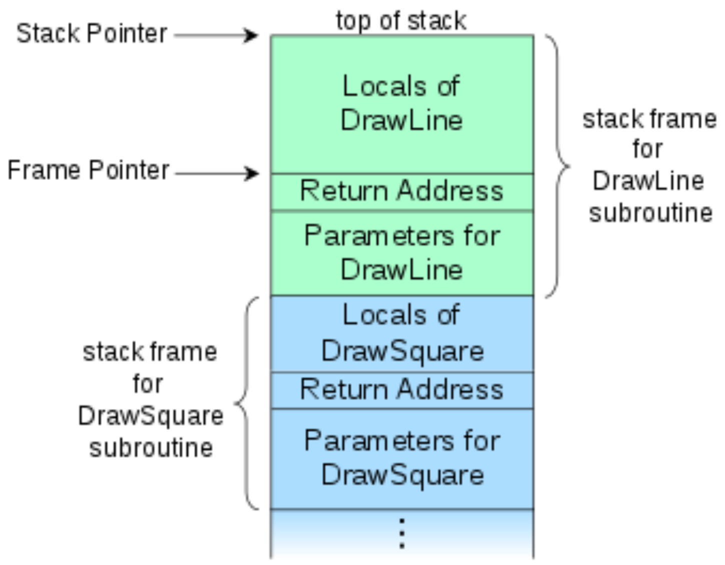
Lab1: Stack and Stack Frame in Linux (6 Points)

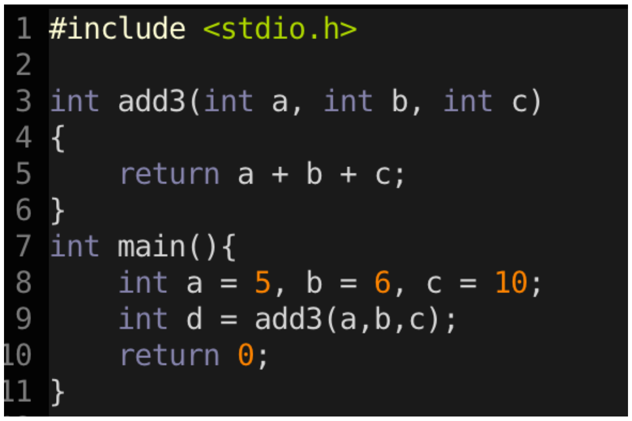


Objectives and Targets

The **stack** is a segment of memory where data like your local variables and function calls get added and/or removed in a last-in-first-out (LIFO) manner. When you compile a program, the compiler enters through the main function and a **stack frame** is created on the stack. A frame, also known as an activation record is the collection of all data on the stack associated with one subprogram call. The main function and all the local variables are stored in an initial frame.

In this lab, you'll re-do the experiment that I did in class, but in a Linux environment.

Step 1: In a Linux environment (e.g. Manjaro environment Link). Download the lab1.c Link.



```c
#include <stdio.h>

int add3(int a, int b, int c)
{
    return a + b + c;
}
int main(){
    int a = 5, b = 6, c = 10;
    int d = add3(a,b,c);
    return 0;
}
```

source code for lab1

Step 2: Compile the code with `gcc` by typing the following command in your terminal.

```
gcc -m32 -no-pie -o lab1 lab1.c
```

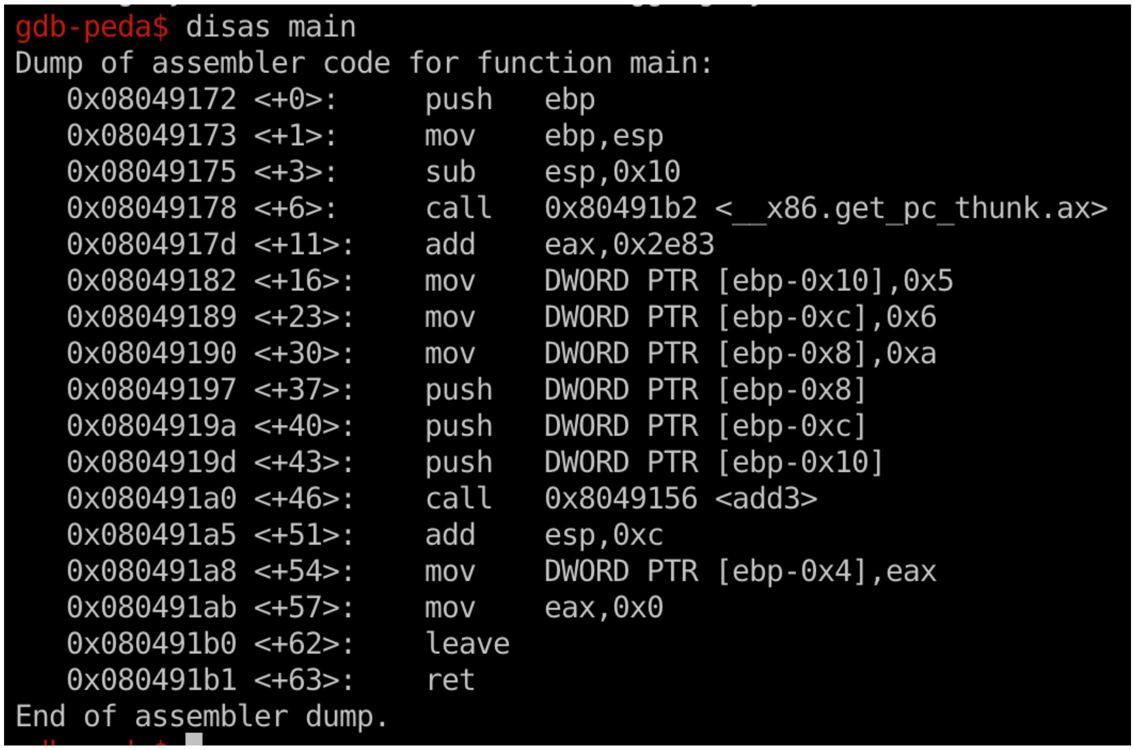
Step 3: Use `gdb` to reverse engineer the output ELF file.

```
gdb lab1
```

P.S. If you're not familar with `gdb`, now it's the best time to check this tutorial Link.

Step 4: Disassemble the `main` function by typing the following command and answer the following question(s):

```
disas main
```



```
gdb-peda$ disas main
Dump of assembler code for function main:
   0x08049172 <+0>:     push   ebp
   0x08049173 <+1>:     mov    ebp,esp
   0x08049175 <+3>:     sub    esp,0x10
   0x08049178 <+6>:     call   0x80491b2 <__x86.get_pc_thunk.ax>
   0x0804917d <+11>:    add    eax,0x2e83
   0x08049182 <+16>:    mov    DWORD PTR [ebp-0x10],0x5
   0x08049189 <+23>:    mov    DWORD PTR [ebp-0xc],0x6
   0x08049190 <+30>:    mov    DWORD PTR [ebp-0x8],0xa
   0x08049197 <+37>:    push   DWORD PTR [ebp-0x8]
   0x0804919a <+40>:    push   DWORD PTR [ebp-0xc]
   0x0804919d <+43>:    push   DWORD PTR [ebp-0x10]
   0x080491a0 <+46>:    call   0x8049156 <add3>
   0x080491a5 <+51>:    add    esp,0xc
   0x080491a8 <+54>:    mov    DWORD PTR [ebp-0x4],eax
   0x080491ab <+57>:    mov    eax,0x0
   0x080491b0 <+62>:    leave
   0x080491b1 <+63>:    ret
End of assembler dump.
```

Assembly code for main function

Q1: What's the meaning of the first three lines (1 point):

```
0x08049172 <+0>:    push   ebp
0x08049173 <+1>:    mov    ebp,esp
0x08049175 <+3>:    sub    esp,0x10
```

Q2: What's the meaning of these three lines (1 point):

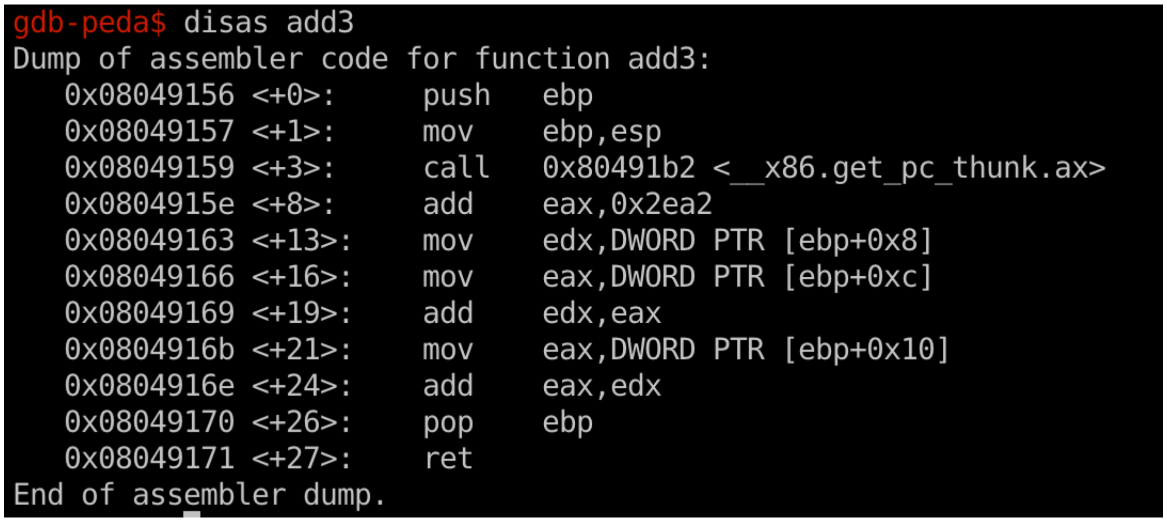
```
0x08049182 <+16>:   mov    DWORD PTR [ebp-0x10],0x5
0x08049189 <+23>:   mov    DWORD PTR [ebp-0xc],0x6
0x08049190 <+30>:   mov    DWORD PTR [ebp-0x8],0xa
```

Q3: What's the meaning of these four lines (1 point):

```
0x08049197 <+37>:   push   DWORD PTR [ebp-0x8]
0x0804919a <+40>:   push   DWORD PTR [ebp-0xc]
0x0804919d <+43>:   push   DWORD PTR [ebp-0x10]
0x080491a0 <+46>:   call   0x8049156 <add3>
```

Step 5: Disassemble the `add3` function by typing the following command and answer the following question(s):

```
disas add3
```



```
gdb-peda$ disas add3
Dump of assembler code for function add3:
   0x08049156 <+0>:     push   ebp
   0x08049157 <+1>:     mov    ebp,esp
   0x08049159 <+3>:     call   0x80491b2 <__x86.get_pc_thunk.ax>
   0x0804915e <+8>:     add    eax,0x2ea2
   0x08049163 <+13>:    mov    edx,DWORD PTR [ebp+0x8]
   0x08049166 <+16>:    mov    eax,DWORD PTR [ebp+0xc]
   0x08049169 <+19>:    add    edx,eax
   0x0804916b <+21>:    mov    eax,DWORD PTR [ebp+0x10]
   0x0804916e <+24>:    add    eax,edx
   0x08049170 <+26>:    pop    ebp
   0x08049171 <+27>:    ret
End of assembler dump.
```

Assembly code for add3 function

Q4: What's the meaning of the first two lines (1 point):

```
0x08049156 <+0>:    push   ebp
0x08049157 <+1>:    mov    ebp,esp
```

Q5: What's the meaning of the last two lines (1 point):

```
0x08049170 <+26>:   pop    ebp
0x08049171 <+27>:   ret
```

Q6: Which register are being used to store the summation result (a+b+c)? Why?(1 point):

Deliverables:

- A detailed project report (**lab1_report.pdf**) in **PDF format** to describe what you have done, including diagrams and code snippets (if needed).

Submission

- Check lab due date on the course website. Late submission will not be accepted.
- The assignment should be submitted to D2L directly.
- Your submission should include file **(lab1_report.pdf)**

- No copy or cheating is tolerated. If your work is based on others', please give clear attribution. Otherwise, you **WILL FAIL** this course.