typedef

The C programming language provides a keyword called **typedef**, which you can use to give a type a new name.

Syntax

So, how do you actually declare a typedef? All you must do is provide the old type name followed by the type that should represent it throughout the code. Here's how you would declare size_t to be an unsigned integer:

```
typedef unsigned int size_t;
```

Following is an example to define a term **BYTE** for one-byte numbers:

```
typedef unsigned char BYTE;
```

After this type definitions, the identifier BYTE can be used as an abbreviation for the type **unsigned char, for example:**.

```
BYTE  b1, b2;
```

By convention, uppercase letters are used for these definitions to remind the user that the type name is really a symbolic abbreviation, but you can use lowercase, as follows:

```
typedef unsigned char byte;
```

Although typedef is thought of as being a storage class, it isn't really. It allows you to introduce synonyms for types which could have been declared some other way. The new name becomes equivalent to the type that you wanted, as this example shows.

```
typedef  int  aaa, bbb, ccc;
typedef  int ar[15] ,  arr[9][6];
typedef  char c, *cp, carr[100];

/* now declare some objects */

/* all ints */
aaa     int1;
bbb     int2;
ccc     int3;

ar      yyy;    /* array of 15 ints */
arr     xxx;    /* 9*6 array of int */

c       ch;     /* a char */
cp      pnt;    /* pointer to char */
carr    chry;   /* array of 100 char */
```

You can use **typedef** to give a name to user defined data type as well. For example you can use typedef with structure to define a new data type and then use that data type to define structure variables

directly as follows:

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

typedef struct Books
{
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
} Book;

int main( )
{
  Book book;

  strcpy( book.title, "C Programming");
  strcpy( book.author, "Nuha Ali");
  strcpy( book.subject, "C Programming Tutorial");
  book.book_id = 6495407;

  printf( "Book title : %s\n", book.title);
  printf( "Book author : %s\n", book.author);
  printf( "Book subject : %s\n", book.subject);
  printf( "Book book_id : %d\n", book.book_id);

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Book  title : C Programming
Book  author : Nuha Ali
Book  subject : C Programming Tutorial
Book  book_id : 6495407
```

typedef vs #define

The **#define** is a C-directive which is also used to define the aliases for various data types similar to**typedef** but with three differences:

The **typedef** is limited to giving symbolic names to types only where as **#define** can be used to define alias for values as well, like you can define 1 as ONE etc.

The **typedef** interpretation is performed by the compiler where as **#define** statements are processed by the pre-processor.

Following is a simplest usage of #define:

```c
#include <stdio.h>

#define TRUE  1
#define FALSE 0

int main( )
{
   printf( "Value of TRUE : %d\n", TRUE);
   printf( "Value of FALSE : %d\n", FALSE);

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Value of TRUE : 1
Value of FALSE : 0
```