

CS2263

Lab 4

Date: March 14th, 2024

Student:

Name: Will Ross

Number: #3734692

Email: will.ross@unb.ca

Due Date: March 18th, 2024

Contents

- [Lab 4 - Name - Studnet ID- CS2263](#)
 - [Contents](#)
 - [Exercise 1](#)
 - 1.1 Run the program ex1.c in the gdb debugger (see Laboratory #2 instructions). Set up the breakpoint at the function dummy_frame(). Run the program until the breakpoint.
 - A. The screenshot showing the output form the backtrace bt. How many frames are there on the memory stack?
 - B. The screenshot showing the output info frame 0. What are the frame boundaries of the main function? (Hint: compare the values under "Stack frame at" and "Called by frame at").
 - C. Are the addresses of array elements falling within the range of the main function frame?
 - 1.2 Modify the program ex1.c so that the array a[] is allocated on the heap (use malloc()). Set up the breakpoint at the function dummy_frame(). Run the program until the breakpoint.
 - A. The source code of the modified program
 - B. The screenshot showing the output form the backtrace bt. How many frames are there on the memory stack?
 - C. The screenshot showing the output info frame 0. What are the frame boundaries of the main function? (Hint: compare the values under "Stack frame at" and "Called by frame at").
 - D. Are the addresses of array elements falling within the range of the main function frame? Explain why.
 - [Exercise 2](#)
 - The source code of the modified program
 - The screenshot showing the output. Are you getting the same addresses for the new extended array? Explain why.
 - [Exercise 3](#)
 - Remove any calls to free() function (if you had any) form the program form Exercise 2 and then run it again under valgrind. to Show the complete output (program output, plus the

valgrind messages). For example:

- Modify the program from Exercise 3.1 to eliminate the memory leak. Run the program again under valgrind. Show the modified program source code and the complete output (program output, plus the valgrind messages, if any).

Lab 3

Exercise 1

1.1 Run the program ex1.c in the gdb debugger (see Laboratory #2 instructions). Set up the breakpoint at the function dummy_frame(). Run the program until the breakpoint.

1. gdb ex1
2. **break** dummy_frame
3. run
4. backtrace
5. select-frame 0
6. info frame

A. The screenshot showing the output form the backtrace bt. How many frames are there on the memory stack?

```
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ex1...
(gdb) break dummy_frame
Breakpoint 1 at 0x1400014a8
(gdb) run
Starting program: c:\Users\willr\Documents\GitHub\Cs2263\Labs\Lab4\ex1.exe
[New Thread 7352.0x2f3c]
main: a = 1 0000000005ffe80
main: a = 2 0000000005ffe84
main: a = 3 0000000005ffe88
main: a = 4 0000000005ffe8c
main: a = 5 0000000005ffe90

Thread 1 hit Breakpoint 1, 0x00007ff7952a14a8 in dummy_frame ()
(gdb) backtrace
#0 0x00007ff7952a14a8 in dummy_frame ()
#1 0x00007ff7952a1528 in main ()
(gdb) []
```

From the photo we see that we have two frames frame 1 and frame 0 on the memory stack.

B. The screenshot showing the output info frame 0. What are the frame boundaries of the main function? (Hint: compare the values under "Stack frame at" and "Called by frame at").

```
(gdb) break dummy_frame
Breakpoint 1 at 0x1400014a8
(gdb) run
Starting program: c:\Users\willr\Documents\GitHub\Cs2263\Labs\Lab4\ex1.exe
[New Thread 7352.0x2f3c]
main: a = 1 00000000005ffe80
main: a = 2 00000000005ffe84
main: a = 3 00000000005ffe88
main: a = 4 00000000005ffe8c
main: a = 5 00000000005ffe90

Thread 1 hit Breakpoint 1, 0x00007ff7952a14a8 in dummy_frame ()
(gdb) backtrace
#0 0x00007ff7952a14a8 in dummy_frame ()
#1 0x00007ff7952a1528 in main ()
(gdb) select-frame 0
(gdb) info frame
Stack level 0, frame at 0x5ffe60:
 rip = 0x7ff7952a14a8 in dummy_frame; saved rip = 0x7ff7952a1528
 called by frame at 0x5ffeb0
 Arglist at 0x5ffe50, args:
 Locals at 0x5ffe50, Previous frame's sp is 0x5ffe60
 Saved registers:
  rbp at 0x5ffe50, rip at 0x5ffe58
(gdb) []
```

Stack Frame at: 0x5ffe60 Called by Frame at: 0x5ffeb0

So the frame boundaries are 0x5ffeb0 -> 0x5ffe60

C. Are the addresses of array elements falling within the range of the main function frame?

```
(gdb) frame 1
#1 0x00007ff7952a1528 in main ()
(gdb) info frame
Stack level 1, frame at 0x5ffeb0:
 rip = 0x7ff7952a1528 in main; saved rip = 0x7ff7952a12ee
 caller of frame at 0x5ffe60
 Arglist at 0x5ffea0, args:
 Locals at 0x5ffea0, Previous frame's sp is 0x5ffeb0
 Saved registers:
  rbp at 0x5ffea0, rip at 0x5ffea8
(gdb) info registers
rax      0x1e      30
rbx      0x8       8
rcx      0xffffffff 4294967295
rdx      0x0       0
rsi      0x39      57
rdi      0xa84708 11028232
rbp      0x5ffea0 0x5ffea0
rsp      0x5ffe60 0x5ffe60
r8       0x7ff9a28ce7a0 140710150727584
r9       0x0       0
r10      0x0       0
r11      0x246     582
r12      0xa84740 11028288
r13      0x0       0
r14      0x0       0
r15      0x0       0
rip      0x7ff7952a1528 0x7ff7952a1528 <main+125>
eflags   0x202     [ IF ]
cs       0x33      51
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x53      83
gs       0x2b      43
(gdb) []
```

The addresses 0x5ffeb0 -> 0x5ffe60 are within the main functions stack as we can see from the screenshot above.

1.2 Modify the program ex1.c so that the array `a[]` is allocated on the heap (use `malloc()`). Set up the breakpoint at the function `dummy_frame()`. Run the program until the breakpoint.

A. The source code of the modified program

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

void dummy_frame()
{
    return;
}

int main(int argc, char * * argv)
{

    int i;
    int size = 5;

    int *a = (int*)malloc(size);

    //fill array with 1 to 5
    for(i = 0; i <= size; i++)
    {
        a[i] = i + 1;
    }

    dummy_frame();

    for (i=0; i< size; i++){
        printf("a[%d] = %d at address: %p \n", i, a[i], &a[i]);
    }

    free(a);
    return EXIT_SUCCESS;
}
```

B. The screenshot showing the output from the backtrace `bt`. How many frames are there on the memory stack?

```

c:\Users\willr\Documents\GitHub\Cs2263\Labs\Lab4>gdb ./ex1
GNU gdb (GDB) 14.1
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-w64-mingw32".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./ex1...
(gdb) break dummy_frame
Breakpoint 1 at 0x1400014a8
(gdb) run
Starting program: c:\Users\willr\Documents\GitHub\Cs2263\Labs\Lab4\ex1.exe
[New Thread 2536.0x48b0]

Thread 1 hit Breakpoint 1, 0x00007ff73b9a14a8 in dummy_frame ()
(gdb) backtrace
#0  0x00007ff73b9a14a8 in dummy_frame ()
#1  0x00007ff73b9a1511 in main ()
(gdb)

```

From the photo we see that we have two frames frame 1 and frame 0 on the memory stack.

C. The screenshot showing the output info frame 0. What are the frame boundaries of the main function? (Hint: compare the values under "Stack frame at" and "Called by frame at").

```

<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./ex1...
(gdb) break dummy_frame
Breakpoint 1 at 0x1400014a8
(gdb) run
Starting program: c:\Users\willr\Documents\GitHub\Cs2263\Labs\Lab4\ex1.exe
[New Thread 2536.0x48b0]

Thread 1 hit Breakpoint 1, 0x00007ff73b9a14a8 in dummy_frame ()
(gdb) backtrace
#0  0x00007ff73b9a14a8 in dummy_frame ()
#1  0x00007ff73b9a1511 in main ()
(gdb) select-frame 0
(gdb) info frame
Stack level 0, frame at 0x5ffe70:
 rip = 0x7ff73b9a14a8 in dummy_frame; saved rip = 0x7ff73b9a1511
 called by frame at 0x5ffeb0
 Arglist at 0x5ffe60, args:
 Locals at 0x5ffe60, Previous frame's sp is 0x5ffe70
 Saved registers:
  rbp at 0x5ffe60, rip at 0x5ffe68
(gdb)

```

Stack Frame at: 0x5ffe70 Called by Frame at: 0x5ffeb0

So the frame boundaries are 0x5ffeb0 -> 0x5ffe70

D. Are the addresses of array elements falling within the range of the main function frame? Explain why.

```

(gdb) select-frame 1
(gdb) info frame
Stack level 1, frame at 0x5ffeb0:
  rip = 0x7ff73b9a1511 in main; saved rip = 0x7ff73b9a12ee
  caller of frame at 0x5ffe70
  Arglist at 0x5ffea0, args:
  Locals at 0x5ffea0, Previous frame's sp is 0x5ffeb0
  Saved registers:
    rbp at 0x5ffea0, rip at 0x5ffea8
(gdb) info registers
rax             0x6             6
rbx             0x8             8
rcx             0x7ffe0380      2147353472
rdx             0x6             6
rsi             0x39            57
rdi             0xa546c8        10831560
rbp             0x5ffea0        0x5ffea0
rsp             0x5ffe70        0x5ffe70
r8              0x5             5
r9              0xa547b0        10831792
r10             0x0             0
r11             0x5ffa38        6289976
r12             0xa54700        10831616
r13             0x0             0
r14             0x0             0
r15             0x0             0
rip             0x7ff73b9a1511   0x7ff73b9a1511 <main+102>
eflags          0x202          [ IF ]
cs              0x33            51
ss              0x2b            43
ds              0x2b            43
es              0x2b            43
fs              0x53            83
gs              0x2b            43
(gdb) 

```

No, because of how malloc reserves memory. It reserves a user-defined chunk of memory and the address of the user-defined memory is higher than the addresses of the stack frame.

Exercise 2

The source code of the modified program

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

void dummy_frame()
{
    return;
}

int main(int argc, char * * argv)
{
    int i;
    int size = 5;
    int addedSize = 9;
    int *a;

    //call malloc
    a = (int *) malloc(size);
    printf("\nMalloc Values:\n\n");
    //fill array with 1 to 5
    for(i = 0; i <= size; i++)
    {
        a[i] = i + 1;
    }

    //print the original array
    for (i=0; i< size; i++)
    {
        printf("a[%d] = %d at address: %p \n", i, a[i], &a[i]);
    }

    printf("\nRealloc Values:\n\n");
    //called realloc getting the original values + our new size
    a = (int *) realloc(a, addedSize);

    //add more values to the array
    for(i = size; i < addedSize; i++){
        a[i] = i + 1;
    }

    //print all values
    for (i=0; i< addedSize; i++)
    {
        printf("a[%d] = %d at address: %p \n", i, a[i], &a[i]);
    }
}
```



```
    free(a);  
    return EXIT_SUCCESS;  
}
```

The screenshot showing the output. Are you getting the same addresses for the new extended array? Explain why.

```
c:\Users\willr\Documents\Github\Cs2263\Labs\Lab4>cd "c:\Users\willr\Documents\Github\Cs2263\Labs\Lab4\" && gcc ex2.c -o ex2 && "c:\Users\willr\Documents\Github\Cs2263\Labs\Lab4\ex2  
x2  
  
Malloc Values:  
a[0] = 1 at address: 00000173fc496a60  
a[1] = 2 at address: 00000173fc496a64  
a[2] = 3 at address: 00000173fc496a68  
a[3] = 4 at address: 00000173fc496a6c  
a[4] = 5 at address: 00000173fc496a70  
  
Realloc Values:  
a[0] = 1 at address: 00000173fc496a60  
a[1] = 2 at address: 00000173fc496a64  
a[2] = 3 at address: 00000173fc496a68  
a[3] = 4 at address: 00000173fc496a6c  
a[4] = 5 at address: 00000173fc496a70  
a[5] = 6 at address: 00000173fc496a74  
a[6] = 7 at address: 00000173fc496a78  
a[7] = 8 at address: 00000173fc496a7c  
a[8] = 9 at address: 00000173fc496a80  
  
c:\Users\willr\Documents\Github\Cs2263\Labs\Lab4>
```

Due to us using the realloc function for our array it tells our system to keep the same addresses when using the function to reserve memory space. So when we call realloc to add more elements onto our array it will find our array and build off of it. In our case and it being an integer array the memory pointer increments in 4. Also it isn't always guaranteed that it will continue off the last memory pointer of the original array, if there is allocated memory near the array it will have to skip over it and continue at a later memory location, this is rare though.

Exercise 3

Remove any calls to `free()` function (if you had any) from the program from Exercise 2 and then run it again under `valgrind`. to Show the complete output (program output, plus the `valgrind` messages). For example:

```
valgrind ./a.out
```

Source Code:

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

void dummy_frame()
{
    return;
}

int main(int argc, char * * argv)
{

    int i;
    int size = 5;
    int addedSize = 8;
    int *a;

    //call malloc
    a = malloc(size * sizeof(int));

    printf("\nMalloc Values:\n\n");

    //fill array with 1 to 5
    for(i = 0; i < size; i++)
    {
        a[i] = i + 1;
    }

    //print the original array
    for (i=0; i< size; i++)
    {
        printf("a[%d] = %d at address: %p \n", i, a[i], &a[i]);
    }

    printf("\nRealloc Values:\n\n");

    //called realloc getting the original values + our new size
    int *b = realloc(a, addedSize * sizeof(int));
```

```

    a = b;

    //add more values to the array
    for(i = size; i < addedSize; i++){
        a[i] = i + 1;
    }

    //print all values
    for (i=0; i< addedSize; i++)
    {
        printf("a[%d] = %d at address: %p \n", i, a[i], &a[i]);
    }
    free(a);
    return EXIT_SUCCESS;
}

```

The output from running ./ex3

```

[q3d5k@gc112m36 Lab4]$ cd "/home1/ugrads/q3d5k/github/Cs2263/Labs/Lab4/" && gcc
ex3.c -o ex3 && "/home1/ugrads/q3d5k/github/Cs2263/Labs/Lab4/"ex3

```

Malloc Values:

```

a[0] = 1 at address: 0x7692a0
a[1] = 2 at address: 0x7692a4
a[2] = 3 at address: 0x7692a8
a[3] = 4 at address: 0x7692ac
a[4] = 5 at address: 0x7692b0

```

Realloc Values:

```

a[0] = 1 at address: 0x7696d0
a[1] = 2 at address: 0x7696d4
a[2] = 3 at address: 0x7696d8
a[3] = 4 at address: 0x7696dc
a[4] = 5 at address: 0x7696e0
a[5] = 6 at address: 0x7696e4
a[6] = 7 at address: 0x7696e8
a[7] = 8 at address: 0x7696ec

```

The output from running valgrind ./ex3

```

[q3d5k@gc112m36 Lab4]$ valgrind ./ex3
==555178== Memcheck, a memory error detector
==555178== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==555178== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==555178== Command: ./ex3

```

```
==555178==
```

```
Malloc Values:
```

```
a[0] = 1 at address: 0x4a74040
a[1] = 2 at address: 0x4a74044
a[2] = 3 at address: 0x4a74048
a[3] = 4 at address: 0x4a7404c
a[4] = 5 at address: 0x4a74050
```

```
Realloc Values:
```

```
a[0] = 1 at address: 0x4a744e0
a[1] = 2 at address: 0x4a744e4
a[2] = 3 at address: 0x4a744e8
a[3] = 4 at address: 0x4a744ec
a[4] = 5 at address: 0x4a744f0
a[5] = 6 at address: 0x4a744f4
a[6] = 7 at address: 0x4a744f8
a[7] = 8 at address: 0x4a744fc
```

```
==555178==
```

```
==555178== HEAP SUMMARY:
```

```
==555178==      in use at exit: 32 bytes in 1 blocks
==555178==    total heap usage: 3 allocs, 2 frees, 1,076 bytes allocated
```

```
==555178==
```

```
==555178== LEAK SUMMARY:
```

```
==555178==    definitely lost: 32 bytes in 1 blocks
==555178==    indirectly lost: 0 bytes in 0 blocks
==555178==    possibly lost: 0 bytes in 0 blocks
==555178==    still reachable: 0 bytes in 0 blocks
==555178==           suppressed: 0 bytes in 0 blocks
```

```
==555178== Rerun with --leak-check=full to see details of leaked memory
```

```
==555178==
```

```
==555178== For lists of detected and suppressed errors, rerun with: -s
```

```
==555178== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
[q3d5k@gc112m36 Lab4]$
```

Modify the program from Exercise 3.1 to eliminate the memory leak. Run the program again under valgrind. Show the modified program source code and the complete output (program output, plus the valgrind messages, if any).

Source Code:

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

void dummy_frame()
{
    return;
}
```

```
int main(int argc, char * * argv)
{

    int i;
    int size = 5;
    int addedSize = 8;
    int *a;

    //call malloc
    a = malloc(size * sizeof(int));

    printf("\nMalloc Values:\n\n");

    //fill array with 1 to 5
    for(i = 0; i < size; i++)
    {
        a[i] = i + 1;
    }

    //print the original array
    for (i=0; i< size; i++)
    {
        printf("a[%d] = %d at address: %p \n", i, a[i], &a[i]);
    }

    printf("\nRealloc Values:\n\n");

    //called realloc getting the original values + our new size
    int *b = realloc(a, 8 * sizeof(int));
    a = b;

    //add more values to the array
    for(i = size; i < 8; i++){
        a[i] = i + 1;
    }

    //print all values
    for (i=0; i< 8; i++)
    {
        printf("a[%d] = %d at address: %p \n", i, a[i], &a[i]);
    }
    free(a);
    return EXIT_SUCCESS;
}
```

The output from running ./ex3

```
[q3d5k@gc112m36 Lab4]$ cd "/home1/ugrads/q3d5k/github/Cs2263/Labs/Lab4/" && gcc  
ex3.c -o ex3 && "/home1/ugrads/q3d5k/github/Cs2263/Labs/Lab4/"ex3
```

Malloc Values:

```
a[0] = 1 at address: 0x1b2e2a0  
a[1] = 2 at address: 0x1b2e2a4  
a[2] = 3 at address: 0x1b2e2a8  
a[3] = 4 at address: 0x1b2e2ac  
a[4] = 5 at address: 0x1b2e2b0
```

Realloc Values:

```
a[0] = 1 at address: 0x1b2e6d0  
a[1] = 2 at address: 0x1b2e6d4  
a[2] = 3 at address: 0x1b2e6d8  
a[3] = 4 at address: 0x1b2e6dc  
a[4] = 5 at address: 0x1b2e6e0  
a[5] = 6 at address: 0x1b2e6e4  
a[6] = 7 at address: 0x1b2e6e8  
a[7] = 8 at address: 0x1b2e6ec  
[q3d5k@gc112m36 Lab4]$
```

The fixed memory link output when running `valgrind ./ex3`

```
[q3d5k@gc112m36 Lab4]$ valgrind ./ex3  
==556502== Memcheck, a memory error detector  
==556502== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.  
==556502== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info  
==556502== Command: ./ex3  
==556502==
```

Malloc Values:

```
a[0] = 1 at address: 0x4a74040  
a[1] = 2 at address: 0x4a74044  
a[2] = 3 at address: 0x4a74048  
a[3] = 4 at address: 0x4a7404c  
a[4] = 5 at address: 0x4a74050
```

Realloc Values:

```
a[0] = 1 at address: 0x4a744e0  
a[1] = 2 at address: 0x4a744e4  
a[2] = 3 at address: 0x4a744e8  
a[3] = 4 at address: 0x4a744ec  
a[4] = 5 at address: 0x4a744f0  
a[5] = 6 at address: 0x4a744f4  
a[6] = 7 at address: 0x4a744f8
```

```
a[7] = 8 at address: 0x4a744fc
==556502==
==556502== HEAP SUMMARY:
==556502==      in use at exit: 0 bytes in 0 blocks
==556502==    total heap usage: 3 allocs, 3 frees, 1,076 bytes allocated
==556502==
==556502== All heap blocks were freed -- no leaks are possible
==556502==
==556502== For lists of detected and suppressed errors, rerun with: -s
==556502== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[q3d5k@gc112m36 Lab4]$
```