

Week 5 Programming Assignment

Steffen Petersen — au722120

October 3rd 2022

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.

```
1
2
3 _____ ( _____ ) {
4
5     return (x2 - x1) * (y2 - y1);
6 }
```

The function should have the following signature:

```
1  int area(int x1, int x2, int y1, int y2){
2
3      return (x2 - x1) * (y2 - y1);
4  }
```

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 points. The function does not return any value. Complete the function signature and function body below, so that the main function prints 6 when executed.

```
1
2
3 _____ ( _____ ) {
4
5
6 _____;
7 }
8
9 int main () {
10     int v = 5;
11     increment (&v);
12     printf("%d", v);
13     return 0;
14 }
```

The completed function is below:

```
1 void increment(int *in){
2     *in += 1;
3 }
```

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;

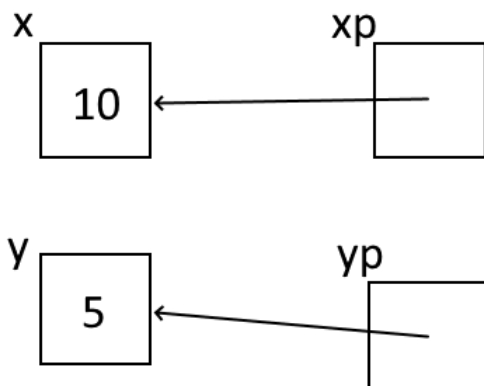
    /* What are values of: x,y,*xp,*yp */

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

    return 0;
}
```

In this code the xp pointer points to x , and the yp pointer points to y .

The value of x at the end is 10 and the value of y is 5, and thus the pointers point to 10 and 5 respectively. Below is the diagram over the variables, and this can be found as `task3Diagram.png` in the `/text/img` folder.



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;
    xp = &x;

    x = 10;

    y = *xp;

    yp = &y;

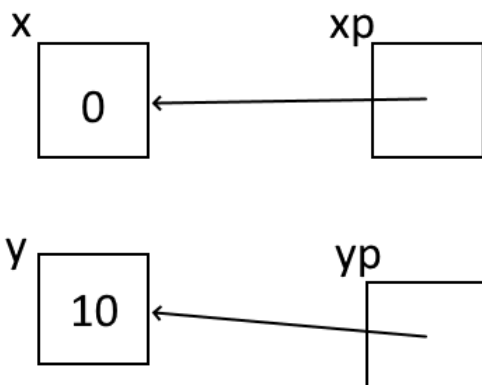
    *xp = 0;

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

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

    return 0;
}
```

Here the value of x ends at 0, and the value of y ends at 10, xp points to x and yp points to y . Below is the diagram for the code, also included as task4Diagram.png in /text/img



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;

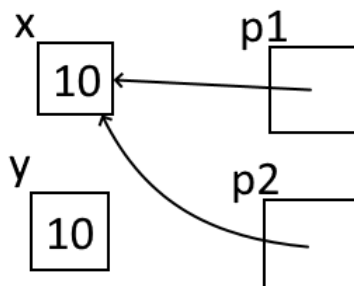
    *p2 = y;

    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;
}
```

The final value of x would be 10 and the final value of y would be 10, and both pointers point to x . Below is the diagram for the variables, it can be found as task5Diagram.png in /text/img



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)`

The edits/answers to this task can be found in the files here in the repository, this one related to `circle.h` and `circle.c`

(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 isJollyJumper(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