C Structures & Unions

C Structures

We have already learnt in the previous tutorials that variables store one piece of information and arrays of certain data types store the information of the same data type. Variables and arrays can handle a great variety of situations. But quite often, we have to deal with the collection of dissimilar data types. For dealing with cases where there is a requirement to store dissimilar data types, C provides a data type called 'structure'. It is a way to group together information belonging to different data types and combine them into one structure.

What are Structures in C?

Structures are usually used when we wish to store data of different data types together. For example, if we want to store information about a book, there could be a number of parameters defining a book.

Books have a title, an author name, the number of pages, and a price. All of the book attributes belong to different data types. The titles and author names must be strings, but the prices and number of pages must be numerical.

One way to store the data is to construct individual arrays, and another method is to use a structure variable. It is to keep in mind that structure elements are always stored in contiguous memory locations.

Creating a struct element

Basic syntax for declaring a struct is:

```
struct structure_name

{

   //structure_elements

} structure_variable;
```

Here’s one example of how a struct is defined and used in main as a user-defined data type.

```
#include <stdio.h>


struct Books

{
   char title[20];

   char author[100];

   float price;

   int pages;

};


int main()

{

   struct Books book1;

   return 0;

}
```

Accessing struct elements

We use the subscript operator, '[' fed with the index number to access individual elements of an array. But in the case of structures, to access any element, we use the dot operator (.). This dot operator is coded between the structure variable name and the structure member that we wish to access.

Before the dot operator, there must always be an already defined structure variable and after the dot operator, there must always be a valid structure element.

Here's one example demonstrating how we access struct elements.

```
#include <stdio.h>


struct Books

{

   char title[20];

   char author[100];

   float price;

   int pages;

};
```

```
int main()

{

   struct Books book1 = {"C Programming", "ABC", 123.99, 300};

   printf("%s\n", book1.title);

   printf("%s\n", book1.author);

   printf("%f\n", book1.price);

   printf("%d\n", book1.pages);

   return 0;

}
```

Output:

```
C Programming

ABC

123.989998

300
```

Additional Features of Structs

1. We can assign the values of a structure variable to another structure variable of the same type using the assignment operator.
2. Structure can be nested within another structure which means structures can have their members as structures themselves.
3. We can pass the structure variable to a function. We can pass the individual structure elements or the entire structure variable into the function as an argument. And functions can also return a structure variable.
4. We can have a pointer pointing to a struct just like the way we can have a pointer pointing to an int, or a pointer pointing to a char variable.

Where are Structs useful?

It is possible to use structures for many different purposes, including:

1. Structures are used to store a large amount of data of varying data types.
2. They are used to send data to the printer.
3. For placing the cursor at an appropriate position on the screen, we can use structure.

4. It can be used in drawing and floppy formatting.
5. We use structures in finding out the list of equipment attached to the computer.

C Unions

Just like Structures, the union is a user-defined data type. All the members in unions share the same memory location. The union is a data type that allows different data belonging to different data types to be stored in the same memory locations. One of the advantages of using a union is that it provides an efficient way of reusing the memory location, as only one of its members can be accessed at a time. A union is used in the same way we declare and use a structure. The difference lies just in the way memory is allocated to their members.

Defining a Union

We use the union keyword to define the union.

The syntax for defining a union is:

```
union union_name{    // union_elements} structure_variable;
```

Here's one example of how a union is defined and used in main as a user-defined data type.

```
#include <stdio.h>


union Books

{

   char title[20];

   char author[100];

   float price;

   int pages;

};


int main()

{

   union Books book1;

   return 0;
}
```

Initialising and accessing union elements

Different from how we used to initialise a struct in one single statement, union elements are initialised one at a time.

And also, one can access only one union element at a time. Altering one union element disturbs the value stored in other union elements.

```
#include <stdio.h>
#include <string.h>

union Books
{
   char title[20];
   char author[100];
   float price;
   int pages;
};

int main()
{
   union Books book1;
   strcpy(book1.title, "C Programming");
   printf("%s\n", book1.title);

   strcpy(book1.author, "ABC");
   printf("%s\n", book1.author);

   book1.price = 123.99;
   printf("%f\n", book1.price);
```

```
  book1.pages = 300;
  printf("%d\n", book1.pages);


  return 0;
}
```

Output:

```
C Programming

ABC

123.989998

300
```

How are Structs and Unions similar?

1. Structures and unions, both are user-defined data types used to store data of different types.
2. The members of structures and unions can be objects of any type, including even other structures and unions or arrays.
3. A union or a structure can be passed by value to functions and can be returned by value by functions.
4. The . operator is used for accessing both union and structure members.

How are Structs and Unions different?

1. The keyword union is used to define a union and a keyword struct is used to define the structure.
2. Within a structure, each member is allocated a unique storage area of location whereas memory allocated to a union is shared by individual members of the union.
3. Individual members can be accessed at a time in structures whereas only one member can be accessed at a time in unions.
4. Changing the value of one of the members of a structure will not affect the values of the other members of the structure, whereas changing the value of one of the members of a union will affect the values of other members in a union.
5. Several members of a structure can be initialised at once, whereas only one member can be initialised in the union.

C Typedef

In C programming, a typedef declaration is used to create shorter, more meaningful, and convenient names for keywords already defined by C like int, float, and char.

What is a typedef in C?

A typedef is a keyword that is used to assign alternative names to existing datatypes. We use typedef with user-defined datatypes when the names of the datatypes become slightly complicated to use in programs. Typedefs can be used to:

- Make a more complex definition reusable by abbreviating it to something less complex.
- Provide more clarity to the code.
- Make it easier to change the underlying data types that we use.
- Make the code more clear and easier to modify.

Following is the syntax for using typedef:

```
typedef <previous_name> <alias_name>
```

For example, we would often want to create a variable of type unsigned long. But, then it becomes a complex task if the need to declare an unsigned long comes for multiple variables. To overcome this problem, we use a typedef keyword. Here is how we use it:

```
#include <stdio.h>

typedef unsigned long ul;


int main()

{

  ul a;

}
```

Applications of Typedef

There are various applications of typedef. Listed below are a few applications of typedef:

- The typedef can be used with arrays, primarily multi-dimensional arrays. It increases the readability of the program.
- Typedefs can also be implemented for defining a user-defined data type like structs or unions with a specific name and type.

Here’s how we use typedefs for defining a struct in C:

```
typedef struct

{

   structure element1;

   structure element2;

   structure element3;

} name_of_type;
```

typedef can be used for providing a pseudo name to pointer variables as well.

```
typedef int *ptr;
```

Advantages of Typedef

- Typedef, as mentioned, increases the readability of the code. If we are using structures, function pointers, or long keywords repeatedly in our code, then using typedefs increases the readability of the code.
- With the help of typedef, we can use the same name for different applications even in different scopes.
- In the case of structures, if we use the typedef, then we do not require to write the struct keyword at the time of variable declaration.
- Typedef increases the portability of the code.