CYB220 Lab 8 – GDB

**Due: Tue Dec 3rd, 2024, 11:59 pm.**

**Points: 50 pts**

**Turn in: this report.**

Tori Overholtzer

Step 1: compile the program with “-g” flag to instrument useful data for the gdb debugger.

(Two files provided for this lab, so you don’t need to do this step.)

* g++ -g -o program-g++-dbg Score\_system\_new.cpp

(version 1: compiled with g++)

* clang++ -g -o program-clang-dbg Score\_system\_new.cpp (version 2: compiled with clang++)

A computer screen shot of a program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Step 2: Run both versions, when the score\_system program asking for comments, enter EXACT ten “A”s

Based on the average score, final grade is: F

looks good? (Yes or No)AAAAAAAA

Comments - Looks good? - AAAAAAAA

**Program-clang-dbg**

**A screenshot of a computer program

Description automatically generated**

**A computer screen shot of a program

Description automatically generated**

**Program-g++-dbg**

**A screenshot of a computer program

Description automatically generated**

**A computer screen shot of a program

Description automatically generated**

Step 3: Practice using gdb to examine variable values and their locations.

PROGRAM-CLANG-GDB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable name** | **Type** | **Location (Mem Address)** | **Value before overflow\*** | **Value after overflow\*** |
| scores[10] | int array | 0x7fffffffdd80 | {1651076199,779647075,1600677166,1819242352,-8656,32767,-137661553,32767,0,0} | {1,2,3,4,5,6,7,8,9,10} |
| name[100] | char array | 0x7fffffffdd10 | “abc”, “\000” repeated 96 times | Tori ‘\000’ repeated 94 times |
| number\_or\_score | int | 0x7fffffffdd0c | 0 | 10 |
| average | double | 0x7fffffffdd00 | 0 | 5 |
| sum | int | 0x7fffffffdcfc | 0 | 1094795585 |
| grade | char | 0x7fffffffdcfb | 88 ‘X’ | 65 A |
| Comments[5] | char array | 0x7fffffffdcf6 | NONE | AAAAA |

PROGRAM-G++-GDB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable name** | **Type** | **Location (Mem Address)** | **Value before overflow\*** | **Value after overflow\*** |
| scores[10] | int array | 0x7fffffffdd10 | {-8472, 32767,-269950720, -771510246, -8880, 32767, -140494144, 32767,4607, 0} | {1,2,3,4,5,6,7,8,9,10} |
| name[100] | char array | 0x7fffffffdd40 | “abc”, “\000” repeats 96 times | “AAAAA”, “\000” repeats 94 times |
| number\_or\_score | int | 0x7fffffffdcfc | 0 | 10 |
| average | double | 0x7fffffffdd08 | 0 | 5 |
| sum | int | 0x7fffffffdd00 | 0 | 55 |
| grade | char | 0x7fffffffdcf7 | 88 ‘X’ | 70 ‘F’ |
| Comments[5] | char array | 0x7fffffffdd3b | NONE | AAAAA |

----------------------Experiment #1----------------------

Debug/run the clang version with gdb: **gdb program-clang-dbg**

A screenshot of a computer program

Description automatically generated

Once gdb started, the first thing to do is to set up break points. Your program will stop before executing the code at the break points.

gdb commands: **break or b**

eg: break <line #> or b <function number>

break 18

A screenshot of a computer screen

Description automatically generated

b main

A screenshot of a computer screen

Description automatically generated

For the location of the variables

Because all variables are local variables (no dynamic variables), we can set up a break point at the place where all variables have been declared.

**break 18** 🡪 set up a break point on line 18

A screenshot of a computer screen

Description automatically generated

To check the variable value before the buffer overflow happens, where should we set the break point?

* at the line to get user input to comments (cin >> comments;)
* Can you find the line number and set break point there? (“list” command may be helpful) // <https://sourceware.org/gdb/current/onlinedocs/gdb.html/List.html>

A computer screen shot of text

Description automatically generated

A computer screen shot of text

Description automatically generated

**B.** After setting up the break points, now run the program.

gdb command: **run or r**

**A computer screen shot of a program code

Description automatically generated**

It should run the program and stop at the first break points (or stop and ask for user input as the program executes).

If you want to do line by line debug, use command **next (or n)**

A computer screen shot of a program

Description automatically generated

If you want to execute the program until next break point, use command **continue (or c)**

A computer screen shot of a program code

Description automatically generated

At break point #1 (break linenum #18), we can take a look at the locations of each local variable.

For example:   
print (or p) &scores 🡪 gives the address of the scores array.

A computer screen shot of a program code

Description automatically generated

print &average 🡪 gives the address of the average variable.

A screen shot of a computer code

Description automatically generated

Or use display, eg. display scores 🡪 display the value stored in scores

A black background with white text

Description automatically generated

At break point #2, we can examine the variable values by using “print” command.

For example: print scores 🡪 prints the scores array’s elements

A screen shot of a computer

Description automatically generated

print sum 🡪 prints the sum variable’s value

A computer screen with blue text

Description automatically generated

Once fill in the “value before overflow” column, use gdb command “next” to execute the next line of code and enter the user input for comments.

A screen shot of a computer

Description automatically generated

Then use “print” commands to print the variable values after the buffer overflow and fill in column #5.

A screenshot of a computer program

Description automatically generated

Here is my finished table for the clang++ version.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable name** | **Type** | **Location (Mem Address)** | **Value before overflow\*** | **Value after overflow\*** |
| scores[10] | int array | df20 | 32,23,32,… | 32,23,32,… |
| name[100] | char array | deb0 | Jiasong | jiasong |
| number\_or\_score | int | Deac | 3 | 3 |
| average | double | dea0 | 29 | 29 |
| sum | int | de9c | 87 | 1094795585 |
| grade | char | de9b | 70 ‘F’ | 65 ‘A’ |
| Comments[5] | char array | de96 | NONE | AAAAA |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable name** | **Type** | **Location (Mem Address)** | **Value before overflow\*** | **Value after overflow\*** |
| scores[10] | int array | 0x7fffffffdd80 | {1651076199,779647075,1600677166,1819242352,-8656,32767,-137661553,32767,0,0} | {1,2,3,4,5,6,7,8,9,10} |
| name[100] | char array | 0x7fffffffdd10 | “abc”, “\000” repeated 96 times | Tori ‘\000’ repeated 94 times |
| number\_or\_score | int | 0x7fffffffdd0c | 0 | 10 |
| average | double | 0x7fffffffdd00 | 0 | 5 |
| sum | int | 0x7fffffffdcfc | 0 | 1094795585 |
| grade | char | 0x7fffffffdcfb | 88 ‘X’ | 65 A |
| Comments[5] | char array | 0x7fffffffdcf6 | NONE | AAAAA |

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

\*Overflow means the buffer overflow after giving comments (see picture below).

﻿

Based on the average score, final grade is: F

looks good? (Yes or No)AAAAAAAA

Comments - Looks good? - AAAAAAAA

----------------------Experiment #2----------------------

(10 pts) Follow the steps before to debug the g++ version with gdb and fill in the table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable name** | **Type** | **Location (Mem Address)** | **Value before overflow\*** | **Value after overflow\*** |
| scores[10] | int array | 0x7fffffffdd10 | {-8472, 32767,-269950720, -771510246, -8880, 32767, -140494144, 32767,4607, 0} | {1,2,3,4,5,6,7,8,9,10} |
| name[100] | char array | 0x7fffffffdd40 | “abc”, “\000” repeats 96 times | “AAAAA”, “\000” repeats 94 times |
| number\_or\_score | int | 0x7fffffffdcfc | 0 | 10 |
| average | double | 0x7fffffffdd08 | 0 | 5 |
| sum | int | 0x7fffffffdd00 | 0 | 55 |
| grade | char | 0x7fffffffdcf7 | 88 ‘X’ | 70 ‘F’ |
| Comments[5] | char array | 0x7fffffffdd3b | NONE | AAAAA |

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen shot of a computer error

Description automatically generated with medium confidence

A computer screen with text on it

Description automatically generated

A screenshot of a computer program

Description automatically generated

For the g++ version:

* (5pts) Required: A screenshot of where you get the locations of variables.

A screenshot of a computer program

Description automatically generated

* (5pts) Required: A screenshot of where you get the value before overflow.

A computer screen shot of a program

Description automatically generated

* (5pts) Required: A screenshot of where you get the value after overflow.

A computer screen shot of a computer

Description automatically generated

(5 pts)Draw a memory layout for the g++ version of program.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable name** | **Type** | **Location (Mem Address)** | **Value before overflow\*** | **Value after overflow\*** |
| scores[10] | int array | 0x7fffffffdd10 | {-8472, 32767,-269950720, -771510246, -8880, 32767, -140494144, 32767,4607, 0} | {1,2,3,4,5,6,7,8,9,10} |
| name[100] | char array | 0x7fffffffdd40 | “abc”, “\000” repeats 96 times | “AAAAA”, “\000” repeats 94 times |
| number\_or\_score | int | 0x7fffffffdcfc | 0 | 10 |
| average | double | 0x7fffffffdd08 | 0 | 5 |
| sum | int | 0x7fffffffdd00 | 0 | 55 |
| grade | char | 0x7fffffffdcf7 | 88 ‘X’ | 70 ‘F’ |
| Comments[5] | char array | 0x7fffffffdd3b | NONE | AAAAA |

Stack grows

|  |
| --- |
| (Higher Address) |
| Command line arguments  empty |
| Stack (Dynamic Memory Layout) empty |
| Heap (Dynamic Memory Layout)  Empty |
| Un-initialized Data Segment (Static Memory Layout) scores[10] 0x7fffffffdd10 |
| Initialized Data Segment (Static Memory Layout) name[100] 0x7fffffffdd40  comments[5] 0x7fffffffdd3b  average 0x7fffffffdd08  sum 0x7fffffffdd00  number\_of\_score 0x7fffffffdcfc  grade 0x7fffffffdcf7 |
| Text/Code Segment (Static Memory Layout)  binary file instructions |
| (Lower Address) |

High address

Df20

scores

Low address

(5 pts) Draw a memory layout for the clang++ version of program.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Variable name** | **Type** | **Location (Mem Address)** | **Value before overflow\*** | | **Value after overflow\*** |
| scores[10] | int array | df20 | 32,23,32,… | | 32,23,32,… |
| name[100] | char array | deb0 | Jiasong | | jiasong |
| number\_or\_score | int | Deac | 3 | | 3 |
| average | double | dea0 | 29 | | 29 |
| sum | int | de9c | 87 | | 1094795585 |
| grade | char | de9b | 70 ‘F’ | | 65 ‘A’ |
| Comments[5] | char array | de96 | NONE | | AAAAA |
| (Higher Address) | | | |
| Command line arguments  empty | | | |
| Stack (Dynamic Memory Layout) empty | | | |
| Heap (Dynamic Memory Layout)  Empty | | | |
| Un-initialized Data Segment (Static Memory Layout) scores[10] df20 | | | |
| Initialized Data Segment (Static Memory Layout) name[100] deb0  number\_of\_score deac  average dea0  sum de9c  grade de9b  comments[5] de96 | | | |
| Text/Code Segment (Static Memory Layout)  binary file instructions | | | |
| (Lower Address) | | | |

(5 pts) Question: the program compiled with clang++ stores variable based on the order of declaration. What about the version of C++? Any pattern?

In the g++ version there appears to be a small pattern. It looks like arrays were stored first and then calculated values like sum, average, exc… and any remaining variables were stored after that.

(5 pts) Question: In the clang version, why did we get sum == 1094795585 after the overflow?

When a buffer overflow happens in comments it overwrites grades and affects the sum value stored next to grades. The sum value has an integer overflow/underflow which makes sum store a very large number instead of what is expected.

(5 pts) Lab Summary (What have you learned in this lab? Anything interesting?)

In this lab I have learned and practiced using GDB to look at variable values before and after program concerns (security issues such as buffer overflows) using breakpoints. I have also used GDB to look at memory addresses and used those addresses to visualize memory blocks of programs. I have learned that various compilers handle their memory allocations differently and that specific issues like buffer overflow will affect different variable values depending on how the variables are stored in memory and thus may have different results from compiler to compiler. I also had to review reading hexadecimal memory addresses to determine location in the memory layout.