ECE 3220 Lab5
Debugging Using GDB Debugger

Marshall Lindsay
mblgh6@mail.missouri.edu
14211662

2/28/2017

Objective:

The objective of this lab was to introduce the GDB debugger and to become familiar with some of its functionality.

Implementation:

The functionality of the GDB debugger was demonstrated on several example files that contained differing errors. By using breakpoints and through careful analysis of the code each error was found and fixed in turn.

Results and Discussion:

To better illustrate the process and results of the lab, each example file will be discussed in turn. A detailed explanation will be given for the first example file while the rest will be summarized to contain only important observations and results.

Lab5_ex1.c - The debugging of this file began with the compilation using gcc and the '-g' argument. After attempting to run the program in the GDB debugger, a segmentation fault was encountered. This error is shown in detail in Figure1. By using the 'backtrace' command, a better idea of the location and context of the error was obtained. After discovering the error was found on line 10, the 'list' command was used to view the source code. From the source code it was easy to see that the segmentation fault was occurring when attempting to access the char* fileName. To double check my suspicion, a breakpoint was set right before the value of the file name was assigned to fileName. After the program should have assigned a value to fileName, the 'print' command was used in an attempt to view the contents of the variable. The results of this print are visible in Figure1. Because I was unable to access the contents of the memory location pointed to by fileName, there must have been an error when reserving memory for the file name. By looking back at the source code it was easy to see that no memory was allocated to hold the file name and that only the pointer was initialized. Given the context of the program, the solution was to initialize fileName as a character array with a finite value, rather than just a pointer. This solution can be seen in Figure2. Figure3 shows a successful run of the Lab5_ex1.c.

Lab5_ex2.c - The debugging of this code began the same way. A segmentation fault was found without any user input on line 5. By listing the source code it was easy to see the cause of the segmentation fault was due to attempting to write data to memory that was not in the scope of the program. By initializing the data to be used by the program (through an indexed char array in this case) and setting the initial values, the segmentation fault was avoided. Figure4 shows the debugging process in GDB. Figure5 shows the correction to Lab5_ex2.c. Figure6 shows a successful run of Lab5_ex2.c.

Lab5_ex3.c - The debugging of this program the same way. The user was prompted for an input in the form of a number. Once the input was given a segmentation fault was encountered. An attempt to backtrace the error lead to an infinite loop, which lead me to believe invalid memory access was not the issue in this example. By listing the program source code and looking at the line where the error occurred (line 5) it was easy to pinpoint the issue. The purpose for this code was to return the factorial of a number using recursion. However the exit case (boundary case?) was left out of the recursion loop leading to an infinite loop. By adding an IF ELSE statement containing the exit case, the segmentation fault was avoided. Figures 7 - 9 show the debugging process, error correction and successful execution respectively.

Lab5_ex4.c - The debugging for this program began the same way as before. When the program was ran in GDB a segmentation fault was found on line 13. When the program source code was listed, it seemed as though the problem came from attempted to copy strings of unequal size. When the strings were adjusted to be the same size the error persisted. After looking up the function 'strcpy' the error became apparent. The declaration of strcpy is: `char *strcpy(char *dest, const char *src).` When we attempted to call this function in Lab5_ex4.c we called it with the parameters of (const char*, char*). This lead to the segmentation fault when attempting to write to a constant string. While the different string sized may have not lead to the segmentation fault (unconfirmed), the fix remains. Even if it is valid, it is bad programming practice. Figures 10 - 12 show the debugging process, the error correction and successful execution respectively.

Lab5_ex5.c - This program does not contain an error detectable by the compiler or the debugger. Though when the source code is inspected, the free() command is used in an unconventional way. If this was intended to be the error, it is not registering as one. Figure13 shows successful execution of Lab5_ex5.c.

Lab5_ex6.c - This program had errors during the compilation. It was missing a library and was not using the malloc function correctly. Both of these errors was caught by gcc and were fixed prior to the GDB debugger. Once the program was compiled successfully, it ran without any more errors. Figure14 shows successful execution of Lab5_ex6.c.

Lab5_ex7.c - This program successfully compiled first try. When the program was ran in GDB, a segmentation fault was found on line 7. Backtrace confirmed this location. When the program source code was listed, the problem became very apparent. Line 7 attempts to dereference a pointer to void in order to view an integer value contained in the variable x. By fixing the pointer to contain the address of x, proper dereferencing was obtained. Figures 15 - 17 show the debugging process, the error correction and the successful execution of Lab5_ex7.c respectively.

Lab5_ex8.c - This program had an error when attempting to compile. The function malloc was used incorrectly and was fixed. After that error was fixed the program compiled successfully. When the program was ran through GDB, a segmentation fault was found on line 9. Backtrace confirmed the origin of the error. When the source code was viewed with 'list' it was easy to pinpoint the error. In an attempt to obtain a string from the user, the scanf function tries to read an inputted string to uninitialized memory. There are several ways to fix this error (the best of which use a different function for user input) however the easiest was used in this case. The segmentation fault was fixed by creating a sort of fixed 'buffer' for the user's input to reside until it it placed in a dynamically allocated array. This buffer was made a size of 10 for this example. Figures 18 - 20 show the debugging process, the error correction and the successful execution of Lab5_ex8.c respectively.

Conclusion:

The goal for this lab was to become familiar with the GDB debugger and some of its functionality. Through the use of different example files containing a multitude of errors, this goal was obtained. By the end of this lab a new familiarity with debugging and, more specifically, the GDB debugger was obtained.

Figures:

```
(gdb) run
Starting program: D:\school\ece3220\lab5/ex1.exe
[New Thread 8108.0x2568]
[New Thread 8108.0x37c4]
Specify file name you would like to print to:
File

Program received signal SIGSEGV, Segmentation fault.
0x76283c93 in msvcrt!_fsopen () from C:\WINDOWS\SysWOW64\msvcrt.dll
(gdb) backtrace
#0  0x76283c93 in msvcrt!_fsopen () from C:\WINDOWS\SysWOW64\msvcrt.dll
#1  0xdf0ffc50 in ?? ()
#2  0x004012c0 in _mingw32_init_mainargs ()
#3  0x004012c0 in _mingw32_init_mainargs ()
#4  0x00354000 in ?? ()
#5  0x76283d12 in msvcrt!fopen () from C:\WINDOWS\SysWOW64\msvcrt.dll
#6  0x00401482 in main () at ex1.c:10
(gdb) list
1       #include <stdio.h>
2
3       int main()
4       {
5               char* fileName;
6               printf("Specify file name you would like to print to: \n");
7               scanf("%s",&fileName);
8
9
10              FILE* file1 = fopen(fileName, "a+");
(gdb) breakpoint 4
Undefined command: "breakpoint".  Try "help".
(gdb) break 4
Breakpoint 1 at 0x40144e: file ex1.c, line 4.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y

error return ../../gdb-7.6.1/gdb/windows-nat.c:1275 was 5
Starting program: D:\school\ece3220\lab5/ex1.exe
[New Thread 14232.0x13e0]
[New Thread 14232.0x328]

Breakpoint 1, main () at ex1.c:6
6               printf("Specify file name you would like to print to: \n");
(gdb)
(gdb) s
Specify file name you would like to print to:
7               scanf("%s",&fileName);
(gdb) s
File
10              FILE* file1 = fopen(fileName, "a+");
(gdb) print filename
No symbol "filename" in current context.
(gdb) filename
Undefined command: "filename".  Try "help".
(gdb) p fileName
$1 = 0x656c6946 <Address 0x656c6946 out of bounds>
(gdb) p *fileName
Cannot access memory at address 0x656c6946
(gdb) p &fileName
$2 = (char **) 0x61ff28
```

Figure1: Screenshot of the debugging process for Lab5_ex1.c

```c
#include <stdio.h>

int main()
{
        char fileName[10];
        printf("Specify file name you would like to print to: \n");
        scanf("%s",&fileName);


        FILE* file1 = fopen(fileName, "a+");

        char c;
        while ((c=getchar()) != EOF){
                fprintf(file1, &c);
        }

        fclose(file1);

        printf("CTRL+d is a correct ending");

        return 0;
}
```

Figure2: Screenshot showing the correction of the segmentation fault error

```
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 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 "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from D:\school\ece3220\lab5\ex1.exe...done.
(gdb) run
Starting program: D:\school\ece3220\lab5/ex1.exe
[New Thread 5828.0x2154]
[New Thread 5828.0x1a38]
Specify file name you would like to print to:
File
Test
this
file
^Z
CTRL+d is a correct ending[Inferior 1 (process 5828) exited normally]
(gdb)
```

Figure3: Screenshot showing successful execution of Lab5_ex1.c

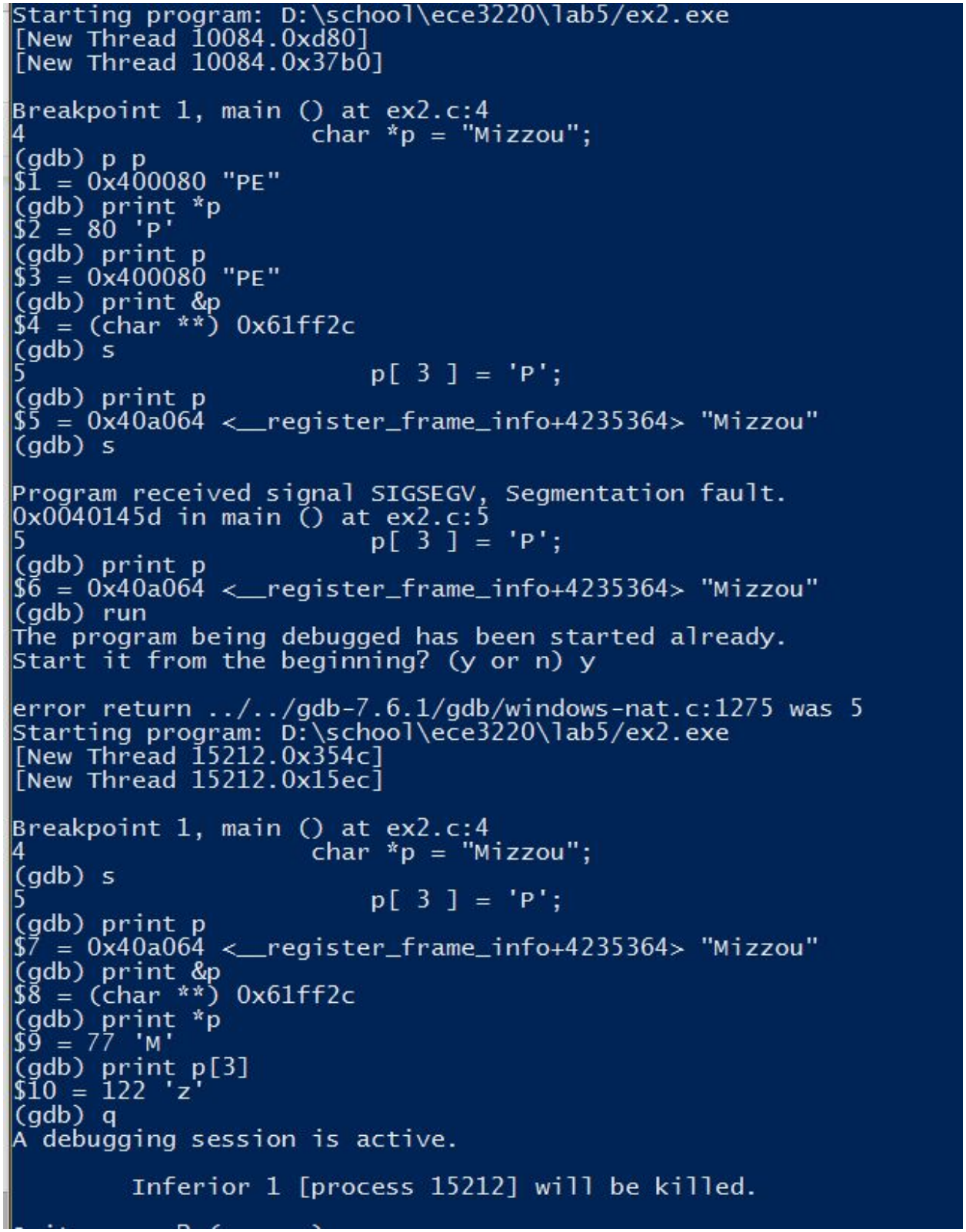

Figure4: Screenshot showing the debugging process for Lab5_ex2.c

```c
#include <stdio.h>
void main()
{
            char p[7] = {'M', 'I', 'Z', 'Z', 'O', 'U', '\0'};
                p[3] = 'P';
                    printf("%s",p);
}
```

Figure5: Screenshot showing the creation of an array to avoid the segmentation fault

```
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 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 "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from D:\school\ece3220\lab5\ex2.exe...done.
(gdb) run
Starting program: D:\school\ece3220\lab5/ex2.exe
[New Thread 10584.0x2468]
[New Thread 10584.0x31b0]
MIZPOU[Inferior 1 (process 10584) exited with code 06]
(gdb) q
```

Figure6: Screenshot showing a successful run of the Lab5_ex2.c

```
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 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 "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from D:\school\ece3220\lab5\ex3.exe...done.
(gdb) run
Starting program: D:\school\ece3220\lab5/ex3.exe
[New Thread 15500.0x37f0]
[New Thread 15500.0x2058]
Enter the number
5

Program received signal SIGSEGV, Segmentation fault.
0x0040144c in factorial (num=-65139) at ex3.c:5
5                   return (num * factorial( num - 1 ));
(gdb) list
1       #include <stdio.h>
2
3       int factorial ( int num )
4       {
5                   return (num * factorial( num - 1 ));
6       }
7
8       int main(void)
9       {
10                  int no,ans;
(gdb) list
11                      printf("Enter the number \n");
12                      scanf( "%d", &no );
13                          ans = factorial( no );
14                              printf("The factorial is %d ", ans );
15                                  return 0;
16      }
(gdb)
```

Figure7: Screenshot showing the debugging process for Lab5_ex3.c

```c
#include <stdio.h>

int factorial ( int num )
{
        if(num == 1){
                return(num);
        }else{
            return (num * factorial( num - 1 ));
        }
}

int main(void)
{
            int no,ans;
                printf("Enter the number \n");
                    scanf( "%d", &no );
                        ans = factorial( no );
                            printf("The factorial is %d ", ans );
                                return 0;
}
```

Figure8: Screenshot showing the addition of an IF ELSE to eliminate an infinite loop

```
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 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 "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from D:\school\ece3220\lab5\ex3.exe...done.
(gdb) run
Starting program: D:\school\ece3220\lab5/ex3.exe
[New Thread 12596.0x20e0]
[New Thread 12596.0x3ee4]
Enter the number
5
The factorial is 120 [Inferior 1 (process 12596) exited normally]
(gdb)
```

Figure9: Screenshot showing a successful run and proper output of Lab5_ex3.c

```
(gdb) run
Starting program: D:\school\ece3220\lab5/ex4.exe
[New Thread 14176.0x10ec]
[New Thread 14176.0x16c4]

Program received signal SIGSEGV, Segmentation fault.
0x76ef60a9 in strcat () from C:\WINDOWS\SysWOW64\msvcrt.dll
(gdb) backtrace
#0  0x76ef60a9 in strcat () from C:\WINDOWS\SysWOW64\msvcrt.dll
#1  0x004012c0 in _mingw32_init_mainargs ()
#2  0x004014a9 in main () at ex4.c:13
(gdb) list
1
2       #include<stdio.h>
3       #include<string.h>
4       void main()
5       {
6                   int i=0;
7                       char arr1[ 15 ] = "Trumantiger";
8                           char arr2[ 20 ];
9                               while(i<20){
10                                              arr2[ i ] = 'A';
(gdb)
11                                              ++i;
12                                                  }
13                                      strcpy( arr1 , arr2 );
14                                          printf("%s",arr1);
15      }
(gdb)
Line number 16 out of range; ex4.c has 15 lines.
(gdb)
```

Figure10: Screenshot showing the debugging process for Lab5_ex4.c

```c
#include<stdio.h>
#include<string.h>
void main()
{
            int i=0;
                char arr1[ 15 ] = "Trumantiger";
                    char arr2[ 15 ];
                        while(i<15){
                                                arr2[ i ] = 'A';
                                                ++i;
                                                    }
                                    strcpy( arr2 , arr1 );
                                        printf("%s",arr1);
}
```

Figure11: Screenshot showing the error correction for Lab5_ex4.c

```
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 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 "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from D:\school\ece3220\lab5\ex4.exe...done.
(gdb) run
Starting program: D:\school\ece3220\lab5/ex4.exe
[New Thread 12080.0x3b0c]
[New Thread 12080.0x3300]
Trumantiger[Inferior 1 (process 12080) exited with code 013]
(gdb)
```

Figure12: Screenshot showing a successful run of Lab5_ex4.c

```
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 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 "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from D:\school\ece3220\lab5\ex5.exe...done.
(gdb) run
Starting program: D:\school\ece3220\lab5/ex5.exe
[New Thread 10912.0x3620]
[New Thread 10912.0x1a0c]
hello[Inferior 1 (process 10912) exited with code 05]
(gdb)
```

Figure13: Screenshot showing successful run of Lab5_ex5.c

```
Reading symbols from D:\school\ece3220\lab5\ex6.exe...done.
(gdb) run
Starting program: D:\school\ece3220\lab5/ex6.exe
[New Thread 13636.0x3e2c]
[New Thread 13636.0x11dc]
Character: 3 and a
Character: 2 and a
Character: 1 and a
 main:00401440
 ch:00AC1708
 ch+1:00AC1709
 ch+2:00AC170A
 [Inferior 1 (process 13636) exited with code 060]
(gdb) list
1       #include <stdio.h>
2       #include <malloc.h>
3       int main(void)
4       {
5               char *ch;
6               ch=malloc(sizeof(char)*2);
7               ch[1]='3';
8               ch[2]='a';
9               while (ch[1] > '0')
10              {
(gdb) list
11                      printf("Character: %c and %c\n", ch[1], ch[2]);
12                      ch[1]--;
13              }
14              printf(" main:%p\n ch:%p\n ch+1:%p\n ch+2:%p\n ",&main,ch,ch+1,ch+2);
15              return ch[1];
16      }
17
(gdb)
```

Figure14: Screenshot showing successful run of Lab5_ex6.c (Note: the list command was used post execution for verification)

```
(gdb) run
Starting program: D:\school\ece3220\lab5/ex7.exe
[New Thread 5720.0x2bc]
[New Thread 5720.0x2e98]

 The value of x is = 1309
Program received signal SIGSEGV, Segmentation fault.
0x00401476 in main () at ex7.c:7
7               printf("\n The pointer points to the value = %d",*p);
(gdb) backtrace
#0  0x00401476 in main () at ex7.c:7
(gdb) list
2       int main()
3       {
4               int x=1309;
5               int *p = NULL;
6               printf("\n The value of x is = %d",x);
7               printf("\n The pointer points to the value = %d",*p);
8               return 1;
9       }
(gdb) list
Line number 10 out of range; ex7.c has 9 lines.
(gdb)
```

Figure15: Screenshot showing the debugging process for Lab5_ex7.c

```c
#include<stdio.h>
int main()
{
        int x=1309;
        int *p = &x;
        printf("\n The value of x is = %d",x);
        printf("\n The pointer points to the value = %d",*p);
        return 1;
}
```

Figure16: Screenshot showing the error correction for Lab5_ex7.c

```
Reading symbols from D:\school\ece3220\lab5\ex7.exe...done.
(gdb) run
Starting program: D:\school\ece3220\lab5/ex7.exe
[New Thread 13724.0x2e90]
[New Thread 13724.0x1204]

 The value of x is = 1309
 The pointer points to the value = 1309[Inferior 1 (process 13724) exited with code 01]
(gdb)
```

Figure17: Screenshot showing the successful run of Lab5_ex7.c

```
(gdb) run
Starting program: D:\school\ece3220\lab5/ex8.exe
[New Thread 11396.0x1044]
[New Thread 11396.0x38d8]
Enter the input string
String

Program received signal SIGSEGV, Segmentation fault.
0x76eeeeab in vswprintf () from C:\WINDOWS\SysWOW64\msvcrt.dll
(gdb) backtrace
#0  0x76eeeeab in vswprintf () from C:\WINDOWS\SysWOW64\msvcrt.dll
#1  0x76eefab8 in vswprintf () from C:\WINDOWS\SysWOW64\msvcrt.dll
#2  0x76ee7c43 in wscanf_s_l () from C:\WINDOWS\SysWOW64\msvcrt.dll
#3  0x76ee4ec8 in scanf () from C:\WINDOWS\SysWOW64\msvcrt.dll
#4  0x0040146e in main () at ex8.c:9
(gdb) list
1       #include<stdio.h>
2       #include<string.h>
3       #include<malloc.h>
4       int main(void)
5       {
6               char *s, *dyn_s;
7               int ln;
8               printf("Enter the input string\n");
9               scanf("%s",s);
10              ln = strlen(s);
(gdb) list
11              dyn_s = (char*)malloc(strlen(s)+1);
12              dyn_s = s;
13              dyn_s[strlen(s)]='\0';
14              printf(dyn_s);
15              return 0;
16      }
17
(gdb) list
Line number 18 out of range; ex8.c has 17 lines.
(gdb)
```

Figure18: Screenshot showing the debugging process for Lab5_ex8.c

```c
#include<stdio.h>
#include<string.h>
#include<malloc.h>
int main(void)
{
        char s[10], *dyn_s;
        int ln;
        printf("Enter the input string\n");
        scanf("%s",s);
        ln = strlen(s);
        dyn_s = (char*)malloc(strlen(s)+1);
        dyn_s = s;
        dyn_s[strlen(s)]='\0';
        printf(dyn_s);
        return 0;
}
```

Figure19: Screenshot showing the error correction for Lab5_ex8.c

```
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from D:\school\ece3220\lab5\ex8.exe...done.
(gdb) run
Starting program: D:\school\ece3220\lab5/ex8.exe
[New Thread 14176.0x3a24]
[New Thread 14176.0x1c34]
Enter the input string
String
String[Inferior 1 (process 14176) exited normally]
(gdb)
```

Figure20: Screenshot showing successful execution of Lab5_ex8.c

Figure21: Screenshot showing the commits for my lab5 repository