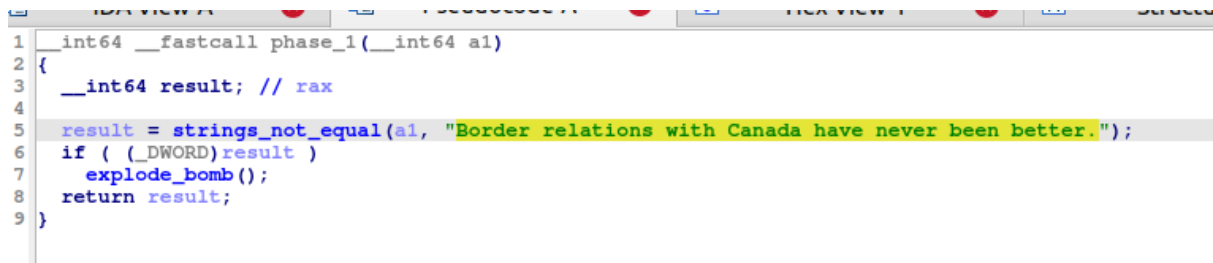


This program says that we have 7 phases to detonate the bomb, but we need to solve only the first four (as per the exercise description)

Using IDA and Ghidra and then using the decompiled code (press F5 to get it), we can see:

FIRST STAGE

A screenshot of the IDA Pro decompiler window showing the assembly and decompiled code for a function named phase_1. The decompiled code is as follows:

```
1  __int64 __fastcall phase_1(__int64 a1)
2  {
3      __int64 result; // rax
4
5      result = strings_not_equal(a1, "Border relations with Canada have never been better.");
6      if ( (_DWORD)result )
7          explode_bomb();
8      return result;
9  }
```

The string "Border relations with Canada have never been better." is highlighted in yellow.

This is the first answer to provide.

SECOND STAGE

The second stage reads 6 numbers and perform some operations, it reads the numbers with a space between them:

From Ghidra, we can see that the first number is equal to 1, and the other are the multiplication by 2 of the current number

```
read_six_numbers(param_1,local_38);
if (local_38[0] != 1) {
    explode_bomb();
}
piVar1 = local_38 + 1;
do {
    if (*piVar1 != piVar1[-1] * 2) {
        explode_bomb();
    }
    piVar1 = piVar1 + 1;
} while (piVar1 != local_20);
return;
```

INPUT: 1 2 4 8 16 32

THIRD STAGE

In ghidra we can change the hexadecimal to integers, in order to see the code in a clear way:

```
void phase_3(undefined8 param_1)

{
    int iVar1;
    uint local_10;
    int local_c [3];

    iVar1 = __isoc99_sscanf(param_1,"%d %d",&local_10,local_c);
    if (iVar1 < 2) {
        explode_bomb();
    }
    switch(local_10) {
    case 0:
        iVar1 = 207;
        break;
    case 1:
        iVar1 = 311;
        break;
    case 2:
        iVar1 = 707;
        break;
    case 3:
        iVar1 = 256;
        break;
    case 4:
        iVar1 = 389;
        break;
    case 5:
        iVar1 = 206;
        break;
    case 6:
        iVar1 = 682;
        break;
    case 7:
        iVar1 = 327;
        break;
    default:
        explode_bomb();
        iVar1 = 0;
    }
    if (iVar1 != local_c[0]) {
        explode_bomb();
    }
    return;
}
```

It asks two numbers in input, the first goes into the switch case and you need to set the second with the same number (last check)

One input example is: 0 207

FOURTH STAGE

This stage asks for two number and we can see that the phase calls another function **func4**

```
void phase_4(undefined8 param_1)
{
    int iVar1;
    uint local_10;
    int local_c [3];

    iVar1 = __isoc99_sscanf(param_1,"%d %d",&local_10,local_c);
    if ((iVar1 != 2) || (14 < local_10)) {
        explode_bomb();
    }
    iVar1 = func4(local_10,0,14);
    if ((iVar1 != 0) || (local_c[0] != 0)) {
        explode_bomb();
    }
    return;
}
```

Func4 code is as follow:

```
int func4(int input,undefined8 param_2,int param_3)
{
    int iVar1;
    int iVar2;

    iVar2 = (param_3 - (int)param_2) / 2 + (int)param_2;
    if (input < iVar2) {
        iVar1 = func4(input,param_2,iVar2 + -1);
        iVar1 = iVar1 * 2;
    }
    else {
        iVar1 = 0;
        if (iVar2 < input) {
            iVar2 = func4(input,iVar2 + 1);
            iVar1 = iVar2 * 2 + 1;
        }
    }
}
```

```
}  
return iVar1;  
}
```

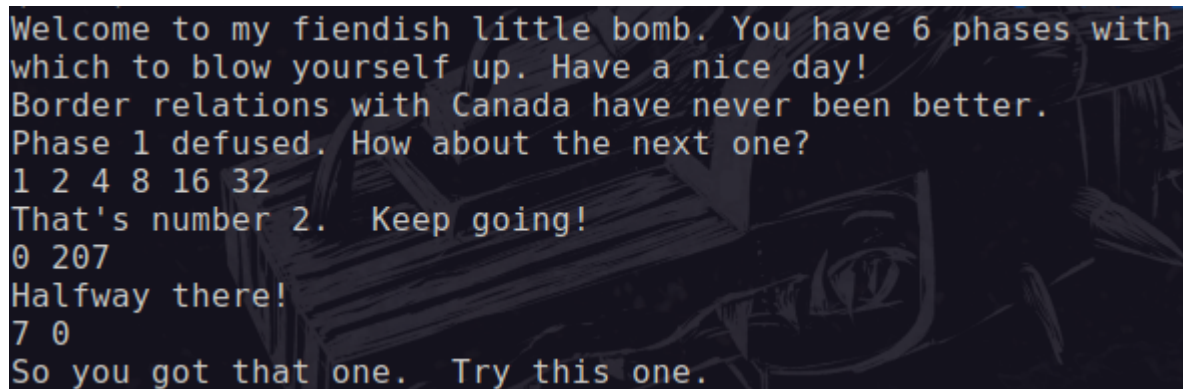
Here we can understand that the function is recursive, and the phase is satisfied only if the result of this function is equal to 0, as the second number we put in input (last if condition of phase4 code).

We can infer from func4 code that it gets the first number as input, along with numbers 0 and 14. It starts seeing if the number we provide is less than $[(14-0) / 2 - 0]$ (ivar2 first line code).

We can satisfy the condition to return 0 using in input:
INPUT: 7 0

And with this we finish the challenge.

The complete sequence of inputs is



```
Welcome to my fiendish little bomb. You have 6 phases with  
which to blow yourself up. Have a nice day!  
Border relations with Canada have never been better.  
Phase 1 defused. How about the next one?  
1 2 4 8 16 32  
That's number 2. Keep going!  
0 207  
Halfway there!  
7 0  
So you got that one. Try this one.
```