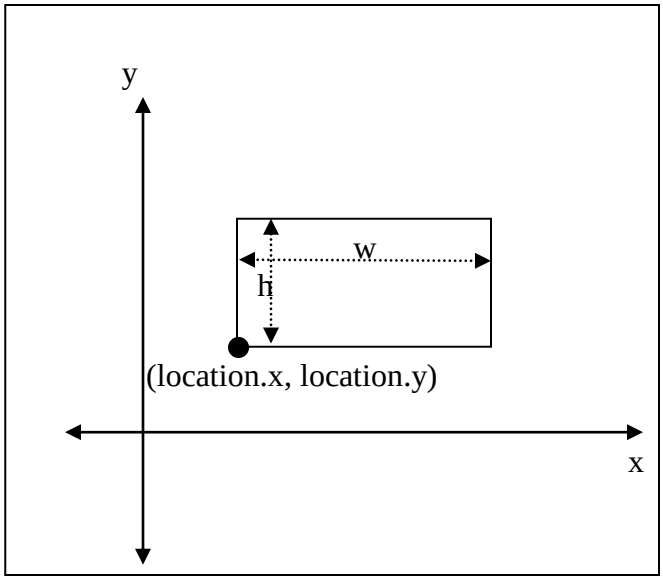
**CS 2123 Data Structures Recitation - Recitation Exercise 04**

## Struct, Arrays, Dynamic memory - functions

- Define a struct Rect as follows:

```
typedef struct Point
{
     double x;
     double y;
} PointT;

typedef struct Rect
{
   PointT location;
   char color;
   double w; //width;
   double h; //height;
} RectT;
```




- Write a program that declares `RectT a, b, *recs;` and asks user to enter `location.x, location.y, w, h` for both `a` and `b`.

- Write a function `int chk_overlap(RectT *r1, RectT *r2)` which can be called to check if `a` and `b` overlap or not. If so return 1, else return 0. Note you need to pass the addresses of a an b to that function!

- Dynamically create an array of 50 rectangles (an array of RectT type data) and save the address of that array to `recs`. Then randomly initialize `location.x, location.y, w, h` of each rectangle
  (e.g., `recs[i].location.x = rand()%20;` etc...)

- Count/print how many of these rectangles overlap with `a` alone, `b` alone, and both `a` and `b`. So you need to keep three counters.

After implementing the program

- Run it with valgrind to see memory usage info

  > valgrind rec04

- Free the allocated memory then run it again with valgrind

- Compile your program with –g option and use gdb and ddd

/* Don't forget to include comments about the problem, yourself and each major step in your program! */

**What to return:** !!!! NO LATE RECITATION ASSIGNMNET WILL BE ACCEPTED !!!

1. Create a directory called LASTNAME_Recitation04 and do all your work under that directory
2. First implement your program which can be named as rec04.c
3. Then compile and run it. Copy/paste the result in an output file, say out04.txt.
3. Finally zip your LASTNAME_Recitation04 directory as a single file LASNAME_Recitation04.zip and Go to BB Learn to submit it as **attachment** before the deadline.

---

You must submit your work using Blackboard Learn and respect the following rules:

1) All assignments must be submitted as .tar archive file unless it is a single pdf file.
2) Assignments must include all source code.
3) Assignments must include an output.txt file which demonstrates the final test output run by the student.
4) If your assignment does not run/compile, the output.txt file should include an explanation of what was accomplished, what the error message was that prevented the student from finishing the assignment and what the student BELIEVES to be the underlying cause of the error.

---