# CS2100 - Tutorial 2 - C: Pointers, Functions, Arrays and Structures

Week 4

## 1. Arrays

(a)

```c
int readArray(int arr[], int limit)
//Purpose: Read at most "limit" number of positive integers into arr
//         User can terminate by entering a negative integer
{
    int numOfItems = 0;
    int input;

    while (numOfItems < limit)
    {
        printf("Input: ");
        scanf("%d", &input);
        if (input < 0)
        {
            break;
        }
        arr[numOfItems++] = input;
    }

    return numOfItems;
}
```

(b).

```c
// Iterative version
// Other solutions possible
void reverseArray(int arr[], int size)
//Purpose: Reverse the items in array arr
{
    int temp;
    int left = 0;
    int right = size - 1;

    while (left < right)
    {
        temp = arr[left];
        arr[left++] = arr[right];
        arr[right--] = temp;
    }
}
```

(c)(i). Similar to iterative soln (swap extreme ends, move further in the middle until left ≥ right)

(ii). Need to pass in the arguments left = 0, right = size - 1

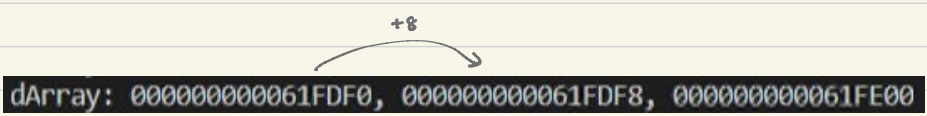
## 2. Arrays and memory

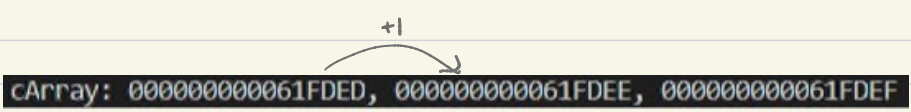
```c
#include <stdio.h>

int main()
{
    int iArray[3];
    double dArray[3];
    char cArray[3];

    printf("iArray: %p, %p, %p\n", &iArray[0], &iArray[1], &iArray[2]);
    printf("dArray: %p, %p, %p\n", &dArray[0], &dArray[1], &dArray[2]);
    printf("cArray: %p, %p, %p\n", &cArray[0], &cArray[1], &cArray[2]);
}
```

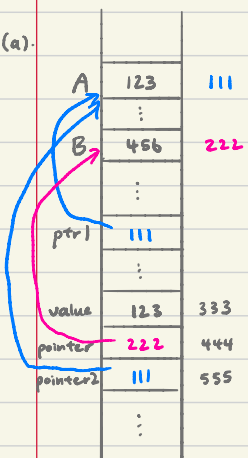

$(4)_{16} = (100)_2$



3. Parameter Passing

(a).




(b)(i). No. Pass-by value ⇒ different memory address

(ii). Yes. *pointer = <newValue>

Dereference

(iii). Yes. *pointer2 = <newValue>

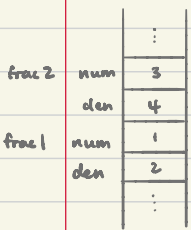
(iv). No. Different memory address.

To change: use parameter int **ptrptr, then pass in &ptr1 as argument

4. **Structures**

(a).

```
In main(), frac1 is at 0x61fe18, frac2 is at 0x61fe10
The address of frac1.num is 0x61fe18, frac1.den is 0x61fe1c
```




(b).

```
8   void swapFractionByValue( struct Fraction a, struct Fraction b)
9   //Purpose: To swap the content of a and b
10  {
11      int tempNum = a.num;
12      int tempDen = a.den;
13
14      a.num = b.num;
15      a.den = b.den;
16
17      b.num = tempNum;
18      b.den = tempDen;
19
20      printf("By Value parameter addressses:\n");
21      printf("a.num: 0x%x, a.den: 0x%x\n", &a.num, &a.den);
22      printf("b.num: 0x%x, b.den: 0x%x\n", &b.num, &b.den);
23  }
```

(c).

```
At the start: fract 1 is (1 / 2), frac2 is (3 / 4)
By Value parameter addressses:
a.num: 0x61fdf0, a.den: 0x61fdf4
b.num: 0x61fdf8, b.den: 0x61fdfc
After swapFractionByValue(): fract 1 is (1 / 2), frac2 is (3 / 4)
```

(d).

```
25  void swapFractionByAddress( struct Fraction* a, struct Fraction* b)
26  //Purpose: To swap the content of a and b
27  {
28      int tempNum = a->num;
29      int tempDen = a->den;
30
31      a->num = b->num;
32      a->den = b->den;
33
34      b->num = tempNum;
35      b->den = tempDen;
36
37      printf("By Address parameter addressses:\n");
38      printf("a.num: 0x%x, a.den: 0x%x\n", &(a->num), &(a->den));
39      printf("b.num: 0x%x, b.den: 0x%x\n", &(b->num), &(b->den));
40  }
```

(e).

```
By Address parameter addressses:
a.num: 0x61fe18, a.den: 0x61fe1c
b.num: 0x61fe10, b.den: 0x61fe14
After swapFractionByAddress(): fract 1 is (3 / 4), frac2 is (1 / 2)
```

matching memory addresses

```
In main(), frac1 is at 0x61fe18, frac2 is at 0x61fe10
The address of frac1.num is 0x61fe18, frac1.den is 0x61fe1c
```