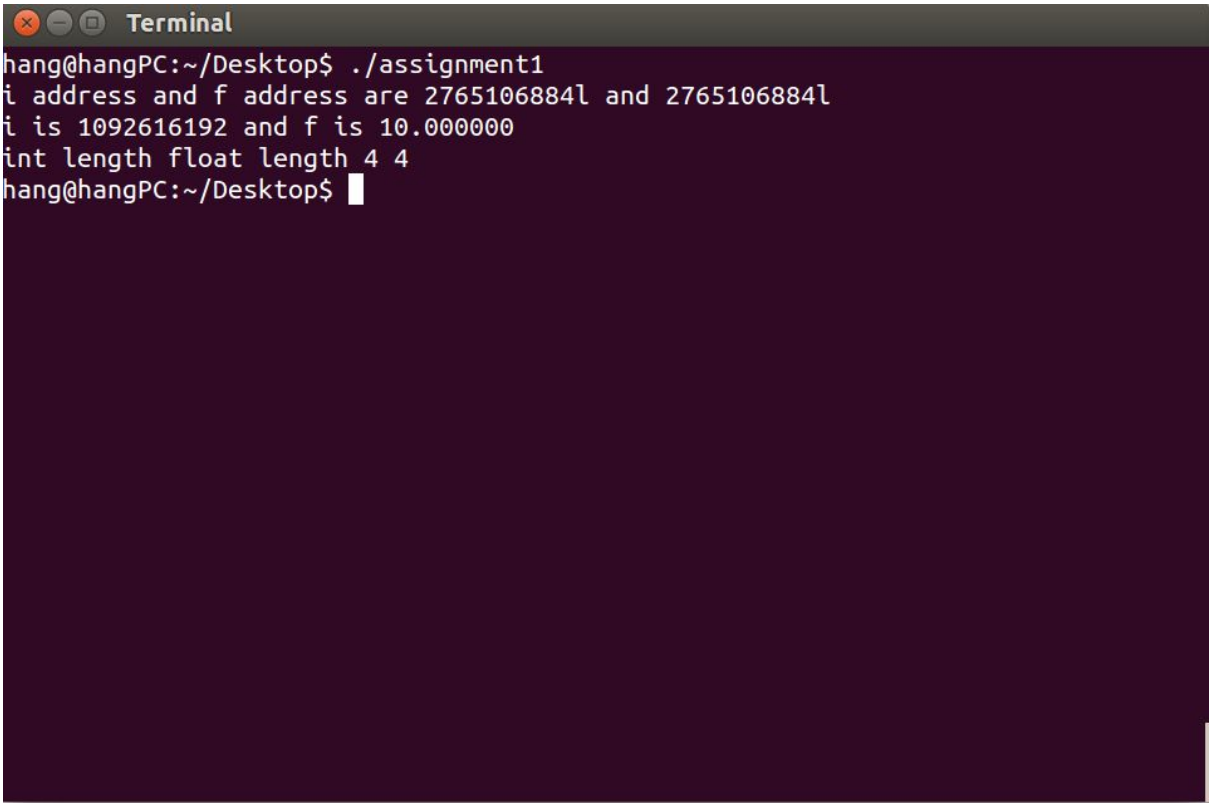
Name: Hang Ngo
Assignment #1: Simple example of aliasing and pointers

```
#include <stdio.h>

main()
{
   // Declare an integer i
   int i;
   // Declare a float pointer f
   float *f;

   // Initialize i
   i=1092616192;

   // Type case float pointer f to point to the address of
integer i
   f=(float *)&i;

   printf("i address and f address are %ul and %ul\n",&i,f);
   printf("i is %d and f is %f\n",i,*f);
   printf("int length float length %d %d\n",sizeof(int),
sizeof(float));
}
```

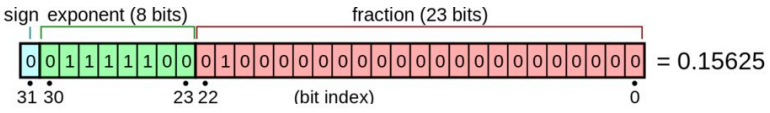
- The output of the code above is:



(The address may vary depending the computer)

- **Explanation:**

+ Since integer (32 bits) and float (32 bits) have different representations, using float pointer to point to the address of an integer would make the value of f and i different from each other.
+ Float number in C is represented using single-precision floating point format below:



The real value assumed by a given 32-bit *binary32* data with a given biased *sign*, exponent *e* (the 8-bit unsigned integer), and a *23-bit fraction* is

$$(-1)^{b_{31}} \times (1.b_{22}b_{21}\dots b_0)_2 \times 2^{(b_{30}b_{29}\dots b_{23})_2 - 127},$$

which in decimal yields

$$\text{value} = (-1)^{\text{sign}} \times \left(1 + \sum_{i=1}^{23} b_{23-i}2^{-i}\right) \times 2^{(e-127)}.$$

+ Converting i to binary, we have:

```
i=1092616192
→ Binary: 0 10000010 01000000000000000000000
```

+ Using the format above, we could calculate floating point value from binary representation of i:

Sign = 0
Exponent = $2^7+2^1 = 130$
Fraction = 2^{-2}
Value = $(-1)^0*2^{130-127}*(1+2^{-2}) = 10.0$
→ This matches with the output that says: `i is 1092616192 and f is 10.000000`
The address of i and f are the same because both f and i all point to the address of i.
Integer and float are both 4 bytes so the length of int and float are 4

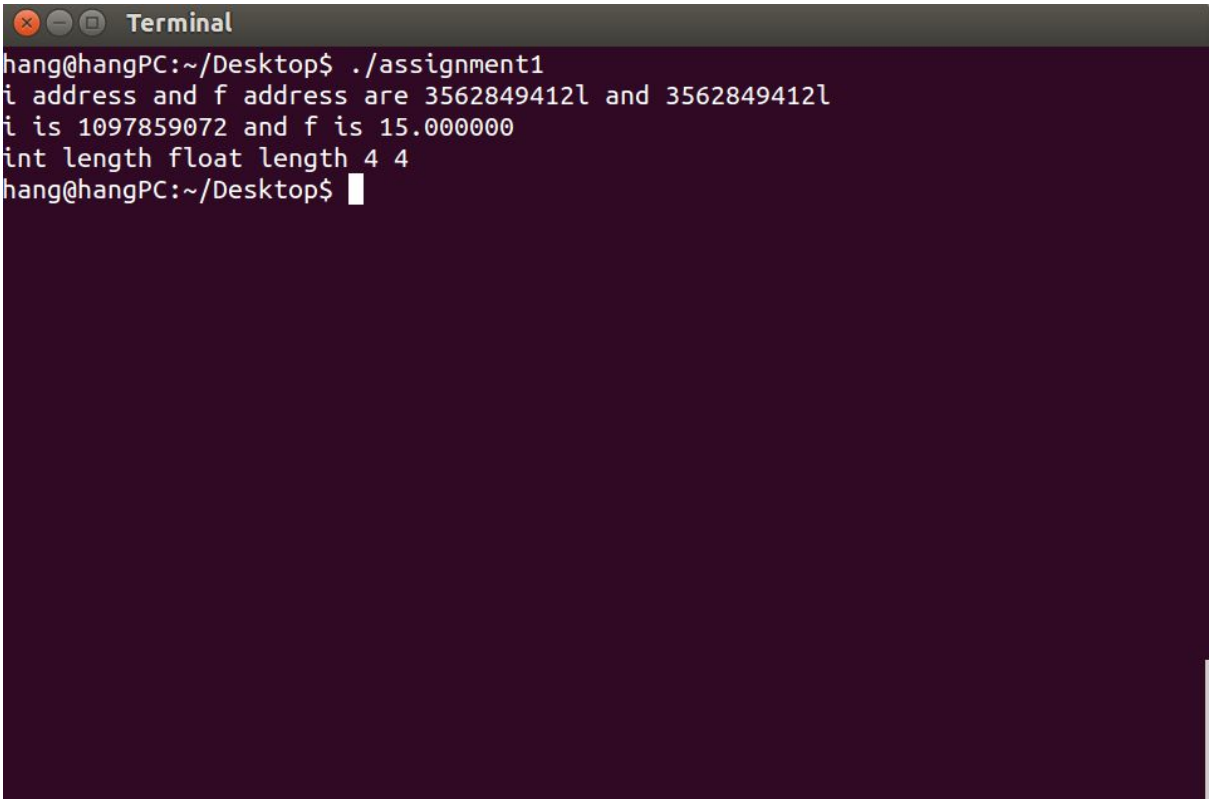
- To get a floating value of 15:

$15_{10} = 1111_2 = 1.111_2*2^3$
Sign = 0 → Sign = 0
$2^3 = 2^{130-127}$ → Exponent = 10000010
Fraction = 1110 0000 0000 0000 0000 000
→ f (15) will be represented as:
$0\ 1000010\ 1110\ 0000\ 0000\ 0000\ 0000\ 000_2 = 2^{20}+2^{21}+2^{22}+2^{24}+2^{30} = 1097859072$
→ We need to set value of i to 1097859072 so that we could get an equivalent floating point value of 15 for f.

- The new code would be:

```
#include <stdio.h>

main()
{
   // Declare an integer i
   int i;
   // Declare a float pointer f
   float *f;

   // Initialize i
   i=1097859072; // The value of i is changed to get f=15

   // Type case float pointer f to point to the address of
integer i
   f=(float *)&i;

   printf("i address and f address are %ul and %ul\n",&i,f);
   printf("i is %d and f is %f\n",i,*f);
   printf("int length float length %d %d\n",sizeof(int),
sizeof(float));
}
```

- Output when run with the end value of 15:



Resources:

https://en.wikipedia.org/wiki/Single-precision_floating-point_format