https://youtu.be/-p99igKSazc

Using the Debugger to Analyze Memory

As you have seen in the last section, binary numbers and hex numbers can be used to represent information. A coding scheme such as an ASCII table makes it possible to convert text into binary form. In the following, we will try to look at computer memory and locate information there.

In the following example, we will use the debugger to look for a particular string in computer memory. Depending on your computer operating system and on the compiler you have installed, there might be several debugging tools available to you. In the following video, we will use the gdb debugger to locate the character sequence "UDACITY" in computer memory. The code below creates an array of characters in computer memory (on the stack, which we will learn more about shortly) and prints it to the console:

```
#include <stdio.h>

int main()
{
    char str1[]= "UDACITY";
    printf("%s",str1);

    return 0;
}
```

Let us try to locate the string in memory using gdb.

https://youtu.be/Spj2jK1-uIE

https://youtu.be/Lopa5WXR1uQ

As you have just seen in the video, the binary ASCII codes for the letters in UDACITY could be located in computer memory by using the address of the variable `str1` from the code example above. The output of gdb can also be observed in the following image:

You can clearly see that using hex numbers to display the information is a much shorter and more convenient form for a human programmer than looking at the binary numbers. Note that hex numbers are usually prepended with "0x".

Computer memory is treated as a sequence of cells. This means that we can use the starting address to retrieve the byte of information stored there. The following figure illustrates the principle:

Computer memory represented as a sequence of data cells (e.g. 01010101) with their respective memory addresses shown on top.

Let us perform a short experiment using gdb again: By adding 1, 2, 3, ... to the address of the string variable `str1`, we can proceed to the next cell until we reach the end of the memory we want to look at.

```
(gdb) x/1xb 0x7ffeefbff940
0x7ffeefbff940: 0x55
(gdb) x/1xb 0x7ffeefbff941
0x7ffeefbff941: 0x44
(gdb) x/1xb 0x7ffeefbff942
0x7ffeefbff942: 0x41
(gdb) x/1xb 0x7ffeefbff943
0x7ffeefbff943: 0x43
(gdb) x/1xb 0x7ffeefbff944
0x7ffeefbff944: 0x49
(gdb) x/1xb 0x7ffeefbff945
0x7ffeefbff945: 0x54
(gdb) x/1xb 0x7ffeefbff946
0x7ffeefbff946: 0x59
(gdb) x/1xb 0x7ffeefbff947
0x7ffeefbff947: 0x00
```

Note that the numbers above represent the string "UDACITY" again. Also note that once we exceed the end of the string, the memory cell has the value 0x00. This means that the experiment has shown that an offset of 1 in a hexadecimal address corresponds to an offset of 8 bits (or 1 byte) in computer memory.

Your Turn

Unfortunately, gdb will not work in Udacity Workspaces, but you can still try this exercise either in your local environment if you have g++ and gdb installed, or you can use OnlineGDB to follow along.

Try to locate the characters of "Udacity" using gdb in your local environment or in the online debugger.

Using GDB Locally

In order to use gdb locally, you will need to compile `main.cpp` with debugging symbols. This can be done with the `-g` option for g++:

```
g++ -g main.cpp
```

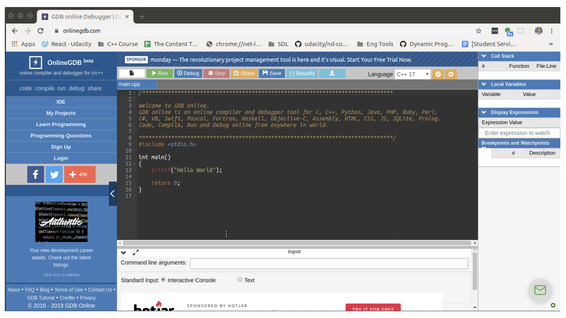
You can then run gdb on the output with:

```
gdb a.out
```

When gdb displays the line `Type <RET> for more, q to quit, c to continue without paging`, be sure to press the RETURN key to continue.

Using GDB Online

To use the OnlineGDB application, simply paste the code into the online editor and press the "Debug" button at the top of the screen.



You can find the gdb "cheat sheet" used in the videos above here.

Outro

https://youtu.be/9oESTYFVCV8