Structures

- C structures are similar to records in Pascal, they allow us to collect together several pieces of related data into one block. The individual pieces of data are called structure elements or structure members
- A structure prototype is formed as follows:

```
struct StructureName {
    member declaration;
    member declaration;
        :
        :
        :
    member declaration;
};
```

- This declaration doesn't allocate any memory, it just provides a <u>template</u> for the structure. It specifies the quantities that are being grouped together
- The individual structure elements can be of any C type, including other structures
- For example, we could have the following for the declaration of a name structure

```
    struct Person {
        char*  first;
        char*  last;
    };
```


- This structure has two elements, the pointers first and last, which are used to access an array of characters. Other variables can have the same names as structure elements, and the same element name can be used in different structure declarations. You will not find this to be a problem, because the compiler recognizes all potential ambiguities.
- There are several ways that we can declare a variable that has a structure type
- One way is to do the following:

  struct StructureName VariableName;

- So with our example name struct we could do the following:

```
struct Person fred, andrew;
```

This declaration creates two variables (called fred and andrew) and allocates space for them. Each variable (structure) has two elements, each element points to an array of characters that may be used to hold the first and last names, respectively.

We use the **.** (dot) operator to form individual instances of a structure element. For example:

```
fred.first = "Fred";
fred.last  = "Flintstone";
```

equally one might write

```
andrew.last = "Choi";
```

- Another way of declaring a structure variable is:

```
struct StructureName {
    member declarations
} VariableName;
```

- In this approach we combine the structure definition with the variable declaration. The structure name is optional in this case, but it is always a good idea to include it
- A structure variable can also be initialized in a way that is reminiscent of initializing an array, as follows:

```
struct StructureName
    VariableName = { element_values } ;
```

- So for our example we could have:

```
struct Name andrew = { "Andrew", "Choi" };
```

- The element values are assigned in the same order as the element declarations
- The **.** (dot) operator is used to extract the individual elements from a structure variable. In the case of our Person structure we have::

```
char*  FirstName;
char*  LastName;
    FirstName  =  andrew.first
    LastName  = andrew.last
```


- In general the syntax is

  VariableName.ElementName

- A slightly different syntax is used for pointers to structures, for example

  ```
  struct Person* ptr;
  ```

- The variable ptr is now a pointer to a structure of type Person. From our previous knowledge about pointers we can use the following to get the value of the first element of the structure that ptr points to

  ```
  (*ptr).first
  ```
  or
  ```
  ptr->first
  ```

- Thus (*ptr). is equivalent to ptr->
  The -> operator takes a pointer to a struct, follows the pointer to the structure value and then extracts the field - this is a shorthand, but it makes sense
- We can have arrays of structures, just as we can have arrays of any other data type. This is often quite convenient, and is done in the following way:

  ```
  struct StructureName VariableName [ size ];
  ```

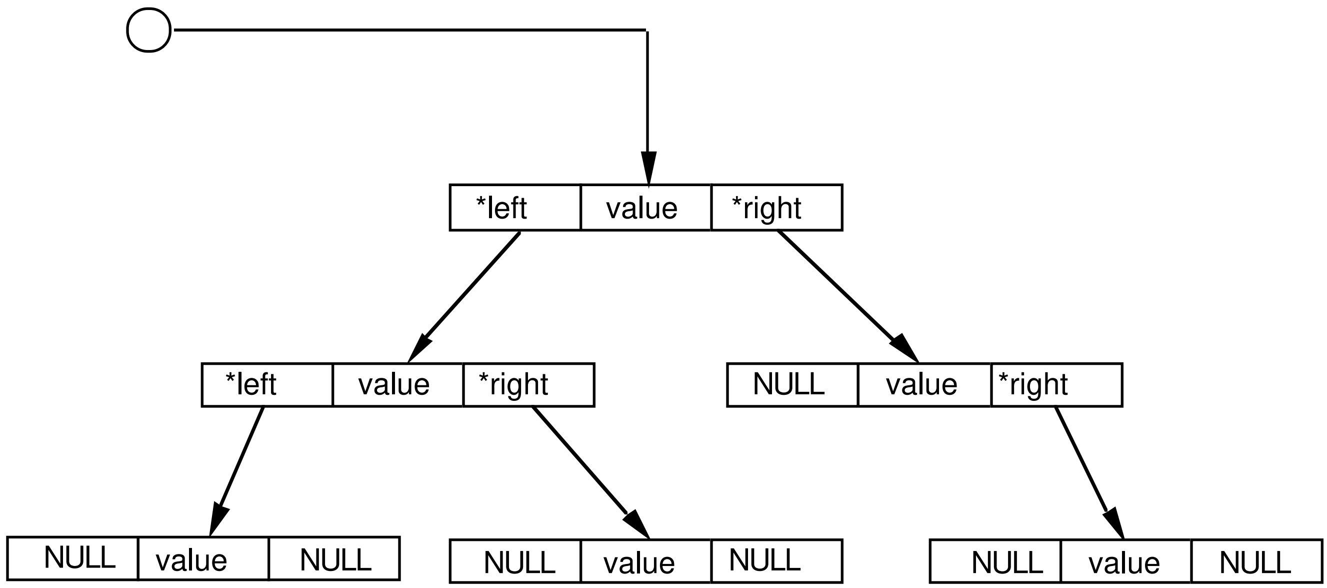

- In the case of our Person structure, we could have an initialized array of names constructed in the following way:

```
struct Person People[] = {
    { "Andrew", "Choi" },
    { "Fred", "Flintstone" },
          :
          :
    { 0, 0 }
};
```

- Here we use an explicit 0 value at the end of the array to indicate the end. There are other ways of doing this, but this is the safest
- We can include pointers to a structure within the declaration of the structure, we use this technique to build linked lists and binary trees
- We can use the following structure declaration for a node in a binary tree

```
struct Node {
    struct Node* left;
    int value;
    struct Node* right;
};
```

- Note that left and right must be pointers to structures, they cannot be structure variables, otherwise we will have a structure that includes two copies of itself



- This use of pointers is perhaps clearest in a linked list:

```
struct ListNode {
    float value;
    struct ListNode* next;
};
```