10.5 Output Formats

By default, GDB prints a value according to its data type. Sometimes this is not what you want. For example, you might want to print a number in hex, or a pointer in decimal. Or you might want to view data in memory at a certain address as a character string or as an instruction. To do these things, specify an *output format* when you print a value.

The simplest use of output formats is to say how to print a value already computed. This is done by starting the arguments of the `print` command with a slash and a format letter. The format letters supported are:

`x`

Regard the bits of the value as an integer, and print the integer in hexadecimal.

`d`

Print as integer in signed decimal.

`u`

Print as integer in unsigned decimal.

`o`

Print as integer in octal.

`t`

Print as integer in binary. The letter ‘`t`’ stands for “two”. [10]

`a`

Print as an address, both absolute in hexadecimal and as an offset from the nearest preceding symbol. You can use this format used to discover where (in what function) an unknown address is located:

```
(gdb) p/a 0x54320
$3 = 0x54320 <_initialize_vx+396>
```

The command `info symbol 0x54320` yields similar results. See info symbol.

`c`

Regard as an integer and print it as a character constant. This prints both the numerical value and its character representation. The character representation is replaced with the octal escape ‘`\nnn`’ for characters outside the 7-bit ASCII range.

Without this format, GDB displays `char`, `unsigned char`, and `signed char` data as character constants. Single-byte members of vectors are displayed as integer data.

`f`

Regard the bits of the value as a floating point number and print using typical floating point syntax.

`s`

Regard as a string, if possible. With this format, pointers to single-byte data are displayed as null-terminated strings and arrays of single-byte data are displayed as fixed-length strings. Other values are displayed in their natural types.

Without this format, GDB displays pointers to and arrays of `char`, `unsigned char`, and `signed char` as strings. Single-byte members of a vector are displayed as an integer array.

`z`

Like ‘`x`’ formatting, the value is treated as an integer and printed as hexadecimal, but leading zeros are printed to pad the value to the size of the integer type.

`r`

Print using the ‘`raw`’ formatting. By default, GDB will use a Python-based pretty-printer, if one is available (see Pretty Printing). This typically results in a higher-level display of the value’s contents. The ‘`r`’ format bypasses any Python pretty-printer which might exist.

For example, to print the program counter in hex (see Registers), type

```
p/x $pc
```

Note that no space is required before the slash; this is because command names in GDB cannot contain a slash.

To reprint the last value in the value history with a different format, you can use the `print` command with just a format and no expression. For example, ‘`p/x`’ reprints the last value in hex.

Footnotes

(10)

‘`b`’ cannot be used because these format letters are also used with the `x` command, where ‘`b`’ stands for “byte”; see Examining Memory.

Next: Memory, Previous: Arrays, Up: Data   [Contents][Index]