This is an assignment designed to teach/test some of the common GDB commands in a hands-on manner.

What is GDB:

GDB stands for GNU Debugger (and sometimes incorrectly as General Debugger). It lets you look into your code as it runs and see variable change, functions call, and many more. It has many many many commands, several more than could be covered in a simple document. The simplest ones will be covers later.

How Do I Get GDB:

On a Linux machine (or WSL): `sudo apt-get install gdb`. On Mac, you may instead use `LLDB`. The commands are similar, but you should check online to be sure. On Windows (not WSL), you may run into problems, but the ported version of the program is WinGDB.

How Do I Use GDB:

First, you must compile the commands using the `-g` flag. For example, the command for this homework is `g++ -Wall main.cpp -g.` Next, run the executable using `gdb ./a.out`. This will launch the GDB application. At the prompt you can type any GDB command.

Common GDB Commands:

- `run <arguments>` - run the executable with any optional arguments, can be called when broken on a line to restart with the same arguments
- `quit` – exit the debugger (note, GDB prevents normal exit via ctrl-c)
- `break <position>` - set a breakpoint on a specific line. This will stop program execution immediately before this line runs. `<position>` can be specified via `filename:lineNumber` or `filename:functionSignature`. There is tab-completion to help prevent typos.
- `where` – lists the stack frames currently on the call stack
- `frame <number>` - switch into the frame specified by number
- `print <variable_name>` - print the value of the variable. Also supports most basic operations, such a bracket operator, arithmetic, and sometimes calling user defined functions
- `set <variable_name> = <value>` - set the variable to a specific value
- `list` – prints several lines from the source file around the current line, generally five lines in either direction
- `step` – move to the next logical statement, including into called functions
- `next` – move to the next logical statement in the current scope (skipping function calls)
- `continue` – resume normal execution
- `ctrl-c` – (key-combo, not command) immediately break on the current line. Useful if the program enters an infinite loop.

The Assignment:

- A program is provided to test your knowledge of the debugger. It takes `two command line parameters`: an integer for mode and optionally a Unique Input Notation (UIN). (If a UIN is not provided, a random one is generated). The UIN format is 3 digits, followed by 2 zeros, followed by 4 additional digits. Any similarity to any other UIN type is purely coincidental. The `mode` argument should be in the range 0 to 11 inclusive. Each scenario starts by printing some instructions/prompts. Then the code runs through some obfuscated code simply to hide what it is doing. Finally the code terminates somewhere in the baseTypes file. It is your job to use the debugger to answer the questions provided.

Workflow:

- Start by compiling the program
- Run `gdb ./a.out`
    - This loads the program into GDB
- Type `run 0` at the gdb prompt, this runs the program with only the command line argument 0
- The program will print some information and then some questions
- Use the debugger and the commands listed earlier to answer the prompts
- When done, at the gdb prompt type `run <number>` to run with the next mode