

Exercises for 01-Pointer

Pointers

A bit of recap first:

`int x;` - declares an int variable, x.

`x=13;` - assigns 13 to x.

`int *xp;` - declares a pointer to int, p

`xp = &x;` - assigns the address of x to xp

`*xp = 12;` - assigns 12 to the memory xp points to (same memory as x).

x now has the value 12.

Pointers - I

1.

Write a program that, in the main function:

- declares an integer variable, called **students**.
- assigns the variable 12.

Compile the program. What do you think will happen when you execute it. Execute the program to verify.

2.

Add a printout of the variable **students**.

3.

To the program in (1):

- add a variable, an integer pointer, called **studentsp**.
- assign the variable the address of students.

Compile the program. What do you think will happen when you execute it. Execute the program to verify.

4.

Add a printout of the variable `studentsp`. Try printing it as a decimal using `%d`.

Compile the program. The compiler (hopefully) warns you. Read the warning and think about it.

5.

What happens if you use `%p` to format instead? Read the manual of `printf`.

6.

`studentsp` points `students` so `studentsp` is an indirect way of accessing `students`.

You should now Assign a new value of `students` using `studentsp`.

Hint: use the dereference `` operator*

7.

Look at the following code.

```
long salary;  
long *salary_ptr;
```

What does the code do?

8.

Look at the following code.

```
long salary;  
long *salary_ptr;  
  
salary = 50000; /*dream on*/  
salary_ptr = &salary;
```

The value of `salary_ptr` is related to `salary`. In what way?

9.

Look at the following code.

```
long salary;
long *salary_ptr;

salary = 50000; /*dream on*/
salary_ptr = &salary;

*salary_ptr = 60000;
```

What is the value of salary after all the lines above have been executed?

Pointers - II

Recap:

When calling/invoking a function, the supplied arguments are copied to the function - regardless of whether they initially were variables or literals, e.g.:

```
int x=2;
int y=4;
int res = add(x,y); /* the values of x and y are copied to the function add */
res = add(12, 34); /* the values 12 and 34 are copied to the function add */
```

20.

Look at the following function.

```
void swap (int x, int y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

What does it do?

21.

Call the function `swap` with 12 and 34 as arguments. What are the values of 12 and 34 after the function has been executed?

Hint: it really is a trick question. Stop wasting energy!

22.

Declare two integer variables, **girls** and **boys**, and assign them values. Call the function **swap**. What are the values of **boys** and **girls**?

Hint: don't ask the supervisors what will happen. . . . write the code, compile and test it!!. Have fun!

22.

Look at the following function.

```
void swap (int *x, int *y) {  
    int tmp = *x;  
    *x = *y;  
    *y = tmp;  
}
```

What does it do?

23.

Declare two integer variables, **girls** and **boys**, and assign them values. Call the function **swap** with **girls** and **boys** as arguments.

Does it compile without warning?

24.

Declare two integer variables, **girls** and **boys**, and assign them values. Call the function **swap** with the addresses of **girls** and **boys** as arguments.

What are the values of **boys** and **girls** after the code has been executed?

25.

Look at the following function.

```
#define swap_int(a,b) int tmp=a; a=b; b=tmp;
```

What does it do?

26.

Declare two integer variables, `girls` and `boys`, and assign them values. Call the macro `swap_int` with the addresses of `girls` and `boys` as arguments.

What are the values of `boys` and `girls` after the code has been executed?

27.

Write a function, `doubler`, that takes a pointer to `int` as parameter. The value pointed to by the parameter shall be doubled. The function shall return `void`.

Also make sure to test the function.

28.

Try the function in (27) by supplying `NULL` as argument.

What do you think will happen? Will it compile? Will it execute nicely?

Make the function in (27) safe.

29.

Add a pointer to `int` as a second arg to `doubler`. The function shall now return an integer value (0 if successful doubling, 1 otherwise). The memory pointed to by the second argument should be assigned the resulting doubling. The prototype of the function should look something like:

```
int doubler (int *x, int *result);
```

Make sure to test the function.

Pointers - III

30.

What is the size of a `char *`?

31.

What is the size of a `int *`?

32.

What is the size of a `long *`?

33.

What is the size of a `long long *`?

34.

What is the size of the variable a `char a[123]`?

35.

Look at the following code:

```
char a[123];  
char *ap;  
ap = a;
```

What are the sizes of `a` and `ap`?

36.

Look at the following code:

```
char a[]="AS Roma";  
char *ap;  
ap = a;
```

What are the sizes of `a` and `ap`?

37.

Look at the following code:

```
long l[123];  
long *lp;
```

What are the sizes of `l` and `lp`?

Pointers - IV

Recap:

Pointers and arrays are not the same thing. Really, they're not!

40.

Traverse the `int` array in the code below and print out the integers one by one.

```
int a[] = {9,8,7,6,5,4,3,2,1};
```

41.

Traverse the `int *` in the code below and print out the integers one by one. You can assume the number of elements are 9.

```
int a[] = {9,8,7,6,5,4,3,2,1};  
int *ap=a;
```

42.

Traverse the `char *` in the code below and print out the integers one by one. You can assume that the string is ending with a terminator sign.

```
char *str = "Henrik is really stupid. Yup, he is";
```

43. (really optional!)

Contemplate on this.

```
#include <stdio.h>  
#include <string.h>  
  
int main() {  
  
    int a[] = { 4,3,2,1 };  
    int *ap=a;  
  
    printf (" a[0]:    %d\n", a[0]);  
    printf (" *ap:      %d\n", *ap);  
    printf (" a[1]:    %d\n", a[1]);  
    printf (" *(ap+1): %d\n", *(ap+1));  
}
```

```
printf (" *(1+ap): %d\n", *(1+ap));  
printf (" 1[ap]:   %d\n", 1[ap]);  
  
return 0;  
}
```