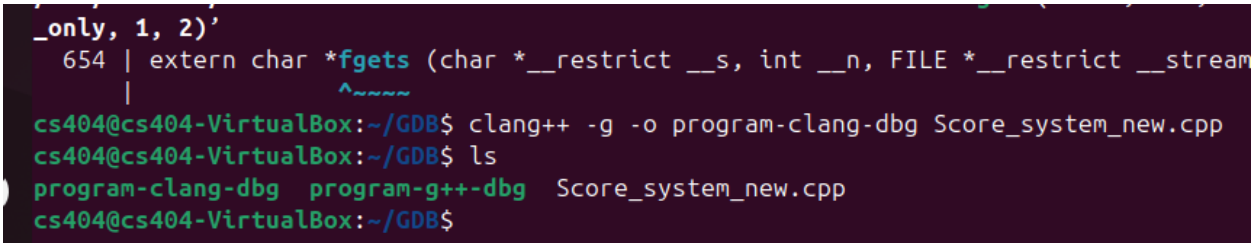
CYB220 Lab 8 – GDB

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.)

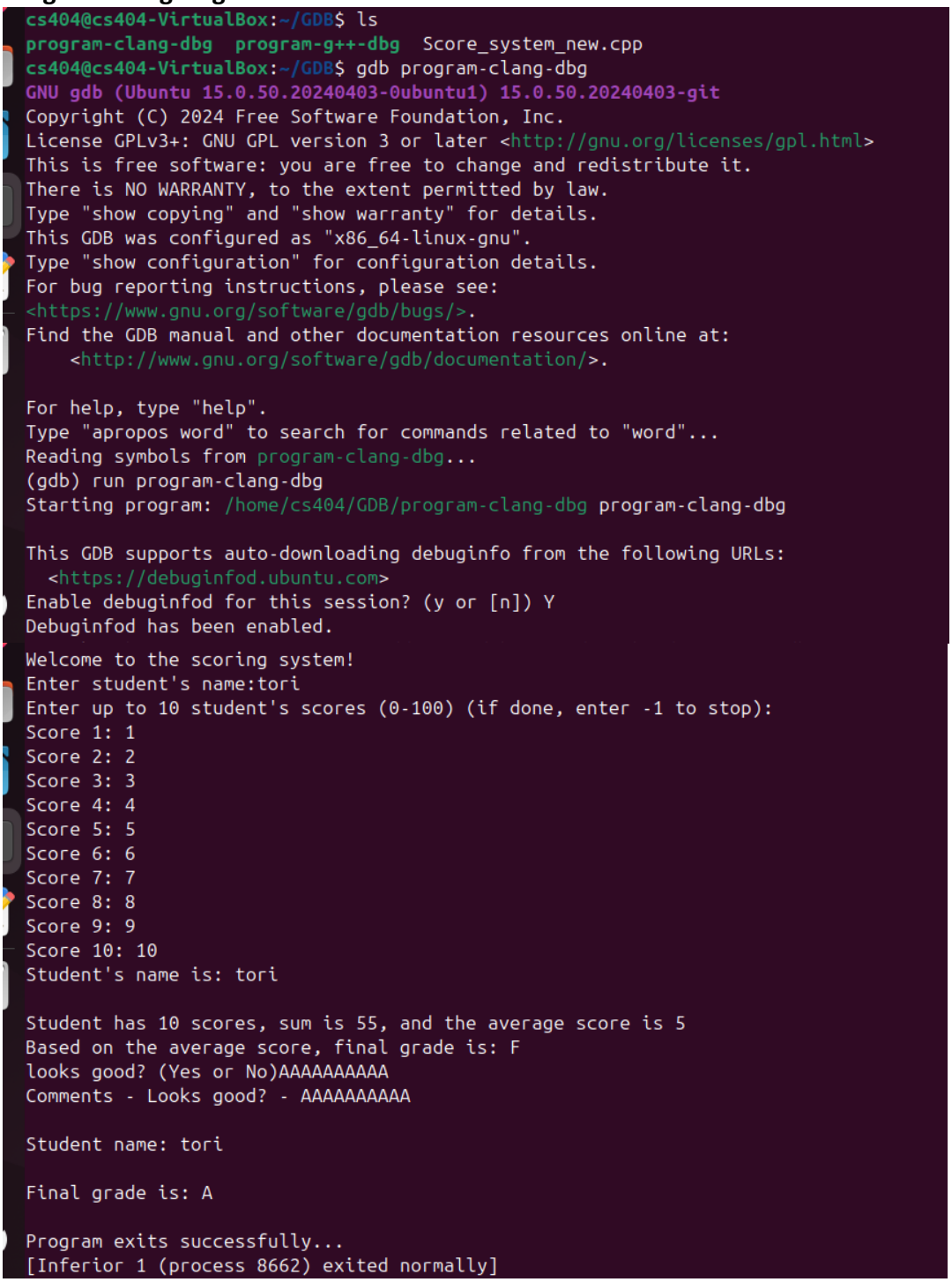
- `g++ -g -o program-g++-dbg Score_system_new.cpp`
  (version 1: compiled with g++)
- `clang++ -g -o program-clang-dbg Score_system_new.cpp`
  (version 2: compiled with clang++)



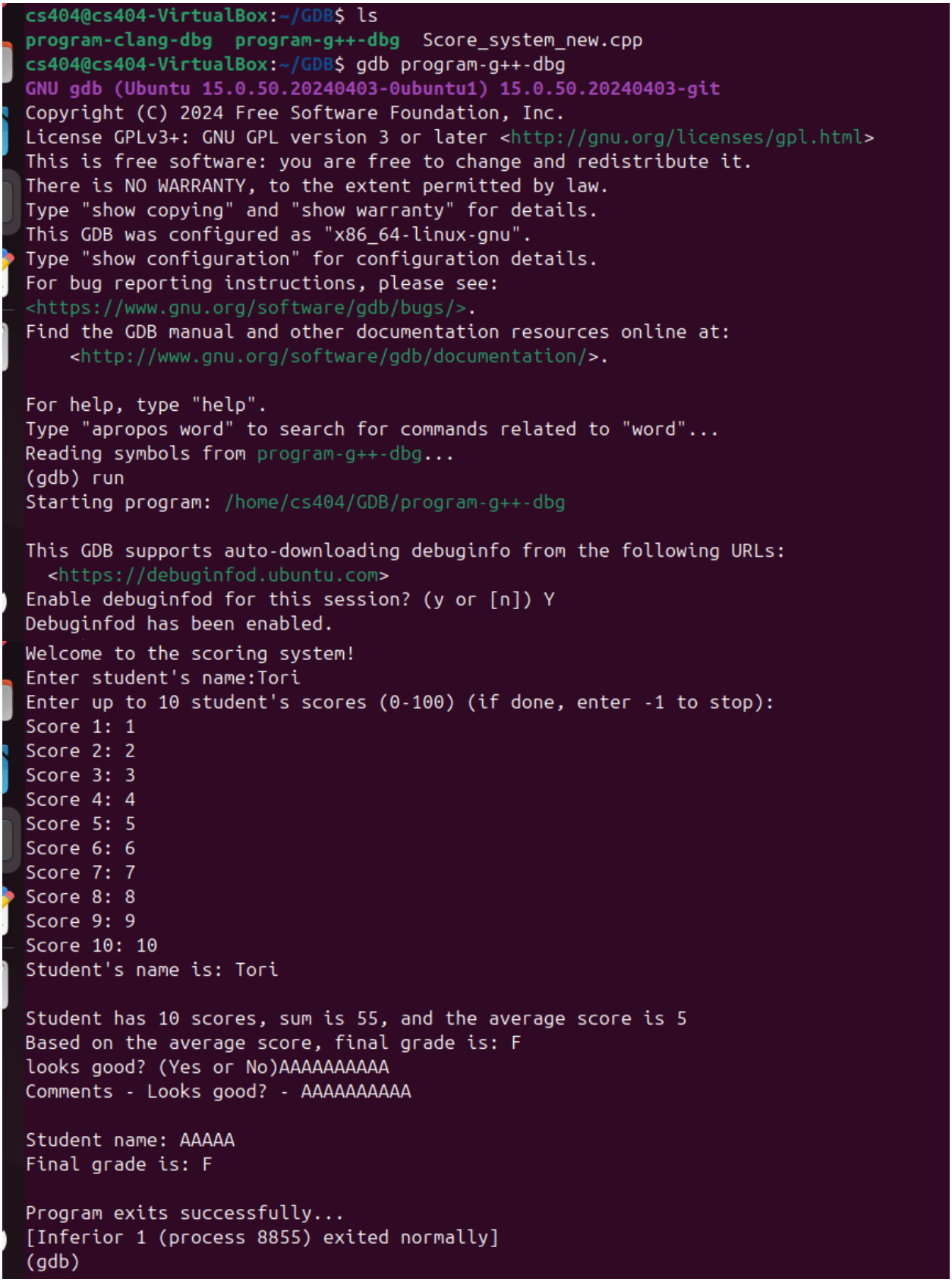
Step 2: Run both versions, when the score_system program asking for comments, enter EXACT ten "A"s

Based on the average score, final grade is: F looks good? (Yes or No)AAAAAAAA Comments - Looks good? - AAAAAAAA

Program-clang-dbg



Program-g++-dbg



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

PROGRAM-CLANG-GDB

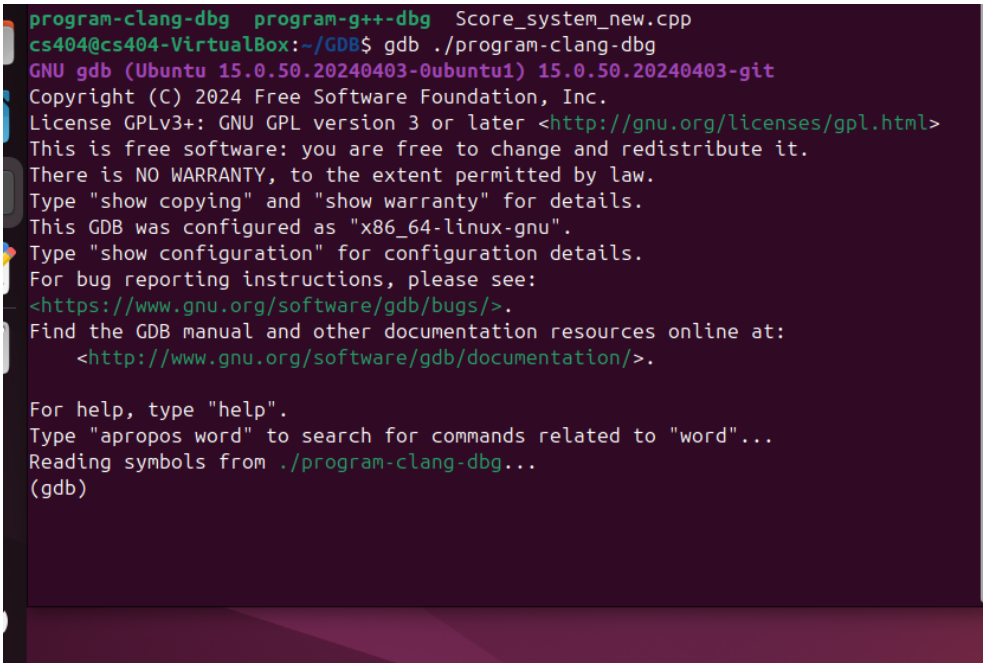
Variable name	**Type**	**Location (Mem Address)**	**Value before overflow***	**Value after overflow***
scores[10]	int array	0x7fffffffdd80	{1651076199,779647075,1600677166, 1819242352,-8656,32767,-137661553,32767,0,0}	{1,2,3,4,5,6,7,8,9,10}
name[100]	char array	0x7fffffffdd10	“abc”, “\000” repeated 96 times	Tori ‘\000’ repeated 94 times
number_or_score	int	0x7fffffffd0c	0	10
average	double	0x7fffffffd00	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`**

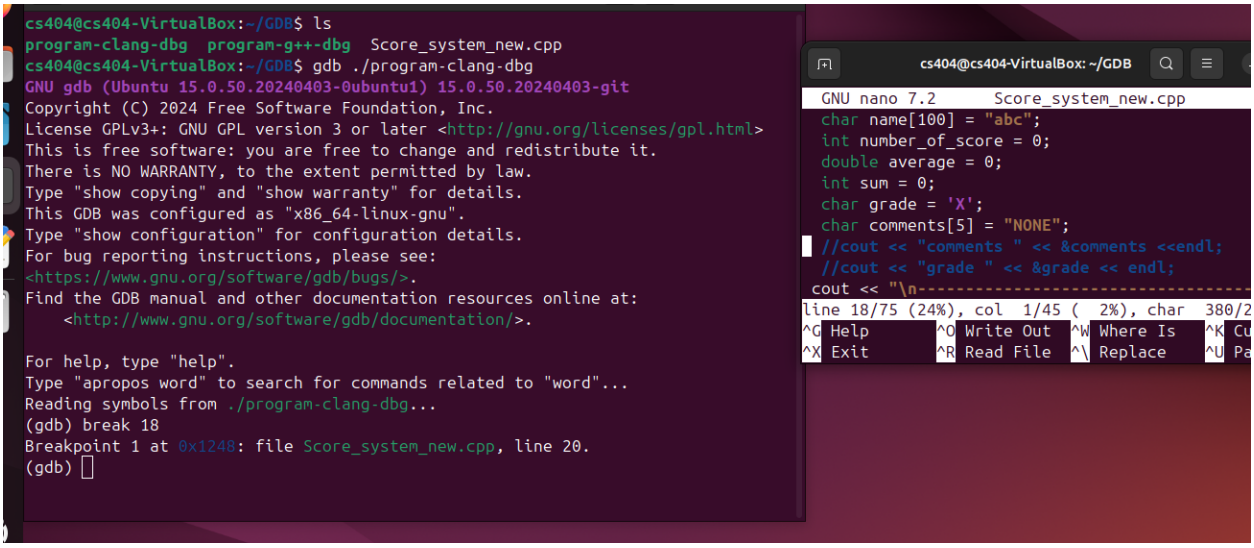


Once gdb started, the first thing to do is to set up break points. Your program will stop before executing the code at the break points.

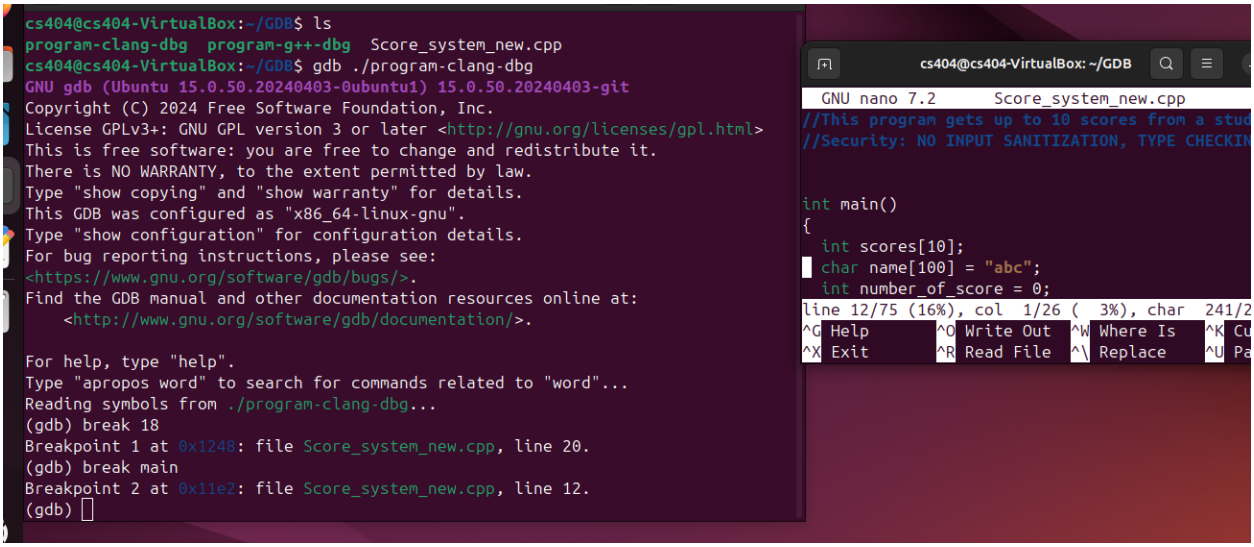
gdb commands: **break or b**

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

break 18



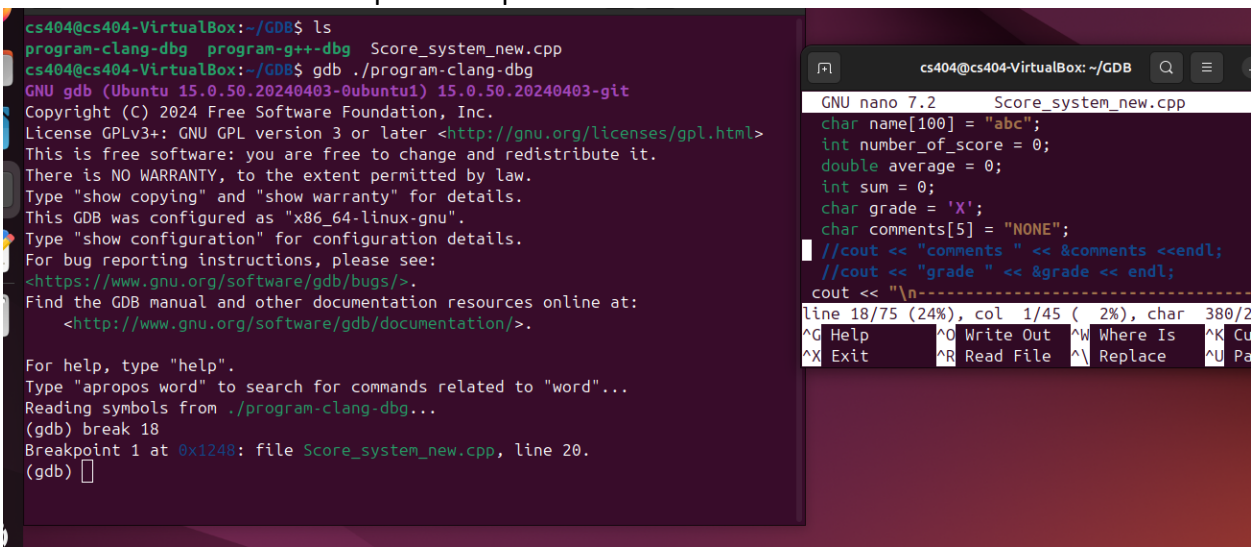
b main



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



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

```
60          grade = 'F';
(gdb) list
61
62        cout << "Based on the average score, final grade is: " <<
63
64        cout << "looks good? (Yes or No)";
65        cin.ignore();
66        cin >> comments;  //BUFFER OVERFLOW, POSSIBLY OVERWRITE T
67
68        cout << "Comments - Looks good? - " << comments << endl <
69        cout << "Student name: " << name << endl;
70        cout << "Final grade is: " << grade << endl <<endl;
(gdb) list
71
72        cout << "Program exits successfully..." <<endl;
```

```
66        cin >> comments;  //BUFFER OVERFLOW, POSSIBLY OVERWRITE
67
68        cout << "Comments - Looks good? - " << comments << endl
69        cout << "Student name: " << name << endl;
70        cout << "Final grade is: " << grade << endl <<endl;
(gdb) list
71
72        cout << "Program exits successfully..." <<endl;
73
74      }
(gdb) list
End of the file was already reached, use "list ." to list the curr
(gdb) break 66
Breakpoint 3 at 0x158e: file Score_system_new.cpp, line 66.
(gdb)
```

B. After setting up the break points, now run the program.
gdb command: **run or r**

```
Breakpoint 3 at 0x158e: file Score_system_new.cpp, line 66.
(gdb) run
Starting program: /home/cs404/GDB/program-clang-dbg

This GDB supports auto-downloading debuginfo from the following UR
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) Y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_d

Breakpoint 2, main () at Score_system_new.cpp:12
12          char name[100] = "abc";
(gdb)
```

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)**

```
This GDB supports auto-downloading debuginfo from the following UR
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) Y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_d

Breakpoint 2, main () at Score_system_new.cpp:12
12          char name[100] = "abc";
(gdb) next
13          int number_of_score = 0;
(gdb) next
14          double average = 0;
(gdb)
```

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

```
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_d

Breakpoint 2, main () at Score_system_new.cpp:12
12          char name[100] = "abc";
(gdb) next
13          int number_of_score = 0;
(gdb) next
14          double average = 0;
(gdb) continue
Continuing.

Breakpoint 1, main () at Score_system_new.cpp:20
20        cout << "\n---------------------------------------\n";
(gdb)
```

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.

```
(gdb) continue
Continuing.

Breakpoint 1, main () at Score_system_new.cpp:20
20        cout << "\n-----------------------------
(gdb) print &Scores
No symbol "Scores" in current context.
(gdb) print &scores
$1 = (int (*)[10]) 0x7fffffffdd80
(gdb)
```

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

```
Breakpoint 1, main () at Score_system_new.cpp:20
20        cout << "\n-------------------------------
(gdb) print &Scores
No symbol "Scores" in current context.
(gdb) print &scores
$1 = (int (*)[10]) 0x7fffffffdd80
(gdb) print &average
$2 = (double *) 0x7fffffffdd00
(gdb)
```

Or use display, eg. display scores → display the value stored in `scores`

```
20        cout << "\n---------------------------------------\n";
(gdb) print &Scores
No symbol "Scores" in current context.
(gdb) print &scores
$1 = (int (*)[10]) 0x7fffffffdd80
(gdb) print &average
$2 = (double *) 0x7fffffffdd00
(gdb) display scores
1: scores = {1651076199, 779647075, 1600677166, 1819242352, -8656, 32767, -137661553, 32767, 0, 0}
(gdb)
```

At break point #2, we can examine the variable values by using "print" command.
For example: print scores → prints the scores array's elements

```
Student has 10 scores, sum is 55, and the average score is 5
Based on the average score, final grade is: F
looks good? (Yes or No)
Breakpoint 3, main () at Score_system_new.cpp:66
66        cin >> comments;  //BUFFER OVERFLOW, POSSIBLY OVERWRITE THE GRADE VARIABLE TO CHANGE THE F
1: scores = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb) print scores
$3 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb)
```

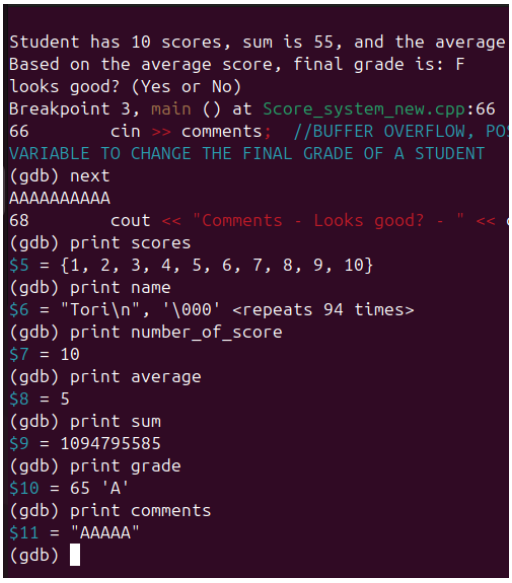
print sum → prints the sum variable's value

```
Based on the average score, final grade is: F
looks good? (Yes or No)
Breakpoint 3, main () at Score_system_new.cpp:66
66        cin >> comments;  //BUFFER OVERFLOW, POSSIBLY OVERWRITE THE GRADE VARIABLE TO CHANGE THE F
1: scores = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb) print scores
$3 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb) print sum
$4 = 55
(gdb)
```

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.

```
Breakpoint 3, main () at Score_system_new.cpp:66
66        cin >> comments;  //BUFFER OVERFLOW, POSSIBLY OVERWRITE TH
VARIABLE TO CHANGE THE FINAL GRADE OF A STUDENT
1: scores = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb) next
AAAAAAAAAA
68        cout << "Comments - Looks good? - " << comments << endl <<
1: scores = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb)
```

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



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

```
Breakpoint 2, main () at Score_system_new.cpp:12
12          char name[100] = "abc";
(gdb) continue
Continuing.

Breakpoint 1, main () at Score_system_new.cpp:20
20          cout << "\n------------------------------
(gdb) print &scores
$12 = (int (*)[10]) 0x7fffffffdd80
(gdb) print &name
$13 = (char (*)[100]) 0x7fffffffdd10
(gdb) print &number_of_score
$14 = (int *) 0x7fffffffdd0c
(gdb) print &average
$15 = (double *) 0x7fffffffdd00
(gdb) print &sum
$16 = (int *) 0x7fffffffdcfc
(gdb) print &grade
$17 = 0x7fffffffdcfb "X"
(gdb) print &comments
$18 = (char (*)[5]) 0x7fffffffdcf6
(gdb)
```

```
(gdb) print &comments
$18 = (char (*)[5]) 0x7fffffffdcf6
(gdb) print scores
$19 = {1651076199, 779647075, 1600677166, 1819242352, -8656, 32767, -137661553, 32767,
  0, 0}
(gdb) print name
$20 = "abc", '\000' <repeats 96 times>
(gdb) print number_of_score
$21 = 0
(gdb) print average
$22 = 0
(gdb) print sum
$23 = 0
(gdb) print grade
$24 = 88 'X'
(gdb) print comments
$25 = "NONE"
(gdb)
```

```
66          cin >> comments;  //BUFFER OVERFLOW, POSSIBLY OVERWRITE THE GRADE VARIABLE TO
(gdb) next
AAAAAAAAAA
68          cout << "Comments - Looks good? - " << comments << endl << endl;
(gdb) print scores
$26 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb) print name
$27 = "Tori\n", '\000' <repeats 94 times>
(gdb) print number_of_score
$28 = 10
(gdb) print average
$29 = 5
(gdb) print sum
$30 = 1094795585
(gdb) print grade
$31 = 65 'A'
(gdb) print comments
$32 = "AAAAA"
(gdb)
```

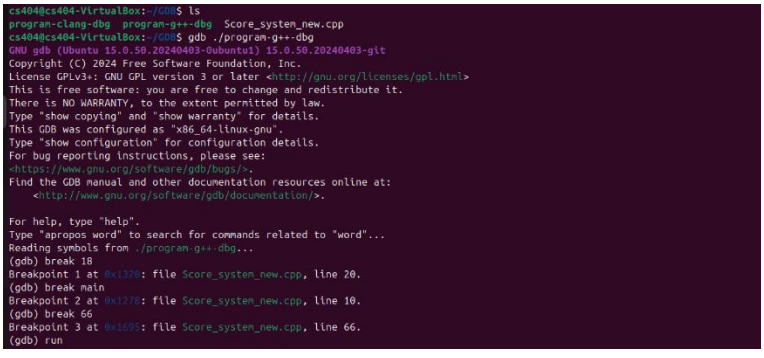
*Overflow means the buffer overflow after giving comments (see picture below).

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.

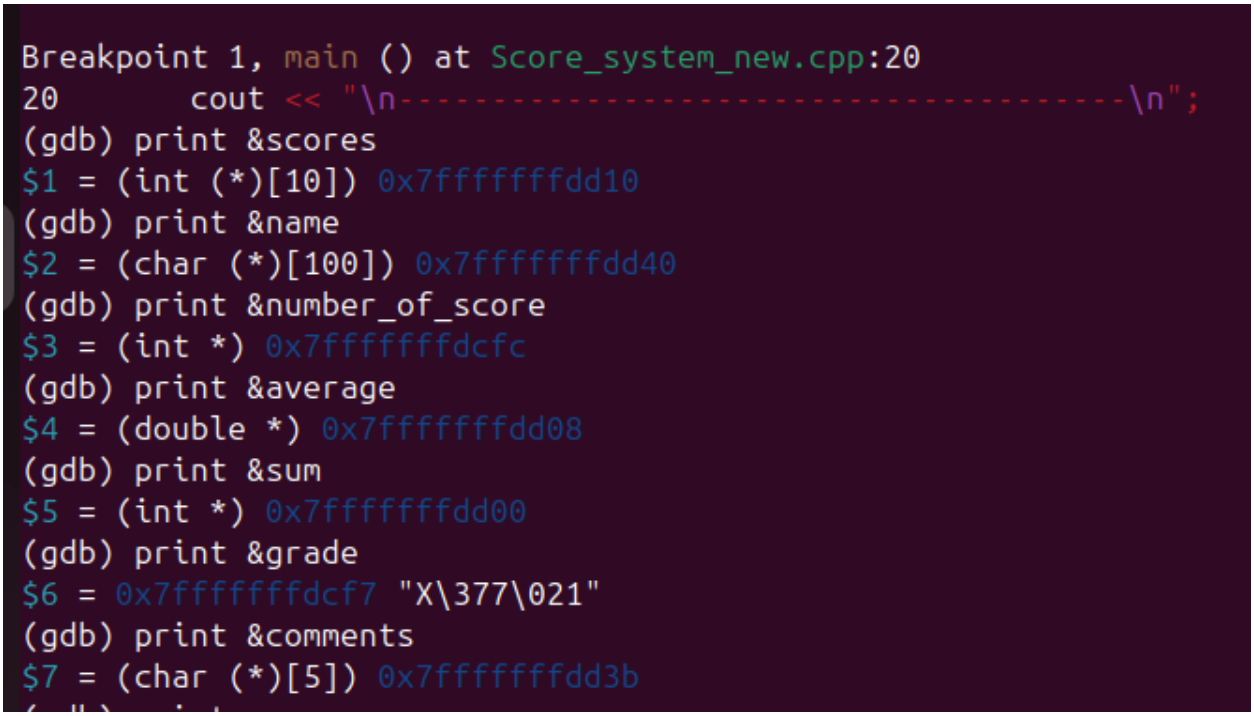
Variable name	**Type**	**Location (Mem Address)**	**Value before overflow***	**Value after overflow***
scores[10]	int array	0x7fffffffdd10	{-8472, 32767,-269950720, -771510246, -8880, 32767, -140494144, 32767,4607, 0}	{1,2,3,4,5,6,7,8,9,10}
name[100]	char array	0x7fffffffdd40	“abc”, “\000” repeats 96 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



```
(gdb) run
Starting program: /home/cs404/GDB/program-g++-dbg

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) Y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc3000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 2, main () at Score_system_new.cpp:10
10
(gdb) continue
Continuing.

Breakpoint 1, main () at Score_system_new.cpp:20
20        cout << "\n---------------------------------------\n";
(gdb) print &scores
$1 = (int (*)[10]) 0x7fffffffdd10
(gdb) print &name
$2 = (char (*)[100]) 0x7fffffffdd40
(gdb) print &number_of_score
$3 = (int *) 0x7fffffffdcfc
(gdb) print &average
(gdb) print &average
$4 = (double *) 0x7fffffffdd08
(gdb) print &sum
$5 = (int *) 0x7fffffffdd00
(gdb) print &grade
$6 = 0x7fffffffdcf7 "X\377\021"
(gdb) print &comments
$7 = (char (*)[5]) 0x7fffffffdd3b
(gdb) print score
No symbol "score" in current context.
(gdb) print scores
$8 = {-8472, 32767, -269950720, -771510246, -8880, 32767, -140494144, 32767, 4607, 0}
(gdb) print name
$9 = "abc", '\000' <repeats 96 times>
(gdb) print number_of_score
$10 = 0
(gdb) print average
$11 = 0
(gdb) print sum
$12 = 0
(gdb) print grade
$13 = 88 'X'
(gdb) print comments
$14 = "NONE"
(gdb) continue
Continuing.
---------------------------------------
Welcome to the scoring system!
Enter student's name:Tori
Enter up to 10 student's scores (0-100) (if done, enter -1 to stop):
Score 1: 1
Score 2: 2
Score 3: 3
Score 4: 4
Score 5: 5
Score 6: 6
Score 7: 7
Score 8: 8
Score 9: 9
Score 10: 10
Student's name is: Tori

Student has 10 scores, sum is 55, and the average score is 5
Based on the average score, final grade is: F
looks good? (Yes or No)
Breakpoint 3, main () at Score_system_new.cpp:66
66        cin >> comments;  //BUFFER OVERFLOW, POSSIBLY OVERWRITE THE GRADE VARIABLE TO CHANGE THE FINAL GRADE OF A STUDENT
(gdb) next
AAAAAAAAAA
68        cout << "Comments - Looks good? - " << comments << endl << endl;
(gdb) print scores
$15 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb) print name

Student has 10 scores, sum is 55, and the average score is 5
Based on the average score, final grade is: F
looks good? (Yes or No)
Breakpoint 3, main () at Score_system_new.cpp:66
66        cin >> comments;  //BUFFER OVERFLOW, POSSIBLY OVERWRITE THE GRADE VARIABLE TO CHANGE THE FINAL GRADE OF A STUDENT
(gdb) next
AAAAAAAAAA
68        cout << "Comments - Looks good? - " << comments << endl << endl;
(gdb) print scores
$15 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb) print name
$16 = "AAAAA", '\000' <repeats 94 times>
(gdb) print number_of_score
$17 = 10
(gdb) print adverage
No symbol "adverage" in current context.
(gdb) print average
$18 = 5
(gdb) print sum
$19 = 55
(gdb) print grade
$20 = 70 'F'
(gdb) print comments
$21 = "AAAAA"
(gdb)
```

For the g++ version:

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



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

```
Breakpoint 1, main () at Score_system_new.cpp:20
20        cout << "\n---------------------------------------\n";
(gdb) print &scores
$1 = (int (*)[10]) 0x7fffffffdd10
(gdb) print &name
$2 = (char (*)[100]) 0x7fffffffdd40
(gdb) print &number_of_score
$3 = (int *) 0x7fffffffdcfc
(gdb) print &average
$4 = (double *) 0x7fffffffdd08
(gdb) print &sum
$5 = (int *) 0x7fffffffdd00
(gdb) print &grade
$6 = 0x7fffffffdcf7 "X\377\021"
(gdb) print &comments
$7 = (char (*)[5]) 0x7fffffffdd3b
(gdb) print score
No symbol "score" in current context.
(gdb) print scores
$8 = {-8472, 32767, -269950720, -771510246, -8880, 32767, -140494144, 32767, 4607, 0}
(gdb) print name
$9 = "abc", '\000' <repeats 96 times>
(gdb) print number_of_score
$10 = 0
(gdb) print average
$11 = 0
(gdb) print sum
$12 = 0
(gdb) print grade
$13 = 88 'X'
(gdb) print comments
$14 = "NONE"
(gdb) continue
Continuing.
```

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

```
Breakpoint 3, main () at Score_system_new.cpp:66
66        cin >> comments;  //BUFFER OVERFLOW, POSSIBLY OVERWRITE THE GRADE VARIABLE TO CHANGE THE FINAL GRADE OF A STUDENT
(gdb) next
AAAAAAAAAA
68        cout << "Comments - Looks good? - " << comments << endl << endl;
(gdb) print scores
$15 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
(gdb) print name
$16 = "AAAAA", '\000' <repeats 94 times>
(gdb) print number_of_score
$17 = 10
(gdb) print adverage
No symbol "adverage" in current context.
(gdb) print average
$18 = 5
(gdb) print sum
$19 = 55
(gdb) print grade
$20 = 70 'F'
(gdb) print comments
$21 = "AAAAA"
(gdb)
```

(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

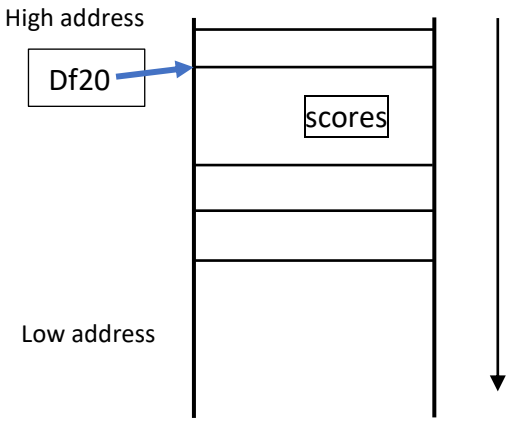
Stack grows

(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)

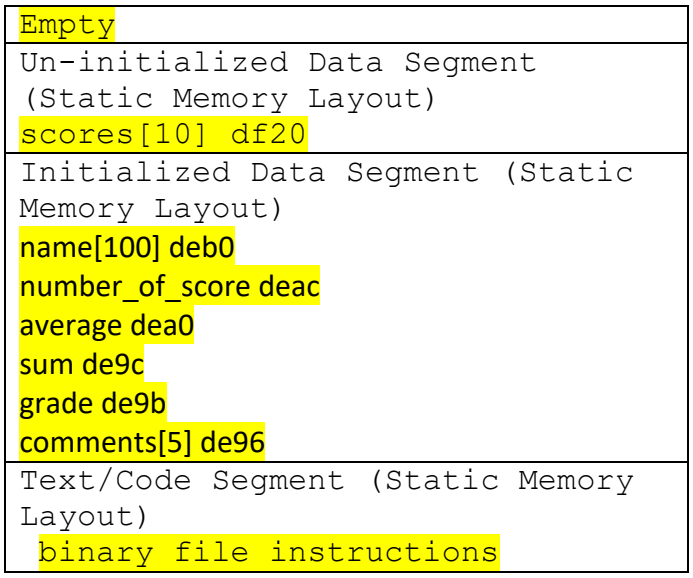


(5 pts) Draw a memory layout for the clang++ version of program.

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) ⬆



(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.