Laboratory 9 – Pointers

07.12.2021

1. [3points] Please follow the instructions step by step:

a) Declare a variable of type **float** x and two pointers to **float** pa, pb.
b) Assign a value to the variable x.
c) Assign the address of the variable x to the pointer pa.
d) Assign a null pointer constant to the pb pointer. NULL is called a null pointer constant. NULL is defined in e.g. <stdio.h>. Just use it!
e) Print the addresses of all variables.
f) Print the values of all variables.
g) Copy the value of the pa pointer into the pb pointer.
h) Print the values of all variables.
i) Use the pb pointer to modify the value of the variable x.
j) Print the values of all variables.
k) Print the values that are at the addresses that the pointers store.

Test data:

The addresses will differ on different computers!

```
e) A: &x = 0x7ffcecd0dd4c,
          &pa = 0x7ffcecd0dd40,
    &pb = 0x7ffcecd0dd38

f) V1: x = 12.51,
     pa = 0x7ffcecd0dd4c,
     pb = (nil)    <-- The printf function prints a null pointer constant as (nil).

h) V2: x = 12.51,
     pa = 0x7ffcecd0dd4c,
     pb = 0x7ffcecd0dd4c

j) V3: x = 3.14,
      pa = 0x7ffcecd0dd4c,
      pb = 0x7ffcecd0dd4c

k) V4: *pa = 3.14,
      *pb = 3.14
```

2. [3points] Please follow the instructions step by step:

a) Declare two variables of type **double** a, b.
b) Declare two pointers to type **double** p1, p2. Set them to NULL when declaring.
c) Print the addresses of all variables.
d) Print the values of the pointers.
e) Assign the address of the variable a to pointer p1.

f) Assign the address of the variable b to pointer p2.
g) Print the values of the pointers.
h) Use the scanf function and the pointers p1 and p2 to set values to a and b.

Remember!
The scan function uses addresses that are the values of p1 and p2. The function cannot modify the values of p1 and p2, but it can use them to modify a and b.
You can also pass the addresses of a and b to the scanf function directly without using p1 and p2, but this is not the subject of this exercise.

i) Print the values of all variables.
j) Declare 4 variables of **double** type: sum, diff, prod, quot. Using the pointers p1 and p2 and not using the variables a and b, calculate the sum, difference, product and quotient of a and b.
k) Print the results using the variables: sum, diff, prod, quot, p1, p2.
Don't use variables a and b.

Test data:

The addresses will differ on different computers!

```
c)A:  &a = 0x7ffec1871798,
       &b = 0x7ffec1871790,
      &p1 = 0x7ffec1871788,
      &p2 = 0x7ffec1871780

d) V1: p1 = (nil),
       p2 = (nil)

g) V2: p1 = 0x7ffec1871798,
       p2 = 0x7ffec1871790

h) Enter any two real numbers: 12.3 13.4

i) V3: p1 = 0x7ffec1871798,
       p2 = 0x7ffec1871790,
        a = 12.300000,
        b = 13.400000

k) 12.30 + 13.40 = 25.70
   12.30 - 13.40 = -1.10
   12.30 * 13.40 = 164.82
   12.30 / 13.40 = 0.92
```

3. [5points] Conversion between coordinate systems.
In mathematics, the polar coordinate system is a two-dimensional coordinate system in which each point on a plane is determined by a distance from a reference point and an angle from a reference direction.

The **Cartesian** coordinates **x** and **y** can be converted to **polar** coordinates **r** and **ϕ** with $r \geq 0$ and φ in the interval $(-\pi, \pi]$ by:

$r = \sqrt{x^2 + y^2}$

$\varphi = atan2(y, x)$, where **atan2** returns the principal value of the arc tangent of y/x, expressed in **radians**.

The **polar** coordinates **r** and **ϕ** can be converted to the **Cartesian** coordinates **x** and **y** by using the trigonometric functions sine and cosine: $\begin{matrix} x = r\cos\varphi \\ y = r\sin\varphi \end{matrix}$, where **sin** returns the sine of an angle of **ϕ** radians, **cos** returns the cosine of an angle of **ϕ** radians.

One radian is equivalent to 180/PI degrees.

To convert degrees to radians or radians to degrees use the constant PI.

```
#define PI (4.0*atan(1))
```

In the main function:
a) Declare two variables of the type float x and y.
b) Use the scanf function to assign values to x and y.
c) Declare two variables of the type float r and t.
d) Call the CartToPolar function to which you pass the values of the x and y variables and the addresses of the r and t variables.
e) Print the values of the polar coordinates r and t.
f) Declare two variables of the type float x1 and y1.
g) Call the PolarToCart function to which you will pass the addresses of the variables x1 and y1 and the values of the variables r and t.
h) Print the Cartesian coordinates x1 and y1.

The CartToPolar and PolarToCart functions take **4** parameters: two float type variables, two float pointers. Functions do not use return statements and are of type void. When defining functions CartToPolar and PolarToCart, use the appropriate formulas to convert between the coordinate systems.

Test data:

```
Enter cartesian coordinate x: 1
Enter cartesian coordinate y: 1

Polar coordinate:
rho = 1.41
theta = 45.00 degree

Cartesian coordinate:
x = 1.00
y = 1.00

Enter cartesian coordinate x: 1
Enter cartesian coordinate y: 8
```

Polar coordinate:
rho = 8.06
theta = 82.87 degree

Cartesian coordinate:
x = 8.00
y = 1.00

4. [3points] Write a C function that swaps the values of two float variables. The type of swap function is void. The swap function has two parameters, pointers to float.

a) In the main function, declare two variables of the float type. Initialize the variables using the rand function. Divide the value returned by rand by RAND_MAX.

b) Print the values of float variables.

c) Call the swap function and pass the addresses of variables of type float.

d) Print the values of float variables.

Test data:
Before: x = 0.840188, y = 0.394383
After: x = 0.394383, y = 0.840188

5. [6points] Swap and arrays.

a)Here is the main function. Fill in the blanks.

```
#include  <stdio.h>
#include  <stdlib.h>
#define SIZE 10

int main (void){
   float x[SIZE], y[SIZE];

   fill_array(…, …);
   fill_array(…, …);

   print_two_arrays(…, …, …);

   swap_arrays(…, …, …);

   printf("\n");
   print_two_arrays(…, …, …);

   swap_half_array(…, …);

   printf("\n");
   print_one_array(…, …);
```

```
    return 0;
}
```

a) Write a function fill_array that takes two parameters: the address of the first element in the array and the address of the first element outside the array. The body of the function is two lines. The first line is a while loop. The second line is to assign each element of the array a value generated by the rand function. The elements of the array are accessed using the first parameter, which is a pointer.

b) The print_two_arrays function will print all the elements of both arrays. The body of the function is two lines. Use a for loop. The function has the following parameters, the address of the first element in the first array, the address of the first element in the second array, number of elements.

c) The function swap_arrays swaps the values between the arrays passed to it when it is called. The body of the function is two lines. Use a for loop. The function has the following parameters, the address of the first element in the first array, the address of the first element in the second array, number of elements. In the for loop, call the swap function from the previous exercise.

d) The function swap_half_array reverses the array passed to it when it is called. The body of the function is two lines. Use a while loop. The function swap_half_array takes two parameters: the address of the first element in the array and the address of the last element in the array. In the while loop, call the swap function from the previous exercise.

e) The print_one_array function prints all the elements of the array. The body of the function is two lines. Use a for loop. The function has the following parameters: the address of the first element in the array and the address of the first element outside the array.

Test data:

```
x[0] = 0.840188, y[0] = 0.477397
x[1] = 0.394383, y[1] = 0.628871
x[2] = 0.783099, y[2] = 0.364784
x[3] = 0.798440, y[3] = 0.513401
x[4] = 0.911647, y[4] = 0.952230
x[5] = 0.197551, y[5] = 0.916195
x[6] = 0.335223, y[6] = 0.635712
x[7] = 0.768230, y[7] = 0.717297
x[8] = 0.277775, y[8] = 0.141603
x[9] = 0.553970, y[9] = 0.606969

x[0] = 0.477397, y[0] = 0.840188
x[1] = 0.628871, y[1] = 0.394383
x[2] = 0.364784, y[2] = 0.783099
x[3] = 0.513401, y[3] = 0.798440
x[4] = 0.952230, y[4] = 0.911647
x[5] = 0.916195, y[5] = 0.197551
x[6] = 0.635712, y[6] = 0.335223
x[7] = 0.717297, y[7] = 0.768230
x[8] = 0.141603, y[8] = 0.277775
x[9] = 0.606969, y[9] = 0.553970

x[0] = 0.606969
```

x[1] = 0.141603
x[2] = 0.717297
x[3] = 0.635712
x[4] = 0.916195
x[5] = 0.952230
x[6] = 0.513401
x[7] = 0.364784
x[8] = 0.628871
x[9] = 0.477397

Next lab 10 – Dynamic memory allocation.