# RESEARCH

## STRUCT AND TYPEDEF

JULIO EMANUEL GONZALEZ GONZALEZ

# STRUCT.

## Structures.

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.).

### Create a Structure

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;
}
```

Now you can easily create multiple structure variables with different values, using just one structure:

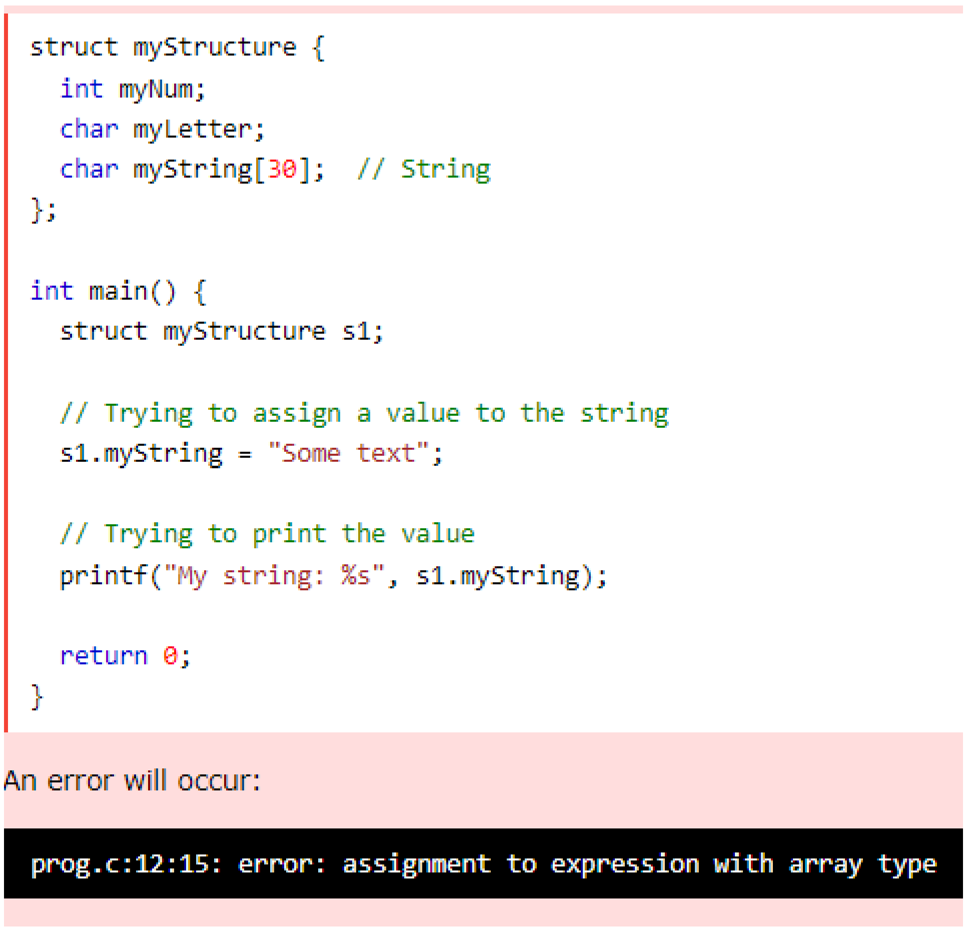
**Example.**

```
// Create different struct variables
struct myStructure s1;
struct myStructure s2;

// Assign values to different struct variables
s1.myNum = 13;
s1.myLetter = 'B';

s2.myNum = 20;
s2.myLetter = 'C';
```

**What About Strings in Structures?**

Remember that strings in C are actually an array of characters, and unfortunately, you can't assign a value to an array like this:

**Example.**



```c
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];  // String
};

int main() {
  struct myStructure s1;

  // Trying to assign a value to the string
  s1.myString = "Some text";

  // Trying to print the value
  printf("My string: %s", s1.myString);

  return 0;
}
```

An error will occur:

```
prog.c:12:15: error: assignment to expression with array type
```

However, there is a solution for this! You can use the strcpy() function and assign the value to s1.myString, like this:

**Example.**

```c
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30]; // String
};

int main() {
  struct myStructure s1;

  // Assign a value to the string using the strcpy function
  strcpy(s1.myString, "Some text");

  // Print the value
  printf("My string: %s", s1.myString);

  return 0;
}
```

Result:

```
My string: Some text
```

**Simpler Syntax**

You can also assign values to members of a structure variable at declaration time, in a single line.

Just insert the values in a comma-separated list inside curly braces {}. Note that you don't have to use the strcpy() function for string values with this technique:

**Example.**

```c
// Create a structure
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  // Create a structure variable and assign values to it
  struct myStructure s1 = {13, 'B', "Some text"};

  // Print values
  printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);

  return 0;
}
```

The order of the inserted values must match the order of the variable types declared in the structure (13 for int, 'B' for char, etc).

**Copy Structures**

You can also assign one structure to another.

In the following example, the values of s1 are copied to s2:

**Example.**

```c
struct myStructure s1 = {13, 'B', "Some text"};
struct myStructure s2;

s2 = s1;
```

**Modify Values**

If you want to change/modify a value, you can use the dot syntax (.).

And to modify a string value, the strcpy() function is useful again:

**Example.**

```c
struct myStructure {
  int myNum;
  char myLetter;
  char myString[30];
};

int main() {
  // Create a structure variable and assign values to it
  struct myStructure s1 = {13, 'B', "Some text"};

  // Modify values
  s1.myNum = 30;
  s1.myLetter = 'C';
  strcpy(s1.myString, "Something else");

  // Print values
  printf("%d %c %s", s1.myNum, s1.myLetter, s1.myString);

  return 0;
}
```

Modifying values are especially useful when you copy structure values:

**Example.**

```c
// Create a structure variable and assign values to it
struct myStructure s1 = {13, 'B', "Some text"};

// Create another structure variable
struct myStructure s2;

// Copy s1 values to s2
s2 = s1;

// Change s2 values
s2.myNum = 30;
s2.myLetter = 'C';
strcpy(s2.myString, "Something else");

// Print values
printf("%d %c %s\n", s1.myNum, s1.myLetter, s1.myString);
printf("%d %c %s\n", s2.myNum, s2.myLetter, s2.myString);
```

**Real-Life Example.**

Use a structure to store different information about Cars:

**Example.**

```
struct Car {
  char brand[50];
  char model[50];
  int year;
};

int main() {
  struct Car car1 = {"BMW", "X5", 1999};
  struct Car car2 = {"Ford", "Mustang", 1969};
  struct Car car3 = {"Toyota", "Corolla", 2011};

  printf("%s %s %d\n", car1.brand, car1.model, car1.year);
  printf("%s %s %d\n", car2.brand, car2.model, car2.year);
  printf("%s %s %d\n", car3.brand, car3.model, car3.year);

  return 0;
}
```

## typedef.

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.

**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 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.

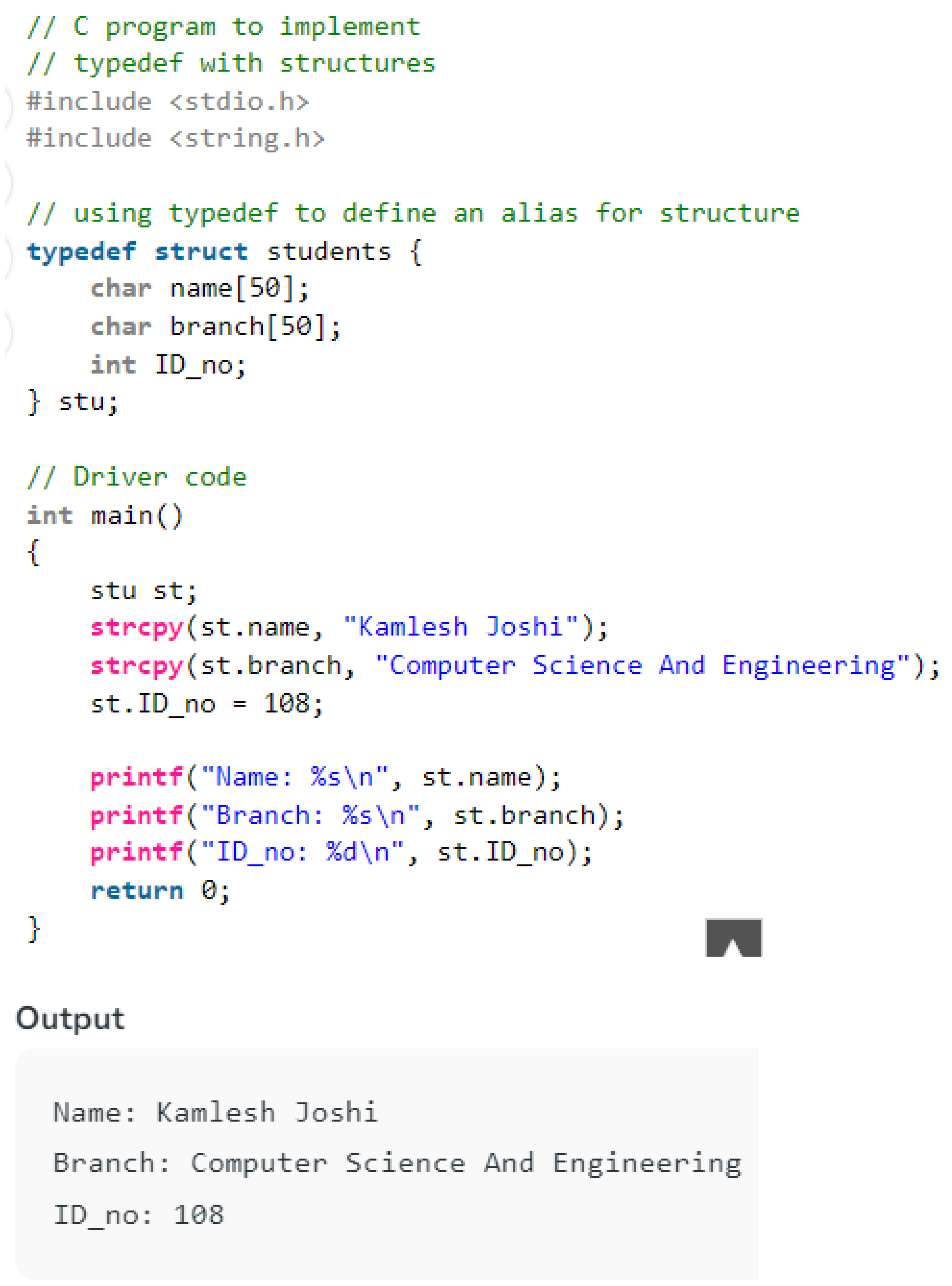
The typedef keyword can also be used with pointers to declare multiple pointers in a single statement.

It can be used with arrays to declare any number of variables.

**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
```

**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 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
```

**typedef with Array**

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

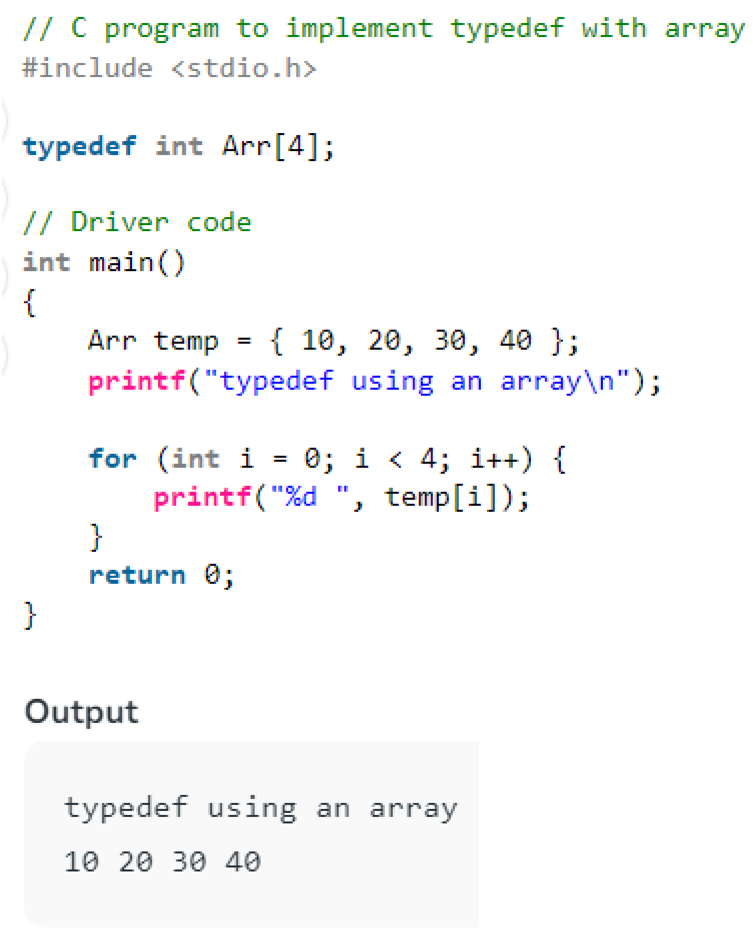
**Example.**

```
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 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

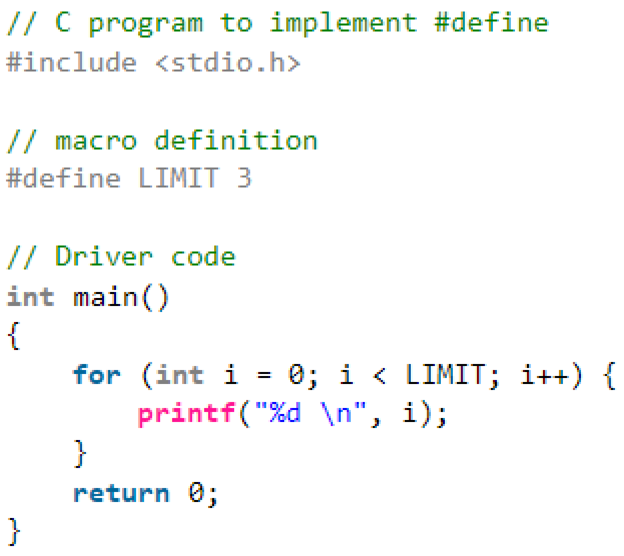
```
typedef using an array
10 20 30 40
```

## typedef vs #define.

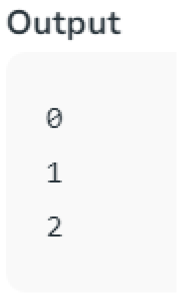
The following are the major difference between the typedef and #define in C:

- #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.
- Preprocessors interpret #define statements, while the compiler interprets typedef statements.
- There should be no semicolon at the end of #define, but a semicolon at the end of typedef.
- 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
```

## FAQs on typedef in C

### What is typedef in C?

The C typedef statement defines an alias or a nickname for the already existing data type.

### What is typedef struct?

The typedef struct is the statement used to define an alias for the structure data type.

### What is typedef enum?

The typedef enum is used to define the alias for the enumeration data type.

## References

[1] "C structures (structs)," W3schools.com. [Online]. Available: https://www.w3schools.com/c/c_structs.php. [Accessed: 11-Jan-2024].

[2] K. Follow, "C typedef," GeeksforGeeks, 07-Oct-2022. [Online]. Available: https://www.geeksforgeeks.org/typedef-in-c/. [Accessed: 11-Jan-2024].