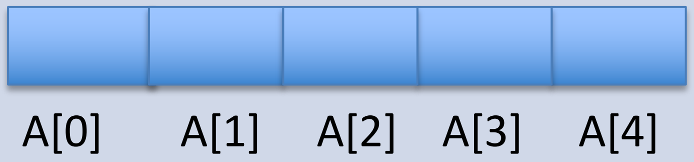
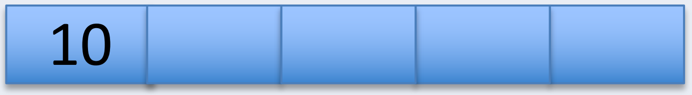
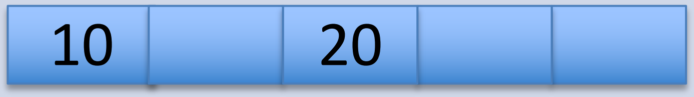
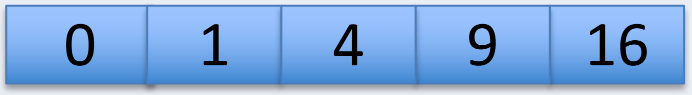
COMP10002 Workshop Week 5

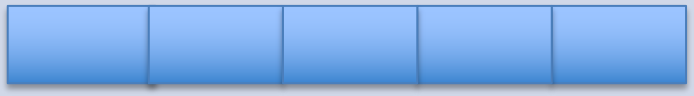
1	Arrays: working with arrays, searching, sorting, 7.6, 7.7
2	Walk-through a sample quiz?
3	Lab: Implement 7.6, 7.7, 7.8, 7.9, 7.10, 7.11
Lab Works	Discuss Exercises 7.6 and 7.7; and through them Confirm that you understand arrays and the operations that manipulate them; then Design and implement solutions to as many as you can get through of Exercises 7.3, 7.4, 7.6, 7.7, 7.8, 7.9, 7.10, and 7.11; The more, the better! (You can do more of them in the Week 6 Workshops).
NOTE	

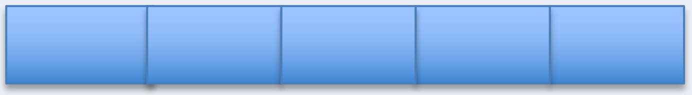
array: for storing and processing a sequence of values

1	The statement **`int A[5];`**	
2	is equivalent to declaring 5 variables, each is of data type **`int`**:	
3	**`A[0] = 10;`**	

4	**`i= 2;`**
	`A[i]= 20;`
		

5	**`for (i=0; i<5; i++) {`**
	    `A[i]= i*i;`
	`}`
		

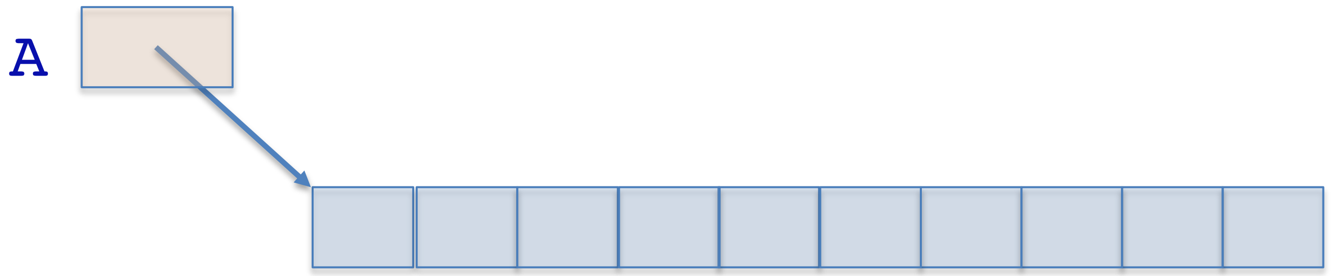
6	**`for (i=1; i<5; i++) {`**
	    `A[i]= A[i-1];`
	`}`
		

7	**`for (i=0; i<5; i++) {`**
	    `scanf ( “%d”, &A[i] );`
	`}`
		

Array name is a pointer

```
int A[10];
```

- the name `A` is a pointer to the first of the 10 int cells,
- after that, the system won't remember the value `10`,
- so, we need to remember that value ourself, and do not fall to a trap of wrongly using `A[10]` or `A[11]` — a very serious *Seg Fault* error!



Declaring Array

Conventional declaration:

```
#define MAX_N 1000
int A[MAX_N]= {10, 20, 30};
int     n= 3;
```

An array is defined by 3 "objects":

- array name `A`
- "buddy" variable `n`, indicating the number of elements currently in use
- a constant `MAX_N`, indicating the maximal capacity of the array

Then, remember:

- Always keep `n` correct
- Always make sure that `n <= MAX_N`

Example: input values for array

The Task: read and store (in an array) *at most* 10000 integer for later processing.

```
#define MAX_N 10000

// declares array
int A[MAX_N];
int    n= 0;

// reads array
int i, x;
for ( ; scanf(“%d”, &x) ==1 ; ) {
   ???
}
```

Working with Arrays:
normally use a function for a major array operation!

Passing array A to a function? We can just pass:

- A: the array name (= pointer to the 1st cell), and
- n: the actual number of elements in the array A.

The pair [ ] indicates that **A** is an array, not specifying its size.

Example:

```
void square_it(int A[], int n) {
    int i;
    for (i=0; i<n; i++) {
        A[i] *= A[i];
    }
}
```

The second parameter supplies the number of elements in the array.

```
...main...
int arr[MAX_N]= {1,2,3,4,5}, num=5;     // now arr[3] has value 3
square_it(arr, num);                    // now arr[3] has value 9
```

Searching for a specific element

The Task:

Input: Given an array of `n` int, and an int `x`.

Output: an index `i` so that `A[i]==x` .

?

- What the function prototype?
- What are *special cases*, what should we do then?

```
??? search( ??? )
```

// help!

Searching for a specific element

The Task:

Input: Given an array of `n` int, and an int `x`.

Output: an index `i` so that `A[i]==x` .

```
// returns index i such as A[i]==x
int search(int A[], int n, int x) {
   int i;
   for(i=0; i<n; i++) {
      if ( A[i] ??? ) {
          ???; //
      }
   }
}
```

Note: Sometimes, we might need to search for something more complicated than `A[i]==x` ...

?

Searching for *a* minimal element in array

The Task:

Input: Given an array of `n` int, and an int `x`.

Output: an index `i` so that `A[i]` is the smallest .

?

- What are special cases, what should we do then?

```
// returns the index of the minimal element in A[]
int min_index(int A[], int n) {
  int i, min_i;
  for (i=0; i<n; i++) {
    // do something with A[i]

  }
  return min_i;
}
```

Sorting

The Task:

Input: Given an array of n elements.

Output: The same input array, but the its elements are re-arranged in some *order* (such as *increasing*)

Example: sort array in non-decreasing order)

Input : `int A[5]= {7,2,5,5,6}, n= 5;`

Output: `A[ ]` becomes `{2,5,5,6,7}`

Algorithms:

- Insertion Sort
- Quick Sort
- …

7.6

An alternative sorting algorithm is selection sort. It goes like this: scan the array to determine the location of the largest element, and swap it into the last position. Then repeat the process, concentrating at each stage on the elements that have not yet been swapped into their final position.

Write a function

```
   void selection_sort (int A[], int n)
```

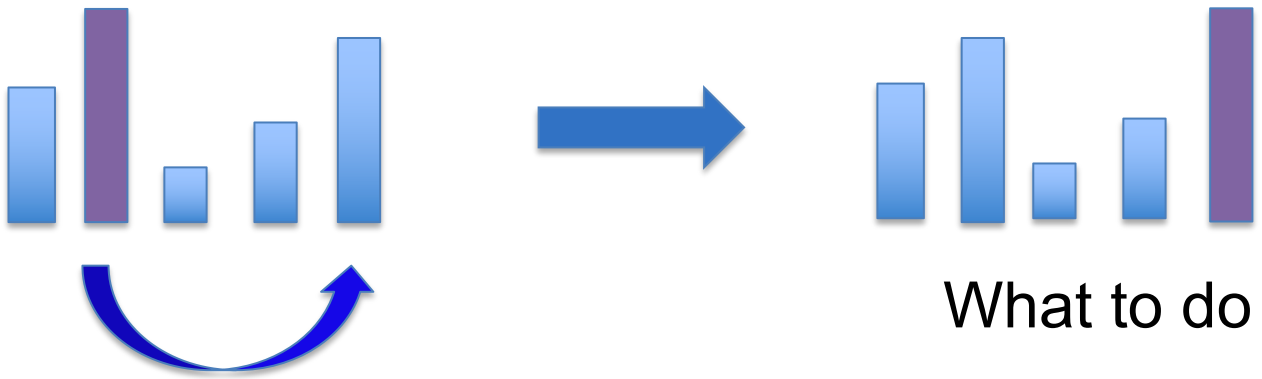
that orders the `n` *elements in array* `A`.

7.6: understanding

<u>Selection sort</u>: scan the array to determine the location of the largest element, and swap it into the last position. Then repeat the process …

So in the first round:

- *look at all elements from* `A[0]` *to* `A[n-1]`,
- *determine the location (?) of the largest element,*
- *swap it into the last position, ie.* `A[n-1]`.



What to do next?

7.6: recursive version (DIY: iterative (non-recursive) version)

Selection sort: scan the array to determine the location of the largest element, and swap it into the last position. Then repeat the process …

```
// sorts array A[] in increasing order
// using selection sort, in recursive manner
void rec_sel_sort(int a[], int n) {
  // base case

  // general case

}
```

Design together: Ex. 7.7 (grok W05)

7.7: Write a function that takes as arguments an integer array `A` and an integer `n` that indicates how many elements of `A` may be accessed, and returns the value of the integer in `A` that appears most frequently. If there is more than one value in `A` of that maximum frequency, then the smallest such value should be returned. The array `A` may not be modified.

Note: This is a grok's task, not a quiz's task, so you also need to write a main program that read in an array and test your function.

Now please turn on your grok if haven't done so…

This `week5.pdf` can be downloaded from
`github.com/anhvir/c102`

If you want it, please download right now. It will be removed in about 5 minutes.
OR, you can also email Anh and ask for the slides later.

Lab: Group+Individual Works: Ex 7.7-7.9 and others.
OR, in the first 15 min, stay with Anh in the main room to go through one sample quiz

Write a *function* that

7.2: [W06] sorts an array in deceasing order

7.3: [W06] sorts an array and removes duplicates

7.4: [W06] computes frequency of each value in an array

7.6: [W06] performs selection sort

7.7: **[W05]** returns the value that appears most frequently in an array `A` of integers. On tie, returns the smallest one. The array `A` may not be modified.

7.8: [W5X] returns the *k*-smallest of `int A[]`, not modifying `A`

7.9: [W05] returns the number of *ascending runs* in `int A[]`. For example, array **`{10,13,16,18,15,22,21}`** has 3 runs.

7.10: [W5X] returns the number of inversions in `int A[]`. For example, the above array has 3 inversions: 2 caused by **`15`**, and 1 by **`21`**.

*** and all other exercises from `W05, W05X, W06, W06X`

```
NOTE: In grok, when you write a function, you also need
to write a simple main function for testing your
function.
In the exam/test you only write the main function only
asked for!
```