typedef command

the C programming language provides a keyword called **typedef** to set an alternate name to an existing data type.

The **typedef** keyword in C is very useful in assigning a convenient alias to a built-in data type as well as any derived data type such as a struct, a union or a pointer.

Sometimes it becomes clumsy to use a data type with a longer name (such as "**struct structname**" or "**unsigned int**") every time a variable is declared. In such cases, we can assign a handy shortcut to make the code more readable.

typedef Syntax

In general, the **typedef** keyword is used as follows –

```
typedef existing_type new_type;
```

typedef Examples

Example 1

In C language, the keyword "**unsigned**" is used to declare unsigned integer variables that can store only non-negative values.

C also has a keyword called "**short**" that declares an integer data type that occupies 2 bytes of memory. If you want to declare a variable that is **short** and can have only non-negative values, then you can combine both these keywords (unsigned and short):

```
short unsigned int x;
```

If there are going to be many variables to be declared of this type, it will not be very convenient to use these three keywords every time. Instead, you can define an **alias** or a shortcut with the **typedef** keyword as follows –

```
typedef short unsigned int USHORT;
```

```
USHORT x;
```

Example 2

```c
#include <stdio.h>

int main() {
    typedef short unsigned int USHORT;
    typedef signed long int SLONG;

    static SLONG x = 100;
    USHORT y = 200;

    printf("Size of SLONG: %d \nSize of USHORT: %d", sizeof(SLONG), sizeof(USHORT));

    return 0;
}
```

```
<terminated> (exit value: 0) text.exe [C/C++ Application
Size of SLONG: 4
Size of USHORT: 2
```

Defining a Structure using Typedef

```c
#include <stdio.h>

int main() {

    typedef unsigned long int ULONG;
    typedef short int SHORT;

    struct mystruct {
        ULONG a;
        SHORT b;
    };

    typedef struct mystruct STR;
    STR s1 = {10, 20};
    printf("%ld %u", s1.a, s1.b);

    return 0;
}
```

```
<terminated> (exit value: 0) text.exe [C/C++ Application
10 20
```

Typedef for Struct Pointer

```c
#include <stdio.h>

int main() {

   typedef unsigned long int ULONG;
   typedef short int SHORT;

   typedef struct mystruct {
      ULONG a;
      SHORT b;
   } STR;

   STR s1 = {10, 20};
   typedef STR * strptr;
   strptr ptr = &s1;

   printf("%d %d \n", s1.a, s1.b);
   printf("%d %d", ptr->a, ptr->b);

   return 0;
}
```

```
<terminated> (exit value: 0) text.exe [C/C++ Application
10 20
10 20
```

Typedef for Union

```c
#include <stdio.h>

int main() {

   typedef unsigned long int ULONG;
   typedef short int SHORT;

   typedef union myunion {
      char a;
      int b;
      double c;
   } UNTYPE;

   UNTYPE u1;
   u1.c = 65.50;

   typedef UNTYPE * UNPTR;
   UNPTR ptr = &u1;

   printf("a:%c b: %d c: %lf \n", u1.a, u1.b, u1.c);
   printf("a:%c b: %d c: %lf \n", ptr->a, ptr->b, ptr->c);

   return 0;
}
```

```
<terminated> (exit value: 0) text.exe [C/C++ Application
a:  b: 0 c: 65.500000
a:  b: 0 c: 65.500000
```

typedef vs #define in C

In C language, **#define** is a preprocessor directive. It is an effective method to define a constant. **#define** is a preprocessor directive, while **typedef** is evaluated at the time of compilation.

- **typedef** is limited to giving symbolic names to types only.
- **#define** can be used to define alias for values as well. For example, you can define "1" as "ONE".
- **typedef** interpretation is performed by the compiler.
- **#define** statements are processed by the pre-processor.