Dynamic Memory Allocation

Tute 8

Structures: Revision

- vector_t to store points in a 2D plane was defined as

```
typedef struct {
double x;
double y;
} vector_t
```

- Then variables of type vector_t can be declared as

```
vector_t pt1, pt2;
```

Structures …

- `pt1` and `pt2` will have their own copies of `x` and `y` that can be referred as

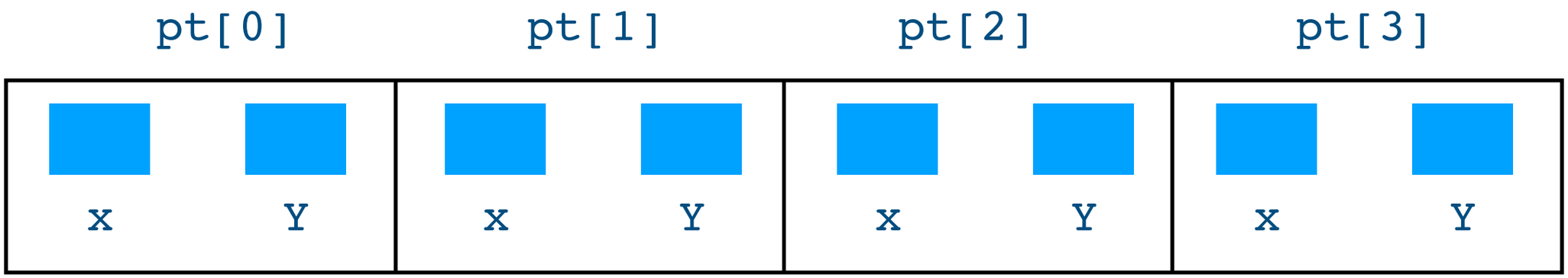
`pt1.x` and `pt2.x`

`pt1.y` and `pt2.y`

Structures …

- You can declare an array of type `vector_t` in the same way

```
#define MAX_SIZE 4
vector_t pt[MAX_SIZE]
```



Problems with static allocation

- We end up wasting a lot of memory
- You would normally set `MAX_SIZE` to a really large number but use only a few of them
- Variables created in local scope are destroyed when you return
- You can’t return an array from a function if you declare it inside the function.

Dynamic Memory Allocation

- `malloc()` function

```
void *malloc(size_t size)
```

- Allocates a *chunk* of memory of `size` bytes

- Our last array declaration can be modified as

```
vector_t *pt = malloc(MAX_SIZE*sizeof(vector_t))
```

Dynamic Memory …

- Still, this doesn't solve our problem. We are still allocating `sizeof(vector_t)*MAX_SIZE` bytes

- Solution: allocate small chunks of memory and resize it at run time as required.

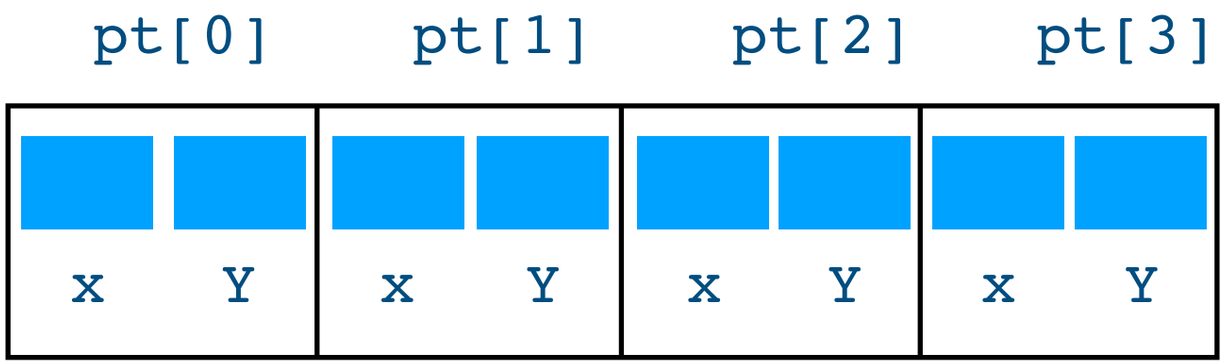
- `realloc()` function

```
void *realloc(void *ptr, size_t size)
```

- `*ptr` is a pointer to previously allocated memory (say using `malloc()` function

Dynamic Memory…

```
vector_t *pt = malloc(MAX_SIZE*sizeof(vector_t))
```




```
*pt = realloc(pt, 2*MAX_SIZE*sizeof(vector_t))
```