

Solutions for Pointer

Pointers

Pointers - I

Most exercises have source code. Look in the **Solutions/src** folder.

1.

You'll see nothing happening, but can be pretty sure that an int has been declared and assigned 12.

2.

Same as for (1) but now the value of students is printed to stdout.

3.

Sames as for (2). A pointer has also been created.

4.

A pointer (in this case an address) is most likely bigger than the max value of an int so the compiler warns about this.

5.

Yes, it works.

6.

We can assign students by dereferencing the **studentsp** variable. Since **studentsp** has been assigned the address of **students** we thereby assign **student** the value.

Think of it like we're getting the address of students, stores the address on a piece of paper (studentsp) and then go to that address and assigns a value

7.

Creates a long variable and a pointer to long variable.

8.

salary is assigned the value 5000. salary_ptr is assigned the address of salary, so salary_ptr points to the memory where the value of salary is stored.

9.

salary has the value 60000 after the new line has been executed.

Pointers - II

20.

It swaps the values of x and y. Since a and y are local variables (with values copied in to them when the function is invoked) it really doesn't do anything useful.

21.

12 is 12. 34 is 34.

It's math :)

Nobel award music is playing in the background now!

22.

See (20).

23.

No. Tons of warnings. When running the program it crashes since we're reading (when dereferencing the pointers x and y which points to memory outside of our program).

24.

By using the addresses the swap function has swapped the values of the variables girls and boys.

25.

see below (26).

26.

It substitutes the call to `swap_int` with the ... ahh, let's look at the code after the preprocessor has been invoked instead. Let's simply do `gcc -E pointer-25.c`

This will give us the following code:

```
...
snip
...
int main() {

    int girls = 12;
    int boys = 11;

    printf ("girls: %d  boys: %d\n",
            girls, boys);

    int tmp=x; x=y; y=x;;

    printf ("girls: %d  boys: %d\n",
            girls, boys);

    return 0;
}
```

27.

See source code.

28.

The program compiles but crashes when executed. It crashes, with a segmentation fault, since we're accessing memory outside of our program.

29.

See source code.

Pointers - III

30.

8 (on the author's computer/OS).

31.

8 (on the author's computer/OS).

32.

8 (on the author's computer/OS).

32.

8 (on the author's computer/OS).

33.

8 (on the author's computer/OS).

34.

123 slots or 123 bytes big - there are 123 charactes in the array. Each character is one byte big. The size of an array is known so the compiler has no problems knowing this.

35.

a has the size 123 - see (34).

ap is 8 bytes big. It is a pointer, and pointers are (as you may have found out before) of this size on the author's computer.

36.

Both are of size 8. "AS Roma" has 7 characters and thogether with the trailing \0 there are 8 characters in the array.

8 is the size of a pointer ... bla bla bla

37.

984 slots or 123 bytes big - there are 123 `long long` in the array. Each `long long` is 8 byte big on the author's computer/OS. The size of an array is known so the compiler has no problems knowing this.

BTW: $1238 = 984^$*

*Note: If you want to calculate thing in a Unix environment you can use the command `bc` - either interactively or in a one-liner like this: `echo "123*8" | bc`*

Pointers - IV

40.

See the source code.

41.

See the source code.

42.

See the source code.