

Last Name:

First Name:

## Computer Science C.Sc. 342

### Take Home TEST No.2 *CSc or CPE*

*Submit by 11:59 PM, March 22, 2017*

#### ***Objective:***

The objective of this take home test is for students to

1. Run and debug a recursive function call on three different platforms: x86 Intel on Microsoft's Visual Studio, MIPS on MARS Simulator, and on a 64-bit Intel processor running Linux. Display and explain all frames on stack.
2. Measure and plot the time it takes to compute Factorial (N), for N= 10, 100, 1000, 10,000.

Example of a recursive procedure that calculates the factorial of a number and its code in both C and MIPS can be found in the textbook and is shown below.

#### ***Create and explain Stack Frames for the recursive function call factorial(5)***

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

void main()
{
    int N_fact=factorial(5);
}
```

1. Compile and run this program in Debug mode in .NET environment.

For each **call level** display Frame on stack and write down the address on stack and value of

- Argument at current level
- local variable ( if any) at current level
- return address at current level
- EIP
- EBP
- ESP

You may use arrow to point a specific location on stack frame.

At the end of calls you should display 5 frames on the stack as shown in FIGURE 1.

Last Name:

First Name:

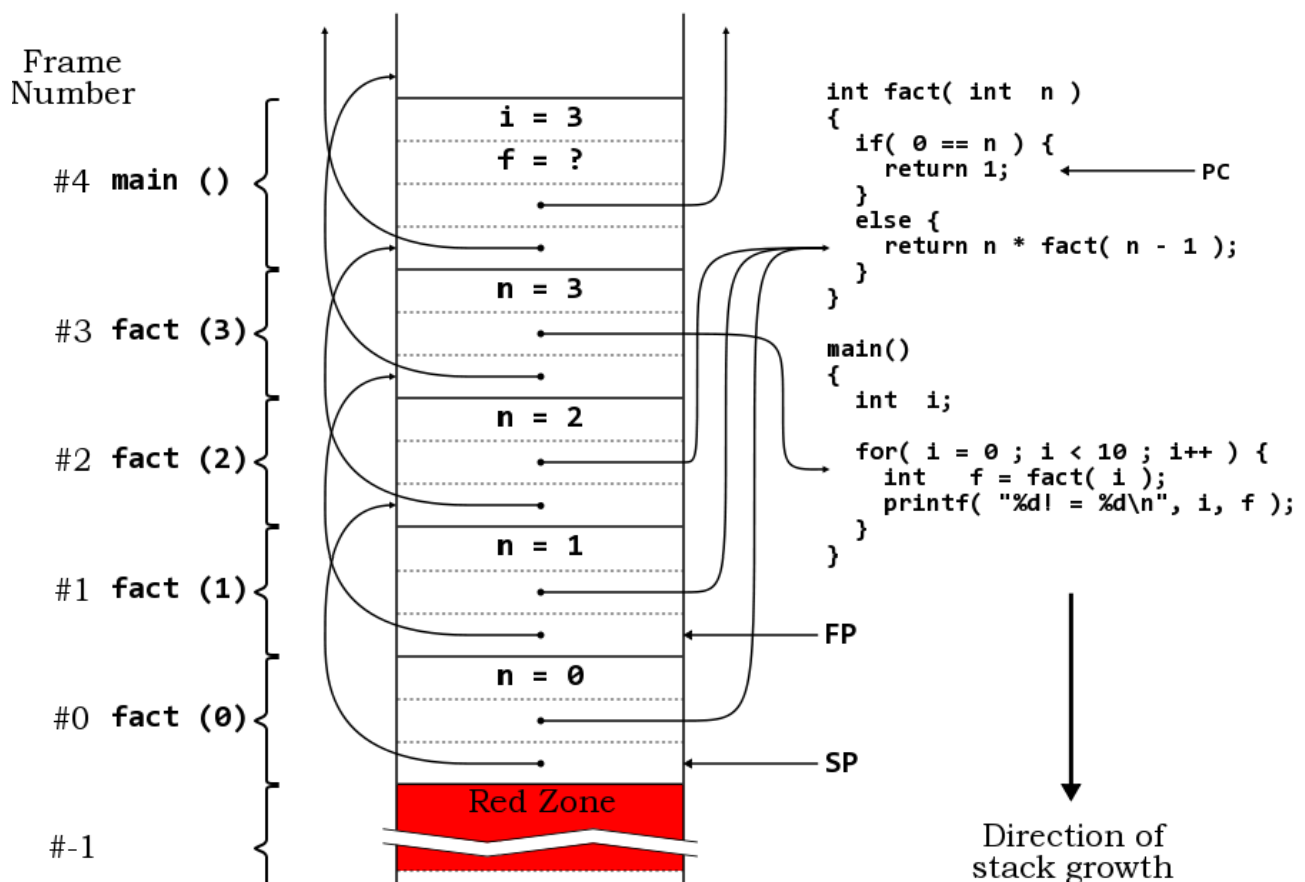


FIGURE 1. All arrows have to show labels to addresses on stack and corresponding values.

Please explain the return process – specify instructions and arguments used at each nested level when returning.

- (Optional) Create a lean version of the factorial() function. Instead of using CALL instruction ( generated by compiler), create function call using similar to JAL instruction in MIPS - save the return address and then jump to function. Do not push and pop unnecessary information on stack ( such as registers ebx, ecx, etc.) on stack.
- Please repeat Section 1 using MIPS instructions and run the program on a simulator MARS. You can use example described in the section on nested procedure calls in the textbook.
- Please repeat Section 1 using GCC, GDB in LINUX environment, and run the program in command mode using GDB. You can use example described in the section on nested procedure calls in the textbook.

Last Name:

First Name:

## Sample screenshots for X86, MS Visual Studio in Debug mode

The screenshot displays the MS Visual Studio debugger in X86 mode, showing the execution of a C++ program with a recursive factorial function. The assembly view at the top shows the function's logic, with a yellow cursor at the base case return statement. The registers pane on the right shows the current state of registers, with EBP pointing to the saved main() frame and EAX containing the return value. The memory view at the bottom shows the stack frame for factorial(5), with annotations for return addresses and arguments.

**Assembly Code:**

```
1: int factorial(int N){
00401300 55      push    ebp
00401301 8B EC   mov     ebp,esp
00401303 81 EC 00 00 00 00 sub     esp,0C0h
00401309 53      push    ebx
0040130A 56      push    esi
0040130B 57      push    edi
0040130C 8D BD 40 FF FF FF lea     edi,[ebp-0C0h]
0040130D B9 30 00 00 00 mov     ecx,30h
0040130F B8 CC CC CC CC mov     eax,0CCCCCCCch
00401310 F3 AB   rep     stos dword ptr es:[edi]
2: if (N == 1) return 1;
0040131D 83 7D 08 01 cmp     dword ptr [N],1
00401322 75 07   jne     factorial+2Bh (04013EBh)
00401324 B8 01 00 00 00 mov     eax,1
00401329 E8 13 jmp     factorial+3Eh (04013FEh)
3: return N*factorial(N - 1);
0040132B 8B 45 08 mov     eax,dword ptr [N]
0040132D 83 E8 01 sub     eax,1
00401330 50      push    eax
00401331 F2 E4 FD FF FF call    factorial (04011DBh)
00401337 83 C4 04 add     esp,4
0040133A 0F AF 45 08 imul    eax,dword ptr [N]
4: }
0040133E 5F      pop     edi
4: }
0040133F 5E      pop     esi
00401340 5B      pop     ebx
```

**Registers:**

- EAX = 00000000
- EBX = 7EFDE000
- ECX = 00000000
- EDX = 00000001
- ESI = 00000000
- EDI = 0015FAC4
- EIP = 004013DE
- ESP = 0015F9F8
- EBP = 0015FAC4
- EFL = 00000200

**Memory:**

Address: 0x0015F61C

factorial(1) return address: 0x0015FAC4

factorial(2) return address: 0x0015FAC8

factorial(3) return address: 0x0015FAC0

factorial(4) return address: 0x0015FAC4

factorial(5) return address: 0x0015FAC8

Last Name:

First Name:

## Sample screenshots for MIPS, Simulator MARS environment

The screenshot displays the MARS MIPS simulator interface, which is divided into several panels:

- Text Segment:** Shows assembly code with columns for Bkpt, Address, Code, Basic, and Source. The code includes instructions like `addiu $4, $0, 0x00000005`, `jal 0x00400010`, `addi $16, $2, 0x00000000`, `syscall`, `addi $29, $29, 0xffffffff`, `sw $31, 0x00000004($29)`, `sw $4, 0x00000000($29)`, `slti $8, $4, 0x00000001`, `beq $8, $0, 0x00000003`, `addi $2, $0, 0x00000001`, `addi $29, $29, 0x00000008`, `jr $31`, `addi $4, $4, 0xffffffff`, `jal 0x00400010`, `lw $4, 0x00000000($29)`, `lw $31, 0x00000004($29)`, `addi $29, $29, 0x00000008`, `mul $2, $4, $2`, and `jr $31`.
- Data Segment:** Shows memory addresses and their corresponding values. The values are mostly zero, except for `0x00000005` at address `0x00400008` and `0x00000038` at address `0x00400038`. A red arrow points to the value `0x00000005` at address `0x00400008`, and a blue arrow points to the value `0x00000038` at address `0x00400038`.
- Registers:** Shows the state of MIPS registers. The `$ra` register (address `0x00400038`) contains the value `0x00000038`, which is highlighted in green. The `$sp` register (address `0x00000028`) contains the value `0x7fffffec`, which is highlighted in green. The `$pc` register (address `0x00400010`) contains the value `0x00400038`, which is highlighted in orange.

Annotations in the Data Segment panel include:

- Argument at current level address: `0x7fffec` value: `4`
- Return address on stack address: `0x7fffeff0` value: `0x00400038`



Last Name:

First Name:

## Sample screenshots for 64 bit Intel processor, GDB

```
=> 0x00000000004004f6 <+0>:    push    %rbp
0x00000000004004f7 <+1>:    mov     %rsp,%rbp
0x00000000004004fa <+4>:    sub     $0x10,%rsp
0x00000000004004fe <+8>:    mov     %edi,-0x4(%rbp)
0x0000000000400501 <+11>:   cmpl    $0x1,-0x4(%rbp)
0x0000000000400505 <+15>:   jne     0x40050e <factorial(int)+24>
0x0000000000400507 <+17>:   mov     $0x1,%eax
0x000000000040050c <+22>:   jmp     0x40051f <factorial(int)+41>
0x000000000040050e <+24>:   mov     -0x4(%rbp),%eax
0x0000000000400511 <+27>:   sub     $0x1,%eax
0x0000000000400514 <+30>:   mov     %eax,%edi
0x0000000000400516 <+32>:   callq   0x4004f6 <factorial(int)>
0x000000000040051b <+37>:   imul    -0x4(%rbp),%eax
0x000000000040051f <+41>:   leaveq  0x0000000000400520 <+42>:   retq
```

End of assembler dump.

(gdb) nexti 3

```
0x00000000004004fe      1      int factorial(int N){
```

```
1: x/i $pc
```

```
=> 0x4004fe <factorial(int)+8>: mov     %edi,-0x4(%rbp)
```

```
(gdb) printf "rbp:%x\nrsp:%x\n", $rbp, $rsp
```

```
rbp:ffffdde0
```

```
rsp:ffffddd0
```

```
(gdb) █
```

```
1: x/i $pc
=> 0x4004fe <factorial(int)+8>: mov     %edi,-0x4(%rbp)
(gdb) printf "rbp:%x\nrsp:%x\n", $rbp, $rsp
rbp:ffffdde0
rsp:ffffddd0
(gdb) nexti
2      if(N == 1) return 1;
1: x/i $pc
=> 0x400501 <factorial(int)+11>    cmpl    $0x1,-0x4(%rbp)
(gdb) x/12xw $rsp
0x7fffffffddd0: 0x00000000 0x00000000 0x00000000 0x00000001
0x7fffffffddde: 0xffffde00 0x00007fff 0x0040051b 0x00000000
0x7fffffffdd10: 0xffffde20 0x00007fff 0xffffde10 0x00000002
(gdb) p $rip
$15 = (void (*)(void)) 0x400501 <factorial(int)+11>
(gdb) █
```

Argument during factorial(1)

Return address during factorial(1)

Saved RBP of factorial(2)