#include <stdio.h>

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

int main(void)
{
    /* Define a variable p of type point, and initialize all its members inline! */
    point p = {1,2};
    point q;
    q = p; // q.x = 1 and q.y=2
    q.x = 2;

    /* Demonstrate we have a copy and that they are now different. */
    if (p.x != q.x)
        printf("The members are not equal! %d != %d", p.x, q.x);
    return 0; }

struct mystruct {
    int a;
    char* b; }
   //note: could put st here instead
struct mystruct st;
char* pb = (char*)&st + offsetof(struct mystruct, b);

- `offsetof` tells you the offset of a variable within a structure (stddef.h)
- should set "pb" to be a pointer to member "b" within structure "mystruct".
#include<stdio.h>

typedef struct
{
    char *name;
    int number;
}TELEPHONE;

int main()
{
    TELEPHONE index;
    TELEPHONE *ptr_myindex;
    ptr_myindex = &index;
    ptr_myindex->name = "Jane Doe";
    ptr_myindex->number = 12345;
    printf("Name: %s\n", ptr_myindex->name);
    printf("Telephone number: %d\n", ptr_myindex->number);
    return 0; }

#include<stdio.h>

typedef struct
{
    int i;
    float PI;
    char A;
} RECORD;

int main()
{
    RECORD *ptr_one;
    ptr_one = (RECORD *) malloc (sizeof(RECORD));
    (*ptr_one).i = 10;
    (*ptr_one).PI = 3.14;
    (*ptr_one).A = 'a';
    printf("First value: %d\n", (*ptr_one).i);
    printf("Second value: %f\n", (*ptr_one).PI);
    printf("Third value: %c\n", (*ptr_one).A);
    free(ptr_one);
    return 0;
}
A struct declaration consists of a list of fields, each of which can have any type. The total storage required for a struct object is the sum of the storage requirements of all the fields, plus any internal padding.