Sara Kazemi
CSCI 212
09/02/2019

Assignment 2

2.10 Programming Exercises

1. Testing out the echoDecHex program:

```
pi@kazemi:~/assign2 $ gcc -Wall -o echoDecHex echoDecHex.c
pi@kazemi:~/assign2 $ ./echoDecHex
Enter an unsigned decimal integer: 50
Enter a bit pattern in hexadecimal: FA
50 is stored as 0x00000032, and
0x000000fa represents the unsigned decimal integer 250
```

2. This program only supports up to eight digits of hexadecimal and each hexadecimal digit represents 4 bits, so, 8 * 4 = **32 bits**

```
pi@kazemi:~/assign2 $ ./echoDecHex
Enter an unsigned decimal integer: 2147483647
Enter a bit pattern in hexadecimal: 7FFFFFFF
2147483647 is stored as 0x7fffffff, and
0x7fffffff represents the unsigned decimal integer 2147483647
pi@kazemi:~/assign2 $ ./echoDecHex
Enter an unsigned decimal integer: 2147483648
Enter a bit pattern in hexadecimal: ffffffff
2147483648 is stored as 0x80000000, and
0xffffffff represents the unsigned decimal integer 4294967295
pi@kazemi:~/assign2 $ ./echoDecHex
Enter an unsigned decimal integer: 9
Enter a bit pattern in hexadecimal: fffffffff
9 is stored as 0x00000009, and
0xffffffff represents the unsigned decimal integer 4294967295
pi@kazemi:~/assign2 $ ./echoDecHex
Enter an unsigned decimal integer: 4294967295
Enter a bit pattern in hexadecimal: fffffffff
4294967295 is stored as 0xffffffff, and
0xffffffff represents the unsigned decimal integer 4294967295
```

3. After modifying the program to convert the hexadecimal integer to a signed decimal integer, the largest integer that can now be input while getting correct output is $7FFFFFFF_{16}$ or 2147483647_{10}. If a larger integer is input, overflow occurs and the output wraps back around. For instance, entering the next highest hexadecimal integer, 80000000_{16} yields the lowest 32-bit signed integer, -2147483648_{10}.

```
pi@kazemi:~/assign2 $ ./echoDecHex
Enter an unsigned decimal integer: 4294967295
Enter a bit pattern in hexadecimal: 80000000
4294967295 is stored as 0xffffffff, and
0x80000000 represents the signed deciminal integer -2147483648
```

4. The compiler allocates four bytes (represented by 8 hexadecimal digits) of memory for memory addresses.

```
pi@kazemi:~/assign2 $ ./echoDecHexAddr
Enter a decimal integer (0 to quit): 214
Enter a bit pattern in hexadecimal: FF
214 is stored as 0x000000d6 at 0x7ec60264, and
0x000000ff represents the decimal integer 255 stored at 0x7ec60260
```

2.12 Programming Exercise
Using the numbers you get, explain where the variables anInt and aFloatare stored in memory and what is stored in each location.

Tracing through the intAndFloat program using the gdb debugger, we can see that anInt and aFloat are stored in little endian order. This means that the least significant digit is stored in the lowest value memory address, with higher significant digits stored in sequentially higher value memory addresses.

For example, the integer **anInt** is stored in memory addresses 0x7efff44 (last or least significant 4 bits), 0x7efff45 (third 4 bits), 0x7efff46 (second 4 bits), and 07efff47 (first or most significant 4 bits).

Similarly, the floating point number **aFloat** is stored in memory addresses 0x7efff240 (last 4 bits), 0x7efff241 (third 4 bits), 0x7efff242 (second 4 bits), and 07efff43 (first 4 bits).

```
Reading symbols from ./intAndFloat...done.
(gdb) li
1       /* intAndFloat.c
2        * Using printf to display an integer and a float.
3        * 2017-09-29: Bob Plantz
4        */
5       #include <stdio.h>
6
7       int main(void)
8       {
9         int anInt = 19088743;
10        float aFloat = 19088.743;
(gdb) br 12
Breakpoint 1 at 0x1043c: file intAndFloat.c, line 12.
(gdb) run
Starting program: /home/pi/assign2/intAndFloat

Breakpoint 1, main () at intAndFloat.c:12
12        printf("The integer is %d and the float is %f\n", anInt, aFloat);
(gdb) print anInt
$1 = 19088743
(gdb) print aFloat
$2 = 19088.7422
(gdb) printf "anInt = %i and aFloat = %f\n", anInt, aFloat
anInt = 19088743 and aFloat = 19088.742188
(gdb) printf "anInt = %#010x and aFloat = %#010x\n", anInt, aFloat
anInt = 0x01234567 and aFloat = 0x00004a90
(gdb) print &anInt
$3 = (int *) 0x7efff244
(gdb) print &Float
No symbol "Float" in current context.
(gdb) print &aFloat
$4 = (float *) 0x7efff240
(gdb) x/1dw 0x7efff244
0x7efff244:     19088743
(gdb) x/1fw 0x7efff240
0x7efff240:     19088.7422
(gdb) x/1xw 0x7efff244
0x7efff244:     0x01234567
(gdb) x/1xb 0x7efff244
0x7efff244:     0x67
(gdb) x/4xb 0x7efff244
0x7efff244:     0x67    0x45    0x23    0x01
(gdb) x/1xw 0x7efff2440x7efff240
Invalid number "0x7efff2440x7efff240".
(gdb) x/1xw 0x7efff240
0x7efff240:     0x4695217c
(gdb) x/4xb 0x7efff240
0x7efff240:     0x7c    0x21    0x95    0x46
(gdb) cont
Continuing.
The integer is 19088743 and the float is 19088.742188
[Inferior 1 (process 19849) exited normally]
(gdb) q
```

2.14 Programming Exercise

Write a program that displays the memory address of each character in a string and its ASCII value (in hexadecimal).

```
#include <stdio.h>
int main()
{
    char *strPtr = "Hello, world!\n";
    printf("Address:  Contents\n");
    while(*strPtr != '\0') // NULL is sentinel character
    {
        printf("%p:  ", strPtr);
        printf("0x%02x\n", *strPtr);
        strPtr++; // next char
    }
    // print final, null sentinel character
    printf("%p:  ", strPtr);
    printf("0x%02x\n", *strPtr);
    return 0;
}
```

```
pi@kazemi:~/assign2 $ ./helloWorld
Address:  Contents
0x1055c:  0x48
0x1055d:  0x65
0x1055e:  0x6c
0x1055f:  0x6c
0x10560:  0x6f
0x10561:  0x2c
0x10562:  0x20
0x10563:  0x77
0x10564:  0x6f
0x10565:  0x72
0x10566:  0x6c
0x10567:  0x64
0x10568:  0x21
0x10569:  0x0a
0x1056a:  0x00
```

2.16 Programming Exercises

1. The following program reads in a user-entered string one byte at a time and echoes the entire string back

```
/* echoChar1.c
 * Echoes a string  entered by the user.
 */

#include <unistd.h>
#include <string.h>

int main(void)
{
  char aString[200];
  char *stringPtr = aString;

  write(STDOUT_FILENO, "Enter a text String: ",
        strlen("Enter a text String: ")); // prompt user
  read(STDIN_FILENO, stringPtr, 1);                  // one character of string
  while(*stringPtr != '\n')
  {
    stringPtr++;
    read(STDOUT_FILENO, stringPtr, 1); // get next character
  }

  // now echo for user
  write(STDOUT_FILENO, "You entered:\n", strlen("You entered:\n"));
  stringPtr = aString;
  do
  {
    write(STDOUT_FILENO, stringPtr, 1);
    stringPtr++;
  } while (*stringPtr != '\n');
  write(STDOUT_FILENO, stringPtr, 1);

  return 0;
}
```

```
Enter a text String: hello
You entered:
hello
```

2. The program is now modified so that the entered string gets modified so that the new line character is now replaced with the NUL character so the string is stored as a C-string and can be echoed with fprint:

```c
/* echoChar2.c
 * Echoes a string  entered by the user.
 */

#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main(void)
{
  char aString[200];
  char *stringPtr = aString;

  write(STDOUT_FILENO, "Enter a text String: ",
        strlen("Enter a text String: ")); // prompt user
  read(STDIN_FILENO, stringPtr, 1);                  // one character of string
  while(*stringPtr != '\n')
  {
    stringPtr++;
    read(STDOUT_FILENO, stringPtr, 1); // get next character
  }
  // now pointer is at /n
  *stringPtr = '\0'; // replace with NUL character

  // echo by printing string for user
  printf("You entered:\n%s\n", aString);
  return 0;
}
```

```
Enter a text String: hello world!
You entered:
hello world!
```

3. The last program splits the functionality into multiple files: writeStr.c defines a function that takes an argument, a pointer to the string to be displayed, and returns the number of characters actually displayed. readLn.c defines a function that takes two arguments, one that points to the char array where the characters are going to be stored, and one that specifies the maximum length of the char array. echoString3.c defines the main function that uses these utility functions. There are two header files defined for writeStr and readLn.

```
pi@kazemi:~/assign2 $ ./echoString
ENTER A STRING: hi
You entered:
hi
pi@kazemi:~/assign2 $ ./echoString
ENTER A STRING: dewgrthtjuliodgertyyuuk
You entered:
dewgrthtjuliodgertyyuuk
pi@kazemi:~/assign2 $ ./echoString
ENTER A STRING:
You entered:
```