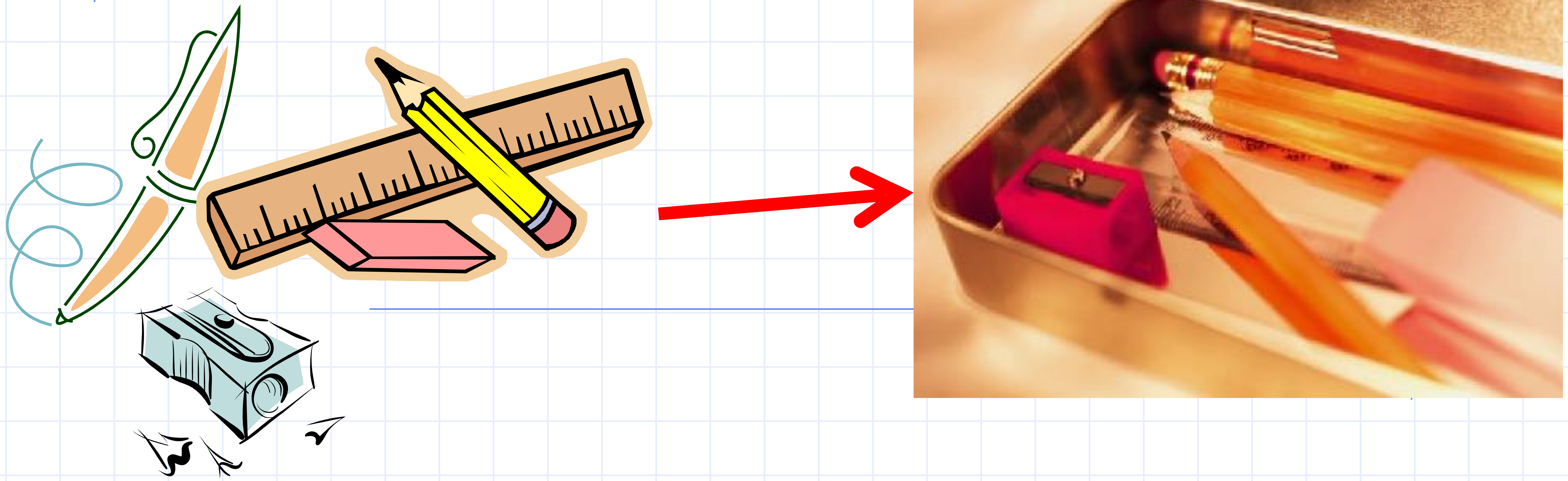
# ESC101: Introduction to Computing

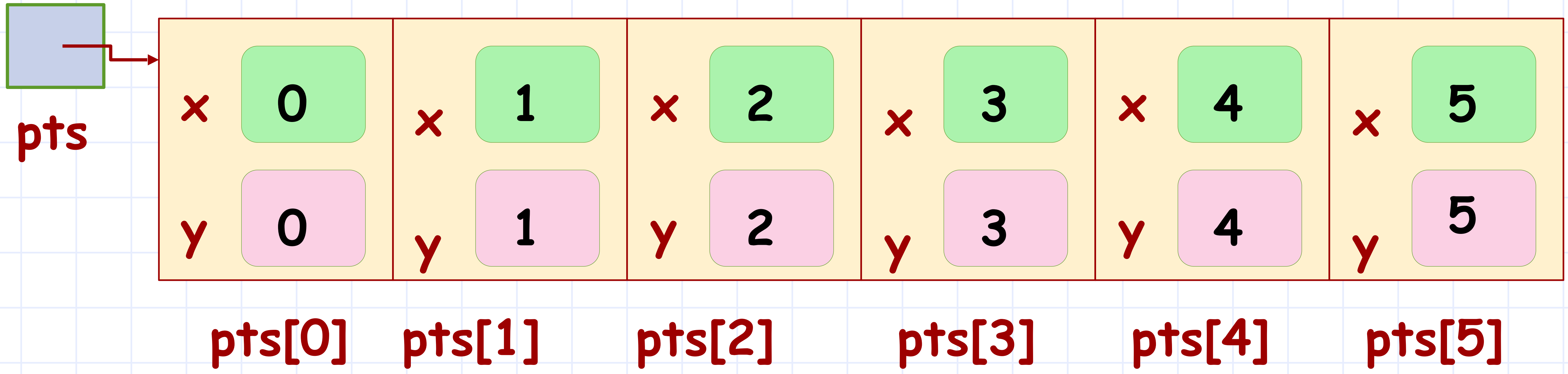
## Structures



## Structures

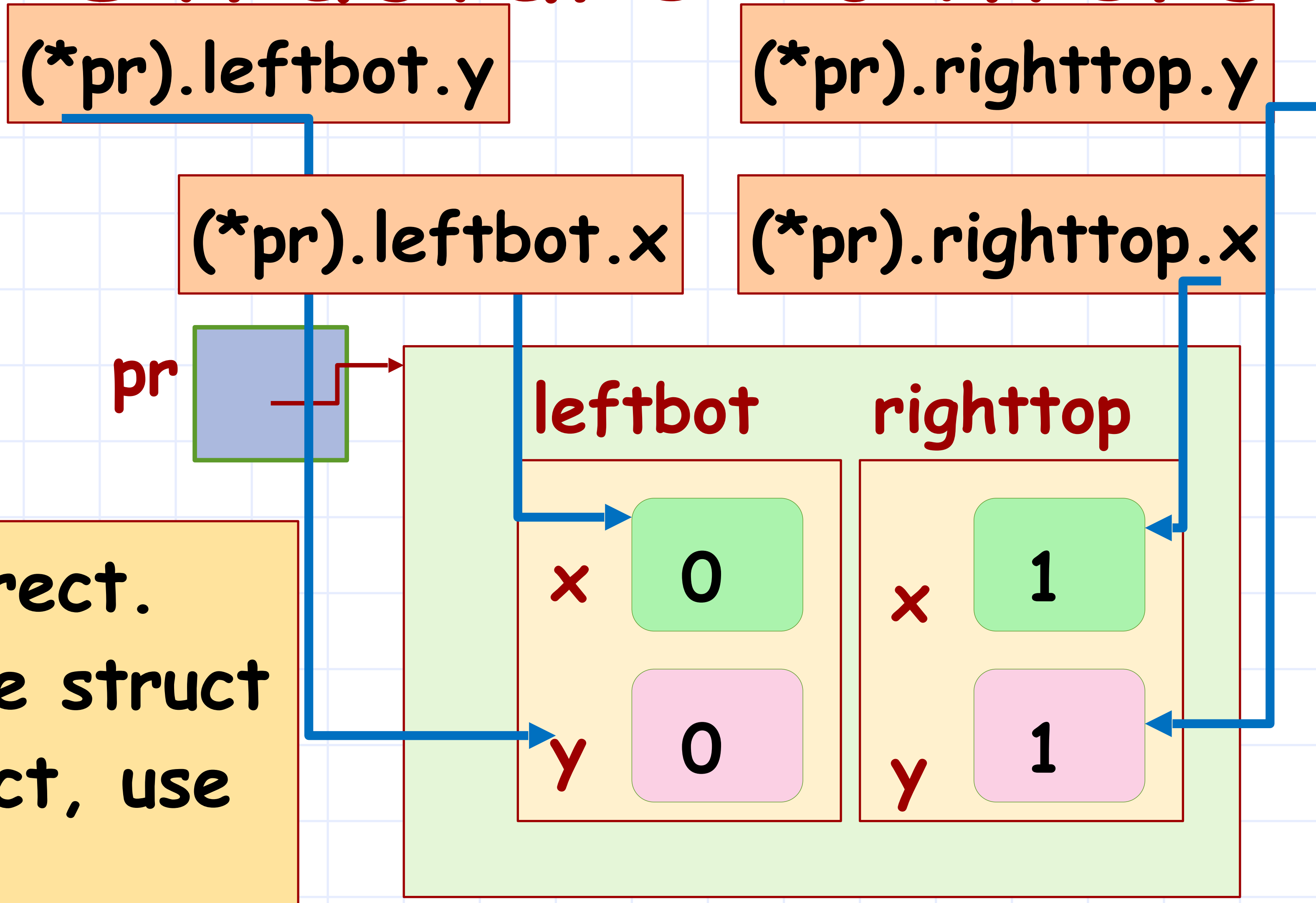
```
struct point {
        int x; int y;
};
struct point pts[6];
int i;
for (i=0; i < 6; i=i+1) {
        pts[i].x = i;
        pts[i].y = i;
}
```

State of memory after the code executes.



# Structure Pointers

```
struct point {
      int x; int y;};
struct rect {
   struct point leftbot;
   struct point righttop;
};
struct rect *pr;
```

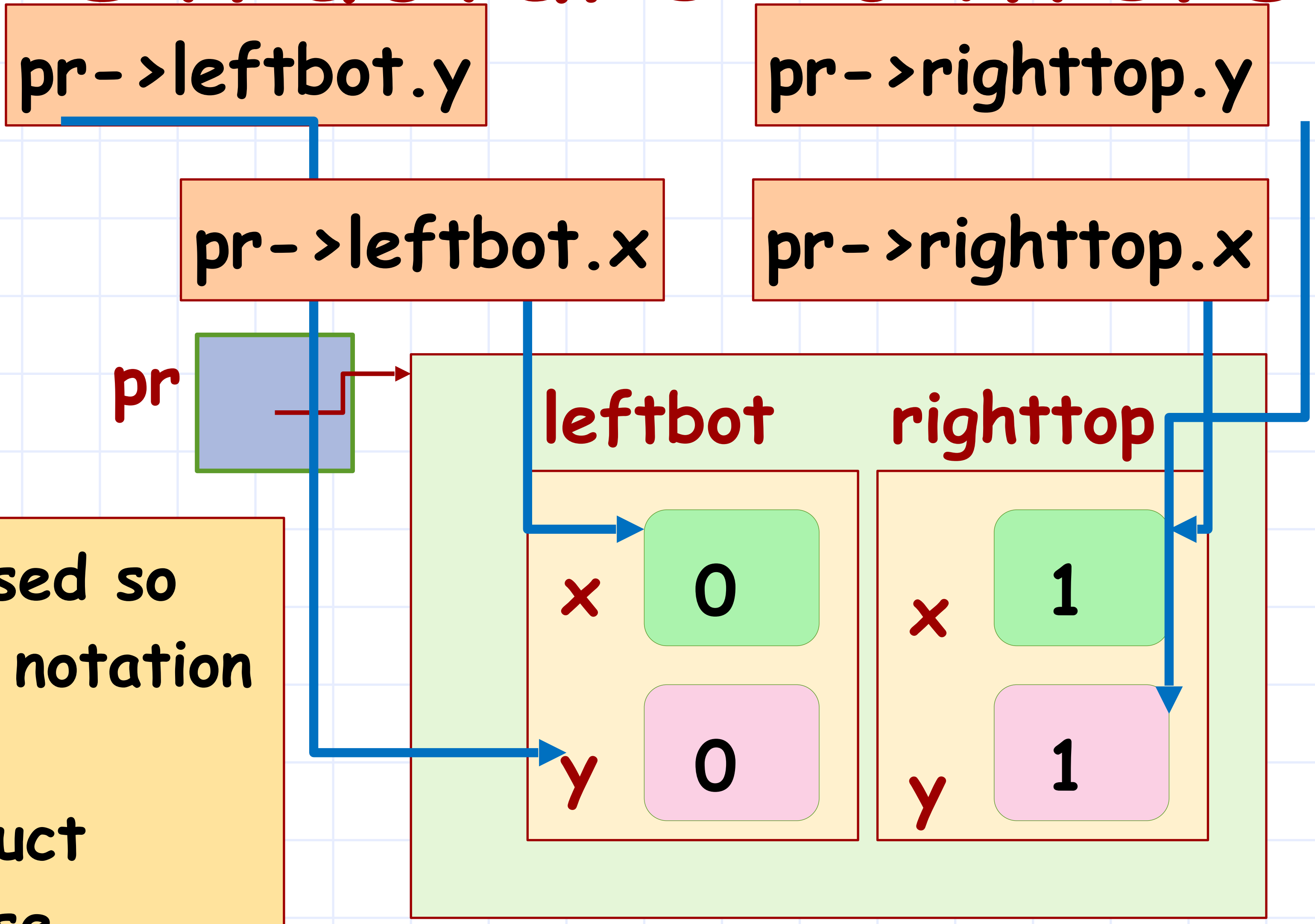


1. pr is pointer to struct rect.
2. To access a field of the struct pointed to by struct rect, use
   (*pr).leftbot
   (*pr).righttop
3. Bracketing (*pr) is essential here. * has lower precedence than .
4. To access the x field of leftbot, use (*pr).leftbot.x

Addressing fields
via the structure's pointer

```
struct point {
      int x; int y;};
struct rect {
   struct point leftbot;
   struct point righttop;
};
struct rect *pr;
```

# Structure Pointers



1. Pointers to structures are used so frequently that a shorthand notation (->) is provided.
2. To access a field of the struct pointed to by struct rect, use

   pr->leftbot
3. -> is one operator. To access the x field of leftbot, use

   pr->leftbot.x
3. -> and . have same precedence and are left-associative. Equivalent to (pr->leftbot).x

pr->leftbot is equivalent to (*pr).leftbot

Addressing fields via the structure's pointer (shorthand)

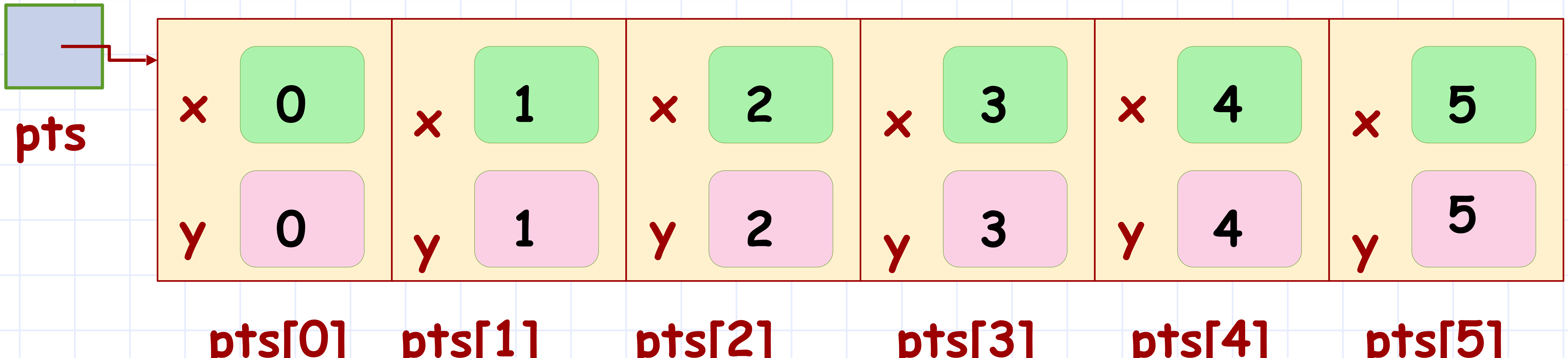
## Passing struct to functions

- When a struct is passed directly, it is passed by copying its contents
  - Any changes made inside the called function are lost on return
  - This is same as that for simple variables
- When a struct is passed using pointer,
  - Change made to the contents using pointer dereference are visible outside the called function

# Dynamic Allocation of struct

- Similar to other data types
- sizeof(…) works for struct-s too

```
struct point* pts;
int i;
pts = (struct point*) malloc(6 * sizeof(struct point));
for (i = 0; i < 6; i++)
    pts[i] = make_point(i, i);
```



## Exercise

- Write a program to read in two polynomials and add them.
- Input

  Len of Polynomial 1 *len1*

  *len1* terms consisting of *e* exponent and *c* coefficient as integers

  Len of Polynomial 2 *len2*

  *len2* terms consisting of *e* exponent and *c* coefficient as integers

```
#include <stdio.h>
#include <stdlib.h>
struct term
{
   int exp;
   int coeff;
};
void polyadd( struct term *p1, struct term *p2,int p1len,int p2len);

int main()
{
   struct term *p1, *p2;
   int p1len, p2len;

   scanf("%d",&p1len);
   p1 = (struct term*) malloc(sizeof(struct term)*p1len);
   for( int i=0; i<p1len; i++)
      scanf("%d %d",&(p1[i].exp), &(p1[i].coeff) );

   scanf("%d",&p2len);
   p2 = (struct term*) malloc(sizeof(struct term)*p2len);
   for( int i=0; i<p2len; i++)
      scanf("%d %d",&(p2[i].exp), &(p2[i].coeff));

   polyadd( p1, p2, p1len, p2len);
   free(p1); free(p2);
   return 0;
}
```

```
void polyadd( struct term *p1, struct term *p2,int p1len,int p2len)
{
  int i=0,j=0;
  while( i<p1len && j<p2len )
  {
    if(p1[i].exp == p2[j].exp) {
      printf("%d %d ",p1[i].exp, p1[i].coeff+p2[j].coeff );
      i++; j++;
    }
    else if( p1[i].exp < p2[j].exp) {
      printf("%d %d ",p1[i].exp, p1[i].coeff );
      i++;
    }
    else {
      printf("%d %d ",p2[j].exp, p2[j].coeff );
      j++;
    }
  }
  while( i<p1len ) {
    printf("%d %d ",p1[i].exp, p1[i].coeff );
    i++;
  }
  while( j<p2len ) {
    printf("%d %d ",p2[j].exp, p2[j].coeff );
    j++;
  }
}
```

# (Re)defining a Type - typedef

- When using a structure data type, it gets a bit cumbersome to write struct followed by the structure name every time.
- Alternatively, we can use the typedef command to set an alias (or shortcut).

```
struct point {
      int x; int y;
};
typedef struct point Point;
struct rect {
   Point leftbot;
   Point righttop;
};
```

- We can merge struct definition and typedef:

```
typedef struct point {
      int x; int y;
} Point;
```