Meng Wai Chan
Apr 30, 2023

TAKE HOME TEST 1

Meng Wai Chan
Professor Gertner
CSc 343
April 30, 2023

Meng Wai Chan
Apr 30, 2023

Table of Contents

I. Objective.....2

II. Description of Specifications and Functionality..... 3

Factorial - Windows Visual Studio..... 3

Factorial - GDB.....5

Factorial - MIPS.....7

Factorial - Time Analysis..... 8

GCD - Windows Visual Studio.....9

GCD - GDB..... 10

GCD - MIPS..... 12

III. Conclusion.....13

Meng Wai Chan
Apr 30, 2023

I. Objective

The objective of this take home test is to understand and demonstrate how recursive function calls allocate memory, and how recursive functions are executed. Also understanding how control and parameters are transferred from one level of recursion to the next level.

II. Description of Specifications and Functionality

Factorial - Windows Visual Studio

```
int factorial(int N)
{
    if (N == 1)
        return 1;
    return (N*factorial(N - 1));
}
void main()
{
    int N_fact = factorial(5);
}
```

```
Registers
EAX = 0024C000
EBX = 00491000
ECX = 0024C000
EDX = 00000001
ESI = 00241325
EDI = 006FF5CC
EIP = 00241718
ESP = 006FF500
EBP = 006FF5CC
EFL = 00000246

0x006FF5D4 = 00000001
```

```
int factorial(int N)
{
002416F0 55                    push        ebp
002416F1 8B EC                 mov         ebp,esp
002416F3 81 EC C0 00 00 00     sub         esp,0C0h
002416F9 53                    push        ebx
002416FA 56                    push        esi
002416FB 57                    push        edi
002416FC 8D BD 40 FF FF FF     lea         edi,[ebp-0C0h]
00241702 B9 30 00 00 00        mov         ecx,30h
00241707 B8 CC CC CC CC        mov         eax,0CCCCCCCCh
0024170C F3 AB                 rep stos    dword ptr es:[edi]
0024170E B9 00 C0 24 00        mov         ecx,offset _3D704958_factorial@cpp (024C000h)
00241713 E8 EB FA FF FF        call        @__CheckForDebuggerJustMyCode@4 (0241203h)
    if (N == 1)
00241718 83 7D 08 01           cmp         dword ptr [N],1
0024171C 75 07                 jne         factorial+35h (0241725h)
        return 1;
0024171E B8 01 00 00 00        mov         eax,1
00241723 EB 13                 jmp         factorial+48h (0241738h)
    return (N*factorial(N - 1));
00241725 8B 45 08              mov         eax,dword ptr [N]
00241728 83 E8 01              sub         eax,1
0024172B 50                    push        eax
0024172C E8 72 FB FF FF        call        factorial (02412A3h)
00241731 83 C4 04              add         esp,4
00241734 0F AF 45 08           imul        eax,dword ptr [N]
}
00241738 5F                    pop         edi
00241739 5E                    pop         esi
0024173A 5B                    pop         ebx
0024173B 81 C4 C0 00 00 00     add         esp,0C0h
00241741 3B EC                 cmp         ebp,esp
00241743 E8 C5 FA FF FF        call        __RTC_CheckEsp (024120Dh)
00241748 8B E5                 mov         esp,ebp
0024174A 5D                    pop         ebp
0024174B C3                    ret
```

Meng Wai Chan
Apr 30, 2023

```
0x006FF590  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF5B2  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc a4 f6 6f 00 31 17 24 00
0x006FF5D4  01 00 00 00 7c f7 6f 00 25 13 24 00 00 10 49 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF5F6  Factorial(1) cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF618  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF63A  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF65C  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF67E  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF6A0  cc cc cc cc 7c f7 6f 00 31 17 24 00 02 00 00 00 54 f8 6f 00 25 13 24 00 00 10 49 00 cc cc cc cc cc cc
0x006FF6C2  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF6E4  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF706  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF728  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF74A  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF76C  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc 54 f8 6f 00 31 17 24 00 03 00 00 00 2c f9 6f 00 25 13
0x006FF78E  24 00 00 10 49 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF7B0  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF7D2  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF7F4  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF816  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF838  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc 2c f9 6f 00 31 17
0x006FF85A  24 00 04 00 00 00 10 fa 6f 00 25 13 24 00 00 10 49 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF87C  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF89E  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF8C0  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF8E2  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF904  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF926  cc cc cc cc cc cc 10 fa 6f 00 af 17 24 00 05 00 00 00 25 13 24 00 25 13 24 00 00 10 49 00 cc cc cc cc
0x006FF948  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x006FF96A  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
```

Factorial(2)

Factorial(3)

Factorial(4)

Factorial(5)

Each color Frame is how the a recursive function allocate their memory with (10 fa 6f 00) Being the saved EBP of main(), return address for factorial(5) is (af 17 24 00) and (05 00 00 00) is the argument during execution of factorial(5). In factorial(4) the EBP is (2c f9 6f 00) and the argument is (04 00 00 00). In factorial(3) the EBP is (54 f8 6f 00) and the argument is (03 00 00 00). In factorial(2) the EBP is (7c f7 6f 00) and the argument is (02 00 00 00). In factorial(1) the EBP is (a4 f6 6f 00) and the argument is (01 00 00 00).

Meng Wai Chan
Apr 30, 2023

Factorial - GDB

```
Breakpoint 1, fact (n=5) at main.cpp:2
2           if( n == 1)
(gdb) disassemble
Dump of assembler code for function _Z4facti:
   0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     sub    $0x10,%rsp
   0x0000555555555135 <+12>:    mov    %edi,-0x4(%rbp)
=> 0x0000555555555138 <+15>:    cmpl   $0x1,-0x4(%rbp)
   0x000055555555513c <+19>:    jne    0x555555555145 <_Z4facti+28>
   0x000055555555513e <+21>:    mov    $0x1,%eax
   0x0000555555555143 <+26>:    jmp    0x555555555156 <_Z4facti+45>
   0x0000555555555145 <+28>:    mov    -0x4(%rbp),%eax
   0x0000555555555148 <+31>:    sub    $0x1,%eax
   0x000055555555514b <+34>:    mov    %eax,%edi
   0x000055555555514d <+36>:    call   0x555555555129 <_Z4facti>
   0x0000555555555152 <+41>:    imul   -0x4(%rbp),%eax
   0x0000555555555156 <+45>:    leave
   0x0000555555555157 <+46>:    ret
End of assembler dump.
```

```
(gdb) x/12xw $rsp
0x7fffffffdfc0: 0x00000000      0x00000000      0x00000000      0x00000005
0x7fffffffdfd0: 0xffffdff0      0x00007fff      0x5555516e      0x00005555
0x7fffffffdfe0: 0x00000000      0x00000000      0x00000000      0x00000000
(gdb) n 2

Breakpoint 1, fact (n=4) at main.cpp:2
2           if( n == 1)
(gdb) x/12xw $rsp
0x7fffffffdfa0: 0x00000000      0x00000000      0x00000000      0x00000004
0x7fffffffdfb0: 0xffffdfd0      0x00007fff      0x55555152      0x00005555
0x7fffffffdfc0: 0x00000000      0x00000000      0x00000000      0x00000005
(gdb) n 2

Breakpoint 1, fact (n=3) at main.cpp:2
2           if( n == 1)
(gdb) x/12xw $rsp
0x7fffffffdf80: 0x00000000      0x00000000      0x00000000      0x00000003
0x7fffffffdf90: 0xffffdfb0      0x00007fff      0x55555152      0x00005555
0x7fffffffdfa0: 0x00000000      0x00000000      0x00000000      0x00000004
(gdb) n 2

Breakpoint 1, fact (n=2) at main.cpp:2
2           if( n == 1)
(gdb) x/12xw $rsp
0x7fffffffdf60: 0x00000000      0x00000000      0x00000000      0x00000002
0x7fffffffdf70: 0xffffdf90      0x00007fff      0x55555152      0x00005555
0x7fffffffdf80: 0x00000000      0x00000000      0x00000000      0x00000003
(gdb) n 2

Breakpoint 1, fact (n=1) at main.cpp:2
2           if( n == 1)
(gdb) x/12xw $rsp
0x7fffffffdf40: 0x00000002      0x00000000      0x00000000      0x00000001
0x7fffffffdf50: 0xffffdf70      0x00007fff      0x55555152      0x00005555
0x7fffffffdf60: 0x00000000      0x00000000      0x00000000      0x00000002
```

Meng Wai Chan
Apr 30, 2023

Stack Frame

```
(gdb) bt
#0  fact (n=1) at main.cpp:2
#1  0x0000555555555152 in fact (n=2) at main.cpp:4
#2  0x0000555555555152 in fact (n=3) at main.cpp:4
#3  0x0000555555555152 in fact (n=4) at main.cpp:4
#4  0x0000555555555152 in fact (n=5) at main.cpp:4
#5  0x000055555555516e in main () at main.cpp:8
```

Assembly Code

```
mengwai@mengwai-VirtualBox:~/Desktop/
        .file   "main.cpp"
        .text
        .globl  _Z4facti
        .type   _Z4facti, @function
_Z4facti:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        cmpl    $1, -4(%rbp)
        jne     .L2
        movl    $1, %eax
        jmp     .L3
.L2:
        movl    -4(%rbp), %eax
        subl    $1, %eax
        movl    %eax, %edi
        call    _Z4facti
        imull   -4(%rbp), %eax
.L3:
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   _Z4facti, .-_Z4facti
        .globl  main
        .type   main, @function
```

```
main:
.LFB1:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $5, %edi
        call    _Z4facti
        movl    %eax, -4(%rbp)
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
```

As you can see above the recursive functions have 5 stack frames with the main() frame at the bottom. The return address is 0x55555152 0x00005555. RBP is 0xffffdff0 0x00007ffff for factorial(5), RBP is 0xffffdfd0 0x00007ffff for factorial(4), RBP is 0xffffdfb0 0x00007ffff for factorial(3), RBP is 0xffffdf90 0x00007ffff for factorial(2), RBP is 0xffffdf70 0x00007ffff for factorial(1).

Meng Wai Chan
Apr 30, 2023

Factorial - MIPS

Factorial.asm

```
main:
li $a0, 5
jal fact
addi $s0, $v0, 0
syscall

fact:
addi $sp, $sp, -8
sw $ra, 4($sp)
sw $a0, 0($sp)

slti $t0, $a0, 1
beq $t0, $zero, L1

addi $v0, $zero, 1
addi $sp, $sp, 8
jr $ra

L1: addi $a0 , $a0, -1
jal fact

lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8
mul $v0, $a0, $v0
jr $ra
```

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000078
$v1	3	0x00000000
$a0	4	0x00000005
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000001
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000078
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00400008
pc		0x0040000c
hi		0x00000000
lo		0x00000078

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Va Factorial(1)	Value (+18)	Value (+1c) Factorial(2)
0x7fffefc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00400038	0x00000001	0x00400038	0x00000002
0x7fffefe0	0x00400038	0x00000003	0x00400038	0x00000004	0x00400038	0x00000005	0x00400008	0x00000000
0x7ffff000	0x00000000	0x0 Factorial(3)	0x00000000	Factorial(4) 00	0x00000000	0x Factorial(5)	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

In the MIPS Code we can see the stack frame of each factorial recursion. For fact(5) the return address is to main() with 0x00400008. For fact(4) the return address is to fact(5) with 0x00400038. For fact(3) the return address is to fact(4) with 0x00400038. For fact(2) the return address is to fact(3) with 0x00400038. For fact(1) the return address is to fact(2) with 0x00400038. The final answer for factorial(5) is held in $s0.

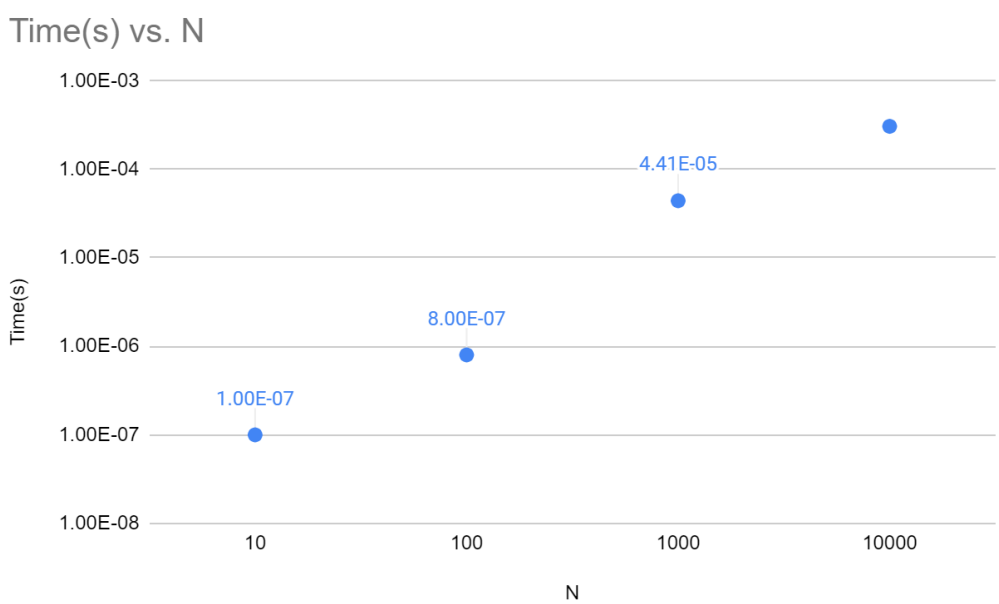
Meng Wai Chan
Apr 30, 2023

Factorial - Time Analysis

fact.cpp > main()

```cpp
#include <iostream>
#include <chrono>
using namespace std;

int factorial(int N){
    if (N == 1)
        return 1;
    return N*factorial(N-1);
}

int main(){
    typedef chrono::high_resolution_clock time;
    typedef chrono::milliseconds ms;
    typedef chrono::duration<float> fsec;

    auto t0 = time::now();
    factorial(10);
    auto t1 = time::now();

    fsec fs = t1 - t0;
    ms d = chrono::duration_cast<ms>(fs);
    cout<< fs.count()<<"s\n";
}
```

N	10	100	1000	10000
Time(s)	1e-07	8e-07	4.41e-5	3.054e-4



Meng Wai Chan
Apr 30, 2023

GCD - Windows Visual Studio

Project1 | (Global Scope) | gcd(int a, int b)

```
1  int gcd(int a, int b) {
2      if (b == 0)
3          return a;   ≤1ms elapsed
4      return gcd(b, a%b);
5  }
6  int main() {
7      int gcdOutput = gcd(8, 24);
8  }
```

Registers

```
EAX = 007AC000
EBX = 00B41000
ECX = 007AC000
EDX = 00000001
ESI = 007A1325
EDI = 00CFFA74
EIP = 007A171E
ESP = 00CFF9A8
EBP = 00CFFA74
EFL = 00000246

0x00CFFA7C = 00000008
```

```
int gcd(int a, int b) {
007A16F0 55                  push        ebp
007A16F1 8B EC               mov         ebp,esp
007A16F3 81 EC C0 00 00 00   sub         esp,0C0h
007A16F9 53                  push        ebx
007A16FA 56                  push        esi
007A16FB 57                  push        edi
007A16FC 8D BD 40 FF FF FF   lea         edi,[ebp-0C0h]
007A1702 B9 30 00 00 00      mov         ecx,30h
007A1707 B8 CC CC CC CC      mov         eax,0CCCCCCCCh
007A170C F3 AB               rep stos    dword ptr es:[edi]
007A170E B9 00 C0 7A 00      mov         ecx,offset _8F72488B_gcd@cpp (07AC000h)
007A1713 E8 F0 FA FF FF      call        @__CheckForDebuggerJustMyCode@4 (07A1208h)
    if (b == 0)
007A1718 83 7D 0C 00         cmp         dword ptr [b],0
007A171C 75 05               jne         gcd+33h (07A1723h)
        return a;
007A171E 8B 45 08            mov         eax,dword ptr [a]
007A1721 EB 14               jmp         gcd+47h (07A1737h)
    return gcd(b, a%b);
007A1723 8B 45 08            mov         eax,dword ptr [a]
007A1726 99                  cdq
007A1727 F7 7D 0C            idiv        eax,dword ptr [b]
007A172A 52                  push        edx
007A172B 8B 45 0C            mov         eax,dword ptr [b]
007A172E 50                  push        eax
007A172F E8 D0 F9 FF FF      call        gcd (07A1104h)
007A1734 83 C4 08            add         esp,8
}
007A1737 5F                  pop         edi
007A1738 5E                  pop         esi
007A1739 5B                  pop         ebx
007A173A 81 C4 C0 00 00 00   add         esp,0C0h
007A1740 3B EC               cmp         ebp,esp
007A1742 E8 CB FA FF FF      call        __RTC_CheckEsp (07A1212h)
007A1747 8B E5               mov         esp,ebp
007A1749 5D                  pop         ebp
007A174A C3                  ret
```

Meng Wai Chan
Apr 30, 2023

```
0x00CFFA5A  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc 50 fb cf 00 34 17 7a 00
0x00CFFA7C  08 00 00 00 00 00 00 00 2c fc cf 00 25 13 7a 00 00 10 b4 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFA9E  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFAC0  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFAE2  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFB04  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFB26  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFB48  cc cc cc cc cc cc cc cc 2c fc cf 00 34 17 7a 00 18 00 00 00 08 00 00 00 14 fd cf 00 25 13 7a 00 00 10
0x00CFFB6A  b4 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFB8C  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFBAE  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFBD0  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFBF2  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFC14  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc 14 fd cf 00 b1 17 7a 00 08 00
0x00CFFC36  00 00 18 00 00 00 25 13 7a 00 25 13 7a 00 00 10 b4 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x00CFFC58  cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
```

gcd(8,0)

gcd(24,8)

gcd(24,8)

Each color Frame is how the a recursive function allocate their memory with (14 fd cf 00) Being the saved EBP of main(), the return address for gcd(8, 24) is (b1 17 7a 00) and (08 00 00 00) and (18 00 00 00) are the arguments during execution of gcd(8, 24). In gcd(24, 8) the EBP is (2c fc cf 00) and the argument is (18 00 00 00) and (08 00 00 00). In gcd(8, 0) the EBP is (50 fb cf 00) and the argument is (08 00 00 00) and (00 00 00 00).

GCD - GDB

```
Breakpoint 1, gcd (a=8, b=24) at gcd.cpp:2
2           if (b == 0)
(gdb) x/12xw $rsp
0x7fffffffdfc0: 0x00000000      0x00000000      0x00000018      0x00000008
0x7fffffffdfd0: 0xffffdff0      0x00007fff      0x55555179      0x00005555
0x7fffffffdfe0: 0x00000000      0x00000000      0x00000000      0x00000000
```

```
Breakpoint 1, gcd (a=24, b=8) at gcd.cpp:2
2           if (b == 0)
(gdb) x/12xw $rsp
0x7fffffffdfa0: 0x00000000      0x00000000      0x00000008      0x00000018
0x7fffffffdfb0: 0xffffdfd0      0x00007fff      0x5555515b      0x00005555
0x7fffffffdfc0: 0x00000000      0x00000000      0x00000018      0x00000008
```

```
Breakpoint 1, gcd (a=8, b=0) at gcd.cpp:2
2           if (b == 0)
(gdb) x/12xw $rsp
0x7fffffffdf80: 0x00000000      0x00000000      0x00000000      0x00000008
0x7fffffffdf90: 0xffffdfb0      0x00007fff      0x5555515b      0x00005555
0x7fffffffdfa0: 0x00000000      0x00000000      0x00000008      0x00000018
```

```
(gdb) bt
#0  gcd (a=8, b=0) at gcd.cpp:5
#1  0x000055555555515b in gcd (a=24, b=8) at gcd.cpp:4
#2  0x000055555555515b in gcd (a=8, b=24) at gcd.cpp:4
#3  0x0000555555555179 in main () at gcd.cpp:7
```

Meng Wai Chan
Apr 30, 2023

```
(gdb) disassemble
Dump of assembler code for function _Z3gcdii:
   0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     sub    $0x10,%rsp
   0x0000555555555135 <+12>:    mov    %edi,-0x4(%rbp)
   0x0000555555555138 <+15>:    mov    %esi,-0x8(%rbp)
   0x000055555555513b <+18>:    cmpl   $0x0,-0x8(%rbp)
   0x000055555555513f <+22>:    jne    0x555555555148 <_Z3gcdii+31>
   0x0000555555555141 <+24>:    mov    $0x0,%eax
   0x0000555555555146 <+29>:    jmp    0x55555555515c <_Z3gcdii+51>
   0x0000555555555148 <+31>:    mov    -0x4(%rbp),%eax
   0x000055555555514b <+34>:    cltd
   0x000055555555514c <+35>:    idivl  -0x8(%rbp)
   0x000055555555514f <+38>:    mov    -0x8(%rbp),%eax
   0x0000555555555152 <+41>:    mov    %edx,%esi
   0x0000555555555154 <+43>:    mov    %eax,%edi
   0x0000555555555156 <+45>:    call   0x555555555129 <_Z3gcdii>
   0x000055555555515b <+50>:    nop
=> 0x000055555555515c <+51>:    leave
   0x000055555555515d <+52>:    ret
End of assembler dump.
```

```
mengwai@mengwai-VirtualBox:~/Desktop/
        .file   "gcd.cpp"
        .text
        .globl  _Z3gcdii
        .type   _Z3gcdii, @function
_Z3gcdii:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movl    %esi, -8(%rbp)
        cmpl    $0, -8(%rbp)
        jne     .L2
        movl    $0, %eax
        jmp     .L3
.L2:
        movl    -4(%rbp), %eax
        cltd
        idivl   -8(%rbp)
        movl    -8(%rbp), %eax
        movl    %edx, %esi
        movl    %eax, %edi
        call    _Z3gcdii
        nop
.L3:
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   _Z3gcdii, .-_Z3gcdii
        .globl  main
        .type   main, @function
```

```
main:
.LFB1:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $24, %esi
        movl    $8, %edi
        call    _Z3gcdii
        movl    %eax, -4(%rbp)
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
```

Meng Wai Chan
Apr 30, 2023

GCD - MIPS

```
.data
a: .word 8
b: .word 24

.text
.globl main
main:
    lw $a0,a
    lw $a1,b
    jal gcd
    add $a0,$v0,$zero
    li $v0,1
    syscall
li $v0, 10
syscall

gcd:
    addi $sp, $sp, -12
    sw $ra, 0($sp)
    sw $s0, 4($sp)
    sw $s1, 8($sp)

    add $s0, $a0, $zero
    add $s1, $a1, $zero
    addi $t1, $zero, 0
    beq $s1, $t1, return
    add $a0, $zero, $s1
    div $s0, $s1
    mfhi $a1

    jal gcd

exit:
    lw $ra, 0 ($sp)
    lw $s0, 4 ($sp)
    lw $s1, 8 ($sp)
    addi $sp,$sp , 12
    jr $ra
return:
    add $v0, $zero, $s0
    j exit
```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffefc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00400058	0x00000018
0x7fffefe0	0x00000008	0x00400058	0x00000008	0x00000018	0x00400014	0x00000000	0x00000000	0x00000000
0x7ffff000	0x00000 gcd(24,8)	0x00000000	0x00000000	0x00000 gcd(8,24)	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

current $sp ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000008
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000

III. Conclusion

In this take home test I gained an important lesson, which emphasized on how each recursive function called is shown and executed in the stack frame. It also helps us gain an insight on where each stack frame is stored in memory, giving us a better understanding on how stack frames function.