Struct and typedef and how to vinculate them to a pointer

Owner	A Angel David Sanchez
Tags	Embedded software

Struct

- A **struct** in the C programming language (and many derivatives) is a composite data type (or record) declaration that defines a physically grouped list of variables under one name in a block of memory, allowing the different variables to be accessed via a single pointer or by the struct declared name which returns the same address
- Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a **member** of the structure.
- Unlike an array, a structure can contain many different data types (int, float, char, etc.).
- Because the contents of a struct are stored in contiguous memory, the sizeof operator must be used to get the number of bytes needed to store a particular type of struct, just as it can be used for primitives.

You can create a structure by using the `struct` keyword and declare each of its members inside curly braces:

```
struct MyStructure {   // Structure declaration
     int myNum;           // Member (int variable)
    char myLetter;       // Member (char variable)
}; // End the structure with a semicolon
```

To access the structure, you must create a variable of it.

Use the `struct` keyword inside the `main()` method, followed by the name of the structure and then the name of the structure variable:

Create a struct variable with the name "s1":

```
struct myStructure {
    int myNum;
    char myLetter;
};

int main() {
    struct myStructure s1;
    return 0;
}
```

Access Structure Members

To access members of a structure, use the dot syntax (`.`):

Example

```
// Create a structure called myStructure
struct myStructure {
    int myNum;
    char myLetter;
};
    int main() {
// Create a structure variable of myStructure called s1
    struct myStructure s1;
// Assign values to members of s1
    s1.myNum = 13;
    s1.myLetter = 'B';
    // Print values
    printf("My number: %d\n", s1.myNum);
    printf("My letter: %c\n", s1.myLetter);
    return 0;
}
```

Typedef

- **typedef** is a reserved keyword in the programming languages C, C++, and Objective-C. It is used to create an additional name (*alias*) for another data type, but does not create a new type.
- The **typedef** is a keyword that is used to provide existing data types with a new name. The C typedef keyword is used to redefine the name of already existing data types.
- When names of datatypes become difficult to use in programs, typedef is used with user-defined datatypes, which behave similarly to defining an alias for commands.
- As such, it is often used to simplify the syntax of declaring complex data structures consisting of struct and union types, although it is also commonly used to provide specific descriptive type names for integer data types of varying sizes.

C typedef Syntax

```
typedef existing_name alias_name;
```

After this declaration, we can use the *alias_name* as if it were the real *existing_name* in out C program.

Example of typedef in C

```
typedef long long ll;
```

Below is the C program to illustrate how to use typedef.

C

```
// C program to implement typedef
#include <stdio.h>
// defining an alias using typedef
typedef long long ll;
// Driver code
int main()
{
    // using typedef name to declare variable
```

```
    ll var = 20;
    printf ( "%ld" , var);
    return 0;
}
```

Output

```
20
```

Use of typedef in C

Following are some common uses of the typedef in C programming:

- The typedef keyword gives a meaningful name to the existing data type which helps other users to understand the program more easily.
- It can be used with structures to increase code readability and we don’t have to type struct repeatedly.
- The typedef keyword can also be used with pointers to declare multiple pointers in a single statement.
- It can be used with arrays to declare any number of variables.

1. typedef struct

typedef can also be used with structures in the C programming language. A new data type can be created and used to define the structure variable.

Example 1: Using typedef to define a name for a structure

C

```
// C program to implement
// typedef with structures
#include <stdio.h>
#include <string.h>
// using typedef to define an alias for structure
typedef struct students {
    char name[50];
```

```
    char branch[50];
    int ID_no;
} stu;
// Driver code
int main()
{
    stu st;
    strcpy (st.name, "Kamlesh Joshi" );
    strcpy (st.branch, "Computer Science And Engineering" );
    st.ID_no = 108;
    printf ( "Name: %s\n" , st.name);
    printf ( "Branch: %s\n" , st.branch);
    printf ( "ID_no: %d\n" , st.ID_no);
    return 0;
}
```

Output

```
Name: Kamlesh Joshi
Branch: Computer Science And Engineering
ID_no: 108
```

2. typedef with Pointers

typedef can also be used with pointers as it gives an alias name to the pointers. Typedef is very efficient while declaring multiple pointers in a single statement because pointers bind to the right on the simple declaration.

Example:

```
typedef int* Int_ptr;
Int_ptr var, var1, var2;
```

In the above statement var, var1, and var2 are declared as pointers of type int which helps us to declare multiple numbers of pointers in a single statement.

Example 2: Using typedef to define a name for pointer type.

C

```
// C program to implement
// typedef with pointers
#include <stdio.h>
typedef int * ptr;
// Driver code
int main()
{
  ptr var;
  *var = 20;
  printf ( "Value of var is %d" , *var);
  return 0;
}
```

Output

```
Value of var is 20
```

3. typedef with Array

typedef can also be used with an array to increase their count.

Example:

```
typedef int arr[20]
```

Here, arr is an alias for an array of 20 integer elements.

```
// it's same as Arr[20], two-Arr[20][23];
arr Arr, two-Arr[23];
```

Example 3: Using typedef to define an alias for Array.

C

```
// C program to implement typedef with array
#include <stdio.h>
```

```
typedef int Arr[4];

// Driver code

int main()

{

  Arr temp = { 10, 20, 30, 40 };

  printf ( "typedef using an array\n" );

  for ( int i = 0; i < 4; i++) {

  printf ( "%d " , temp[i]);

  }

  return 0;

}
```

Output

```
typedef using an array
10 20 30 40
```

C typedef vs #define

The following are the major **difference between the typedef and #define** in C:

1. #define is capable of defining aliases for values as well, for instance, you can define 1 as ONE, 3.14 as PI, etc. Typedef is limited to giving symbolic names to types only.
2. Preprocessors interpret #define statements, while the compiler interprets typedef statements.
3. There should be no semicolon at the end of #define, but a semicolon at the end of typedef.
4. In contrast with #define, typedef will actually define a new type by copying and pasting the definition values.

Below is the C program to implement #define:

C

```
// C program to implement #define

#include <stdio.h>

// macro definition
```

```
#define LIMIT 3
// Driver code
int main()
{
  for ( int i = 0; i < LIMIT; i++) {
  printf ( "%d \n" , i);
  }
  return 0;
}
```

Output

```
0
1
2
```