CS 221 Pointers and Arrays Evaluation Assignment

Lily Larsen

October 28, 2022

1.

a.

If the installation of the library was intended to be local to my user I would copy or symlink any executables to `~/.local/bin` and any shared object files to `~/.local/lib`. I would additionally need to ensure `~/.local/lib` and `~/.local/bin` are present on the path by adding `PATH="\$HOME/.local/bin:\$HOME/.local/lib:\$PATH"` to my `.bashrc`. For a system-wide installation of the library I would copy or symlink executables to `/usr/bin` and shared object files to `/usr/lib`.

b.

```
# If not running interactively, don't do anything
[[ $- != *i* ]] && return

# helpfully display a steam locomotive to the user (requiring the sl package) and then switch to a more useful shell
sl
PS3='How about zsh instead? Or maybe fish if you are daring? '
select shell in zsh fish
do
        ${shell}
done
```

c.

```
cat Logistics.tex | grep that -o | wc -w
```

2.

a.

https://github.com/ld405/cs-221-eval-assignments/

b.

If the desired version was recent I would use `git revert HEAD~<N>` where N is the number of commits I wish to revert. Otherwise, I would run `git log` to find the commit hash of the desired version, and then `git revert <HASH>` to revert to the desired hash.

c.

Git history acts similarly linked list with each commit pointing to the hash of the previous commit. Creating a branch allows you to create a fork in the linked list. Functionally, branches allow you to store and reconcile multiple parallel versions of a codebase within a repository.

You can create a branch with `git branch <NAME>`, and you can switch branches with `git checkout <NAME>`. As a shortcut, you can create a branch and switch to it with `git checkout -b <NAME>`.

Branches are essential for modern software development. Companies often employ a pattern of software development where developers must work in their own branches, either individual to them, or individual to the feature being developed. The developer when finished with the feature creates a pull request (or a merge request, or even an email containing a patch, depending on the git provider being used) describing the changes they made and asking for it to be merged into a central main branch.

In an individual project branches can be useful as well. For instance: you may have useful features which are still in progress that can be stowed the away on an experimental branch so that the main branch remains stable.

If you wish to merge branches you can do so with `git merge <NAME>` to merge the named branch into the currently checked out branch.

3.

a.

Let's start by running f on a series of inputs...

https://github.com/Id405/cs-221-eval-assignments/blob/main/eval-4/3a_stages/test.c

```c
#include <stdio.h>

int f(int x, int y) {
  int r = 1;
  while (y > 1) {     // loop for floor(log2(y)) + 1 iterations
    if (y % 2 == 1) { // branches if y is odd
      r = x * r;      // then r is x^(number of odd numbers encountered)
    }
    x = x * x; // then x is 2^(floor(log2(y)) + 1)
    y = y / 2; // y is halved each iteration
  }
  return r * x; // r * x = x^(floor(log2(y)) + 1) * x^(number of odd numbers
                // encountered)
}

int main() {
  printf("| x | y | f(x, y)\n");
  for (int x = 0; x < 10; x++) {
    for (int y = 0; y < 4; y++) {
      printf("| %d | %d | %d\n", x, y, f(x, y));
    }
  }
}
```

```
$ make test
gcc test.c -o target/test -Wall && ./target/test
| x | y | f(x, y)
| 0 | 0 | 0
| 0 | 1 | 0
| 0 | 2 | 0
| 0 | 3 | 0
| 1 | 0 | 1
| 1 | 1 | 1
| 1 | 2 | 1
| 1 | 3 | 1
| 2 | 0 | 2
| 2 | 1 | 2
| 2 | 2 | 4
| 2 | 3 | 8
| 3 | 0 | 3
| 3 | 1 | 3
| 3 | 2 | 9
| 3 | 3 | 27
| 4 | 0 | 4
| 4 | 1 | 4
| 4 | 2 | 16
| 4 | 3 | 64
| 5 | 0 | 5
| 5 | 1 | 5
| 5 | 2 | 25
| 5 | 3 | 125
| 6 | 0 | 6
| 6 | 1 | 6
| 6 | 2 | 36
| 6 | 3 | 216
| 7 | 0 | 7
| 7 | 1 | 7
| 7 | 2 | 49
| 7 | 3 | 343
| 8 | 0 | 8
| 8 | 1 | 8
| 8 | 2 | 64
| 8 | 3 | 512
| 9 | 0 | 9
| 9 | 1 | 9
| 9 | 2 | 81
| 9 | 3 | 729
```

A common pattern emerges: $f(x, y) = x^y$ when $x > 0$ and $y > 0$.

For example...

```
$ make test
gcc test.c -o target/test -Wall && ./target/test
| x | y | f(x, y)
...
| 1 | 1 | 1       = 1^1
| 1 | 2 | 1       = 1^2
| 1 | 3 | 1       = 1^3
...
| 2 | 1 | 2       = 2^1
| 2 | 2 | 4       = 2^2
| 2 | 3 | 8       = 2^3
...
| 3 | 1 | 3       = 3^1
| 3 | 2 | 9       = 3^2
| 3 | 3 | 27      = 3^3
...
| 4 | 1 | 4       = 4^4
| 4 | 2 | 16      = 4^2
| 4 | 3 | 64      = 4^3
...
| 5 | 1 | 5       = 5^1
| 5 | 2 | 25      = 5^2
| 5 | 3 | 125     = 5^3
```

So, we can assume the code is meant to represent exponentiation, and fails in the case of $x = 0$, or $y = 0$.

Let's add two cases to fix this...

https://github.com/Id405/cs-221-eval-assignments/blob/main/eval-4/3a_stages/fix.c

```c
#include <stdio.h>

int f(int x, int y) {
  if (y == 0) { // so that 0^0 = 1
    return 1;
  }

  if (x == 0) {
    return 0;
  }

  int r = 1;
  while (y > 1) {
    if (y % 2 == 1) {
      r = x * r;
    }
    x = x * x;
    y = y / 2;
  }
  return r * x;
}

int main() {
  printf("| x | y | f(x, y)\n");
  for (int x = 0; x < 10; x++) {
    for (int y = 0; y < 4; y++) {
      printf("| %d | %d | %d\n", x, y, f(x, y));
    }
  }
}
```

Now, let's test the output of the updated function...

```
$ make fix
gcc fix.c -o target/fix -Wall && ./target/fix
| x | y | f(x, y)
| 0 | 0 | 1       = 0^0
| 0 | 1 | 0       = 0^1
| 0 | 2 | 0       = 0^2
| 0 | 3 | 0       = 0^3
| 1 | 0 | 1       = 1^0
| 1 | 1 | 1       = 1^1
| 1 | 2 | 1       = 1^2
| 1 | 3 | 1       = 1^3
| 2 | 0 | 1       = 2^0
| 2 | 1 | 2       = 2^1
| 2 | 2 | 4       = 2^2
| 2 | 3 | 8       = 2^3
| 3 | 0 | 1       = 3^0
| 3 | 1 | 3       = 3^1
| 3 | 2 | 9       = 3^2
| 3 | 3 | 27      = 3^3
| 4 | 0 | 1       = 4^0
| 4 | 1 | 4       = 4^1
| 4 | 2 | 16      = 4^2
| 4 | 3 | 64      = 4^3
...
```

The function now operates as expected.

b.

Let's start by cleaning up the code and fixing all of the GCC warnings.

https://github.com/Id405/cs-221-eval-assignments/blob/main/eval-4/3b_stages/fix1.c

```c
#include <stdio.h> // NEW: include required libraries
#include <stdlib.h>

struct point2d {
  int x;
  int y;
};

int main(int argc,
         char *argv[]) { // NEW: change type of argc to int, and argv to *char[]
  int m = atoi(argv[1]);
  struct point2d *p = malloc(sizeof(struct point2d));
  p->x = atoi(argv[2]);
  p->y = atoi(argv[3]);
  struct point2d *z = malloc(sizeof(struct point2d));
  z->x = 0;
  z->y = z->y - (m * p->x);
  printf("x coord: %d\n", p->x);
  printf("y coord: %d\n", p->y);
  return 0;
}
```

Now, let's try running the code

```
$ make fix1 && ./target/fix1
gcc fix1.c -o target/fix1 -Wall && ./target/fix1
make: *** [makefile:6: fix1] Segmentation fault (core dumped)
```

and, we get a segmentation fault. Let's try now running it through GDB with the backtrace option.

```
$ gcc fix1.c -ggdb -o /tmp/fix1.out; and gdb /tmp/fix1.out -ex run -ex bt
GNU gdb (GDB) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /tmp/fix1.out...
Starting program: /tmp/fix1.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".
```

```
Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7dcea3b in ?? () from /usr/lib/libc.so.6
#0  0x00007ffff7dcea3b in ?? () from /usr/lib/libc.so.6
#1  0x00007ffff7dc2f34 in atoi () from /usr/lib/libc.so.6
#2  0x000055555555517b in main (argc=1, argv=0x7fffffffda38) at fix1.c:11
(gdb) exit
A debugging session is active.

Inferior 1 [process 3488971] will be killed.

Quit anyway? (y or n) y
```

So, GDB points to line 11 as the offending line of code. After further inspection of the code we can see that this is because the program expects to be passed three integers as arguments. Let's try running the program with some random integers as arguments then.

```
$ ./target/fix1 32 47 12
x coord: 47
y coord: 12

$ ./target/fix1 92 26 12
x coord: 26
y coord: 12
```

The program appears to print the second two arguments provided to it.

Let's now annotate the source code with this knowledge.

https://github.com/Id405/cs-221-eval-assignments/blob/main/eval-4/3b_stages/annotate.c

```c
#include <stdio.h>
#include <stdlib.h>

struct point2d {
  int x;
  int y;
};

int main(int argc, char *argv[]) {
  int m = atoi(argv[1]); // store the first argument passed to the program as an
                         // int in the variable "m"
  struct point2d *p = malloc(sizeof(struct point2d)); // allocate a point2d "p"
  p->x = atoi(
      argv[2]); // set the x coordinate of the point2d to the second argument
  p->y = atoi(
      argv[3]); // set the y coordinate of the point2d to the third argument
  struct point2d *z =
      malloc(sizeof(struct point2d)); // allocate another point2d "z"
  z->x = 0;                           // set the x coordinate of z to 0
  // set the y coordinate of z to be itself subtracted
  // from m multiplied by the x coordinate of "p".
  // however, before this, the y coordinate of z is uninitialized and, z is left
  // unused in the rest of the program
  z->y = z->y - (m * p->x);
  printf("x coord: %d\n", p->x); // print the x and y coordinates of p
  printf("y coord: %d\n", p->y);
  return 0; // note: p and z are not freed
```

```
}
```

It seems that the program is calculating a point z from our inputs but doesn't print it out. Since a program that prints out the last 2 of 3 arguments you provide it as a point is not a very useful program, we can assume that the program is intended to print out this point instead. Let's try modifying the program such that it prints z instead.

https://github.com/Id405/cs-221-eval-assignments/blob/main/eval-4/3b_stages/fix2.c

```c
#include <stdio.h>
#include <stdlib.h>

struct point2d {
  int x;
  int y;
};

int main(int argc, char *argv[]) {
  int m = atoi(argv[1]);
  struct point2d *p = malloc(sizeof(struct point2d));
  p->x = atoi(argv[2]);
  p->y = atoi(argv[3]);
  struct point2d *z = malloc(sizeof(struct point2d));
  z->x = 0;
  // while we are here lets make the assumption that the intended variable was
  // p->y as this makes z mathematically the y intercept of a line crossing the
  // point p with the slope m.
  z->y = p->y - (m * p->x);
  printf("x coord: %d\n", z->x); // print z instead of p
  printf("y coord: %d\n", z->y);
  return 0;
}
```

```
$ make fix2 && ./target/fix2 3 6 18
gcc fix2.c -o target/fix2 -Wall
x coord: 0
y coord: 0

$ make fix2 && ./target/fix2 3 6 16
gcc fix2.c -o target/fix2 -Wall
x coord: 0
y coord: -2

$ make fix2 && ./target/fix2 1 1 1
gcc fix2.c -o target/fix2 -Wall
x coord: 0
y coord: 0

$ make fix2 && ./target/fix2 1 1 12
gcc fix2.c -o target/fix2 -Wall
x coord: 0
y coord: 11

$ make fix2 && ./target/fix2 1 0 12
gcc fix2.c -o target/fix2 -Wall
x coord: 0
```

```
y coord: 12

$ make fix2 && ./target/fix2 1 12 0
gcc fix2.c -o target/fix2 -Wall
x coord: 0
y coord: -12
```

Now the program seems to work in a fashion that can be made reason of. The program prints out the y intercept given a line passing through the point (argument 2, argument, 3) with slope m. Finally, lets revisit that segmentation fault, and note about p and z not being freed. We could use Valgrind/GDB to further debug this, but it is fairly clear that to solve the segfault we just need to prompt the user when argc < 3, and that we need to free p and z on exit.

https://github.com/Id405/cs-221-eval-assignments/blob/main/eval-4/3b_stages/fix3.c

```c
#include <stdio.h>
#include <stdlib.h>

struct point2d {
  int x;
  int y;
};

int main(int argc, char *argv[]) {
  if (argc < 3) { // validate we have the proper number of arguments supplied
    printf("provide three arguments in the format: slope, x, y");
    return 0;
  }
  int m = atoi(argv[1]);
  struct point2d *p = malloc(sizeof(struct point2d));
  p->x = atoi(argv[2]);
  p->y = atoi(argv[3]);
  struct point2d *z = malloc(sizeof(struct point2d));
  z->x = 0;
  z->y = p->y - (m * p->x);
  printf("x coord: %d\n", z->x);
  printf("y coord: %d\n", z->y);
  free(p); // free p and z
  free(z);
  return 0;
}
```

```
$ make fix3 && ./target/fix3 12 3 5
gcc fix3.c -o target/fix3 -Wall
x coord: 0
y coord: -31

$ make fix3 && ./target/fix3
gcc fix3.c -o target/fix3 -Wall
provide three arguments in the format: slope, x, y
```

The segfault is solved, and due to the simplicity of the code, we can simply match up each malloc with its corresponding free statement to ensure there are no memory leaks.