

Last Name:

First Name:

Computer Science C.Sc. 342

Take Home TEST No.2 CSc or CPE

Submit by 11:00 PM April 25, 2022

1. Please hand write and sign statements affirming that you will not cheat here and in your submission:
"I will neither give nor receive unauthorized assistance on this TEST. I will use only one computing device to perform this TEST, I will not use cell while performing this TEST".

2. Students should allocate no more than total 12 hours to prepare report for this TEST.

Please write

Start Time and date:

End TIME and date:

3. You can use any resources.

4. You are not allowed to communicate with fellow students or other professionals.

5. THIS IS NOT A TEAM TEST!!

You must submit:

1. Report must have OBJECTIVE section, and Table of Content with links to corresponding sections.
2. Short Video presentation, no more than 2 min.
3. Source code and project files you have used. You must include README file with instructions on how to run your examples.

The report file and video file names should be:

YOUR_LAST_NAME_TAKE_HOME_TEST_2,

LAST_NAME_VIDEO_TAKE_HOME_TEST_2

LAST_NAME_SOURCE_README__TAKE_HOME_TEST_2.ZIP

Last Name:

First Name:

Objective:

The objective of this take home test is for students to

1. Run and debug a recursive function call on four different platforms: x86 Intel on Microsoft's Visual Studio, Raspberry Pi ARM processor 32 bit, 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.
3. **Required part :** Repeat tutorial example 1 and 2 to compute GCD(a,b) using recursive version of EUCLIDEAN algorithm for two integers $a > 0$, $b > 0$. To refresh GCD(a,b) computation please refer to last 3 pages of this assignment.
4. **What to Submit:** report, working project files and how to use, 2 min video presentation.

Tutorial 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:

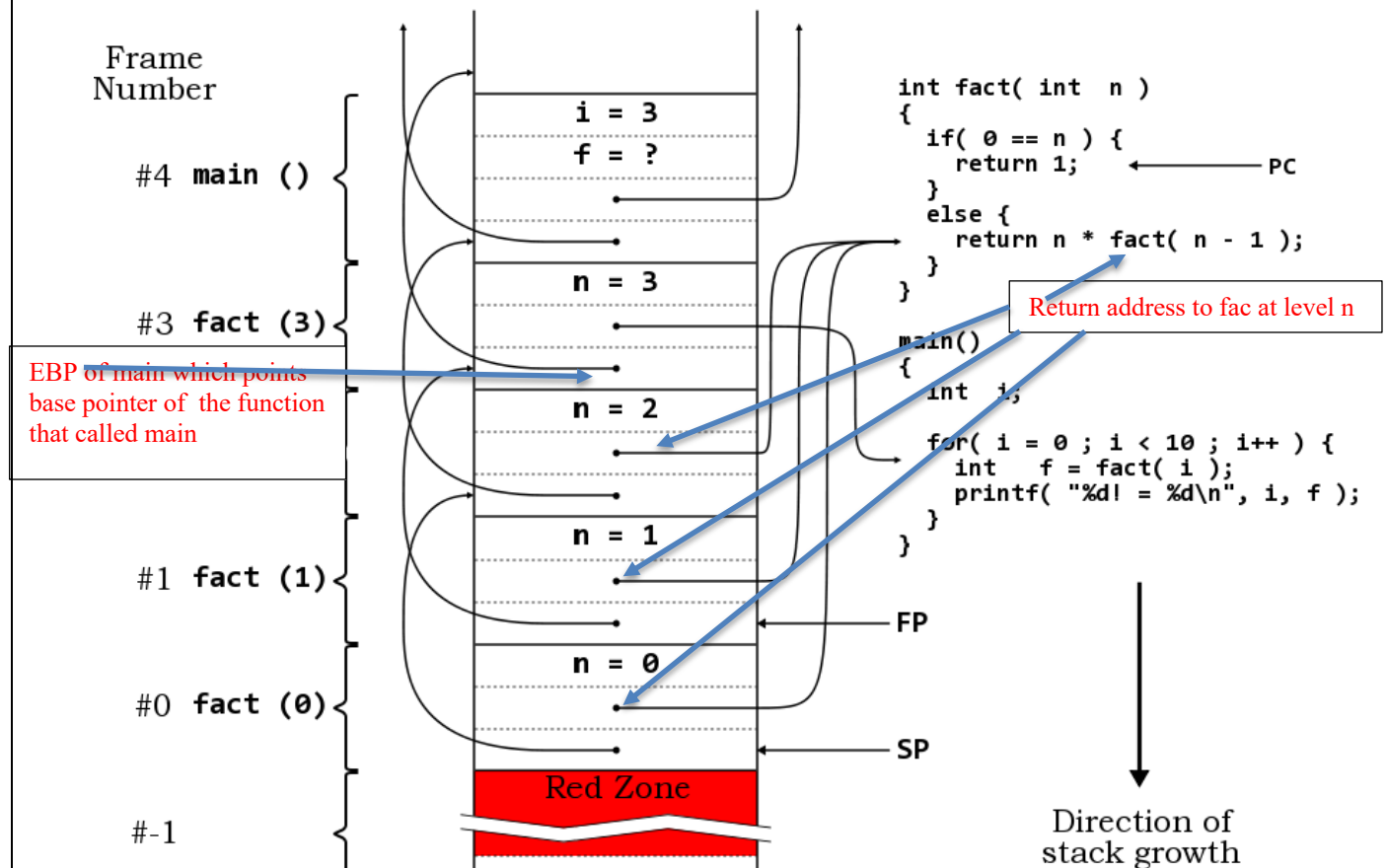


FIGURE 1. All arrows have to show labels to addresses on stack and corresponding values.

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 interface in X86 architecture. The assembly window shows the execution of a `factorial` function. The memory window shows the stack, with various memory locations highlighted and labeled.

Assembly Window:

```
1: int factorial(int N){
004013C0 55      push    ebp
004013C1 8B EC   mov     ebp,esp
004013C3 81 EC C0 00 00 00 sub     esp,0C0h
004013C9 53      push    ebx
004013CA 56      push    esi
004013CB 57      push    edi
004013CC 8D BD 40 FF FF FF lea     edi,[ebp-0C0h]
004013D2 B9 30 00 00 00 mov     ecx,30h
004013D7 B8 CC CC CC CC mov     eax,0CCCCCCCCh
004013DC F3 AB   rep stos dword ptr es:[edi]
2: if (N == 1) return 1;
004013DE 83 7D 08 01 cmp     dword ptr [N],1
004013E2 75 07   jne     factorial+28h (004013EBh)
004013E4 B8 01 00 00 00 mov     eax,1
004013E9 EB 13   jmp     factorial+3Eh (004013FEh)
3: return N*factorial(N - 1);
004013EB 8B 45 08 mov     eax,dword ptr [N]
004013EE 83 E8 01 sub     eax,1
004013F1 50      push    eax
004013F2 E8 F4 FD FF FF call    factorial (004011DBh)
004013F7 83 C4 04 add     esp,4
004013FA 0F AF 45 08 imul    eax,dword ptr [N]
4: }
004013FE 5F      pop     edi
4: }
004013FF 5E      pop     esi
00401400 5B      pop     ebx
```

Registers:

- EAX = CCCCCCCC
- EBX = 7EFDE000
- ECX = 00000000
- EDX = 00000001
- ESI = 00000000
- EDI = 0015FAC4
- EIP = 004013DE
- ESP = 0015F9F8
- EBP = 0015FAC4
- EFL = 00000200
- 0x0015facc = 00000005

Memory Window:

Address: 0x0015F61C

Memory dump showing various values, including:

- 0x0015F61C: 00 00 47 00 00 00 00 00 7f 00 00 14 f7 15 00 5a 3c be 77 8b 00 00 00 00 9c b1 c3 77
- 0x0015F638: 14 f7 15 00 ce 57 c0 77 d3 3c be 77 cf c0 b9 75 40 04 00 00 00 00 00 00 00 47 00
- 0x0015F654: 50 01 47 00 50 01 47 00 64 f7 15 00 b8 50 47 00 00 10 00 00 64 f7 15 00 01 00 00 00
- 0x0015F670: 77 01 00 00 04 fe 69 0f 50 01 47 00 fe ff ff ff 00 88 47 00 cf ee 5b 0f 00 00 56 0f
- 0x0015F68C: 01 00 00 00 7f 00 00 b4 f6 15 00 3c f8 15 00 00 00 00 00 e0 fd 7e cc cc cc cc cc
- 0x0015F6A8: cc
- 0x0015F6C4: cc
- 0x0015F6E0: cc
- 0x0015F6FC: cc
- 0x0015F718: cc
- 0x0015F734: cc
- 0x0015F750: cc
- 0x0015F76C: 01 00 00 00 14 f9 15 00 00 00 00 00 e0 fd 7e cc cc cc cc cc cc cc cc cc cc cc cc
- 0x0015F788: cc
- 0x0015F7A4: cc
- 0x0015F7C0: cc
- 0x0015F7DC: cc
- 0x0015F7F8: cc
- 0x0015F814: cc
- 0x0015F830: cc
- 0x0015F84C: 00 00 00 00 e0 fd 7e cc
- 0x0015F868: cc
- 0x0015F884: cc
- 0x0015F8A0: cc
- 0x0015F8BC: cc
- 0x0015F8D8: cc
- 0x0015F8FC: cc
- 0x0015F910: cc
- 0x0015F92C: cc
- 0x0015F948: cc
- 0x0015F964: cc
- 0x0015F980: cc
- 0x0015F99C: cc
- 0x0015F9B8: cc
- 0x0015F9D4: cc
- 0x0015F9F0: f7 13 40 00 04 00 00 00 a8 fb 15 00 00 00 00 00 e0 fd 7e cc cc cc cc cc cc cc cc
- 0x0015FA0C: cc
- 0x0015FA28: cc
- 0x0015FA44: cc
- 0x0015FA60: cc
- 0x0015FA7C: cc
- 0x0015FA98: cc
- 0x0015FAB4: cc
- 0x0015FAD0: 00
- 0x0015FAD4: 00

Annotations:

- Return address to fac: 0x004913F7
- Saved EBP of main(): 0x0015FAC4
- Return address during execution of factorial(5): 0x0015FAC4
- Argument during execution of factorial(5): 0x0015FAC4
- factorial(1): 0x0015F61C
- factorial(2): 0x0015F638
- factorial(3): 0x0015F654
- factorial(4): 0x0015F670
- factorial(5): 0x0015F68C
- EBP of main: 0x0015FBA8
- Saved at ebp = 0x0015FAC4 at fact(5) level

Last Name:

First Name:

Sample screenshots for MIPS, Simulator MARS environment

The screenshot displays the MARS MIPS simulator interface with four main panels:

- Text Segment:** A table of instructions with columns for Bkpt, Address, Code, Basic, and Source. The instruction at address 0x24040005 is highlighted in yellow.
- Data Segment:** A table of memory values with columns for Address, Value (+0), Value (+4), Value (+8), Value (+c), Value (+10), Value (+14), Value (+18), and Value (+1c). The value at address 0x7ffffef0 is highlighted in red.
- Labels:** A table of labels with columns for Label and Address. The label 'factasm' is highlighted in yellow.
- Registers:** A table of registers with columns for Name, Number, and Value. The register 'ra' is highlighted in yellow.

Annotations on the Data Segment panel:

- A red arrow points to the value at address 0x7ffffef0, labeled "Argument at current level address: 0x7ffffef0 value: 4".
- A blue arrow points to the value at address 0x7ffffef4, labeled "Return address on stack address: 0x7ffffef0 value: 0x00400038".

Last Name:

First Name:

Sample screenshots for 64 bit Intel processor, GDB

```
=> 0x0000000004004f6 <+0>:      push  %rbp
0x0000000004004f7 <+1>:      mov   %rsp,%rbp
0x0000000004004fa <+4>:      sub   $0x10,%rsp
0x0000000004004fe <+8>:      mov   %edi,-0x4(%rbp)
0x000000000400501 <+11>:     cmpl  $0x1,-0x4(%rbp)
0x000000000400505 <+15>:     jne   0x40050e <factorial(int)+24>
0x000000000400507 <+17>:     mov   $0x1,%eax
0x00000000040050c <+22>:     jmp   0x40051f <factorial(int)+41>
0x00000000040050e <+24>:     mov   -0x4(%rbp),%eax
0x000000000400511 <+27>:     sub   $0x1,%eax
0x000000000400514 <+30>:     mov   %eax,%edi
0x000000000400516 <+32>:     callq 0x4004f6 <factorial(int)>
0x00000000040051b <+37>:     imul  -0x4(%rbp),%eax
0x00000000040051f <+41>:     leaveq
0x000000000400520 <+42>:     retq

End of assembler dump.
(gdb) nexti 3
0x0000000004004fe      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
0x7fffffffdd00: 0xffffde00 0x00007fff 0x0040051b 0x00000000
0x7fffffffdd00: 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)

END TUTORIAL EXAMPLE.

Review of GCD algorithm: