WIKIPEDIA
The Free Encyclopedia

Static variable

In computer programming, a **static variable** is a variable that has been allocated "statically", meaning that its lifetime (or "extent") is the entire run of the program. This is in contrast to shorter-lived automatic variables, whose storage is stack allocated and deallocated on the call stack; and in contrast to objects, whose storage is dynamically allocated and deallocated in heap memory.

Variable lifetime is contrasted with scope (where a variable can be used): "global" and "local" refer to scope, not lifetime, but scope often implies lifetime. In many languages, global variables are always static, but in some languages they are dynamic, while local variables are generally automatic, but may be static.

In general, **static memory allocation** is the allocation of memory at compile time, before the associated program is executed, unlike dynamic memory allocation or automatic memory allocation where memory is allocated as required at run time.[1]

History

Static variables date at least to ALGOL 60 (1960), where they are known as ***own*** *variables*:

> A declaration may be marked with the additional declarator **own.** This has the following effect: upon a re-entry into the block, the values of **own** quantities will be unchanged from their values at the last exit, while the values of declared variables that are not marked with **own** is undefined.
>
> —Revised report on ALGOL 60, section "5. Declarations", p. 14

This definition is subtly different from a static variable: it only specifies behavior, and hence lifetime, not storage: an own variable can be allocated when a function is first called, for instance, rather than at program load time.

The use of the word *static* to refer to these variables dates at least to BCPL (1966), and has been popularized by the C programming language, which was heavily influenced by BCPL. The BCPL definition reads:

> (1) Static data items:
> Those data items whose extents lasts as long as the program execution time; such data items have manifest constant Lvalues. Every static data item must have been declared either in a function or routine definition, in a global declaration or as a label set by colon.
>
> —The BCPL Reference Manual, 7.2 Space Allocation and Extent of Data Items

Note that BCPL defined a "dynamic data item" for what is now called an *automatic* variable (local, stack-allocated), not for heap-allocated objects, which is the current use of the term *dynamic allocation*.

The static keyword is used in C and related languages both for static variables and other concepts.

Addressing

The absolute address addressing mode can only be used with static variables, because those are the only kinds of variables whose location is known by the compiler at compile time. When the program (executable or library) is loaded into memory, static variables are stored in the data segment of the program's address space (if initialized), or the BSS segment (if uninitialized), and are stored in corresponding sections of object files prior to loading.

Scope

In terms of scope and extent, static variables have extent the entire run of the program, but may have more limited scope. A basic distinction is between a *static global variable*, which has global scope and thus is in context throughout the program, and a *static local variable*, which has local scope. A static local variable is different from a local variable as a static local variable is initialized only once no matter how many times the function in which it resides is called and its value is retained and accessible through many calls to the function in which it is declared, e.g. to be used as a count variable. A static variable may also have module scope or some variant, such as internal linkage in C, which is a form of file scope or module scope.

Example

An example of a static local variable in C:

```c
#include <stdio.h>

void Func() {
  static int x = 0;
  // |x| is initialized only once across five calls of |Func| and the variable
  // will get incremented five times after these calls. The final value of |x|
  // will be 5.
  x++;
  printf("%d\n", x);  // outputs the value of |x|
}

int main() {
  Func();  // prints 1
  Func();  // prints 2
  Func();  // prints 3
  Func();  // prints 4
  Func();  // prints 5

  return 0;
}
```

Object-oriented programming

In object-oriented programming, there is also the concept of a *static member variable*, which is a "class variable" of a statically defined class, i.e., a member variable of a given class which is shared across all instances (objects), and is accessible as a member variable of these objects. A class variable of a dynamically defined class, in languages where classes can be defined at run time, is allocated when the class is defined and is not static.

Object constants known at compile-time, such as string literals, are usually allocated statically. In object-oriented programming, the virtual method tables of classes are usually allocated statically. A statically defined value can also be global in its scope ensuring the same immutable value is used throughout a run for consistency.

See also

- Constant (computer programming)
- Global variable
- Static method
- Thread-local storage

Notes

1. Jack Rons. "What is static memory allocation and dynamic memory allocation?" (http://www.merithub.com/q/58-static-memory-allocation-dynamic-memory-allocation.aspx). MeritHub [An Institute of Career Development]. Retrieved 2011-06-16. "The compiler allocates required memory space for a declared variable. By using the addressof operator, the reserved address is obtained and this address may be assigned to a pointer variable. Since most of the declared variables have static memory, this way of assigning pointer value to a pointer variable is known as static memory allocation. Memory is assigned during compilation time."

References

- Kernighan, Brian W.; Ritchie, Dennis M. (1988). *The C Programming Language* (https://archive.org/details/cprogramminglang00bria) (2nd ed.). Upper Saddle River, NJ: Prentice Hall PTR. ISBN 0-13-110362-8.
- *The C++ Programming Language* (special edition) by Bjarne Stroustrup (Addison Wesley, 2000; ISBN 0-201-70073-5)