

Chapter 14 Functions

Original slides from Gregory Byrd, North Carolina State University

Modified slides by Chris Wilcox,
Colorado State University

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Function

- **Smaller, simpler, subcomponent of program**
- **Provides abstraction**
 - hide low-level details, give high-level structure
 - easier to understand overall program flow
 - enables separable, independent development
- **C functions**
 - zero or multiple arguments passed in
 - single result returned (optional)
 - return value is always a particular type
- In other languages, called procedures, subroutines, ...

CS270 - Fall 2014 - Colorado State University

2

Example of High-Level Structure

```
main()
{
    SetupBoard(); /* place pieces on board */
    DetermineSides(); /* choose black/white */

    /* Play game */
    do {
        WhitesTurn();
        BlacksTurn();
    } while (NoOutcomeYet());
}
```

Structure of program
is evident, even without
knowing implementation.

CS270 - Fall 2014 - Colorado State University

3

Functions in C

- **Declaration** (also called prototype)

```
int Factorial(int n);
```

type of
return value

name of
function

types of all
arguments

- **Function call** -- used in expression

```
a = x + Factorial(f + g);
```

1. evaluate arguments

2. execute function

3. use return value in expression

CS270 - Fall 2014 - Colorado State University

4

Function Definition

- State type, name, types of arguments

- must match function declaration
- give name to each argument (doesn't have to match declaration)

```
int Factorial(int n)
{
    int i;
    int result = 1;
    for (i = 1; i <= n; i++)
        result *= i;
    return result; ←
}
```

gives control back to calling function and returns value

Why Declaration?

- Since function definition also includes return and argument types, why is declaration needed?
- Use might be seen before definition.**
Compiler needs to know return and arg types and number of arguments.
- Definition might be in a different file, written by a different programmer.**
 - include a "header" file with function declarations only
 - compile separately, link together to make executable

Example

```
double ValueInDollars(double amount, double rate);
main() ←
{
    ...
    dollars = ValueInDollars(francs,
        DOLLARS_PER_FRANC);
    printf("%f francs equals %f dollars.\n",
        francs, dollars);
    ...
}
double ValueInDollars(double amount, double rate)
{
    return amount * rate;
}
```

function declaration (prototype)

function call (invocation)

function definition (code)

Implementing Functions: Overview

- Activation record (stack frame)
 - information about each function, including arguments and local variables
 - stored on run-time stack

Calling function

push new activation record
copy values into arguments
call function
get result from stack

execute code
put result in activation record
pop activation record from stack
return

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Run-Time Stack

- Recall that local variables are stored on the run-time stack in an **activation record**
- Stack Pointer (R6)** is a pointer to the next free location in the stack, and is used to push and pop values on and off the stack.
- Frame pointer (R5)** is a pointer to the beginning of a region of the activation record that stores local variables for the current function
- When a new function is **called**, its activation record is **pushed** on the stack; when it **returns**, its activation record is **popped** off the stack.

CS270 - Fall 2014 - Colorado State University 9

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Run-Time Stack

The diagram illustrates the state of memory during a function call across three stages:

- Before call:** Shows a vertical stack of memory cells. The bottom section is labeled "main". Above it is a section labeled "locals" containing "y", "x", and "w". The stack pointer R6 points to the top of the "locals" section, and the frame pointer R5 points to the start of the "locals" section.
- During call:** Shows the stack after the function has been called. The "locals" section now contains "Watt" instead of "y", "x", and "w". The stack pointer R6 has moved to point to the top of the new "locals" section. The frame pointer R5 still points to the start of the original "locals" section.
- After call:** Shows the stack after the function has returned. The "locals" section has been popped back to its original state, containing "y", "x", and "w". The stack pointer R6 points to the top of this restored section, and the frame pointer R5 still points to its original position.

CS270 - Fall 2014 - Colorado State University 10

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Activation Record

```
• int NoName(int a, int b)
{
    int w, x, y;
    .
    .
    .
    return y;
}
```

Name	Type	Offset	Scope
a	int	4	NoName
b	int	5	NoName
w	int	0	NoName
x	int	-1	NoName
y	int	-2	NoName

A vertical stack diagram shows the activation record structure. It includes sections for "locals" (y, x, w) and "args" (a, b). The stack pointer R5 is shown pointing to the top of the "locals" section. The label "bookkeeping" is placed near the bottom of the stack. A legend on the right identifies the colors: red for "locals" and green for "args".

CS270 - Fall 2014 - Colorado State University 11

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Activation Record Bookkeeping

- Return value**
 - space for value returned by function
 - allocated even if function does not return a value
- Return address**
 - save pointer to next instruction in calling function
 - convenient location to store R7 in case another function (JSR) is called
- Dynamic link**
 - caller's frame pointer
 - used to pop this activation record from stack

CS270 - Fall 2014 - Colorado State University 12

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Example Function Call

```

• int Volta(int q, int r)
{
    int k;
    int m;
    ...
    return k;
}

int Watt(int a)
{
    int w;
    ...
    w = Volta(w,10);
    ...
    return w;
}

```

CS270 - Fall 2014 - Colorado State University 13

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Calling the Function

- w = Volta(w, 10);
- ; push second arg

AND R0, R0, #0	
ADD R0, R0, #10	new R6 → 25
ADD R6, R6, #-1	R6 → 10
STR R0, R6, #0	R5 → 25
- ; push first argument

LDR R0, R5, #0	
ADD R6, R6, #-1	
STR R0, R6, #0	
- ; call subroutine

JSR Volta	xFD00
-----------	-------

Note: Caller needs to know number and type of arguments, doesn't know about local variables.

CS270 - Fall 2014 - Colorado State University 14

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Starting the Callee Function

- ; leave space for return value

ADD R6, R6, #-1	
new R6 →	m
new R5 →	k
- ; push return address

ADD R6, R6, #-1	xFCFB
STR R7, R6, #0	x3100
R6 →	25
R5 →	10
- ; push caller's frame ptr

STR R7, R6, #0	25
ADD R6, R6, #-1	10
STR R5, R6, #0	25
R6 →	25
R5 →	25
- ; set new frame pointer

ADD R5, R6, #-1	
STR R5, R6, #0	
R6 →	
- ; allocate space for locals

ADD R6, R6, #-2	xFD00
-----------------	-------

CS270 - Fall 2014 - Colorado State University 15

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Ending the Callee Function

- return k;
- ; copy k into return value

LDR R0, R5, #0	R6 → -43
STR R0, R5, #3	R5 → 217
- ; pop local variables

ADD R6, R5, #1	xFCFB
; pop dynamic link (into R5)	x3100
LDR R5, R6, #0	217
ADD R6, R6, #1	25
; pop return addr (into R7)	10
LDR R7, R6, #0	25
ADD R6, R6, #1	25
; return control to caller	dyn link
RET	ret addr
	ret val
	a

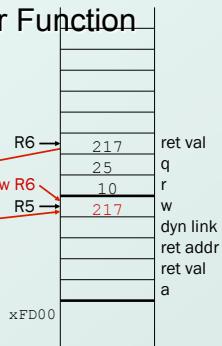
CS270 - Fall 2014 - Colorado State University 16

Resuming the Caller Function

```

• w = Volta(w,10);
• JSR Volta
; load return value
; from top of stack
LDR R0, R6, #0
; perform assignment
STR R0, R5, #0
; pop return value
ADD R6, R6, #1
; pop arguments
ADD R6, R6, #2

```



Summary of LC-3 Function Call Implementation

1. **Caller** pushes arguments (last to first).
2. **Caller** invokes subroutine (JSR).
3. **Callee** allocates return value, pushes R7 and R5.
4. **Callee** allocates space for local variables.
5. **Callee** executes function code.
6. **Callee** stores result into return value slot.
7. **Callee** pops local vars, pops R5, pops R7.
8. **Callee** returns (JMP R7).
9. **Caller** loads return value and pops arguments.
10. **Caller** resumes computation...