Understanding “extern” keyword in C

I’m sure that this post will be as interesting and informative to C virgins (i.e. beginners) as it will be to those who are well versed in C. So let me start with saying that extern keyword applies to C variables (data objects) and C functions. Basically extern keyword extends the visibility of the C variables and C functions. Probably that’s is the reason why it was named as extern.

Though (almost) everyone knows the meaning of declaration and definition of a variable/function yet for the sake of completeness of this post, I would like to clarify them. Declaration of a variable/function simply declares that the variable/function exists somewhere in the program but the memory is not allocated for them. But the declaration of a variable/function serves an important role. And that is the type of the variable/function. Therefore, when a variable is declared, the program knows the data type of that variable. In case of function declaration, the program knows what are the arguments to that functions, their data types, the order of arguments and the return type of the function. So that’s all about declaration. Coming to the definition, when we define a variable/function, apart from the role of declaration, it also allocates memory for that variable/function. Therefore, we can think of definition as a super set of declaration. (or declaration as a subset of definition). From this explanation, it should be obvious that a variable/function can be declared any number of times but it can be defined only once. (Remember the basic principle that you can’t have two locations of the same variable/function). So that’s all about declaration and definition.

Now coming back to our main objective: Understanding “extern” keyword in C. I’ve explained the role of declaration/definition because it’s mandatory to understand them to understand the “extern” keyword. Let us first take the easy case. Use of extern with C functions. By default, the declaration and definition of a C function have “extern” prepended with them. It means even though we don’t use extern with the declaration/definition of C functions, it is present there. For example, when we write.

```
int foo(int arg1, char arg2);
```

There’s an extern present in the beginning which is hidden and the compiler treats it as below.

```
extern int foo(int arg1, char arg2);
```

Same is the case with the definition of a C function (Definition of a C function means writing the body of the function). Therefore whenever we define a C function, an extern is present there in the beginning of the function definition. Since the declaration can be done any number of times and definition can be done only once, we can notice that declaration of a function can be added in several C/H files or in a single C/H file several times. But we notice the actual definition of the function only once (i.e. in one file only). And as the extern extends the visibility to the whole program, the functions can be used (called) anywhere in any of the files of the whole program provided the declaration of the function is known. (By knowing the declaration of the function, C compiler knows that the definition of the function exists and it goes ahead to compile the program). So that’s all about extern with C functions.

Now let us the take the second and final case i.e. use of extern with C variables. I feel that it more interesting and information than the previous case where extern is present by default with C functions. So let me ask the question, how would you declare a C variable without defining it? Many of you would see it trivial but it’s important question to understand extern with C variables. The answer goes as follows.

```
extern int var;
```

Here, an integer type variable called var has been declared (remember no definition i.e. no memory allocation for var so far). And we can do this declaration as many times as needed. (remember that declaration can be done any number of

times) So far so good. 🙂

Now how would you define a variable. Now I agree that it is the most trivial question in programming and the answer is as follows.

```
    int var;
```

Here, an integer type variable called var has been declared as well as defined. (remember that definition is the super set of declaration). Here the memory for var is also allocated. Now here comes the surprise, when we declared/defined a C function, we saw that an extern was present by default. While defining a function, we can prepend it with extern without any issues. But it is not the case with C variables. If we put the presence of extern in variable as default then the memory for them will not be allocated ever, they will be declared only. Therefore, we put extern explicitly for C variables when we want to declare them without defining them. Also, as the extern extends the visibility to the whole program, by externing a variable we can use the variables anywhere in the program provided we know the declaration of them and the variable is defined somewhere.

Now let us try to understand extern with examples.

Example 1:

```
int var;
int main(void)
{
   var = 10;
   return 0;
}
```

Run on IDE

Analysis: This program is compiled successfully. Here var is defined (and declared implicitly) globally.

Example 2:

```
extern int var;
int main(void)
{
  return 0;
}
```

Run on IDE

Analysis: This program is compiled successfully. Here var is declared only. Notice var is never used so no problems.

Example 3:

```
extern int var;
int main(void)
{
 var = 10;
 return 0;
}
```

Run on IDE

Analysis: This program throws error in compilation. Because var is declared but not defined anywhere. Essentially, the var isn't allocated any memory. And the program is trying to change the value to 10 of a variable that doesn't exist at all.

Example 4:

```
#include "somefile.h"
extern int var;
```

```
int main(void)
{
 var = 10;
 return 0;
}
```

Run on IDE

Analysis: Supposing that somefile.h has the definition of var. This program will be compiled successfully.

Example 5:

```
extern int var = 0;
int main(void)
{
 var = 10;
 return 0;
}
```

Run on IDE

Analysis: Guess this program will work? Well, here comes another surprise from C standards. They say that..if a variable is only declared and an initializer is also provided with that declaration, then the memory for that variable will be allocated i.e. that variable will be considered as defined. Therefore, as per the C standard, this program will compile successfully and work.

So that was a preliminary look at “extern” keyword in C.

I’m sure that you want to have some take away from the reading of this post. And I would not disappoint you. 🙂
In short, we can say

1. Declaration can be done any number of times but definition only once.
2. “extern” keyword is used to extend the visibility of variables/functions().
3. Since functions are visible through out the program by default. The use of extern is not needed in function declaration/definition. Its use is redundant.
4. When extern is used with a variable, it’s only declared not defined.
5. As an exception, when an extern variable is declared with initialization, it is taken as definition of the variable as well.