CC-112 Programming Fundamentals

C Functions

Nazar Khan
Department of Computer Science
University of the Punjab

Modularizing Programs in C

- The best way to develop and maintain a large program is to divide it into several smaller pieces, each more manageable than the original program.
- A function is invoked by a function call.
- The function call specifies the function by name and provides information (as arguments) that the called function needs to perform its task.
- The purpose of *information hiding* is to give functions access only to the information they need to complete their tasks.
- This is a means of implementing the *principle of least privilege*, one of the most important principles of good software engineering.

Functions

- A *local variable* is known only in a function definition.
- Other functions are not allowed to know the names of a function's local variables, nor is any function allowed to know the im- plementation details of any other function.

The *square* Function

The *maximum* Function

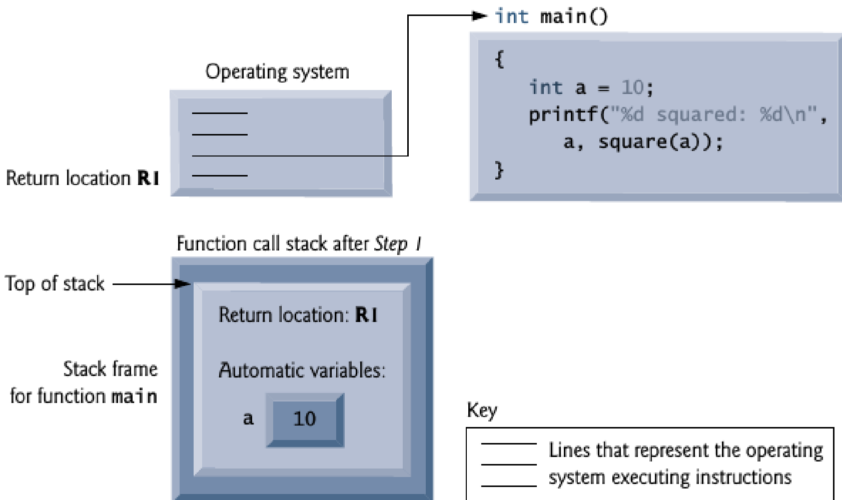
Function Prototypes: A Deeper Look

- A function prototype declares the function's name, return type and declares the number, types, and order of the parameters the function expects to receive.
- Function prototypes enable the compiler to verify that functions are called correctly.
- The compiler ignores variable names mentioned in the function prototype.
- Arguments in a *mixed-type expression* are converted to the same type via the C standard's usual arithmetic conversion rules.

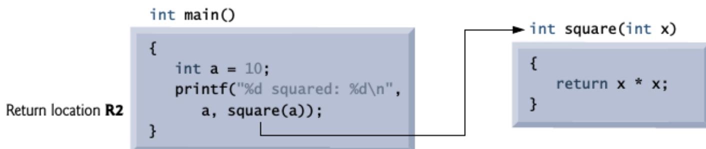
Function Call Stack and Stack Frames

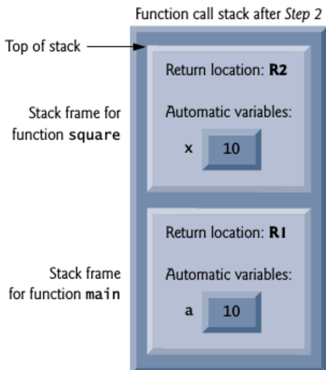
- Stacks are known as last-in, first-out (LIFO) data structures – the last item pushed (inserted) on the stack is the first item popped (removed) from the stack.
- A called function must know how to return to its caller.
- So return address of calling function is pushed onto the *program execution stack* when the function is called.
- If a series of function calls occurs, successive return addresses are pushed onto the stack in last-in, first-out order.
- So the last function to execute will be the first to return to its caller.
- The program execution stack contains the memory for the local variables used in each invocation of a function during a program's execution.
- This data is known as the *stack frame* of the function call.
- When a function call is made, the stack frame for that function call is pushed onto the program execution stack.

Function Call Stack and Stack Frames



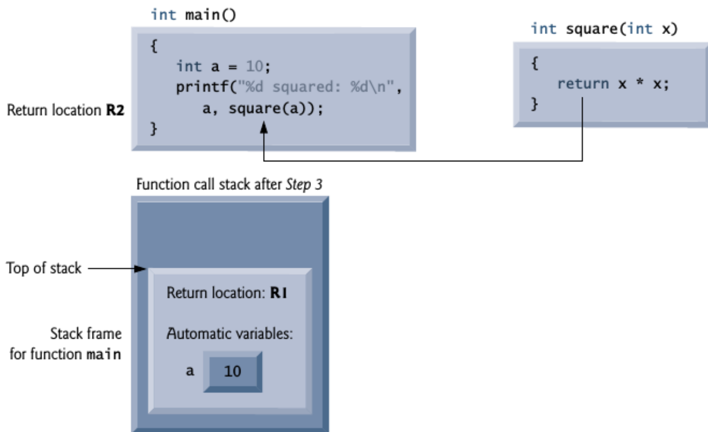
Function Call Stack and Stack Frames





Function Call Stack and Stack Frames

- When the function returns to its caller, the stack frame for this function call is popped off the stack and those local variables are no longer known to the program.



Function Call Stack and Stack Frames

- The amount of memory in a computer is finite.
- So only a certain amount of memory can be used to store stack frames on the program execution stack.
- If there are more function calls than can have their stack frames stored on the program execution stack, an error known as a *stack overflow* occurs.
- The application will compile correctly, but its execution will fail with a stack overflow.

Headers

- Each standard library has a corresponding header containing
    1. function prototypes for all of that library's functions, and
    2. definitions of various symbolic constants needed by those functions.
- You can create and include your own headers.

Some C Standard Library Files

- `<assert.h>` contains information for adding diagnostics that aid program debugging.
- `<ctype.h>` contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.
- `<errno.h>` defines macros that are useful for reporting error conditions.
- `<float.h>` contains the floating-point size limits of the system.
- `<limits.h>` contains the integral size limits of the system.
- `<locale.h>` contains function prototypes and other information that enables a program to be modified for the current locale on which it's running. The notion of locale enables the computer system to handle different conventions for expressing data such as dates, times, currency amounts and large numbers throughout the world.

Some C Standard Library Files

- `<math.h>` contains function prototypes for math library functions.
- `<setjmp.h>` contains function prototypes for functions that allow bypassing of the usual function call and return sequence.
- `<signal.h>` contains function prototypes and macros to handle various conditions that may arise during program execution.
- `<stdarg.h>` defines macros for dealing with a list of arguments to a function whose number and types are unknown.
- `<stddef.h>` contains common type definitions used by C for performing calculations.
- `<stdio.h>` contains function prototypes for the standard input/output library functions, and information used by them.
- `<stdlib.h>` contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers and other utility functions.

Some C Standard Library Files

- `<string.h>` contains function prototypes for string-processing functions.
- `<time.h>` contains function prototypes and types for manipulating the time and date.

Passing Arguments By Value and By Reference

- When an argument is *passed by value*, a copy of its value is made and passed to the called function.
- Changes to the copy in the called function do not affect the original variable's value.
- When an argument is *passed by reference*, the caller allows the called function to modify the original variable's value.
- All calls in C are call-by-value.
- It's possible to achieve call-by-reference by using address operators and indirection operators.

```
scanf( "%d" ,  &num  );
```

"%d" — by value; &num — by reference