COS 214: Software Modelling

Assignment 3

Name: Caleb Johnstone

Student Number: u19030119

Email: caleb.johnstone@tuks.co.za

Answers to questions

Question 1.1

1)

Compile and run with debugging: g++ -g marks.cpp -o marks

Run the program: ./marks

Error produced: Floating point exception (core dumped)

2)

Run the program using GDB: gdb ./marks

3)

Run the program in GDB: run

Error produced: Arithmetic exception

Line number: 17

Function arguments when the error occurred: a=-2, b=0

4)

List command: list

5)

backtrace command: backtrace

Interpretation of what the stack trace means: A stack trace shows the sequence of function calls that have been placed on the program's runtime stack. When an error occurs the sequence of function calls can be seen in reverse, so from last to first, by popping function calls off the stack. In this case first call is to the main function which in turn calls the improve function with arguments: a=-2, b=0

6)

Move up one level to level 1: up

7)

List command: list

Lines that are printed this time: Lines 6-15, which are the main function (Lines 7-13) and the function signature for the improve function (Line 15).

These lines are run because level 1 of the stack frame is being examined which is the level which the main function is at.

8)

Command to print the value of the highest variable: p highest

Value that is printed: 0

9)

The improve functions returns ((double)(a / b)) * 100. This requires for a to be divided by b, but b equals zero at the point that the program crashes. The program crashes because division by zero is undefined and therefore causes an arithmetic exception.

Question 1.2

1)

Compile and run with debugging: g++ -g capture.cpp -o capture

Run the program: ./capture

2)

Command to run the program in Valgrind: valgrind --leak-check=full ./capture

3)

Process ID on the output (repeatedly printed down the left-hand side of the output): 157

4)

The first error seen in the output:

==157== Invalid write of size 4

This error is because line 4, which is:

```
marks[10] = 0;
```

attempts to set the element at index 10 of the integer array, pointed to by the marks pointer, to zero. This is not possible as the array is of size 10 and therefore the only valid integer indexes are 0 to 9, both inclusive.

5)

Stack trace below the first error:

==157== at 0x10916B: capture() (capture.cpp:4)

==157== by 0x109180: main (capture.cpp:9)

This output shows that the error occurred at line 4 inside of the capture function and that the capture function was called from the main function on line 9.

6)

In C++, an int is allocated 4 bytes. The array pointed to by the marks integer pointer has 10 elements and is therefore allocated 40 bytes, because 4 bytes x 10 = 40 bytes. The capture function does not assign values to any of the elements indexed 0 to 9 inclusive, so the memory is allocated for the 10 integers but it is never used and never deallocated. The heap-allocated memory was never freed, so the program no longer has a pointer to it and it is therefore definitely lost.

7)

The memory leak can be fixed by deallocating the memory pointed to by the marks integer pointer, so by adding the following code at the end of the capture function:

delete[] marks;

marks = nullptr;

The line:

```
marks[10] = 0;
```

must be removed or changed to have a valid index, but that is a separate issue as it is not the cause of the memory leak. Not deallocating memory is the cause of the memory leak.