---



# Overview of stabs

**Stabs** refers to a format for information that describes a program to a debugger. This format was apparently invented by the University of California at Berkeley, for the `pdx` Pascal debugger; the format has spread widely since then.



## Overview of debugging information flow

The GNU C compiler compiles C source in a `.c' file into assembly language in a `.s' file, which is translated by the assembler into a `.o' file, and then linked with other `.o' files and libraries to produce an executable file.

With the `-g' option, GCC puts additional debugging information in the `.s' file, which is slightly transformed by the assembler and linker, and carried through into the final executable. This debugging information describes features of the source file like line numbers, the types and scopes of variables, and functions, their parameters and their scopes.

For some object file formats, the debugging information is encapsulated in assembler directives known collectively as `stab' (symbol table) directives, interspersed with the generated code. Stabs are the native format for debugging information in the a.out and xcoff object file formats. The GNU tools can also emit stabs in the coff and ecoff object file formats.

The assembler adds the information from stabs to the symbol information it places by default in the symbol table and the string table of the `.o' file it is building. The linker consolidates the `.o' files into one executable file, with one symbol table and one string table. Debuggers use the symbol and string tables in the executable as a source of debugging information about the program.

## Overview of stab format

There are three overall formats for stab assembler directives differentiated by the first word of the stab. The name of the directive describes what combination of four possible data fields will follow. It is either `.stabs` (string), `.stabn` (number), or `.stabd` (dot).

The overall format of each class of stab is:

```
.stabs "string",type,0,desc,value
.stabn          type,0,desc,value
.stabd          type,0,desc
```

In general, in `.stabs` the *string* field contains name and type information. For `.stabd` the value field is implicit and has the value of the current file location. Otherwise the value field often contains a relocatable address, frame pointer offset, or register number, that maps to the source code element described by the stab.

The real key to decoding the meaning of a stab is the number in its type field. Each possible type number defines a different stab type. The stab type further defines the exact interpretation of, and possible values for, any remaining `"string"`, *desc*, or *value* fields present in the stab. Table A (see section Table A: Symbol types from stabs) lists in

numeric order the possible type field values for stab directives. The reference section that follows Table A describes the meaning of the fields for each stab type in detail. The examples that follow this overview introduce the stab types in terms of the source code elements they describe.

For `.stabs` the "*string*" field holds the meat of the debugging information. The generally unstructured nature of this field is what makes stabs extensible. For some stab types the string field contains only a name. For other stab types the contents can be a great deal more complex.

The overall format is of the "*string*" field is:

```
"name[:symbol_descriptor]
     [type_number[=type_descriptor ...]]"
```

*name* is the name of the symbol represented by the stab.

The *symbol_descriptor* following the `:' is an alphabetic character that tells more specifically what kind of symbol the stab represents. If the *symbol_descriptor* is omitted, but type information follows, then the stab represents a local variable. For a list of symbol_descriptors, see section Table C: Symbol descriptors.

Type information is either a *type_number*, or a `*type_number*='. The *type_number* alone is a type reference, referring directly to a type that has already been defined.

The `*type_number*=' is a type definition, where the number represents a new type which is about to be defined. The type definition may refer to other types by number, and those type numbers may be followed by `=' and nested definitions.

In a type definition, if the character that follows the equals sign is non-numeric then it is a *type_descriptor*, and tells what kind of type is about to be defined. Any other values following the *type_descriptor* vary, depending on the *type_descriptor*. If a number follows the `=' then the number is a *type_reference*. This is described more thoroughly in the section on types. See section Table D: Type Descriptors, for a list of *type_descriptor* values.

All this can make the "*string*" field quite long. When the "*string*" part of a stab is too long, the compiler splits the `.stabs` directive into two `.stabs` directives. Both stabs duplicate exactly all but the "*string*" field. The "*string*" field of the first stab contains the first part of the overlong string, marked as continued with a double-backslash at the end. The "*string*" field of the second stab holds the second half of the overlong string.

# A simple example in C source

To get the flavor of how stabs describe source information for a C program, let's look at the simple program:

```
main()
{
        printf("Hello world");
}
```

When compiled with `-g', the program above yields the following `.s' file. Line numbers have been added to make it easier to refer to parts of the `.s' file in the description of the stabs that follows.

# The simple example at the assembly level

```
1  gcc2_compiled.:
2  .stabs "/cygint/s1/users/jcm/play/",100,0,0,Ltext0
3  .stabs "hello.c",100,0,0,Ltext0
4  .text
5  Ltext0:
6  .stabs "int:t1=r1;-2147483648;2147483647;",128,0,0,0
7  .stabs "char:t2=r2;0;127;",128,0,0,0
8  .stabs "long int:t3=r1;-2147483648;2147483647;",128,0,0,0
9  .stabs "unsigned int:t4=r1;0;-1;",128,0,0,0
10 .stabs "long unsigned int:t5=r1;0;-1;",128,0,0,0
11 .stabs "short int:t6=r1;-32768;32767;",128,0,0,0
12 .stabs "long long int:t7=r1;0;-1;",128,0,0,0
13 .stabs "short unsigned int:t8=r1;0;65535;",128,0,0,0
14 .stabs "long long unsigned int:t9=r1;0;-1;",128,0,0,0
```

```
15 .stabs "signed char:t10=r1;-128;127;",128,0,0,0
16 .stabs "unsigned char:t11=r1;0;255;",128,0,0,0
17 .stabs "float:t12=r1;4;0;",128,0,0,0
18 .stabs "double:t13=r1;8;0;",128,0,0,0
19 .stabs "long double:t14=r1;8;0;",128,0,0,0
20 .stabs "void:t15=15",128,0,0,0
21       .align 4
22 LC0:
23       .ascii "Hello, world!\12\0"
24       .align 4
25       .global _main
26       .proc 1
27 _main:
28 .stabn 68,0,4,LM1
29 LM1:
30       !#PROLOGUE# 0
31       save %sp,-136,%sp
32       !#PROLOGUE# 1
33       call ___main,0
34       nop
35 .stabn 68,0,5,LM2
36 LM2:
37 LBB2:
38       sethi %hi(LC0),%o1
39       or %o1,%lo(LC0),%o0
40       call _printf,0
41       nop
42 .stabn 68,0,6,LM3
43 LM3:
44 LBE2:
45 .stabn 68,0,6,LM4
46 LM4:
47 L1:
48       ret
49       restore
50 .stabs "main:F1",36,0,0,_main
51 .stabn 192,0,0,LBB2
52 .stabn 224,0,0,LBE2
```

This simple "hello world" example demonstrates several of the stab types used to describe C language source files.

---

Go to the first, previous, next, last section, table of contents.