Storage class in C programming

A storage class defines the scope (visibility) and the lifetime of a variable or / and functions in the C program. The preceding type specifications can be changed. Here are the storage classes that can be used in the C program.

A storage class defines the scope (visibility) and the lifetime of a variable or / and functions in the C program. The preceding type specifications can be changed. Here are the storage classes that can be used in the C program.

auto

register

static

extern

Storage class auto in C

The storage class **auto** is the default storage class for all local variables:

```
{ int  diemthi ; auto int  diemthi ; }
```

The above example to define two variables in the same storage class, auto can be used inside the function, for example local variables.

The class stores the register in C

The **register** class can be used to define local variables and can be stored in a registry instead of RAM. This means that this variable has a maximum size equivalent to the registration size.

```
{ register int  hocphi ; }
```

The register class is only used for variables that require quick access like counters. Note that the definition of 'register' does not mean that the variable can be stored in a register. It means that it can be stored in hardware-dependent registers and with certain restrictions.

Static storage class in C

The **static** storage class instructs the compiler to keep local variable values ??that exist during the program's lifetime instead of creating and destroying it each time it runs over that range. Therefore, create a **local static** variable that allows them to store values ??with call functions.

This static class can be applied to global variables. When this happens, it causes the scope of the variable to be limited to the file it declares.

In programming C, when static is used for the class, it leads to only one copy of the declaration class shared by all objects using this class.

```
#include /* phan khai bao ham */ void  ham ( void ); static int  biendem  = 4 ;
```

You may not understand this example because you use global variables, which will be introduced in the next article. Compile and run the above C program to see the results:

The class stores extern in C

Logos that store **extern** are used to refer to global variables that are seen by all program files.

When you have multiple files and you define local variables or functions, it will be used in other files. To understand this problem, extern is used to declare global variables or functions in other files.

The extern keyword is used when two or more files share the same variable or function as the example below:

First file: file1.c

```
#include int  biendem  ; extern void  ham_extern ();  main () {  biendem  = 5 ;
```

The second file: file2.c

```
#include extern int  biendem ; void  ham_extern ( void ) {  printf ( "biendem c
```

Here, *extern* is the keyword used to declare *biendem* in the second line where it is defined in the first file, *main.c.* Now, if you are using a command prompt, you compile 2 files as follows:

```
$gcc main . c support . c
```

It will provide a.out executable program, when this program is run it will print the following result:

```
5
```

According to Tutorialspoint

Previous article: Constant in programming C

Next lesson: Operator in programming C