

I have written a sample code for this using www.onlinegdb.com

#include <stdio.h>

int main()

{

int var = 2020;

int& varref = var;

int\* varptr = &var;

int\*\* vardptr = &varptr;

int \*\*\* vartptr = & vardptr;

return 0;

}

Word length can be more than 8 bytes.

I my case it was 64.

So output was like this

When I debugged the code with gdb, I got following values:

(gdb) p varptr

$3 = (int \*) 0x7fffffffebcc

(gdb) x/1xw 0x7fffffffebcc

0x7fffffffebcc: 0x000007e4

(gdb)

varptr is stored at 0x7fffffffebcc

which contains address of variable var.

$5 = 2020

(gdb) p varref

$6 = (int &) @0x7fffffffebbc: 2020

(gdb) p varptr

$7 = (int \*) 0x7fffffffebbc

(gdb) x varptr

0x7fffffffebbc: 0x000007e4

(gdb) p vardptr

$8 = (int \*\*) 0x7fffffffebc0

(gdb) x vardptr

0x7fffffffebc0: 0xffffebbc

(gdb) p vartptr

$9 = (int \*\*\*) 0x7fffffffebc8

(gdb) x vartpr

No symbol "vartpr" in current context.

(gdb) x vartptr

0x7fffffffebc8: 0xffffebc0

(gdb)

varref and varptr, both, are pointing to the the variable var.

In above picture, varref and varptr have the same values except “@” in front of value in varref.

References are basically pointers with different syntax (syntactical sugar in c++ jargon).

“@” is basically a directive to the compiler to treat the value as reference.

If I print the value of these references and pointers using “x” command in gdb, it gives values in 2020 in decimal and 0x000007e4 in hex respectively.

vardptr (double pointer) is pointer to pointer varptr. So it contains address of varptr.

vartptr (triple pointer) is pointer to pointer (vardptr) to pointer (varptr).

gdb p and x command output shows the values contained by them.