Week 5: Structured data and pointers

Please make sure to submit your solutions **by next Monday**.

In the beginning of each question, it is described what kind of answer that you are expected to submit. If *Text and code answer* is stated, then you need to submit BOTH some argumentation/description and some code; if just (*Text answer*) or (*Code answer*) then just some argumentation/description OR code. The final answer to the answers requiring text should be **one pdf document** with one answer for each text question (or text and code question). When you hand-in, add a link to your GitHub repository in the beginning of your pdf file. Make sure that you have committed your code solutions to that repository.

Note: the **Challenge** exercises are *optional*, the others mandatory (i.e. you **have** to hand them in).

Exercises

(1) (Text answer) (Old exam question) A function `area` calculates and returns the area of a rectangle as an integer. The input rectangle is given as four integer coordinates: `x1`, `x2`, `y1`, `y2`. Complete the function signature below.

```
int area_of_rectangle( Int x1, int x2, int y1, int y2 ) {
  return (x2 - x1) * (y2 - y1);
}
```

Link to repository:

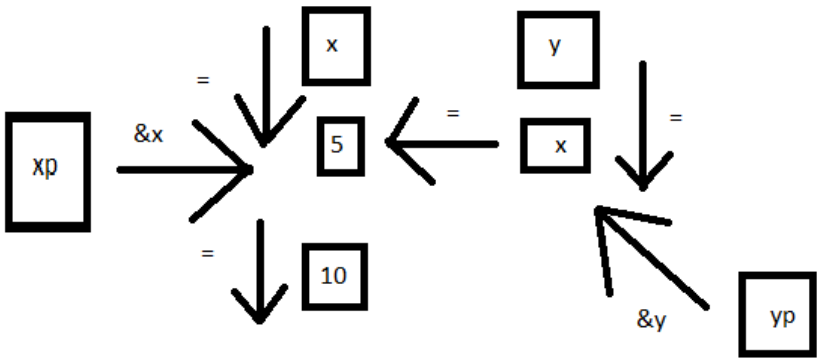
https://github.com/Aarhus-University-ECE/assignment-5-Teun0n

(2) (Text answer) (Old exam question) The function `increment` takes a pointer to an integer and adds `1` to the integer value to which it pints. The function does not return any value. Complete the function signature and function body below, so that the main function prints `6` when executed.

```
int increment( Int *v ) {
    *v = 6;
}

int main () {
  int v = 5;
  increment(&v);
  printf("%d", v);
  return 0;
}
```

(3) (Text answer) Consider the following code. At the end of the function, what are the values for $x, y, *xp, *yp$? Using pen and paper, draw a diagram (like in the lectures) to explain your answer. Your submission must include your diagram. The following diagram formats are allowed: PDF, JPG and PNG.

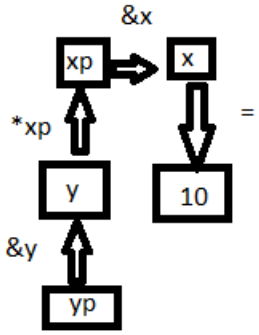
```
#include <stdio.h>

int main(void)
{
  int x;
  int y;

  int *xp;
  int *yp;

  x = 5;
  y = x;

  xp = &x;
  yp = &y;

  x = 10;

  /* What are values of: x,y,*xp,*yp */

  printf("x=%d, y=%d, *xp=%d, *yp=%d\n", x,y,*xp,*yp);

  return 0;
}
```

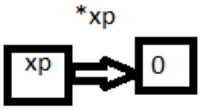


x=10 xp=10 y=5 yp=5

(4) (Text answer) Consider the following code. At the end of the function, what are the values for $x, y, *xp, *yp$? Using pen and paper, draw a diagram (like in the lectures) to explain your answer. Remember to include your diagram (in PDF, JPG or PNG format) in your submission.

```
#include <stdio.h>

int main(void)
{
  int x;
  int y;

  int *xp;
  int *yp;

  x = 5;



  xp = &x;

  x = 10;

  y = *xp;

  yp = &y;

  *xp = 0;

  /* What are values of: x,y,*xp,*yp */

  printf("x=%d, y=%d, *xp=%d, *yp=%d\n", x,y,*xp,*yp);

  return 0;
}
```

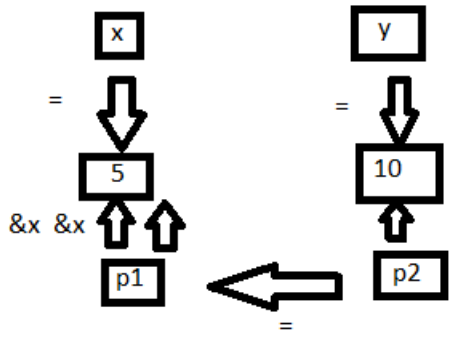




xp=0 (gets defined to 0 in last line.
x=0 (points to xp, therefor gets redefined with xp)
yp=10 (got value from xp, when x=10)
y=10 (point to y)

(5) (Text answer) Once again, consider the following code. At the end of the function, what are the values for $x, y, *xp, *yp$? Using pen and paper, draw a diagram (like in the lectures) to explain your answer. Remember to include your diagram (in PDF, JPG or PNG format) in your submission.

```
#include <stdio.h>

int main(void)
{
  int x;
  int y;

  int *p1;
  int *p2;

  x = 5;
  y = 10;

  p1 = &x;
  p2 = p1;

  *p2 = y;

  p1 = &x;

  /* What are values of: x,y,*xp,*yp */


  printf("x=%d, y=%d, *p1=%d, *p2=%d\n", x,y,*p1,*p2);

  return 0;
}
```



x=10 (because p1 points to x
p1=10 (because p2 points to p1
y=10 (because its defined as 10.
p2=10 (because it point to y)

(6) (Code answer) In the lecture we discussed how to represent a geometric *point* using a C struct. Let's now consider a geometric *circle*: a circle consists of three integers: x coordinate of the centre point, y coordinate of the centre point, and a *radius*.

(a) Write a C struct that represents a *circle* using a C struct with an integer representing the radius (named `r`) and a *point* (named `p` and using the struct shown in class).

(b) Create an array of five circles $c[5]$ such that circle c_i has centre point (i, i) and radius i

(c) Create a function *circleIsValid* that takes a pointer to a circle as input, and returns *true* if the radius of the circle is positive ($r > 0$) and *false* otherwise. The function should have the following signature: `int CircleIsValid(const circle *c)`

(d) Create a function *translate* that takes a pointer to a *circle* c and a pointer to a geometric *point* p (like in the lecture), and adds the coordinate values of p to the centre point coordinate values of c, i.e. it translates the circle by a vector represented by the point p. For example, if c is initially centred at $(5, 10)$ and we pass, as input, a point p with coordinate values $(1, -1)$ then the centre point of c becomes $(6, 9)$ after the *translate* function. The function should have the following signature: `void translate(circle *c, const point *p)`

(7) (Code answer) (PC-2.8.1)

(a) A sequence of $n > 0$ integers is called a *jolly jumper* if the absolute values of the differences between successive elements take on all possible values 1 through $n - 1$. For instance, 1 4 2 3 is a jolly jumper, because the absolute differences are 3, 2, and 1, respectively. As another example, 11 7 4 2 1 6 is a jolly jumper, because the absolute differences are 4, 3, 2, 1, 5 (the order of the differences does not matter). The definition implies that any sequence of a single integer is a jolly jumper. Write a function to determine whether a sequence is a jolly jumper. The function should have the following signature: `int isJollyJumber(const int seq[], int size)` (Hint: use a boolean array, e.g. `bool diffs_found[n]` to keep track of the differences found so far between consecutive numbers. So that `diffs_found[2]` being `true` implies that the absolute difference 2 has already been found.

(b) Write a test program that reads the size and sequence, and uses the function to print out if the sequence is a JollyJumper or not.:

Input A line of input contains an integer $n < 100$ followed by n integers representing the sequence.

Output For the line of input generate a line of output saying "Jolly" or "Not jolly".

Example

```
4
1 4 2 3
Jolly

5
1 4 2 -1 6
Not jolly
```