Structure, Union and Type Definition

By: **Annamalai A**

A structure and union do similar things in `C` and are very useful when grouping data. Let's see more about them.

Structure

A structure is used to group data of different data types under a single name, much like a dictionary in `Python` and vector in `C++`. It uses another data type called `struct` and the syntax looks something like this

```
struct struct_name {
   type data1 = something;
   type data2 = something;
}
```

Why is it useful? We are actually defining three variables in place of two. It is useful because we **group** similar data, or data that contributes to some bigger entity. It can also be said as the *properties* of the *object* we are creating.

Consider an example where we store students' data and many more variables for processing. It will be difficult to read and maintain the code, and sometimes we want to use the same variable name again for some other purpose in the same scope. **A structure fixes all these problems**.

In our example, let's say we are storing students' name, class, roll number and marks.

```
struct student {
   char name[20];
   int class;
   int roll;
   int marks;
}
```

The elements of a structure are stored in **contiguous** memory, meaning each of the variable stored inside occupy consecutive memory spaces. Also, they help in defining custom made data types that makes it easier to classify our data in a long program.

Accessing Elements of `struct`

To access elements in a structure, simply call the variable of a specific structure we want access, followed by the variable stored inside it by using a dot `.`. So the syntax is: `structure_variable.variable`.

```
struct student s1;
s1.name; // Gets the name of the student
s1.marks; // Gets the marks of the student
```

Structure Pointers

Yes, pointers can be used on structures as well. Similar to regular pointers, a structure's pointer follows the syntax: `struct struct_name *pointer`. This variable acts as a pointer specifically to only this `struct` and not any other structure. In our example:

```
struct student *ptr; // ptr acts as pointer to structure 'student'
struct student s1;
ptr = &s1;
```

To access elements, you use the pointer variable in place of the structure name, and it is better to access it using the arrow operator `->` rather than the dot operator.

```
ptr->name; // Pointer gets the name attribute of 'student'
ptr->marks;
```

Union

A union plays a similar role as structure, but differs in how memory works.

In case of the union, multiple data attributes can be written, but *all* members share the same memory location. So when we access one element of the union, the others automatically become undefined. T

The members do not store any value initally, and a value has to assigned each time we are working with it. Moreover, definining another member will *overwrite* the previos member's data and hence it becomes inaccessible.

The union uses the `union` data type and the syntax is similar to structure:

```
union Data {
   int i;
   float f;
   char str[20];
};
```

To access elements of a union, make sure to assign the value of the member we want first and use it, otherwise, the compiler throws an error.

```
union Data d;

d.i = 10;
printf("d.i = %d\n", d.i);

d.f = 3.14;
printf("d.f = %.2f\n", d.f);
```

Structure vs Union

Feature	**struct**	**union**
Memory	Sum of all members	Size of largest member
Valid data	All members valid at once	Only one valid at a time
Use-case	Grouping data	Memory saving

Type Definition

Did you know that you can create custom type names for the ones that already exist. Till now, we saw how to create a custom data type with what has been given such as `struct` or `union`. Now we are going to create *nicknames* to some datatypes.

The `typedef` keyword is used to assign an alias name for existing data types. Once defined, you can use it everywhere in the program instead of the original name, and the compiler knows what data type it is actually handling.

The syntax for `typedef` is: `typedef original_type_name new_type_name`

```
typedef unsigned int UI; // unsigned int can be written as UI everywhere in t
he program

typedef struct {
    char name[20];
    int marks;
    int roll;
    int class;
} Student; // struct Student can be written as Student
```

Note that while using `typedef` on `struct`, we do not define the name given for the structure, rather only the elements, and then followed by the name. This is because the name given to a structure (and union) are just like alias names to them.

But when calling the structure, we write `struct name` everywhere, while now we just write `name`.

So `typedef` can be really useful for datatypes with long names or the ones that are used most frequently.