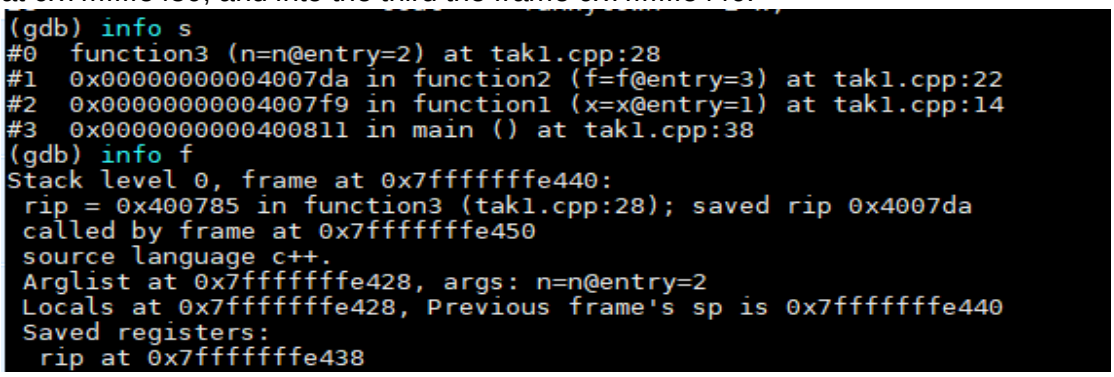
Ervin Hennrich

Project 1

1. The stack is very small, especially when not considering the local values. The stack on delmar has a maximum size of 8192 kbytes at maximum, including stored local values (found this value using ulimit -a). Using the info and x commands (info stack and info frame) I was able to get quite a bit of information about the stack frame. For one iteration I found the starting frame to be at address 0x7fffffffe470, then moving into my first function the next frame to be a 0x7fffffffe460, moving to second function the frame was at 0x7fffffffe450, and into the third the frame 0x7fffffffe440.



Then returning to function1, again with the frame at 0x7fffffffe460 and then back to main, with 0x7fffffffe470.

```
(gdb) info f
Stack level 0, frame at 0x7fffffffe440:
 rip = 0x400785 in function3 (tak1.cpp:28); saved rip 0x4007da
 called by frame at 0x7fffffffe450
 source language c++.
 Arglist at 0x7fffffffe428, args: n=n@entry=2
 Locals at 0x7fffffffe428, Previous frame's sp is 0x7fffffffe440
 Saved registers:
  rip at 0x7fffffffe438
(gdb) info s
#0  function3 (n=n@entry=2) at tak1.cpp:28
#1  0x00000000004007da in function2 (f=f@entry=3) at tak1.cpp:22
#2  0x00000000004007f9 in function1 (x=x@entry=1) at tak1.cpp:14
#3  0x0000000000400811 in main () at tak1.cpp:38
(gdb) c
Continuing.

Breakpoint 2, function1 (x=x@entry=1) at tak1.cpp:16
16              }
(gdb) info f
Stack level 0, frame at 0x7fffffffe460:
 rip = 0x4007f9 in function1 (tak1.cpp:16); saved rip 0x400811
 called by frame at 0x7fffffffe470
 source language c++.
 Arglist at 0x7fffffffe448, args: x=x@entry=1
 Locals at 0x7fffffffe448, Previous frame's sp is 0x7fffffffe460
 Saved registers:
  rip at 0x7fffffffe458
(gdb) info s
#0  function1 (x=x@entry=1) at tak1.cpp:16
#1  0x0000000000400811 in main () at tak1.cpp:38
```

This alone tells me that the stack frame itself is very small, here only occupying a space of 16 addresses.

```
(gdb) x /30bd $fp
0x7fffffffe458: 17      8       64      0       0       0       0       0
0x7fffffffe460: 0       0       0       0       0       0       0       0
0x7fffffffe468: 69      20      33      -9      -1      127     0       0
0x7fffffffe470: 0       0       0       0       0       0
```

In bytes the maximum value for each fame should be 2032(16*127) whereas in practice the values where significantly lower. Because variables were also sometimes stored within this space as well (like above where x is stored at 0x7fffffffe448 right next to the stating frame of my second function), leads me to believe that this value can and will be lower depending upon the instructions within the function. An expanded view shows that

```
(gdb) x /100db $sp
0x7fffffffe430: 0       0       0       0       0       0       0       0
0x7fffffffe438: -38     7       64      0       0       0       0       0
0x7fffffffe440: 0       0       0       0       0       0       0       0
0x7fffffffe448: -7      7       64      0       0       0       0       0
0x7fffffffe450: 64      -27     -1      -1      -1      127     0       0
0x7fffffffe458: 17      8       64      0       0       0       0       0
0x7fffffffe460: 0       0       0       0       0       0       0       0
0x7fffffffe468: 69      20      33      -9      -1      127     0       0
0x7fffffffe470: 0       0       0       0       0       0       0       0
0x7fffffffe478: 72      -27     -1      -1      -1      127     0       0
0x7fffffffe480: 0       0       0       0       1       0       0       0
0x7fffffffe488: 3       8       64      0       0       0       0       0
0x7fffffffe490: 0       0       0       0
```

```
(gdb) x /100db $sp
0x7fffffffe430: 0       0       0       0       0       0       0       0
0x7fffffffe438: -38     7       64      0       0       0       0       0
0x7fffffffe440: 0       0       0       0       0       0       0       0
0x7fffffffe448: -7      7       64      0       0       0       0       0
0x7fffffffe450: 64      -27     -1      -1      -1      127     0       0
0x7fffffffe458: 17      8       64      0       0       0       0       0
0x7fffffffe460: 0       0       0       0       0       0       0       0
0x7fffffffe468: 69      20      33      -9      -1      127     0       0
0x7fffffffe470: 0       0       0       0       0       0       0       0
0x7fffffffe478: 72      -27     -1      -1      -1      127     0       0
0x7fffffffe480: 0       0       0       0       1       0       0       0
0x7fffffffe488: 3       8       64      0       0       0       0       0
0x7fffffffe490: 0       0       0       0
```

function3 didn't need the full amount of space, so variables shared the space with it.

```
(gdb) x /10a $sp
0x7fffffffe430: 0x0     0x4007da <function2(int)+60>
0x7fffffffe440: 0x0     0x4007f9 <function1(int)+29>
0x7fffffffe450: 0x7fffffffe540  0x400811 <main()+14>
0x7fffffffe460: 0x0     0x7ffff7211445 <__libc_start_main+245>
0x7fffffffe470: 0x0     0x7fffffffe548
```

```
(gdb) x /10a $sp
0x7fffffffe430: 0x0     0x4007da <function2(int)+60>
0x7fffffffe440: 0x0     0x4007f9 <function1(int)+29>
0x7fffffffe450: 0x7fffffffe540  0x400811 <main()+14>
0x7fffffffe460: 0x0     0x7ffff7211445 <__libc_start_main+245>
0x7fffffffe470: 0x0     0x7fffffffe548
```

This again shows that the frames do not need the full 2032 bytes but will keep it. It seems that they need about half the amount,1016 bytes.

2.

```
#include <iostream>
#include <stdio.h>

using namespace std;

int main(){

        int locstat[10];

        printf("Address of locstat %p \n", &locstat); //address of first value
        printf("Address of last value in locstat is %p \n", &locstat[9]); //address of last value
        int* locdyn = new int[50];

        printf("Address of locdyn %p \n", &locdyn);
        delete[] locdyn;

}
```

Yes the size of the dynamic array is stored, in bytes, in an address before the dynamically allocated array. Through my testing I have found it to be stored numerous

times before the array. This does not appear to be true for the local array though. I tried setting the initial values of both the dynamic and static arrays and looked in the addresses before the address of the first element of the array. I used printf to display the address of the first element of each array (as well as the last) to make it easier.

```
Address of locstat 0x7fffffffe420
Address of last value in locstat is 0x7fffffffe444
Address of locdyn 0x7fffffffe418

Breakpoint 1, main () at p2.cpp:17
17              delete[] locdyn;
Missing separate debuginfos, use: debuginfo-install glibc-2.1
```

Because the dynamic array created a pointer, I looked for this value before both addresses but only found it before the address of the pointer (the address that holds the pointer value). Then using gdb's x I looked into the memory addresses arbitrarily before that value, usually around 256 less (if the starting address was 0x7fffffffe418 then I would start looking around 0x7fffffffe300). The x command will start at the next memory location every time it is run by default, so it's rather simple (hold down enter) to look through memory (you could also specify the number of memory locations to look through and do it all at once, but I found this to be clunky). I expected a value 4 times whatever the size specified (I checked sizeof(int) to confirm) and found exactly this value. Changing the size of the array many times (here 200), I was able to find this value before the first element every time. One of the most consistent ways to find this value was to look for the malloc instuction, the value was always very close to it.

```
0x7fffffffe350: 0x7c
(gdb)
0x7fffffffe358: 0x7fffffffe418
(gdb)
0x7fffffffe360: 0x602010
(gdb)
0x7fffffffe368: 0x602010
(gdb)
0x7fffffffe370: 0x602000
(gdb)
0x7fffffffe378: 0xc8
(gdb)
0x7fffffffe380: 0x0
(gdb)
0x7fffffffe388: 0x7ffff75b5760 <main_arena>
(gdb)
0x7fffffffe390: 0xc8
(gdb)
0x7fffffffe398: 0x400620 <_start>
(gdb)
0x7fffffffe3a0: 0x7fffffffe530
(gdb)
0x7fffffffe3a8: 0x0
(gdb)
0x7fffffffe3b0: 0x0
(gdb)
0x7fffffffe3b8: 0x7ffff727484c <malloc+76>
(gdb)
0x7fffffffe3c0: 0xc8
(gdb)
0x7fffffffe3c8: 0xc8
```

```
(gdb) x /d 0x7fffffffe350
0x7fffffffe350: 124
(gdb)
0x7fffffffe358: 140737488348184
(gdb)
0x7fffffffe360: 6299664
(gdb)
0x7fffffffe368: 6299664
(gdb)
0x7fffffffe370: 6299648
(gdb)
0x7fffffffe378: 200
(gdb)
0x7fffffffe380: 0
(gdb)
0x7fffffffe388: 140737343346528
(gdb)
0x7fffffffe390: 200
(gdb)
0x7fffffffe398: 4195872
(gdb)
0x7fffffffe3a0: 140737488348464
(gdb)
0x7fffffffe3a8: 0
(gdb)
0x7fffffffe3b0: 0
(gdb)
0x7fffffffe3b8: 140737339934796
(gdb)
0x7fffffffe3c0: 200
(gdb)
0x7fffffffe3c8: 200
(gdb)
0x7fffffffe3d0: 0
(gdb)
0x7fffffffe3d8: 140737349103309
(gdb)
0x7fffffffe3e0: 0
(gdb)
0x7fffffffe3e8: 0
(gdb)
0x7fffffffe3f0: 0
(gdb)
0x7fffffffe3f8: 140737349103561
(gdb)
0x7fffffffe400: 1
(gdb)
0x7fffffffe408: 4196188
(gdb)
0x7fffffffe410: 2
(gdb)
0x7fffffffe418: 6299664
```

3.

```cpp
#include <iostream>
#include <limits.h>
#include <stdlib.h>
#include <cmath>
using namespace std;

int main(){

	int x, y, div, addition, multiplication;

	cout << "Give me two numbers" << endl;
	cin >> x;
	cin >> y;

	if ((y>0) && (x > INT_MAX - y)){   //if x is bigger than int_max - y, then there would be an overflow (and everything else would break too)
		cout << "BAD!";
		return -1;
	}
	else if ((y<0) && (x < INT_MIN - y)){  //same thing as above but for the other end
		cout << "Underflow";
		return -1;
	}

	addition = x+y;
	cout << addition << endl;

	if ( (y && x != 0) && (abs(x) > INT_MAX / abs(y))){  //first check to make sure both x and y arnt 0, because that situation can never overflow (break). Then check that x is greater than int_max / y else there will be overflow
		cout<< "Overflow";                    //needed to use abs() because negatives were breaking it, need only one check this way as well
		return -1;
	}

	multiplication = x * y;
	cout << multiplication<< endl;

	if ( y == 0){                   //just need to check for divide by 0
		cout << "Divide by 0";
		return -1;
	}
	else{
		div = x / y;
		cout << div << endl;
	}


	return 0;
}
```

```
[ephb7d@delmar 3780]$ ./a.out
Give me two numbers
5000
5000
Addition:10000
Mult:25000000
Div:1
[ephb7d@delmar 3780]$
```

```
[ephb7d@delmar 3780]$ ./a.out
Give me two numbers
-500000
-500000
Addition:-1000000
Overflow[ephb7d@delmar 3780]$
```