


# OverIQ.com (/)

Home (/) / C Programming Tutorial (/c-programming-101/) / typedef statement in C

## typedef statement in C

 Last updated on July 27, 2020

The `typedef` is an advance feature in C language which allows us to create an alias or new name for an existing type or user defined type. The syntax of `typedef` is as follows:

**Syntax:** `typedef data_type new_name;`

`typedef : It`  
`data_type : I`  
`new_name : al`



Let's take an example:

```
typedef int myint;
```

Now `myint` is an alias of `int`. From now on we can declare new `int` variables using `myint` instead of `int` keyword.

```
myint i = 0; // this statement is equivalent to int i = 0;
```

This statement declares and initializes a variable `i` of type `int`.

We can even create more than one alias for the same type. For example:

```
typedef int myint, integer;
```

This statement creates two aliases for type `int` namely `myint` and `integer`.

Here are some more examples:

```
1 typedef unsigned long int ulint;  
2 typedef float real;
```

After these two declarations, `ulint` is an alias of `unsigned long int` and `real` is an alias of `float`.

We can write `typedef` declaration anywhere other declarations are allowed. However, it is important to note that the scope of the declarations depends on the location of the `typedef` statement. If the definition is placed outside all functions then the scope is global and any function can use an alias instead of the original name. On the other hand, if the definition is declared inside a function then the scope is local and the only the function which contains the `typedef` statement can use an alias. Consider the following examples:

### Example 1: Declaring a local alias using typedef



([https://sky.carbonads.net/ads/click/x/GTND42](https://sky.carbonads.net/ads/click/x/GTND42&EPOCH=1651184141&LINKID=1&CAMPAIGN=UNITS&TM_TERM=CARBON)  
 segment=placement:overiqcom,)

```

1  #include<stdio.h>
2  void foo(void);
3
4  int main()
5  {
6      typedef unsigned char uchar;
7      uchar ch = 'a';
8      printf("ch inside main() : %c\n", ch);
9      foo();
10     return 0;
11 }
12
13 void foo(void)
14 {
15     // uchar ch = 'a'; // Error
16     unsigned char ch = 'z';
17     printf("ch inside foo() : %c\n", ch);
18 }

```

**Expected Output**

```

1  ch inside main() : a
2  ch inside foo() : z

```

Here typedef definition is inside main() function so we can use alias uchar only inside the main(). Try uncommenting the line 15 and compile the program you will get an error from compiler because alias uchar is not available in the foo() function.

**Example 2: Declaring a global alias using typedef**

```

1  #include<stdio.h>
2
3  typedef unsigned char uchar;
4  void foo(void);
5
6  int main()
7  {
8      uchar ch = 'a';
9      printf("ch inside main() : %c\n", ch);
10     foo();
11     return 0;
12 }
13
14 void foo(void)
15 {
16     uchar ch = 'z';
17     printf("ch inside foo() : %c\n", ch);
18 }

```

**Expected Output:**

```

1  ch inside main() : a
2  ch inside foo() : z

```

Here typedef declaration is above all functions so any function can use alias uchar to declare variables of type unsigned char.

We have seen how to declare aliases for simple types in the following section. In the next section, we will learn how to define aliases for pointer, functions, structures and unions.



(<https://sky.carbonads.net/ads/click/x/GTND42>  
segment=placement:overiqcom,)

## typedef with a pointer

```
typedef int * iptr;
```

After this statement, we can declare a pointer using `iptr`.



```
iptr p;
```

This declaration is same as:

```
int *p;
```

Here are some more examples:

```
1 iptr a, *b; // same as int *a, **b;
2 iptr arr[10]; // same as int *arr[10];
```

In the first declaration, `a` is a pointer to `int` and `b` is pointer to pointer to `int`. In the second declaration, `arr` is an array of 10 integer pointers.

Here is an example:

```
1 #include<stdio.h>
2 typedef int * iptr;
3
4 int main()
5 {
6     int a = 99;
7     iptr p; // same as int *p
8     p = &a;
9
10    printf("%u\n", p);
11    printf("%d\n", *p);
12
13    return 0;
14 }
```

**Expected Output:**

```
1 2686788
2 99
```



(https://sky.carbonads.net/ads/click/x/GTND42  
segment=placement:overiqcom,)

## typedef with an Array



```
typedef int iarr[10];
```

After this declaration, `iarr` is an alias of array of 10 integer elements.

```
iarr a, b, c[5]; // same as int a[10], b[10], c[10][5];
```

In this declaration, `a` and `b` are arrays of 10 integers and `c` is a 2-D array of dimension 10\*5.



Here is an example:

```
1 #include<stdio.h>
2 typedef int iarr[10];
3
4 int main()
5 {
6     int i;
7
8     // same as int a[10] = {12, 43, 45, 65, 67, 87, 89, 91, 14, 19}
9     iarr a = {12, 43, 45, 65, 67, 87, 89, 91, 14, 19};
10
11     for(i = 0; i < 10; i++)
12     {
13         printf("a[%d] = %d\n", i ,a[i]);
14     }
15     return 0;
16 }
```

Expected Output:

```
1 a[0] = 12
2 a[1] = 43
3 a[2] = 45
4 a[3] = 65
5 a[4] = 67
6 a[5] = 87
7 a[6] = 89
8 a[7] = 91
9 a[8] = 14
10 a[9] = 19
```

## typedef with a Structure



(<https://srv.carbonads.net/ads/click/x/GTND42>  
segment=placement:overiqcom,)

```

1  struct book
2  {
3      char title[20];
4      char publisher[20];
5      char author[20];
6      int year;
7      int pages;
8  };
9
10 typedef struct book Book;

```

After this declaration, Book is an alias of struct book. So instead of using struct book to declare new structure variables we can use just use Book.

```
Book b1 = {"The Alchemist", "TDM Publication", "Paulo Coelho", 1978, 331};
```

We can also c

```

1  typedef struct
2  {
3      data_type member1;
4      data_type member1;
5      ...
6  } newname;

```

Let's rewrite structure book definition using this new syntax of typedef.

```

1  typedef struct book
2  {
3      char title[20];
4      char publisher[20];
5      char author[20];
6      int year;
7      int pages;
8  } Book;


```

Here is the program to demonstrate how to use typedef with structures.



(https://scry.carbonads.net/ads/click/x/GTND42  
segment=placement:overiqcom,)

```
1 #include<stdio.h>
2
3 typedef struct book
4 {
5     char title[20];
6     char publisher[20];
7     char author[20];
8     int year;
9     int pages;
10 } Book;
11
12 int main()
13 {
14
15     Book b1 = {
16         "The Zahir",
17         "Harper Perennial" ,
18         "Paulo Coelho",
19
20
21
22
23     printf("Title: %s\n", b1.title);
24     printf("Author: %s\n", b1.author);
25
26     return 0;
27 }
```



105607c43bbd74g02cz08mpodada4noma5np5585p1d-008405ed040046ad14bf3s%3A%2F%2Fpencilsofpromise.or;

Expected Output:

1	Title: The Zahir
2	Author: Paulo Coelho

Similarly, we can use typedef with unions.

# typedef and #define

It is important to mention that typedef is not a preprocessor directive, so its interpretation is handled by the compiler, not by the preprocessor. Recall that #define directive allows us to define an expansion for any text on the other hand typedef is used to create alias for any data type.


However, there are some cases when #define and typedef yield the same result.

The following is one such case:

	#define directive	typedef declaration
	#define uchar unsigned char	typedef unsigned char uchar;
statement to test	uchar ch;	uchar ch;
After translation	unsigned char ch;	unsigned char ch;

Here is the case when #define and typedef yield different results.

	#define directive	typedef declaration
	#define fp float *	typedef float * fp;
statement to test	fp a, b, c;	fp a, b, c;
After translation	float *a, b, c;	float *a, *b, *c;



https://sky.carbonads.net/ads/click/x/GTND42...&utm\_term=CARBON)

In the second case, as soon as preprocessor sees the statement.



```
fp a, b, c;
```

It replaces the occurrence of `fp` it replaces it with `float *`. So the above declaration becomes.

```
float *a, b, c;
```

On the other hand, `typedef` has more semantic meaning so the compiler doesn't just replace as preprocessor does.

The following program demonstrate the difference between `#define` and `typedef`.



(<https://sky.carbonads.net/ads/click/x/GTND42>  
segment=placement:overiqcom,)

```

1  #include<stdio.h>
2  #define ptr int * // replace occurrence of ptr by int *
3  typedef int * iptr; // iptr is an alias of pointer to int or int*
4
5  int main()
6  {
7      ptr a, b, c; // same as int *a, b, c;
8      iptr p1, p2, p3; // same as int *p1, *p2, *p3
9
10     b = 10;
11     c = 20;
12
13     a = &b;
14     p1 = &b;
15
16     p2 = &c;
17     p3 = &c;
18
19     pri
20     pri
21     ret
22
23 }
```

X

X

**Expected Output:**

```

1  Value at a = 10
2  Value at p2 = 20
```

**How it works:**

When preprocessor goes through the program and sees the declaration:

```
ptr a, b, c;
```

It replaces ptr with int \* , so that the above declaration becomes:

```
int *a, b, c;
```

where only a is a pointer to int , b and c are just variable of type int .

On the contrary in the following declaration.

```
iptr p1, p2, p3;
```

The compiler knows that iptr is an alias to a pointer to int , so p1, p2 and p3 are pointer variables of type int .

## Advantages typedef

It makes the program more readable. Certainly, Book b1 is more readable a than writing struct book b1 .

It makes the program portable. Let me explain how .Take a look at the prot and malloc() operator and malloc() function.



(https://sky.carbonads.net/ads/click/x/GTND42  
segment=placement:overiqcom,)



```

1  size_t sizeof(type);
2
3  void *malloc(size_t size);

```



As you can both prototypes uses type `size_t` and we have already told you to treat `size_t` as unsigned int, but that's not entirely true. The C standard says `sizeof()` must return an integer but leaves it up to the implementation to determine which type to return. The reason for this is that C standards committee decided no choice is likely to be the best for every platform. So they created a new type such as `size_t`, `time_t` etc and let the implementation use a typedef to set the name to some specific type. So one system type of `size_t` can be unsigned int, on another, it can be unsigned long int.

← Union Basics in C (/c-programming-101/union-basics-in-c/)

Basics of File Handling in C → (/c-programming-101/basics-of-file-handling-in-c/)



ezoic (https://www.ezoic.com/what-is-ezoic/)

report this ad

## C Programming Tutorial

- Intro to C Programming (/c-programming-101/intro-to-c-programming/)
- Installing Code Blocks (/c-programming-101/installing-code-blocks/)
- Creating and Running The First C Program (/c-programming-101/creating-and-running-the-first-c-program/)
- Basic Elements of a C Program (/c-programming-101/basic-elements-of-a-c-program/)
- Keywords and Identifiers (/c-programming-101/keywords-and-identifiers/)
- Data Types in C (/c-programming-101/data-types-in-c/)
- Constants in C (/c-programming-101/constants-in-c/)
- Variables in C (/c-programming-101/variables-in-c/)
- Input and Output in C (/c-programming-101/input-and-output-in-c/)
- Formatted Input and Output in C (/c-programming-101/formatted-input-and-output-in-c/)
- Arithmetic Operators in C (/c-programming-101/arithmetic-operators-in-c/)
- Operator Precedence and Associativity in C (/c-programming-101/operator-precedence-and-associativity-in-c/)
- Assignment Operator in C (/c-programming-101/assignment-operator-in-c/)
- Increment and Decrement Operators in C (/c-programming-101/increment-and-decrement-operators-in-c/)
- Relational Operators in C (/c-programming-101/relational-operators-in-c/)
- Logical Operators in C (/c-programming-101/logical-operators-in-c/)
- Conditional Operator, Comma operator and sizeof() operator in C (/c-programming-101/conditional-operator-comma-operator-and-sizeof-operator-in-c/)
- Implicit Type Conversion in C (/c-programming-101/implicit-type-conversion-in-c/)
- Explicit Type Conversion in C (/c-programming-101/explicit-type-conversion-in-c/)
- if-else statements in C (/c-programming-101/if-else-statements-in-c/)
- The while loop in C (/c-programming-101/the-while-loop-in-c/)
- The do while loop in C (/c-programming-101/the-do-while-loop-in-c/)
- The for loop in C (/c-programming-101/the-for-loop-in-c/)
- The Infinite Loop in C (/c-programming-101/the-infinite-loop-in-c/)



(https://sky.carbonads.net/ads/click/x/GTND42  
segment-placement:overiqcom,)


- The break and continue statement in C (/c-programming-101/the-break-and-continue-statement-in-c/)
- The Switch statement in C (/c-programming-101/the-switch-statement-in-c/)
- Function basics in C (/c-programming-101/function-basics-in-c/)
- The return statement in C (/c-programming-101/the-return-statement-in-c/)
- Actual and Formal arguments in C (/c-programming-101/actual-and-formal-arguments-in-c/)
- Local, Global and Static variables in C (/c-programming-101/local-global-and-static-variables-in-c/)
- Recursive Function in C (/c-programming-101/recursive-function-in-c/)
- One dimensional Array in C (/c-programming-101/one-dimensional-array-in-c/)
- One Dimensional Array and Function in C (/c-programming-101/one-dimensional-array-and-function-in-c/)
- Two Dimensional Array in C (/c-programming-101/two-dimensional-array-in-c/)
- Pointer Basics in C (/c-programming-101/pointer-basics-in-c/)
- Pointer Arithmetic in C (/c-programming-101/pointer-arithmetic-in-c/)
- Pointers and 1-D arrays (/c-programming-101/pointers-and-1-d-arrays/)
- Pointers and 2-D arrays (/c-programming-101/pointers-and-2-d-arrays/)
- Call by Value and Call by Reference in C (/c-programming-101/call-by-value-and-call-by-reference-in-c/)
- Returning more than one value from function in C (/c-programming-101/returning-more-than-one-value-from-function-in-c/)
- Return in C (/c-programming-101/return-in-c/)
- Passing 2-D Array to a function in C (/c-programming-101/passing-2-d-array-to-a-function-in-c/)
- Array of Pointers in C (/c-programming-101/array-of-pointers-in-c/)
- Void Pointers in C (/c-programming-101/void-pointers-in-c/)
- The malloc() Function in C (/c-programming-101/the-malloc-function-in-c/)
- The calloc() Function in C (/c-programming-101/the-calloc-function-in-c/)
- The realloc() Function in C (/c-programming-101/the-realloc-function-in-c/)
- String Basics in C (/c-programming-101/string-basics-in-c/)
- The strlen() Function in C (/c-programming-101/the-strlen-function-in-c/)
- The strcmp() Function in C (/c-programming-101/the-strcmp-function-in-c/)
- The strcpy() Function in C (/c-programming-101/the-strcpy-function-in-c/)
- The strcat() Function in C (/c-programming-101/the-strcat-function-in-c/)
- Character Array and Character Pointer in C (/c-programming-101/character-array-and-character-pointer-in-c/)
- Array of Strings in C (/c-programming-101/array-of-strings-in-c/)
- Array of Pointers to Strings in C (/c-programming-101/array-of-pointers-to-strings-in-c/)
- The sprintf() Function in C (/c-programming-101/the-sprintf-function-in-c/)
- The sscanf() Function in C (/c-programming-101/the-sscanf-function-in-c/)
- Structure Basics in C (/c-programming-101/structure-basics-in-c/)
- Array of Structures in C (/c-programming-101/array-of-structures-in-c/)
- Array as Member of Structure in C (/c-programming-101/array-as-member-of-structure-in-c/)
- Nested Structures in C (/c-programming-101/nested-structures-in-c/)
- Pointer to a Structure in C (/c-programming-101/pointer-to-a-structure-in-c/)
- Pointers as Structure Member in C (/c-programming-101/pointers-as-structure-member-in-c/)
- Structures and Functions in C (/c-programming-101/structures-and-functions-in-c/)
- Union Basics in C (/c-programming-101/union-basics-in-c/)
- typedef statement in C (/c-programming-101/typedef-statement-in-c/)
- Basics of File Handling in C (/c-programming-101/basics-of-file-handling-in-c/)
- fputc() Function in C (/c-programming-101/fputc-function-in-c/)
- fgetc() Function in C (/c-programming-101/fgetc-function-in-c/)
- fputs() Function in C (/c-programming-101/fputs-function-in-c/)
- fgets() Function in C (/c-programming-101/fgets-function-in-c/)
- fprintf() Function in C (/c-programming-101/fprintf-function-in-c/)
- fscanf() Function in C (/c-programming-101/fscanf-function-in-c/)
- fwrite() Function in C (/c-programming-101/fwrite-function-in-c/)
- fread() Function in C (/c-programming-101/fread-function-in-c/)



105607c43bb6d44a827708c000ad4410f758967d559000c1d10887105c4e400166ad14ef8a73A72F72Epcilsofpromise.org



(https://sky.carbonads.net/ads/click/x/GTND42  
segment-placement:overiqcom,)

 **ezoic** (<https://www.ezoic.com/what-is-ezoic/>)

[report this ad](#)

## Recent Posts

- Machine Learning Assignment Online Help for the Busy and Tired Students: Get Help from Experts (/machine-learning-assignment-online-help-for-the-busy-and-tired-students-get-help-from-experts/)
- 4 Ways to Install Python 3 on Linux (/4-ways-to-install-python-3-on-linux/)
- Finance Assignment Online Help for the Busy and Tired Students: Get Help from Experts (/finance-assignment-online-help-for-the-busy-and-tired-students-get-help-from-experts/)
- Top 9 Machine Learning Algorithms for Data Scientists (/top-9-machine-learning-algorithms-for-data-scientists/)
- Data Science Learning Path or Steps to become a data scientist Final (/data-science-learning-path-or-steps-to-become-a-data-scientist-final/)
- Enable Edit Button in Shutter In Linux Mint 19 and Ubuntu 18.04 (/enable-edit-button-in-shutter-in-linux-mint-19-and-ubuntu-18-04/)
- Python 3 time module (/python-3-time-module/)
- Pygments Tutorial (/pygments-tutorial/)
- How to use Virtualenv? (/how-to-use-virtualenv/)
- Installing MySQL (Windows, Linux and Mac) (/installing-mysql-windows-linux-and-mac/)
- What is if \_\_name\_\_ == '\_\_main\_\_' in Python ? (/what-is-if-\_\_name\_\_-\_\_main\_\_-in-python/)
- Installing GoAccess (A Real-time web log analyzer) (/installing-goaccess-a-real-time-web-log-analyzer/)
- Installing Isso (/installing-isso/)



(<https://scv.carbonads.net/ads/click/x/GTND42>  
segment-placement:overiqcom,)



 ezoic  
(<https://www.ezoic.com/what-is-ezoic/>)

[report this ad](#)

[Home \(/\)](#)

[Terms \(/terms-of-use/\)](#)

[Privacy Policy \(/privacy-policy/\)](#)

[Sitemap \(/sitemap.xml/\)](#)

[Contact \(/contact/\)](#)

[Blog \(/blog/\)](#)

[Facebook \(https://www.facebook.com/OverIQ/\)](https://www.facebook.com/OverIQ/)

[Twitter \(https://twitter.com/infoOverIQ\)](https://twitter.com/infoOverIQ)

[Github \(https://github.com/overiq\)](https://github.com/overiq)

[C Tutorial \(/c-programming-101/\)](#)

[Python Tutorial \(/python-101/\)](#)

[Django Tutorial \(/django-1-11/\)](#)

[Flask Tutorial \(/flask-101/\)](#)

[MySQL Connector/Python Tutorial \(/mysql-connector-python-101/\)](#)

[SQLAlchemy Tutorial \(/sqlalchemy-101/\)](#)

[C Programming Examples \(/c-examples/\)](#)

© 2022 OverIQ.com

[Back to top](#)



(<https://sky.carbonads.net/ads/click/x/GTND42>  
segment=placement:overiqcom,)



(<https://srv.carbonads.net/ads/click/x/GTND42>  
segment=placement:overiqcom,)