CS 341 Assignment 0

Debugging

Provided Files:

```
/
├── heap.hpp
├── heap.cpp
├── djikstra.cpp
├── makefile
└── prettyprinters.py
```

Problem Statement:

1. Given to you is the source code for a C++ implementation of Djiktra's Algorithm for calculating the shortest path.

   All the signatures of the classes and their methods are explained in doxygen style comments in the source itself.

   Run `make` to compile the program (with debug symbols), and `make gdb` to start debugging the program

   (a) `heap.cpp` uses the `class VertexHeap` which houses the heap data in the member `int heap[]`, whose length is dynamically determined by the member `int size`.

   By default printing `VertexHeap` instances in GDB produces the following output:

```
$2 = {index2HeapIdx = std::unordered_map with 4 elements = {[2] = 3,
[1] = 2, [3] = 0, [4] = 1}, heapIdx2index = std::unordered_map with 4 elements
= {[3] = 2, [2] = 1, [0] = 3, [1] = 4}, heap = 0x6b2d10, capacity = 4, size = 4}
```

   which does not contain any information about the member `heap` since it is dynamically allocated (and to GDB it is just a pointer).

   We can customize the formatting GDB uses to print variable information by defining our custom Pretty Printer. The boilerplate for this is already included in the file `prettyprinters.py`. To tell gdb to use the custom pretty printer, you have to issue the command

```
(gdb) source prettyprinters.py
```

   Modify the file `prettyprinters.py` such that printing `VertexHeap` instances in GDB produce readable output. It should at a minimum contain the heap data in a readable format.

   (b) The source files contain 2 subtle bugs that you need to fix. *Use GDB to track down 2 subtle bugs in the program and fix the bugs by modifying the source files..*

   You can log all the gdb commands you used by issuing the following commands before you start debugging

```
(gdb) set logging overwrite on
(gdb) set trace-commands on
(gdb) set logging on
```

   In the submission include the resultant `gdb.txt` file containing your gdb log

Provided Files:

```
/
├── argparse.c
├── argparse.h
├── main.c
└── makefile
```

Problem Statement:

2. Given to you is the source code for a simple implementation of an argument parser in C. All the signatures of the classes and their methods are explained in doxygen style comments in the source itself.

   (a) Compile the program using the command

   ```
   make
   ```

   This compiles the program into `main.out` with debug symbols, so if you wish to break out gdb to walk through this program feel free to do so.

   If you run `main.out`, it will run as expected. But things are not as good as they seem.

   If you execute `make valgrind`, that will run the program called `valgrind` on the executable. This will check for any memory leaks. In the `main.c` file we have provided, you should get 111 bytes which are in use at exit.

   Track down each place where the allocation leads to a memory leak, and explain the reason for the leak. All bytes should be accounted for. All the explanations should be written in the report called `answers.pdf`

   Hint: Lookout for `malloc` and `strdup` both of which lead to memory allocation on the heap.

   (Optional) There is a corner case with a segfault. See if you can find the bug and fix it

   (b) Fix the bugs in `argparse.c`, such that the leaks do not happen. You cannot modify `argparse.h`. You can modify `main.c`, but the final program should not have any memory leaks in the `main.c` that we provide.

Deadline: 4 August 11:59 PM
Submission Instructions:

- Put all your files in the directory structure given below and make a tarball of it using

```
tar -czvf {roll-number}-A0.tar.gz ./{roll-number}-A0
```

- Upload your submission on Moodle
- The template for creating `answers.pdf` will be provided to you via Moodle/Piazza

Submission Directory Structure:

```
roll-number-A0/
├── gdb/
│   ├── heap.hpp
│   ├── djikstra.cpp
│   ├── prettyprinters.py
│   └── gdb.txt
└── valgrind/
    ├── argparse.c
    └── answers.pdf
```