APS 105 Lecture 16 Notes

Last lecture: Scope and Introduction to Arrays

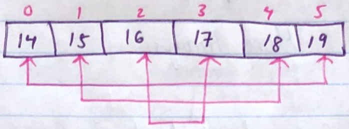
Today: More Arrays

Recap: General form of declaration

```
<type> <identifier> [<size>];          uninitialized

<type> <identifiers[] = { <value list >};     initialized
```

* Index starts by 0
* Size is fixed throughout the program
* Size should be known at compile-time
* Don't need to give size if you're initializing
* If you give size and initialize
  - There were no enough values, rest is set to Ø
    ```
    int list[6] = {1, 2, 3};
    it is similar to
          = {1, 2, 3, 0, 0, 0};
    ```
  - There were more values than the size
    ```
    int list[3] = {1, 2, 3, 4};
                ↓
          COMPILE-ERROR
    ```

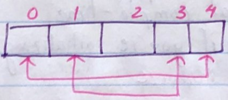
Example: Given an array of any size, write a program That reverses the order of its elements



Steps:

		lower		higher
① Reverse	element	0	with	5
② "	"	1	with	4
③ "	"	2	with	3
		3		2 → DON'T REVERSE

What if # of elements is odd



		lower		higher
① Reverse	element	0	with	4
② "	"	1	with	3
		2		2 → DON'T REVERSE

I need a loop that iterates over lower from 0 to 1 to 2 --- & over higher from size-1 to size-2 ------ until lower >= higher OR as long as lower < higher

```
int main(){
        const int Size = 10;
        double list [Size];
        for (int ind = 0; ind < Size; ind){
                list [ind] = ind+1;
        }              // list initialized to {1,2,3,4,5,6,7,8,9,10};

      int temp;
```

comma separates statements in initialization & alteration

```
      for (int low = 0, high = Size-1; low < high; low++,
                                                   high--){
              temp = list[low];
              list[low] = list[high];
              list[high] = temp;
            swap(&list[low], &list[high]);
      }
   return 0;
}
```

Why not use swap function?

What if I want to pass an array to a function?

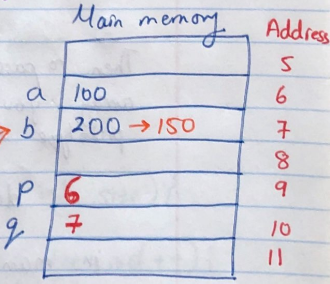
Recall pointers:

```
        int a=100, b= 200;
        int *p, *q;

        p = &a; ①
        q= &b; ②

        *q = *p + 50;
```

address of

$\underbrace{*q}_{b} = \underbrace{*p}_{a} + 50;$

variable p is pointing to

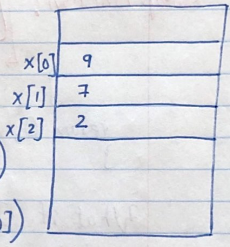


In C, the identifier of array is having special meaning $\longrightarrow$ it is a POINTER to the 1st element in the array

Consider

```
int x[] = {9, 7, 2};
```

We want to make pointers to each element



(*) The first element is already having a pointer to it, i.e. x is that pointer

So, x is same as &(x[0])

*x is same as *(&x[0])
x[0]

Then to pass an array to a function, when you pass the array, you pass the pointer to the 1st element. Function prototype looks like

```
double f(int list[]);
```

in main

```
int x[] = {9, 2, 7};
result = f(x)
```

put identifier only when passing array

Since we're passing a pointer to the 1st element in an array, the function doesn't know the size of the array! Hence if you require size of array in function, you need to pass it.

Create a function that sums elements of an array.

```
        //Function Prototype
        int   sumfunc ( int     [],  int   );
        //Main function
        int main (void){

                 int x [3] = {9, 7, 2};

                int result = sumfunc (x, 3);

            return 0;
        }
        //Function Implementation
        int sumfunc ( int  list [] , int  size){
              int  sum = 0;
              for (int ind=0; ind < size; ind++){
                         sum += list[i];
              }
              return sum;

        }
```

Another example: swap 2 elements of an array

```
void swap(double list [], int i, int j){
        double temp = list[i];
        list[i] = list [j]
        list [j] = temp;
}
```

Since we pass pointer to function, then we're actually changing elements of the array directly in the memory. When swap returns the original values of the array has been swapped