Self-Referential Data Structures

- Structures can hold ***pointers*** to ***instances*** of ***themselves***

```
struct tree {
    char *word;
    int count;
    struct tree *left, *right;
};
```

- Structures ***cannot*** contain ***instances*** of ***themselves***:

```
struct tree {
    char *word
    int count;
    struct tree left, right;
};
```

what is **`sizeof (struct tree)`**?

Dynamic Data Structures

- C library routines **`malloc`** and **`free`** allocate and deallocate memory

`extern void *malloc(unsigned nbytes);`

allocates **`nbytes`** of memory and returns a pointer to the 1st byte

`extern void free(void *p)`

deallocates the memory pointed to by **`p`**, which ***must*** come from **`malloc`**

- To create a new **`treenode`**:

```
typedef struct tree *Tree;
Tree talloc(void) {
    return malloc(sizeof (struct tree));
}
```

- Better yet, provide arguments to ***initialize*** the new **`tree`**:

```
Tree talloc(char *word, int count, Tree left, Tree right) {
    Tree t = malloc(sizeof *t);
    t->word = word; t->count = count;
    t->left = left; t->right = right;
    return t;
}
```

Deallocating Memory

- Delallocate a previously created **`tree`**:

```
void tfree(Tree t) {
     free(t);
}
```

- Other allocation functions:

`extern void *calloc(unsigned n, unsigned nbytes)`

allocates ***and clears*** memory for **`n`** copies of **`nbytes`**, e.g. an array of structures

`extern void *realloc(void *p, unsigned size)`

expands/shrinks the memory pointed by **`p`** to occupy **`nbytes;`** may ***relocate***

- All allocation functions return **`NULL`** if there is ***no memory*** available

Example: Binary Trees

- Function **`insert(Tree *p, char *word)`**

adds **`word`** to the tree rooted at **`p`** if **`word`** isn't already in the tree otherwise, it increments the **`count`** associated with **`word`**

```
void insert(Tree *p, char *word) {
    Tree q = *p;

    if (q) {
        int cond = strcmp(word, q->word);
        if (cond < 0)
            insert(&q->left, word);
        else if (cond > 0)
            insert(&q->right, word);
        else
            q->count++;
    } else
        *p = talloc(strsave(word), 1, NULL, NULL);
}
```

- **`char strsave(char *s)`** makes a copy of string **`s`** and returns it

```
char *strsave(char *s) {
    char *new = malloc(strlen(s) + 1);

    assert(new);
    return strcpy(new, s);
}
```