function prototype. */
```

```
cat -n ch02/proto5a.c
```

```
     1  #include <stdio.h>
     2
     3  #include "proto5.h"
     4
     5  int main(int argc, char **argv)
     6  {
     7    printf("2 + 2 = %d\n", foo(2));
     8    return 0;
     9  }
```

```
cat -n ch02/proto5b.c
```

```
     1  #include "proto5.h"
     2
     3  double foo(double n)
     4  {
     5      return n + n;
     6  }
```

```
gcc ch02/proto5a.c ch02/proto5b.c
```

```
ch02/proto5b.c:3:8: error: conflicting types for 'foo'
double foo(double n)
       ^
ch02/proto5.h:1:5: note: previous declaration is here
int foo(int n);  /* Function prototype. */
    ^
1 error generated.
```

Returning Multiple Values

- **in both Python and C, each function can return only one data item**
- **this item, however, can be a collection**
  - in Python, for instance, this collection can be a tuple
  - in C, for instance, this collection can be a struct
- **so, there are workarounds to return multiple data values**

The void Type

- if a function does not return a value, its return type is void

```
cat -n ch02/void1.c
```

```
     1  #include <stdio.h>
     2
     3  void foo(int n)
     4  {
     5      printf("n**2 = %d\n", n * n);
     6  }
```

- also use void to indicate a function has no input arguments

```
cat -n ch02/void2.c
```

```
     1  #include <stdio.h>
     2
     3  void hello_world(void)
     4  {
     5      printf("hello, world!\n");
     6  }
```

The void Type

- the compiler should detect misuse with void

```
cat -n ch02/void3.c
```

```
     1  #include <stdio.h>
     2
     3  void foo()
     4  {
     5      printf("hello, world!\n");
     6  }
     7
     8  void bar(float x)
     9  {
    10          float z = foo();
    11      float y = foo(x);
    12  }
```

```
gcc -c ch02/void3.c
```

```
ch02/void3.c:10:8: error: initializing 'float' with an expression of incompatible type 'void'
        float z = foo();
              ^   ~~~~~
ch02/void3.c:11:20: warning: too many arguments in call to 'foo'
    float y = foo(x);
              ~~~  ^
ch02/void3.c:11:18: warning: passing arguments to 'foo' without a prototype is deprecated in all versions of C and is not suppo
rted in C2x [-Wdeprecated-non-prototype]
    float y = foo(x);
                  ^
ch02/void3.c:11:11: error: initializing 'float' with an expression of incompatible type 'void'
    float y = foo(x);
          ^   ~~~~~~
2 warnings and 2 errors generated.
```

The static Storage Class

- **static variables are persistent**
- **two types**
  - outside any function: like a global variable, but only to this source file
  - inside a function: retain their value from call to call

The static Storage Class

```
cat -n ch02/static.c
```

```
 1  #include <stdio.h>
 2
 3  static int n = 42;  /* Initialized to 42 at compile time. */
 4
 5  void foo(void)
 6  {
 7    printf("static int n: %d\n\n", n);
 8  }
 9
10  void bar(int a)
11  {
12    static int m = 0;  /* Initialized to 0 at compile time. */
13
14    if (m == 0) {
15      m = a;
16    }
17    printf("static int m: %d (local to function)\n", m);
18    printf("static int n: %d (global to file)\n\n", n);
19  }
20
21  int main(void)
22  {
23    printf("static int n: %d (global to file)\n\n", n);
24
25    foo();
26
27    bar(42);
28    bar(54);
29    bar(54);
30
31    return 0;
32  }
```

The static Storage Class

```
./a.out
```

```
static int n: 42 (global to file)

static int n: 42

static int m: 42 (local to function)
static int n: 42 (global to file)

static int m: 42 (local to function)
static int n: 42 (global to file)

static int m: 42 (local to function)
static int n: 42 (global to file)
```