Assume that we want `p` to point to a `rectangle` structure whose upper left corner is at (10, 25) and whose lower right corner is at (20, 15). Write a series of statements that allocate such a structure and initialize it as indicated.

W 5. Suppose that `f` and `p` are declared as follows:

```
struct {
  union {
    char a, b;
    int c;
  } d;
  int e[5];
} f, *p = &f;
```

Which of the following statements are legal?

(a) `p->b = ' ';`
(b) `p->e[3] = 10;`
(c) `(*p).d.a = '*';`
(d) `p->d->c = 20;`

6. Modify the `delete_from_list` function so that it uses only one pointer variable instead of two (`cur` and `prev`).

W 7. The following loop is supposed to delete all nodes from a linked list and release the memory that they occupy. Unfortunately, the loop is incorrect. Explain what's wrong with it and show how to fix the bug.

```
for (p = first; p != NULL; p = p->next)
  free(p);
```

W 8. Section 15.2 describes a file, `stack.c`, that provides functions for storing integers in a stack. In that section, the stack was implemented as an array. Modify `stack.c` so that a stack is now stored as a linked list. Replace the `contents` and `top` variables by a single variable that points to the first node in the list (the "top" of the stack). Write the functions in `stack.c` so that they use this pointer. Remove the `is_full` function, instead having `push` return either `true` (if memory was available to create a node) or `false` (if not).

9. True or false: If `x` is a structure and `a` is a member of that structure, then `(&x)->a` is the same as `x.a`. Justify your answer.

10. Modify the `print_part` function of Section 16.2 so that its parameter is a *pointer* to a `part` structure. Use the `->` operator in your answer.

11. Write the following function:

```
int count_occurrences(struct node *list, int n);
```

The `list` parameter points to a linked list; the function should return the number of times that `n` appears in this list. Assume that the `node` structure is the one defined in Section 17.5.

12. Write the following function:

```
struct node *find_last(struct node *list, int n);
```

The `list` parameter points to a linked list. The function should return a pointer to the *last* node that contains `n`; it should return `NULL` if `n` doesn't appear in the list. Assume that the `node` structure is the one defined in Section 17.5.

13. The following function is supposed to insert a new node into its proper place in an ordered list, returning a pointer to the first node in the modified list. Unfortunately, the function