Day09 A

Young W. Lim

2017-10-07 Sat

Outline

1. Based on

2. Functions (3) - Recursion
   - Function Prototypes
   - Stack Frames
   - Recursion

Based on

"C How to Program",
Paul Deitel and Harvey Deitel

I, the copyright holder of this work, hereby publish it under the following licenses: GNU head Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

CC BY SA This file is licensed under the Creative Commons Attribution ShareAlike 3.0 Unported License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license compatible with this one.

Function Prototypes

- declares the function's return type
- declares the parameter's number, type, and order

- enable the compiler to verify that the function call is valid
- the compiler ignore the exact variable names of the function prototype

Unresolved References

- indicates to the compiler that the specified function is defined
  - either later in the same file
  - or in a different file
- separate compilation and linking
- the compiler does not attemp to resolve references to such functions
- the linker will resolve *unresolved* references
- if the linker cannot locate a proper function definition,
  the linker issues an error message

Stack

- a stack of dishes
- LIFO (last in first out) data structure
  - the last item pushed on the stack
  - the first item popped from the stack

Function Calls and Returns

- a called function knows how to return to the caller
  - the return address is pushed onto the program execution stack

1. main() calls func1() → push func1's return address
2. func1() calls func2() → push func2's return address
3. func2() calls func3() → push func3's return address

1. func3() returns to func2() → pop func3's return address
2. func2() returns to func1() → pop func2's return address
3. func1() returns to main() → pop func1's return address

Program Execution Stack

- the program execution stack also contains
  the local variables for each invocation of a function
- one stack frame of a function call

- when a function call is made,
  the *stack frame* of that function call is
  pushed onto the *program execution stack*
- when a function return is made,
  the *stack frame* of that function call is
  popped off the *program execution stack*
  - the local variable of that invocation exist no longer

Program Execution Stack

- the size of memory is finite
- only a certain amount of memory can be used

- stack overflow error
  - when there are more function calls
    than can be their stack frames stored on the program execution stack

Recursive Function Call

- function that calls itself either directrly or undirectly
- the base case
  the recursive function simply returns a result
- complex cases
  the recursive function divides the complex problems
  into two smaller problems
  the base problem + a slightly smaller problem
  - viewing this smalller problem as the new given problem
    the procedure recursively applied

Recursive Function Return

- for recursion to terminate,
  each time the recursive function calls the slighty smaller problem
  the sequence of smaller and smaller problems must converge on the base case
- when the function recognizes the base case,
  the result is returned to the previous function call,
  and the combined result is returned to its previous function call
- the sequence of returns ensues all the way up to the original call
  and returns the final result

Recursive Function Calls and Returns

- a called function knows how to return to the caller
  - the return address is pushed onto the program execution stack

1. main() calls func() → push func's 1st return address
2. func() calls func() → push func's 2nd return address
3. func() calls func() → push func's 3rd return address

1. func() returns to func() → pop func's 3rd return address
2. func() returns to func() → pop func's 2nd return address
3. func() returns to main() → pop func's 1st return address