COMP124

Computer Systems

08 | Recursion | String Processing

Dr Stuart Thomason

Recursion

- A recursive subroutine is one that calls itself to perform some subsidiary task
  - Can make programming very efficient (compact code)
  - But can also be hard to understand
- More complex code might have mutual recursion, where two (or more) subroutines call each other recursively
  - First, sub1 calls sub2
  - Then, sub2 calls sub1
  - And so on…
- Very often, the recursive subroutine does most of its work as it returns up through the nested calls (tree structure)

Terminating Case

- There are two parts to a recursive subroutine
  - The general case (ie. the bit that calls itself)
  - The terminating case (ie. the bit that causes it to stop)

- You must have a terminating case otherwise the recursion will carry on forever

- It can be really hard to get your head around recursion, but it helps if you consider what is happening at some arbitrary level

- In other words, think about how the general case will work at some arbitrary point in the execution

Fibonacci Sequence

- The Fibonacci sequence is a series of numbers where each number is the sum of the previous two numbers
  1 1 2 3 5 8 13 21 34 55 89 144

- Thinking of this in terms of recursion…

  - The general case is:
    fib(N) = fib(N-1) + fib(N-2)

  - The terminating case applies when N is either 1 or 2:
    fib(1) = 1
    fib(2) = 1

Fibonacci Sequence – Java Method

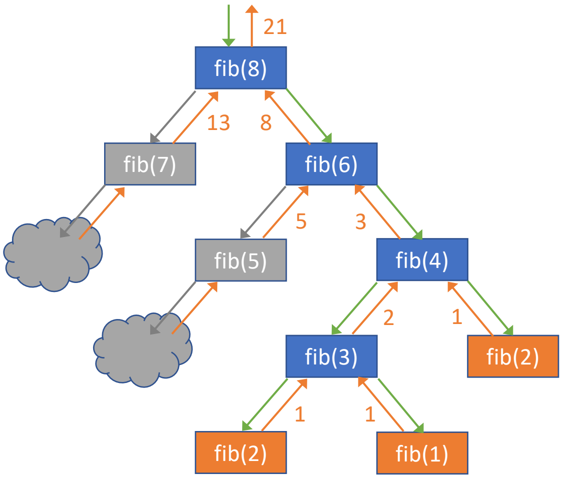
- Find the Nth number in the sequence recursively using Java (or any high level language)

```
int fib(int n) {
    if(n == 1 || n == 2) {
        // Terminating case
        return 1;
    } else {
        // General case
        return fib(n-1) + fib(n-2);
    }
}
```

- To get the 8th number in the sequence you call fib(8)
  (which should return 21)

Nested Calls (Tree Structure)

- To work out fib(8) we need to first work out fib(6) and fib(7)
- But to work out fib(6) we need to first work out fib(4) and fib(5)
- And so on…



Factorial

- The factorial of a number is the product of all the numbers below it (down to 1)

- Example: factorial(4) = 4 x 3 x 2 x 1

- Thinking of this in terms of recursion…

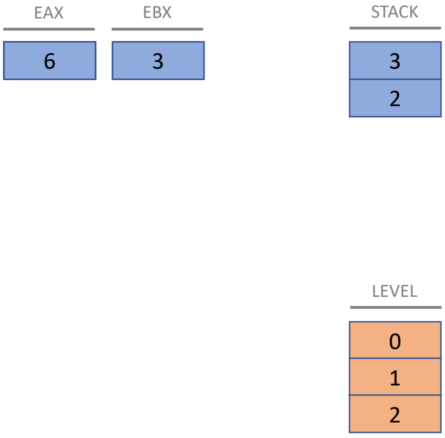
  - The general case is:
    factorial(N) = N x factorial(N-1)

  - The terminating case applies when N is 1:
    factorial(1) = 1

Factorial Subroutine – Assembly Code

- Accumulator stores initial parameter (replaced with result)

```
factorial PROC
    cmp eax, 2
    jl termcase
    push eax
    dec eax
    call factorial
    pop ebx
    mul ebx
    ret
termcase:
    mov eax, 1
    ret
factorial ENDP
```



EAX

6

EBX

3

STACK

3

2

LEVEL

0

1

2

Using Iteration Instead of Recursion

- It is always possible to make an iterative (ie. non-recursive) version of a recursive algorithm

- Sometimes the iterative version will be a lot simpler, usually when the recursive version is tail-recursive

- But many problems are easier to conceptualise, understand and solve with recursive algorithms

- Recursion implicitly uses the stack (of nested calls and frames) as a data structure to simplify the algorithm

- However, stack frames take up memory, which can be wasteful

Non-Recursive Factorial – Assembly Code

- The iterative factorial algorithm is incredibly simple
  (assuming num contains the number we want to calculate)

```
mov eax, 1
mov ecx, num
jecxz finish
floop:
    mul ecx
    loop floop
finish:
```

- The iterative version of an algorithm will use loops and variables instead of nested calls and the stack

Palindrome Checker

- A palindrome is a string that reads the same in both directions
  rotator

- Thinking of this in terms of recursion…

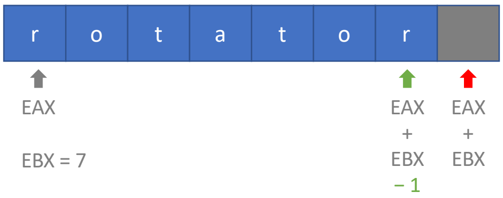
  - The general case is:
    a) letter at each end matches, and
    b) letters in between are also a palindrome

  - The terminating case applies when:
    a) string only has one letter, or
    b) string only has two letters which are the same

Palindrome Checker – Design Decisions

- Can we create a recursive palindrome checker using assembly language?
  - Parameters will be on stack (string address and length)
  - Return 0 (false) or 1 (true) in the accumulator

- Pop length into ebx and address into eax
- We need to get the memory addresses of chars at each end
  - Know where the string starts and how long it is
  - Do some maths to find the address of the end character



r o t a t o r

EAX

EBX = 7

EAX + EBX − 1

EAX + EBX

Palindrome Checker – Design Decisions

- Compare the values at those memory addresses and jump according to the result

- If the characters don’t match, store 0 in eax and return

- If they do match, we need to set up a recursive call
  - Increase eax so it points to the next character along
  - Decrease ebx (by two) so the length is reduced
  - Push those values to the stack
  - Make the recursive call
  - Return the result

- We also need to check for the terminating case
  - Store 1 in eax and return if there is only one char (or less)

Dealing with Strings in Assembly Code

- We create strings as character arrays in the C part of the code

```
char msg[] = “Hello”;
```

72	101	108	108	111	0
2048	2049	2050	2051	2052	2053

- Each character takes up one byte of memory that stores its ASCII value (see lookuptables.com/text/ascii-table)

- Always uses one extra byte to store the string terminator (NUL)

- We can load the address of the start of the string into a register

```
lea eax, msg
```

Accessing a Single Character in Memory

- Each register in a 32-bit system stores 4 bytes of memory

- Moving content of string address into a register will take the next four bytes and treat it as an integer (incorrect)
  `mov edx, [eax]`

EDX | 72 | 101 | 108 | 108 | = 01001000011001010110110001101100 = 1214606444

- To get the single byte (character value) instead…
  `movzx edx, byte ptr [eax]`

EDX | 0 | 0 | 0 | 72 | = 00000000000000000000000001001000 = 72

- Treats address as pointer to a single byte and zero fills register

Palindrome Checker – Assembly Code

- Incoming parameters on stack, returning result in accumulator

```
palindrome PROC
    pop ebx
    pop eax
    cmp ebx, 1
    jle single
    dec ebx
    add ebx, eax
    movzx ecx, byte ptr [eax]
    movzx edx, byte ptr [ebx]
    cmp ecx, edx
    jne failure
    inc eax
    dec ebx
    push eax
    push ebx
    call palindrome
    ret
single:
    mov eax, 1
    ret
failure:
    mov eax, 0
    ret
palindrome ENDP
```

Length of String – C Library Call (strlen)

- The C library has a function that returns the length of a string
- From assembly code
  - Push address of string to the stack
  - Call the strlen external subroutine
  - Result will be in the accumulator
  - Clean up the stack
- Need to include the C string library code in your program

```
#include <string.h>
```

- Most C library routines can be called in this way provided you include the correct C header (.h) at the top of the code
- Be sure not to step into (F11) library routines when you are using the Visual Studio debugger (step over them with F10)