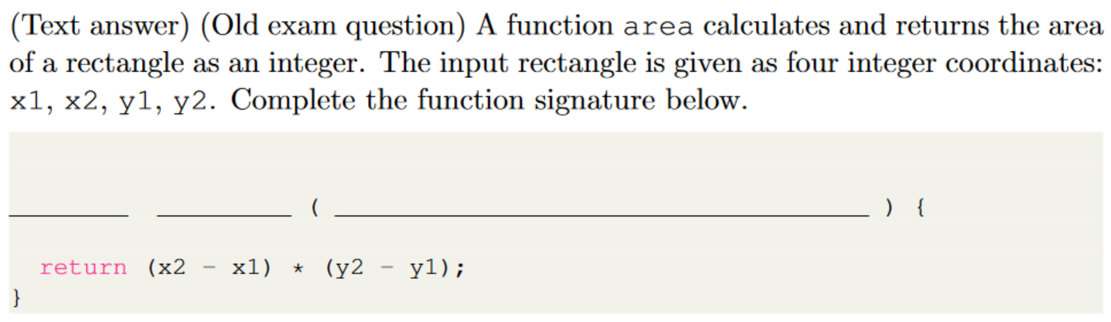
Week 5 Programming Assignment

Steffen Petersen — au722120

October 3rd 2022

1



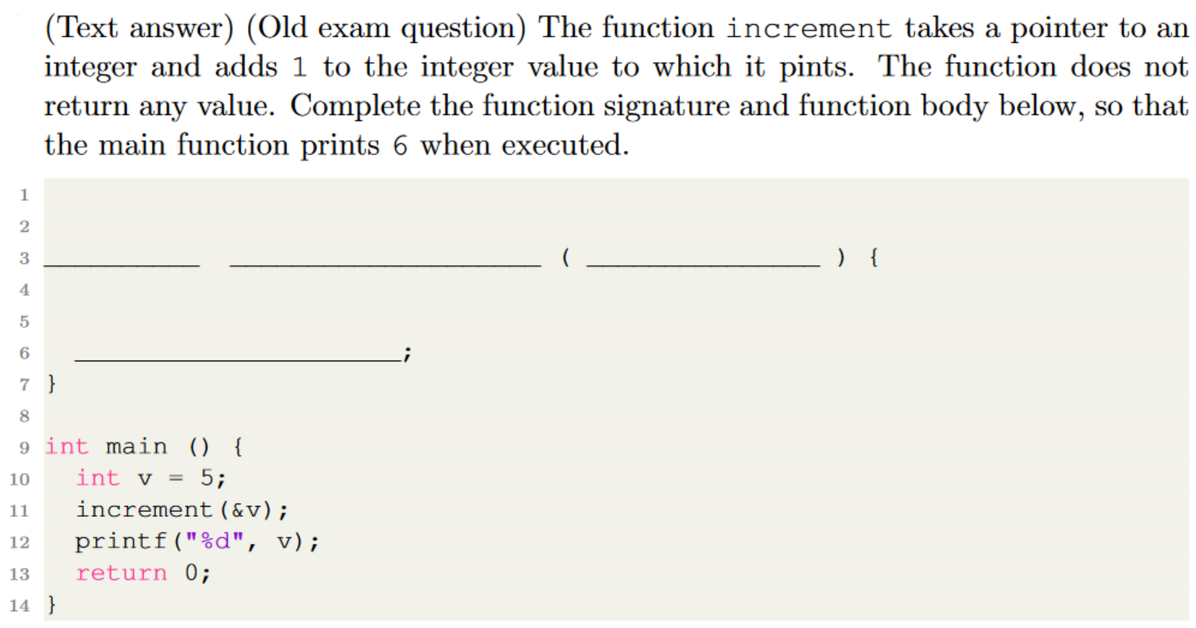
(Text answer) (Old exam question) A function `area` calculates and returns the area of a rectangle as an integer. The input rectangle is given as four integer coordinates: `x1`, `x2`, `y1`, `y2`. Complete the function signature below.

```


__________ ___________ ( ___________________________________________ ) {

  return (x2 - x1) * (y2 - y1);
}
```

Below is my completed function, the same can be found in the task1.c file in the attached ”Week 4 code.zip” folder, which includes a main function to test the get_min function.

```c
int get_min(int list[], int n) // A function that returns the lowest value in the given array list[] of length n.
{
  assert(n > 0); // Precondition: Can't have an array with the length of 0 or lower.
  int lowest = list[0]; // Initialize the first "lowest" value, as the first slot in the array.
  for(int i = 1; i < n; i++){ // Run through the array from 0 -> n-1
    if(list[i] < lowest) // Test whether the current slot in the array has a lower value than the current saved lowest value.
      lowest = list[i]; // If above is true, lowest is reassigned as the value of the current spot in the array.
  }
  return lowest;
}
```

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

Below is my completed function, the same can be found in the task2.c file in the attached ”Week 4 code.zip” folder, which includes a main function to test the reverse function.

```c
void reverse(int list[], int rev_array[], int n) // A function to print an array in reverse
    order through a new array holding the values in reverse.
{
  assert(n > 0); // Precondition: Can't have an array with the length of 0 or lower.
  for(int i = 0; i < n; i++){ // Run through the array from 0 -> n-1
    rev_array[i] = list[n-i-1]; // Assign the reverse array's value i, as the "opposite"
        value of list. So, rev_array's starting value becomes list's end value.
  }
  printf("The reverse of the function is as follows\n");
  for(int i = 0; i < n; i++) { // Print the reversed array
    printf("Slot %d : %d\n", i, rev_array[i]);
  }
}
```

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;
}
```

Below is my completed function, the same can be found in the task3.c file in the attached ”Week 4 code.zip” folder, which includes a main function to test the longest_seq function.

```
int longest_seq(int list[], int n)
{
    assert(n > 0); // precondition, size has to be greater than 0.
    int counter = 0; // Counter for current sequence of zeros
    int countmax = 0; // Record what our longest sequence has been so far.
    int index = -1; // Start index for position of zeros sequence at -1 (as the assignment
        shows.)

    for(int i = 0; i < n; i++){
        if(list[i] == 0){ // Check if current value of array is 0, if so, count 1 up
            counter++;
        }
        else counter = 0; // reset counter if we no longer are getting a sequence of 0's.
        if(counter > countmax){ // If current counter exceeds previous max, update the index
            to match where we started counting this sequence.
            index = i+1-counter;
            countmax = counter; // Also update countmax.
        }
    }
return index;
}
```

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;
}
```

Below is my completed function, the same can be found in the task4.c file in the attached "Week 4 code.zip" folder, which includes a main function to test the count_1_to_20 function.

```
void count_1_to_20(int a[100][150], int count[20])
{
  for(int rst = 0; rst < 20; rst++){
    count[rst] = 0; // Make sure the counter array is reset to 0.
  }
  // Precondition a only contains numbers from 1 to 20. (The programs runs fine regardless)
  for(int i = 0; i < 100; i++){ // Run through the first parameters 100 values
    for(int i2 = 0; i2 < 150; i2++){ // Run through the 2nd parameters 150 values, for each
        of the 100 outer values.
      count[a[i][i2]-1]++; //Counts +1 for each value from 1->20 in the entirity of a.
    }
  }
  for(int i3 = 0; i3 < 20; i3++){ // Output the results.
    printf("Nr. of %d's in the input: %d\n", i3+1, count[i3]);
  }
}
```

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;
}
```

For this function I've used the return type double instead of void, as we would like to return the average value of the contents of the array, which double seems suitable for.
Below is my completed function, the same can be found in the task5.c file in the attached "Week 4 code.zip" folder, which includes a main function to test the average function.

```
double average(double list[], int n) // return type double to retain the best possible
    accuracy.
{
    double average; // Variable to store the ongoing sum and at last the average.
    for(int i = 0; i < n; i++){
        average += list[i];
    }
    average /= n; // Divide the sum of the whole array's values by its length, to get the
        average.
    return average;
}
```

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
```