

SADS 2021 Problem Sheet #4

Problem 4.1: Intel function disassembly and stack frame layout

(6+2+2 = 10 points)

You are given an executable of unknown origin. Since it is never wise to run an executable that you cannot trust, you use `gdb` to disassemble the program. You get the following result:

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000001183 <+0>: push   %rbp
   0x0000000000001184 <+1>: mov    %rsp,%rbp
   0x0000000000001187 <+4>: sub    $0x10,%rsp
   0x000000000000118b <+8>: movl   $0xa,-0x4(%rbp)
   0x0000000000001192 <+15>: mov    -0x4(%rbp),%eax
   0x0000000000001195 <+18>: mov    %eax,%edi
   0x0000000000001197 <+20>: callq 0x1125 <f>
   0x000000000000119c <+25>: leaveq
   0x000000000000119d <+26>: retq
```

End of assembler dump.

```
(gdb) disassemble f
Dump of assembler code for function f:
   0x0000000000001125 <+0>: push   %rbp
   0x0000000000001126 <+1>: mov    %rsp,%rbp
   0x0000000000001129 <+4>: mov    %edi,-0x24(%rbp)
   0x000000000000112c <+7>: movq   $0x0,-0x8(%rbp)
   0x0000000000001134 <+15>: movq   $0x1,-0x10(%rbp)
   0x000000000000113c <+23>: cmpl   $0x0,-0x24(%rbp)
   0x0000000000001140 <+27>: jne    0x1149 <f+36>
   0x0000000000001142 <+29>: mov    $0x0,%eax
   0x0000000000001147 <+34>: jmp    0x1181 <f+92>
   0x0000000000001149 <+36>: movl   $0x1,-0x14(%rbp)
   0x0000000000001150 <+43>: jmp    0x1175 <f+80>
   0x0000000000001152 <+45>: mov    -0x8(%rbp),%rdx
   0x0000000000001156 <+49>: mov    -0x10(%rbp),%rax
   0x000000000000115a <+53>: add    %rdx,%rax
   0x000000000000115d <+56>: mov    %rax,-0x20(%rbp)
   0x0000000000001161 <+60>: mov    -0x10(%rbp),%rax
   0x0000000000001165 <+64>: mov    %rax,-0x8(%rbp)
   0x0000000000001169 <+68>: mov    -0x20(%rbp),%rax
   0x000000000000116d <+72>: mov    %rax,-0x10(%rbp)
   0x0000000000001171 <+76>: addl   $0x1,-0x14(%rbp)
   0x0000000000001175 <+80>: mov    -0x14(%rbp),%eax
   0x0000000000001178 <+83>: cmp    -0x24(%rbp),%eax
   0x000000000000117b <+86>: jl     0x1152 <f+45>
   0x000000000000117d <+88>: mov    -0x10(%rbp),%rax
   0x0000000000001181 <+92>: pop    %rbp
   0x0000000000001182 <+93>: retq
```

You happen to know that the source code has been compiled using `gcc 8.3.00` on `Debian 10.3` for an `Intel x86_64` processor. Note that an `int` is 32 bits and a `long` is 64 bits long on such a system.

- Write a C program (no inline assembler) that does the same as the assembler code shown above. In a first step, lookup what the assembler statements do, comment the assembler code and give memory locations used as variables names. Then try to reconstruct the logic in C.

- b) Explain the layout of the stack frames when $f()$ is executed. Create a diagram showing where the variables, parameters, old base pointer, and return address are located in the stack frame.
- c) The stack pointer (rsp) and the base pointer (rbp) are the same during the execution of the function $f()$ while data is stored “on top of the top of the stack”. Why is this possible in this case?