

# Startrek

## Author

Arjun Lalith

## Startrek1

### Creating the spaceship structure

We start by executing the binary and seeing what it outputs:

```
→ startrek ./startrek
Captains Kirk and Spock both begin in Galaxy 000. Their ships' fuel levels
are both at 100%. Good luck! If you help the Captains accomplish their
mission, you will surely Live Long and Prosper!

Starship: 1      Current Galaxy: 000      Jumps Remaining: 5
Preparing the ship for the next jump...
Do you want to tell the Captain to alter his trajectory (y/n)?
n
Jumped 2 galaxies and landed on 002

Starship: 2      Current Galaxy: 000      Jumps Remaining: 5
Preparing the ship for the next jump...
Do you want to tell the Captain to alter his trajectory (y/n)?
```

We see that there are two ships, they both start at galaxy 0, and start with 5 jumps. Now let's open the binary in ghidra to get a better idea of what's going on under the hood. In the beginning there is the following block of code:

```
for (local_3e4 = 0; (int)local_3e4 < local_68; local_3e4 = local_3e4 + 1)
{
    *(uint*)(local_3a8 + (long)(int)local_3e4 * 0x20) = local_3e4 + 1;
    *(undefined4*)(local_3a8 + (long)(int)local_3e4 * 0x20 + 4) = 0;
    *(int*)(local_3a8 + (long)(int)local_3e4 * 0x20 + 8) = local_6c;
    *(undefined4*)(local_3a8 + (long)(int)local_3e4 * 0x20 + 0xc) = 0;
    lVar13 = (long)local_6c;
```

```

*(undefined8 *) (puVar14 + -0x48) = 0x1015f9;
pvVar11 = malloc(lVar13 << 2);
*(void **) (local_3a8 + (long)(int)local_3e4 * 0x20 + 0x10) = pvVar11;
lVar13 = (long)local_6c;
*(undefined8 *) (puVar14 + -0x48) = 0x10162a;
pvVar11 = malloc(lVar13 << 2);
*(void **) (local_3a8 + (long)(int)local_3e4 * 0x20 + 0x18) = pvVar11;
}

```

This seems to be the initialization of the spaceships, based on the fact that `local_68` is 2, and `local_6c` is 5, matching with the number of spaceships and the amount of jumps each ship starts with. Based on the decompilation, we can tell that `local_3e4` seems to be an array that holds two copies of a structure for spaceships. Based on the initialization of each variable we can determine the size from the following lines:

```

*(uint *) (local_3a8 + (long)(int)local_3e4 * 0x20) = local_3e4 + 1;
*(undefined4 *) (local_3a8 + (long)(int)local_3e4 * 0x20 + 4) = 0;
*(int *) (local_3a8 + (long)(int)local_3e4 * 0x20 + 8) = local_6c;
*(undefined4 *) (local_3a8 + (long)(int)local_3e4 * 0x20 + 0xc) = 0;
*(void **) (local_3a8 + (long)(int)local_3e4 * 0x20 + 0x10) = pvVar11;
*(void **) (local_3a8 + (long)(int)local_3e4 * 0x20 + 0x18) =
pvVar11;

```

We can see that the first four variables are a 4 byte long integer, while the last two variables are pointers

We create the following structure in ghidra based on the sizes in the structure and name the ones we know so far:

Structure Editor - spaceship (startrek-test)					
Offset	Length	Mnemonic	DataType	Name	C
0x0	0x4	int	int	spaceship_num	
0x4	0x4	int	int		
0x8	0x4	int	int	jumps_remaining	
0xc	0x4	int	int		
0x10	0x8	int *	int *		
0x18	0x8	int *	int *		

We now retype `local_3a8` to `spaceship *` and do some variable renaming and the initialization code block ends up looking like this:

```

for (curr_spaceship = 0; curr_spaceship < local_68; curr_spaceship =
curr_spaceship + 1) {
    spaceships[curr_spaceship].spaceship_num = curr_spaceship + 1;
}

```

```

spaceships[curr_spaceship].field1_0x4 = 0;
spaceships[curr_spaceship].jumps_remaining = local_6c;
spaceships[curr_spaceship].field3_0xc = 0;
lVar13 = (long)local_6c;
*(undefined8 *) (puVar14 + -0x48) = 0x1015f9;
piVar11 = (int *)malloc(lVar13 << 2);
spaceships[curr_spaceship].field4_0x10 = piVar11;
lVar13 = (long)local_6c;
*(undefined8 *) (puVar14 + -0x48) = 0x10162a;
piVar11 = (int *)malloc(lVar13 << 2);
spaceships[curr_spaceship].field5_0x18 = piVar11;
}

```

This looks way cleaner, now we need to determine what the other fields do. To do this, we look at other places in the code. We find a `jump` function that takes in the currently selected spaceship. Let's go ahead and retype and rename this parameter for the function and take a look at it:

```

void jump(spaceship *spaceship, uint param_2, long param_3)
{
    uint uVar1;
    uint local_10;

    if (((int)param_2 < 7) && (0 < (int)param_2)) {
        uVar1 = param_2 + spaceship->field1_0x4;
        printf("Jumped %d galaxies and landed on %03d\n", (ulong)param_2,
(ulong)uVar1);
        if (*(int *) (param_3 + (long)(int)uVar1 * 4) == 0) {
            local_10 = uVar1;
            if (*(int *) (param_3 + 4 + ((long)(int)uVar1 + 100) * 4) != 0) {
                local_10 = *(uint *) (param_3 + 4 + ((long)(int)uVar1 + 100) * 4);
                *(undefined4 *) (param_3 + 4 + ((long)(int)uVar1 + 100) * 4) = 0;
                printf("This galaxy contained a wormhole, you were teleported to
galaxy %03d.\n",
                    (ulong)local_10);
            }
        }
        else {
            local_10 = *(uint *) (param_3 + (long)(int)uVar1 * 4);
            *(undefined4 *) (param_3 + (long)(int)uVar1 * 4) = 0;
            printf("This galaxy contained a slipstream, you were teleported to
galaxy %03d.\n",
                (ulong)local_10);
        }
    }
}

```

```

    spaceship->field1_0x4 = local_10;
}
else {
    puts("Impossible trajectory! Are your instruments calibrated??\nYou have just wasted fuel.");
}
spaceship->jumps_remaining = spaceship->jumps_remaining + -1;
return;
}

```

It looks like `param_3` seems to be another structure, as the decompilation is calculating offsets similar to code block we analyzed earlier. We will take a closer look at that later, let's focus on changes to the `spaceship` \* structure. Based on the `printf` statements, we can see that `local_10` is the galaxy number you end up on, which is then set at `spaceship->field0x4`. So we can determine that `field1_0x4` is the `current_galaxy`. The very first `printf` also tells us that `param_2` is the number of galaxies jumped. Let's rename our variables accordingly.

Let's jump back to `main` and see what happens after the call to `jump`:

```

jump(psVar1,uVar9,(long)local_398);
spaceships[curr_spaceship].field4_0x10[spaceships[curr_spaceship].jumps_remaining] = local_3e0;

spaceships[curr_spaceship].field5_0x18[spaceships[curr_spaceship].jumps_remaining] = spaceships[curr_spaceship].current_galaxy;
if (local_70 <= spaceships[curr_spaceship].current_galaxy) {
    spaceships[curr_spaceship].jumps_remaining = 0;
    spaceships[curr_spaceship].field3_0xc = -0x45522ff3;
}

```

We notice that `field4` and `field5` are both arrays indexed by the number of jumps we have remaining. We also see that `field4` stores `local_3e0`, which is a random number from 1-6 that ends up being the amount of distance we traveled. `field5` seems to be the path of galaxies we've traveled. We'll rename `field` to `distancesPath` and `field5` to `galaxyPath`

`field3` is set when the `spaceships current_galaxy` is greater than or equal to `local_70`, which is 100. If we look a couple lines down:

```

for (curr_spaceship = 0; curr_spaceship < local_68; curr_spaceship = curr_spaceship + 1) {
    if (spaceships[curr_spaceship].field3_0xc != 0) {
        local_3d8 = local_3d8 + 1;
    }
}

```

```

if (local_3d8 == local_68) {
    *(undefined8 *) (puVar13 + -0x48) = 0x101ae7;
    puts(
        "The Architects are impressed by your Space Navigation skills. They
        have deemed Kirk and Spock worthy of the Quantum Key!"
    );
    ...
}

```

We see that we need `field3` to be set to a non-zero number for both spaceships in order to get the "Quantum Key". This means that we probably want both spaceships to reach 100. We'll rename `field3` to `reachedEnd`.

In the end we get a `spaceship` struct as such:

Structure Editor - spaceship (startrek-test)					
Offset	Length	Mnemonic	DataType	Name	Col
0x0	0x4	int	int	spaceship_num	
0x4	0x4	int	int	current_galaxy	
0x8	0x4	int	int	jumps_remaining	
0xc	0x4	int	int	reachedEnd	
0x10	0x8	int *	int *	distancesPath	
0x18	0x8	int *	int *	galaxyPath	

## Finding where the key is stored

The code past the `puts` seems to do alot of manipulations to `local_58` on `field4` and `field5` of our spaceships structures. `local_58` is likely where the flag is stored, so we will have to put a breakpoint at the end of this code block. One piece of code is interesting in this code block:

```

for (curr_spaceship = 0; curr_spaceship < 6; curr_spaceship =
curr_spaceship + 1) {
    (&local_58)[curr_spaceship] = (&local_58)[curr_spaceship] ^ local_3c8;
}

```

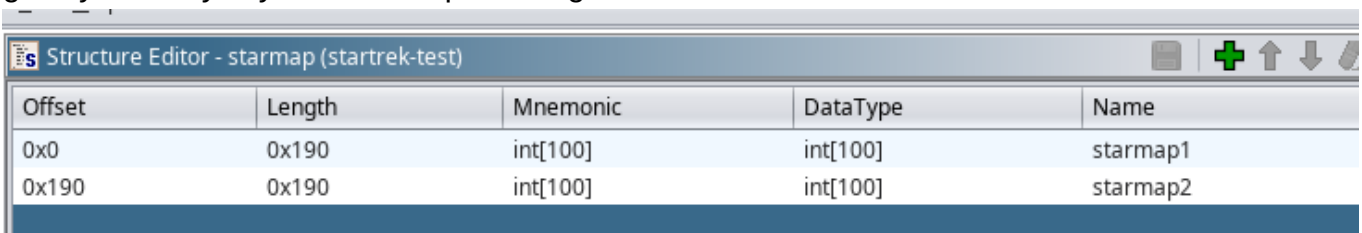
This tells us that `local_58` is an array of 6 elements. Based on other code in the code block also seems to imply that it is a `ulong`, so we can retype and rename the variable as `ulong[6]` `flag`

## Finding the starmap

Let's go back to the `jump` function and take a closer look at how `param_3` is used, which presume to be our `starmap`:

```
if (*(int *)(param_3 + (long)(int)uVar1 * 4) == 0) {
    local_10 = uVar1;
    if (*(int *)(param_3 + 4 + ((long)(int)uVar1 + 100) * 4) != 0) {
        local_10 = *(uint *)(param_3 + 4 + ((long)(int)uVar1 + 100) * 4);
        *(undefined4 *)(param_3 + 4 + ((long)(int)uVar1 + 100) * 4) = 0;
        printf("This galaxy contained a wormhole, you were teleported to
galaxy %03d.\n",
            (ulong)local_10);
    }
}
else {
    local_10 = *(uint *)(param_3 + (long)(int)uVar1 * 4);
    *(undefined4 *)(param_3 + (long)(int)uVar1 * 4) = 0;
    printf("This galaxy contained a slipstream, you were teleported to
galaxy %03d.\n",
        (ulong)local_10);
}
```

We notice the offset calculation being used similar to what was happening for the `spaceship` struct. It looks like there are two `int` arrays, one that is an `int[100]` and the other of unknown length. It looks like that the `starmap` will check the first array to see if there is a galaxy to teleport to, and if there isn't will check the second array as a backup. Once a value in the `starmap` is used, the value is set to 0, meaning that wormhole/slipstream can't be used again. We can assume the second array is also `int[100]` since the end goal is to reach galaxy 100 anyways. We end up creating a struct like this:



Offset	Length	Mnemonic	DataType	Name
0x0	0x190	int[100]	int[100]	starmap1
0x190	0x190	int[100]	int[100]	starmap2

We retype and rename `param_3` in the `jump` function and now our code block looks like this:

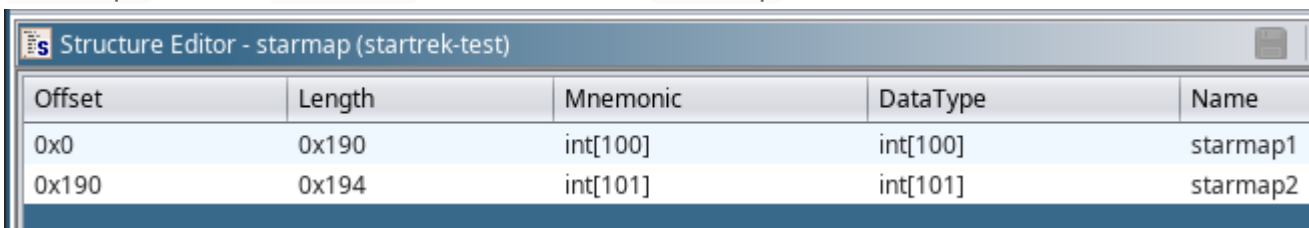
```
if (starmap->starmap1[uVar1] == 0) {
    local_10 = uVar1;
    if (starmap->starmap2[(long)uVar1 + 1] != 0) {
        local_10 = starmap->starmap2[(long)uVar1 + 1];
        starmap->starmap2[(long)uVar1 + 1] = 0;
        printf("This galaxy contained a wormhole, you were teleported to
galaxy %03d.\n",
```

```

        (ulong)local_10);
    }
}
else {
    local_10 = starmap->starmap1[uVar1];
    starmap->starmap1[uVar1] = 0;
    printf("This galaxy contained a slipstream, you were teleported to
    galaxy %03d.\n",
        (ulong)local_10);
}

```

This looks much cleaner and allows us to have a better understanding of what's going on. One thing to note is that `starmap2` actually increases the index by 1, so we need to adjust `starmap2` to be a `int[101]` This is our final `starmap` struct:



Offset	Length	Mnemonic	DataType	Name
0x0	0x190	int[100]	int[100]	starmap1
0x190	0x194	int[101]	int[101]	starmap2

Now let's go back to main and retype the variable that's the third argument in the call to `jump` as `starmap`. Now if we look at the beginning of the `main` function we see our `starmap`:

```

starmap.starmap1[70] = 0x5b;
starmap.starmap1[5] = 0xf;
starmap.starmap1[19] = 0x29;
starmap.starmap1[4] = 0x4b;
starmap.starmap1[28] = 0x32;
starmap.starmap1[35] = 0x60;
starmap.starmap1[44] = 0x52;
starmap.starmap1[53] = 0x5e;
starmap.starmap1[59] = 0x5f;
starmap.starmap2[99] = 0xc;
starmap.starmap2[89] = 0x43;
starmap.starmap2[82] = 0x3e;
starmap.starmap2[77] = 0x29;
starmap.starmap2[53] = 0x17;
starmap.starmap2[48] = 0x1e;
starmap.starmap2[32] = 8;
starmap.starmap2[22] = 3;
USCGSIV{77ad9e80_C31e5tia1_NaV7g@t10n...Check!}```

```

Now we have our full ``starmap`` and ``spaceship`` struct figured out. Now we just need to control our movement so that we can get to 100 in 5 jumps.

```
## Controlling movement
The following two lines control the distance our spaceships go:
```C
    rand_num = rand();
    distance_traveled = rand_num % 6 + 1;
```

`rand()` is a C function that we can overwrite with `LD_PRELOAD`. The following C code can be compiled into an `.so` file to control the numbers called by `rand`:

```
#include<stdio.h>

int i = 0;
int fake_rand[] = {1,1,1,1,1,1,1,1,1,1};

int rand() {
    return fake_rand[i++]-1;
}
```

If we compile this and then hook `rand` with `LD_PRELOAD` we see that the spaceships only ever jump 1 galaxy every time:

```
→ startrek LD_PRELOAD=./test.so ./startrek
Captains Kirk and Spock both begin in Galaxy 000. Their ships' fuel levels
are both at 100%. Good luck! If you help the Captains accomplish their
mission, you will surely Live Long and Prosper!
```

```
Starship: 1      Current Galaxy: 000      Jumps Remaining: 5
Preparing the ship for the next jump...
Do you want to tell the Captain to alter his trajectory (y/n)?
n
Jumped 1 galaxies and landed on 001
```

```
Starship: 2      Current Galaxy: 000      Jumps Remaining: 5
Preparing the ship for the next jump...
n
n
n
n
Do you want to tell the Captain to alter his trajectory (y/n)?
Jumped 1 galaxies and landed on 001
```



```
Starship: 1      Current Galaxy: 001      Jumps Remaining: 4
Preparing the ship for the next jump...
n
Do you want to tell the Captain to alter his trajectory (y/n)?
Jumped 1 galaxies and landed on 002
```

```
Starship: 2      Current Galaxy: 001      Jumps Remaining: 4
Preparing the ship for the next jump...
Do you want to tell the Captain to alter his trajectory (y/n)?
Jumped 1 galaxies and landed on 002
```

## Finding the Path

By trial and error, the correct path for both spaceship 1 and 2 were found. `Startrek2` goes over finding the shortest path algorithmically. The paths are so:

```
spaceship1:
0  +5 5 = 15
15 +4 19 = 41
41 +6 47 = 30
30 +5 35 = 96
96 +4 101 = 100
```

```
spaceship2:
0 +4 4 = 75
75 +1 76 = 41
41 +6 47 = 47
47 +6 53 = 94
94 +6 100 = 100
```

In order to make these jumps our `.so` code needs to be modified to following:

```
#include<stdio.h>

int i = 0;
int fake_rand[] = {5,4,4,1,6,6,5,6,4,6};
int rand() {
```

```
        return fake_rand[i++] - 1;
    }
```

We run the code with `LD_PRELOAD` and get the success message, however the program promptly exits:

```
...
Starship: 1      Current Galaxy: 096      Jumps Remaining: 1
Preparing the ship for the next jump...
Do you want to tell the Captain to alter his trajectory (y/n)?
Jumped 4 galaxies and landed on 100

Starship: 2      Current Galaxy: 094      Jumps Remaining: 1
Preparing the ship for the next jump...
Do you want to tell the Captain to alter his trajectory (y/n)?
Jumped 6 galaxies and landed on 100
The Architects are impressed by your Space Navigation skills. They have
deemed Kirk and Spock worthy of the Quantum Key!
```

## Getting the flag

While the flag is generated in memory, it is not actually printed out. We will need to use `gdb` to break right before the program exits and dump the flag from memory. We break at `*main+2640` as the `jmp` right after all the flag manipulation is done. The flag is stored in `rbp-0x50` so we can just print out the string that sits there:

```
gef> x/s $rbp-0x50
0x7fffffffdc70: "****USCGSIV{148c1252d_U_solved_it@Warp_Sp33d}****"
```

## Flag

```
USCGSIV{148c1252d_U_solved_it@Warp_Sp33d}
```

## Startrek2

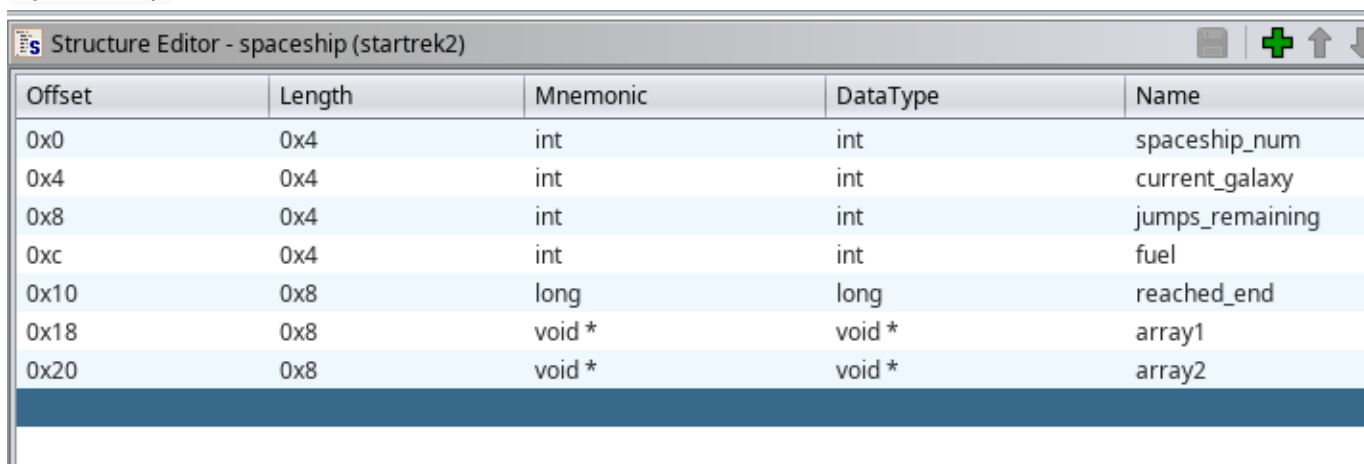
## Adjusting for differences

The following changes are different from `startrek1` to `startrek2`:

- Spaceships are given 8 jumps instead of 5
- Spaceships can jump anywhere from 1-8 galaxies vs the previous 1-6
- There is the option for a ship-to-ship fuel transfer in the beginning, allowing one ship to go farther at the cost of the other one going shorter
- The end goal is galaxy 1600 instead of galaxy 100
- The `spaceship` struct has an extra `int` in it's structure which is related to the fuel transfer

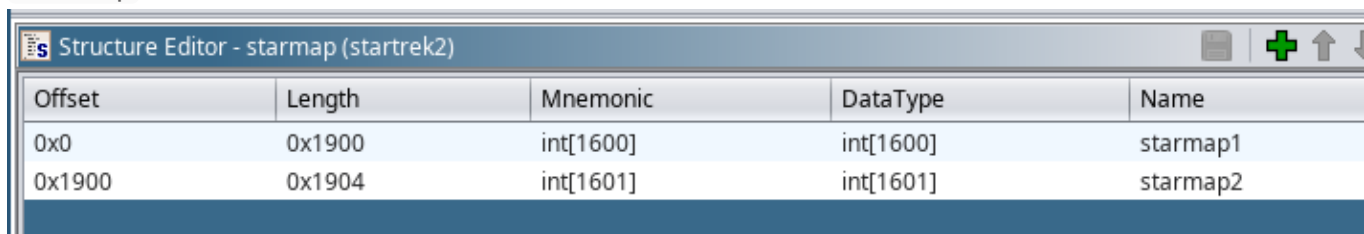
Let's update our structs so that they account for these differences.

`spaceship`:



Offset	Length	Mnemonic	DataType	Name
0x0	0x4	int	int	spaceship_num
0x4	0x4	int	int	current_galaxy
0x8	0x4	int	int	jumps_remaining
0xc	0x4	int	int	fuel
0x10	0x8	long	long	reached_end
0x18	0x8	void *	void *	array1
0x20	0x8	void *	void *	array2

`starmap`:



Offset	Length	Mnemonic	DataType	Name
0x0	0x1900	int[1600]	int[1600]	starmap1
0x1900	0x1904	int[1601]	int[1601]	starmap2

Updating the `starmap` struct reveals the full starmap at the top of the `main` function:

```
starmap.starmap1[8] = 0xce;
starmap.starmap1[2] = 0xb5;
starmap.starmap1[410] = 0x590;
starmap.starmap1[1479] = 0x62c;
starmap.starmap1[1556] = 0x633;
starmap.starmap1[1287] = 0x565;
starmap.starmap1[1006] = 0x621;
starmap.starmap1[369] = 400;
starmap.starmap1[1426] = 0x639;
starmap.starmap1[208] = 0x324;
```

```
starmap.starmap1[884] = 0x5dd;
starmap.starmap1[897] = 0x619;
starmap.starmap1[561] = 0x4f6;
starmap.starmap1[1422] = 0x600;
starmap.starmap1[160] = 0x4db;
starmap.starmap1[209] = 0x3b1;
starmap.starmap1[717] = 0x3dd;
starmap.starmap1[606] = 0x3e5;
starmap.starmap1[18] = 0x55a;
starmap.starmap1[981] = 0x59c;
starmap.starmap1[280] = 0x518;
...many more lines of starmap defintions
```

In order to determine the correct path, we can no longer use trial and error as the starmap is simply too long. We will have to find a way to solve the star map algorithmically.

## Finding optimal paths

The following python script uses a modified version of bfs to determine the optimal paths for both spaceships to take, based on the starmap in the program:

```
from collections import deque
starmap1 = [0] * 1600
starmap2 = [0] * 1601
starmap1[8] = 0xce
starmap1[2] = 0xb5
starmap1[410] = 0x590
starmap1[1479] = 0x62c
starmap1[1556] = 0x633
starmap1[1287] = 0x565
starmap1[1006] = 0x621
starmap1[369] = 400
starmap1[1426] = 0x639
starmap1[208] = 0x324
starmap1[884] = 0x5dd
starmap1[897] = 0x619
starmap1[561] = 0x4f6
starmap1[1422] = 0x600
starmap1[160] = 0x4db
starmap1[209] = 0x3b1
starmap1[717] = 0x3dd
starmap1[606] = 0x3e5
starmap1[18] = 0x55a
starmap1[981] = 0x59c
```

```
starmap1[280] = 0x518
starmap1[587] = 0x2d8
starmap1[1430] = 0x5d5
starmap1[389] = 0x510
starmap1[1460] = 0x62d
starmap1[1348] = 0x622
starmap1[626] = 0x424
starmap1[1451] = 0x5d2
starmap1[383] = 0x2a4
starmap1[1509] = 0x61c
starmap1[675] = 0x313
starmap1[467] = 0x574
starmap1[568] = 0x5e2
starmap1[1104] = 0x4a7
starmap1[856] = 0x413
starmap1[832] = 0x46e
starmap1[928] = 0x52a
starmap1[1549] = 0x61f
starmap1[476] = 0x1e9
starmap1[986] = 0x43d
starmap1[1274] = 0x597
starmap1[399] = 0x624
starmap1[140] = 0x132
starmap1[765] = 0x54c
starmap1[705] = 0x4cc
starmap1[1547] = 0x632
starmap1[1291] = 0x561
starmap1[1435] = 0x5b8
starmap1[132] = 0x416
starmap1[1213] = 0x588
starmap1[978] = 0x4ed
starmap1[275] = 0x62a
starmap1[246] = 0x36e
starmap1[104] = 0x45c
starmap1[1489] = 0x5f0
starmap1[67] = 0x317
starmap1[841] = 0x51f
starmap1[638] = 0x28b
starmap1[370] = 0x331
starmap1[288] = 0x440
starmap1[996] = 0x4e2
starmap1[1076] = 0x620
starmap1[1295] = 0x63b
starmap1[1443] = 0x5c3
starmap1[1535] = 0x624
starmap1[747] = 0x378
```

```
starmap1[1227] = 0x5f8
starmap1[139] = 0x3df
starmap1[77] = 0x1e9
starmap1[818] = 0x4fc
starmap1[1074] = 0x457
starmap1[1121] = 0x55d
starmap1[1498] = 0x613
starmap1[1234] = 0x5b2
starmap1[795] = 0x3db
starmap1[1183] = 0x63e
starmap1[1266] = 0x611
starmap1[376] = 0x584
starmap1[1126] = 0x544
starmap1[322] = 0x45b
starmap1[404] = 0x5c1
starmap1[81] = 0x2df
starmap1[916] = 0x3fc
starmap1[499] = 0x400
starmap1[91] = 0x4d0
starmap1[1084] = 0x54b
starmap1[514] = 0x214
starmap1[270] = 0x501
starmap1[1066] = 0x4df
starmap1[953] = 0x59b
starmap1[125] = 900
starmap1[301] = 0x514
starmap1[1200] = 0x638
starmap1[1196] = 0x52c
starmap1[1223] = 0x550
starmap1[865] = 0x39e
starmap1[156] = 0x5c7
starmap1[118] = 0x32a
starmap1[966] = 0x4c1
starmap1[250] = 0x5df
starmap1[1557] = 0x631
starmap1[16] = 0x51c
starmap1[334] = 0x3fa
starmap1[284] = 0x4ae
starmap1[1298] = 0x595
starmap1[1162] = 0x4d0
starmap1[793] = 0x49a
starmap1[236] = 0x539
starmap1[896] = 0x3b9
starmap1[291] = 0x4a8
starmap1[1301] = 0x5b6
starmap1[279] = 0x520
```

```
starmap1[678] = 0x440
starmap1[954] = 0x4d2
starmap1[1456] = 0x5ce
starmap1[779] = 0x61f
starmap1[124] = 0x2ab
starmap1[725] = 0x60a
starmap1[1358] = 0x5e0
starmap1[946] = 0x484
starmap1[634] = 0x58d
starmap1[864] = 0x600
starmap1[773] = 0x376
starmap1[1141] = 0x4b5
starmap1[298] = 0x298
starmap1[1305] = 0x61a
starmap1[1463] = 0x628
starmap1[677] = 0x43f
starmap1[639] = 0x28b
starmap1[1243] = 0x596
starmap1[1270] = 0x5f3
starmap1[848] = 0x3f0
starmap1[1165] = 0x4b6
starmap1[1500] = 0x62a
starmap2[618] = 0x1b3
starmap2[1298] = 0x118
starmap2[93] = 0x19
starmap2[1050] = 0x1ce
starmap2[850] = 0x3b
starmap2[242] = 0xcd
starmap2[222] = 0x2d
starmap2[1038] = 0x115
starmap2[1504] = 0x3a
starmap2[262] = 0x3f
starmap2[664] = 0x219
starmap2[194] = 0x65
starmap2[225] = 0x2b
starmap2[953] = 0x11f
starmap2[663] = 400
starmap2[1073] = 0x408
starmap2[15] = 2
starmap2[928] = 0xa7
starmap2[72] = 3
starmap2[412] = 0x27
starmap2[1562] = 0x3f3
starmap2[911] = 0x26f
starmap2[318] = 0xf8
starmap2[1409] = 0x2a4
```

```
starmap2[972] = 0x16e
starmap2[615] = 0xc3
starmap2[277] = 0x10a
starmap2[361] = 0xce
starmap2[101] = 0x13
starmap2[1454] = 0x41d
starmap2[98] = 0x2b
starmap2[1031] = 0x172
starmap2[551] = 0x205
starmap2[446] = 0x18e
starmap2[1471] = 0x4a8
starmap2[79] = 0x42
starmap2[996] = 0x8b
starmap2[542] = 0x205
starmap2[1125] = 0x116
starmap2[973] = 0x10a
starmap2[111] = 0xc
starmap2[581] = 0x14f
starmap2[259] = 0x5f
starmap2[1134] = 0x2a3
starmap2[571] = 0x21d
starmap2[1096] = 0x27d
starmap2[606] = 0x244
starmap2[26] = 10
starmap2[1412] = 0x1ff
starmap2[1519] = 0x3e4
starmap2[127] = 0x55
starmap2[1018] = 0x219
starmap2[215] = 0x20
starmap2[97] = 0x4e
starmap2[494] = 0x9f
starmap2[992] = 0x202
starmap2[1380] = 0x27c
starmap2[701] = 0x253
starmap2[295] = 0x4c
starmap2[1089] = 0x2c0
starmap2[1203] = 0x3d2
starmap2[1600] = 0x228
starmap2[309] = 0x108
starmap2[1237] = 0x435
starmap2[139] = 3
starmap2[298] = 0x10
starmap2[1239] = 4
starmap2[1492] = 0xe1
starmap2[703] = 0xb0
starmap2[1313] = 0x5e
```



```
starmap2[1094] = 0x2b2
starmap2[1264] = 0x26d
starmap2[797] = 0x22
starmap2[338] = 0xa2
starmap2[1047] = 0x398
starmap2[782] = 0x2f7
starmap2[806] = 0x1e7
starmap2[528] = 0x16c
starmap2[300] = 0x9c
starmap2[1257] = 0x1e7
starmap2[858] = 0x1b6
starmap2[401] = 0xc9
starmap2[329] = 0xf1
starmap2[1434] = 0x3b6
starmap2[462] = 0x17f
starmap2[143] = 0x57
starmap2[286] = 199
starmap2[254] = 0xdd
starmap2[1027] = 0x1b
starmap2[179] = 99
starmap2[57] = 0x25
starmap2[1379] = 0x1c2
starmap2[772] = 0x1e3
starmap2[811] = 0x9b
starmap2[1110] = 0x2a1
starmap2[132] = 0x37
starmap2[1211] = 0x490
starmap2[1175] = 0x30a
starmap2[1210] = 0x113
starmap2[661] = 0x179
starmap2[178] = 0x43
starmap2[970] = 0x224
starmap2[1522] = 0x415
starmap2[1462] = 0x1a9
starmap2[1491] = 0x331
starmap2[1119] = 0xf2
starmap2[1147] = 0x210
starmap2[1287] = 0x28a
starmap2[1354] = 0x21e
starmap2[652] = 199
starmap2[641] = 0x1db
starmap2[1483] = 0x4da
starmap2[270] = 0xe9
starmap2[1155] = 0x44d
starmap2[940] = 0x394
starmap2[1227] = 0x46b
```

```
starmap2[1448] = 0x372
starmap2[171] = 0x76
starmap2[632] = 0x19a
starmap2[948] = 0x344
starmap2[415] = 0x170
starmap2[76] = 0x11
starmap2[835] = 0x29
starmap2[1384] = 0x288
starmap2[1112] = 0x2cf
starmap2[207] = 0x5a
starmap2[1532] = 0x72
starmap2[469] = 0x58
starmap2[252] = 0xd8
starmap2[1441] = 0x369
```

```
#print(starmap1)
#print(starmap2)
```

```
# get the potential next step, and then the potential state afterwards
```

```
def getAdjacents(node):
```

```
    adjacent = []
```

```
    curr_starmap1 = node['starmap1']
```

```
    curr_starmap2 = node['starmap2']
```

```
    curr_index = node['curr_index']
```

```
    for i in range (1,9):
```

```
        if curr_index+i == 1600:
```

```
            roll = { 'curr_index': curr_index+i, 'increment': i,
'starmap1': curr_starmap1.copy(), 'starmap2': curr_starmap2.copy() }
            adjacent.append(roll)
```

```
            continue
```

```
        if curr_index+i > 1600:
```

```
            continue
```

```
            roll = { 'curr_index': curr_index+i, 'increment': i, 'starmap1':
curr_starmap1.copy(), 'starmap2': curr_starmap2.copy() }
```

```
            if curr_starmap1[curr_index+i] != 0:
```

```
                new_starmap1 = curr_starmap1.copy()
```

```
                new_starmap1[curr_index+i] = 0
```

```
                roll = { 'curr_index': curr_starmap1[curr_index+i], 'increment':
i, 'starmap1': new_starmap1, 'starmap2': curr_starmap2.copy() }
```

```
            elif curr_starmap2[curr_index+i+1] != 0:
```

```
                new_starmap2 = curr_starmap2.copy()
```

```
                new_starmap2[curr_index+i+1] = 0
```

```
                roll = { 'curr_index': curr_starmap2[curr_index+i+1],
'increment': i, 'starmap1': curr_starmap1.copy(), 'starmap2': new_starmap2
```

```

    }

    adjacent.append(roll)

    return adjacent

def bfs(start, end):

    initial_node = { 'curr_index': start, 'increment': 0, 'starmap1':
starmap1.copy(), 'starmap2': starmap2.copy() }
    queue = deque([[initial_node]])

    visited = set()

    queue.append([initial_node])
    while queue:
        path = queue.popleft()

        node = path[-1]

        if node['curr_index'] == end:
            return path

        if node['curr_index'] not in visited:
            visited.add(node['curr_index'])

            for adjacent in getAdjacents(node):
                #print(f'path: {path}')
                #input()
                new_path = list(path)
                new_path.append(adjacent)
                queue.append(new_path)

    return None

path = bfs(0, 1600)

node = path[-1]

print(f'fuel required: {len(path)-1}')
for i in path:
    print(f'{i["curr_index"]} +{i["increment"]}')

starmap1 = node['starmap1']
starmap2 = node['starmap2']

new_path = bfs(0,1600)

```

```
print(f'fuel required: {len(new_path)-1}')
for i in new_path:
    print(f'{i["curr_index"]} +{i["increment"]}')

```

When traversing the graph, we store the current index, plus the increment that was used to get there. This allows us to see exactly what distances we need the spaceships to travel to get to the end. We also store the current state of the starmaps, since we need to close the portal so that it can't be used again just like the code does. After our first run through of the algorithm, we get the end state of the starmaps and then run the algorithm again to determine the path for the second ship. Running this code we get the following:

```
fuel required: 6
0 +0
206 +8
945 +3
287 +7
1192 +4
1592 +8
1600 +8
fuel required: 10
0 +0
8 +8
1308 +8
94 +4
43 +3
51 +8
59 +8
791 +8
1178 +2
1598 +5
1600 +2

```

We see that both ships get to 1600, and that the fuel required adds up to 16, the total amount of fuel we have to work with.

## Getting the flag

We modify our `.so` file with the distances we found, so that we can control `rand()` and get both ships to the end. The result `.so` file is as so:

```
#include<stdio.h>

int i = 0;
int fake_rand[] = {8,8,3,8,7,4,4,3,8,8,8,8,8,2,5,2};

int rand() {
    return fake_rand[i++] - 1;
}
```

Now let's open the program in gdb with `LD_PRELOAD` set, and right before the exit at `*main+5651`. We'll transfer 2 fuel cells to the second ship and then type `n` until both spaceships reach the end. Eventually we hit the breakpoint and we need to dump the flag from memory which is at `rbp-0x70`. We do this and get the flag:

```
gef> x/s $rbp-0x70
0x7fffffffdc40: "~USCGSIV{77ad9e80_C31e5tia1_NaV7g@t10n...Check!}"
```

## Flag

USCGSIV{77ad9e80\_C31e5tia1\_NaV7g@t10n...Check!}