

COMP20005 Workshop Week 8

Preparation:

- open `grok` for exercises
- Programming: open & use `jEdit`, and `minGW/Terminal` if possible

1 Discussion 1: Strings

Ex. 7.15

Other exercises: `Ex 7.12, 7.14`

3 Discussion 2: Working with arrays

some examples

Ex. 7.7 [`grok W7X`]: most frequent element

Ex. 7.8 [`grok W7X`]: k-th smallest element

other exercises: in `grok W7` and `W7X`

LAB


- Assignment: Q&A
- **Work on Assignment**, AND/OR (if `ass1` totally done):
- implement array exercises [individual or group]

Discussion 1: Strings and <string.h>

A string is a pointer to char which points to a valid memory location. The string will then be the sequence of chars from that location until the first-seen zero character '`\0`'.


Examples

```
char *s2="1234";           // s2 points to 1234\0
                           // note that memory is constant
                           // and cannot be changed
```



The diagram shows a pointer variable `s2` with an arrow pointing to the first cell of a memory array. The array contains five cells: '1', '2', '3', '4', and '\0'.

```
char A[10]= "Hello";      //
```



The diagram shows a memory array `A` of size 10. The first six cells contain 'H', 'e', 'l', 'l', 'o', and '\0'. The remaining four cells are empty.

```
s2= A; // now s2 is string and it can hold 9 chars as maximal
printf("string A= %s   s2= %s\n", A, s2);
```

Important functions:

```
strlen(s)           strcpy(s1, s2)       atoi(s2)
```

Strings: array and pointer notations



Array Notation

```
int n= strlen(s);
int i;
for (i=0; i<n; i++) {
    // process character s[i]
    printf("processing %c\n", s[i]);
}
```

OR

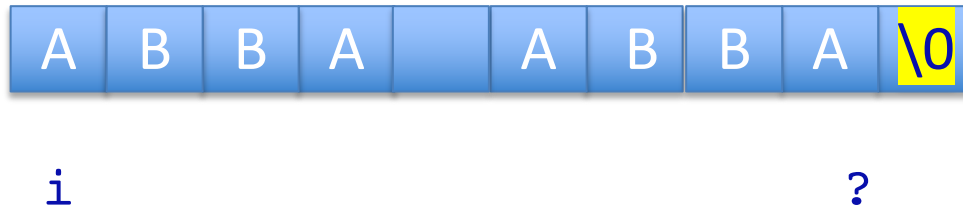
```
int i;
for (i=0; s[i] != '\0'; i++) {
    // process character s[i]
    printf("processing %c\n", s[i]);
}
```

Pointer notation

```
char *p;
for (p=s; *p != '\0'; p++) {
    // process character *p
    printf("processing %c\n", *p);
}
```

Strings: Exercise 7.12 using array notation

Write a function `int is_palindrome(char*)` that returns true if its argument string is a palindrome, that is, reads exactly the same forwards as well as backwards; and false if it is not a palindrome. For example, "rats live on no evil star" is a palindrome according to this definition, while "A man, a plan, a canal, Panama!" is not. (But note that the second one is a palindrome according to a broader definition that allows for case, whitespace characters, and punctuation characters to vary.)

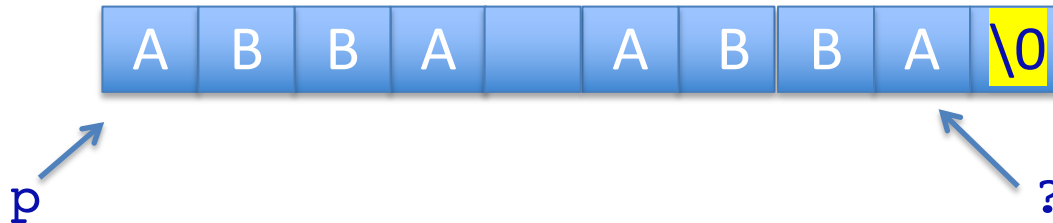


```
// returns 1 if s is a palindrome, 0 otherwise
int is_palindrome(char *s){

}
```

Strings: Exercise 7.12 using pointer notation

Write a function `int is_palindrome(char*)` that returns true if its argument string is a palindrome, that is, reads exactly the same forwards as well as backwards; and false if it is not a palindrome. For example, "rats live on no evil star" is a palindrome according to this definition, while "A man, a plan, a canal, Panama!" is not. (But note that the second one is a palindrome according to a broader definition that allows for case, whitespace characters, and punctuation characters to vary.)



```
// returns 1 if s is a palindrome, 0 otherwise
int is_palindrome(char *s){

}
```

Discussion 2: Working with Arrays

Task: Performing some job with one array (or a number of arrays)

Strategies:

- Define the task clearly as a function
- When implementing function body: if the task is complicated, break it into smaller steps until seeing it can be implemented in C
- For a step, form new function for if needed, ie. if:
 - the step is likely complicated or long
 - a *similar* step will be used somewhere else

With each function:

- each input array: can it be empty? can be be modified?
- the output: index, value or just void? Can it be NOT_FOUND and if yes, how to #define this value?

Remember: The basic way to work with array **A** of **n** elements is *simple*:

```
for (i=0; i<n; i++) {  
    // process element A[i]  
}
```

Example 1: (un-usual) index of max

Write a function that takes as arguments an integer array A with n elements, and returns the index of a largest element of A. The array A may be empty.

```
// returns the index of min element of array  
? index_of_max( ? ) {  
}
```

Example 2: Ex 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 to sort an integer array using selection sort.

First, write the header:

```
? selection_sort (?)
```


Selection Sort: understanding

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

So at first:

- *scan all un-sorted elements from position 0 to position $n-1$ to determine the position of the largest element,*
- *swap it into the last position, ie. position $n-1$.*

After that:

- *Job done for the last position $A[n-1]$*
- *The elements in array $A[0..n-2]$ are still unsorted*



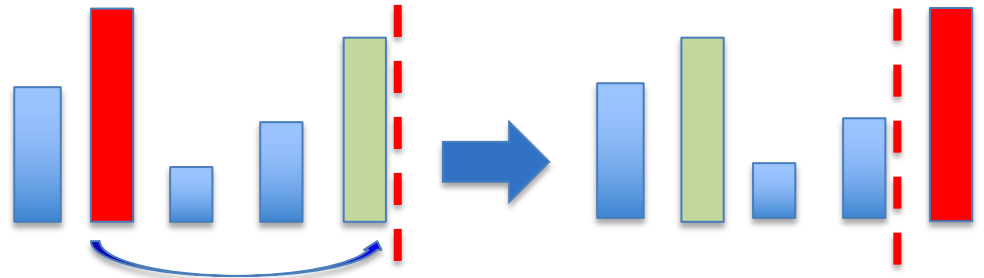
What to do next?

Are we ready to write a function?

Ex 7.6: Write A Function

?: consider $A[0 \dots n-1]$

- *determine position of the largest*
- *swap with the last, ie. $A[n-1]$, then ...*



```
// sort array A in ascending order using selection sort
void sel_sort(int A[], int n) {

}
```

Example 3: Ex 7.7

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

```
// returns the value that appears most frequently in array
? most_frequent( ? ) {
}
```

Example 4: Ex 7.8

7.8: Write a function that takes as arguments an integer array A with n elements, and an integer k; and returns the k smallest integer in A. That is, the value returned should be the one that would move into A[k] if array A were to be sorted. The array A may not be modified. Be sure that you handle duplicates correctly.

```
// returns the value that is k-th smallest, k is ???  
? kth_smallest( ? ) {  
}
```

Looking Ahead

Assignment 1:

- due at 6pm this Friday
- make sure to test your code with dimefox by clicking
LMS → Assignments → Assignment 1 → dimefox
- read the *verification report* carefully, make sure to see **NO compilation message**, **NO** sign **+** or **-** at the start of any line
- final LMS submit Thursday night, enjoy your Friday afternoon.

Quiz 2: 1pm on Friday 7 May. See Practice Quiz.

Before next workshop: do sample quiz for a few time, take note and bring questions to the workshop.

Programming is fun: Check each of the following exercises, make sure you know how to do it, and implement if not yet done and time permitted.

- **ARRAYS:** *Arrays, and array manipulation, are very important!*
 - Searching: Exercises 7.7, 7.8, 7.15
 - Sorting: 7.4 (W07) , 7.2, 7.3, 7.6 (W08)
 - Others: typedef & 2D array – ex 7.11 (W8X)
- **STRINGS:** *are just special arrays*
 - Exercises 7.12, 7.14, 7.15, 7.16 (W8X)

LAB

Do your assignment (highest priority), remember to:

- do testing on dimefox,
- check with marking rubric,
- submit on LMS.

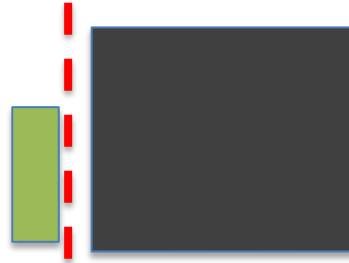
If ass1 completely done, implement exercises from:

- ARRAYS:
 - Searching: Exercises 7.7, 7.8, 7.9,, 7.10, 7.5 (W7X)
 - Sorting: 7.4 (W07) , 7.2, 7.3, 7.6 (W08)
 - Others: typedef & 2D array – ex 7.11 (W8X)
- STRINGS:
 - Exercises 7.12, 7.14, 7.15, 7.16 (W8X)

Additional Materials for self-study

See next pages

Insertion Sort: understanding



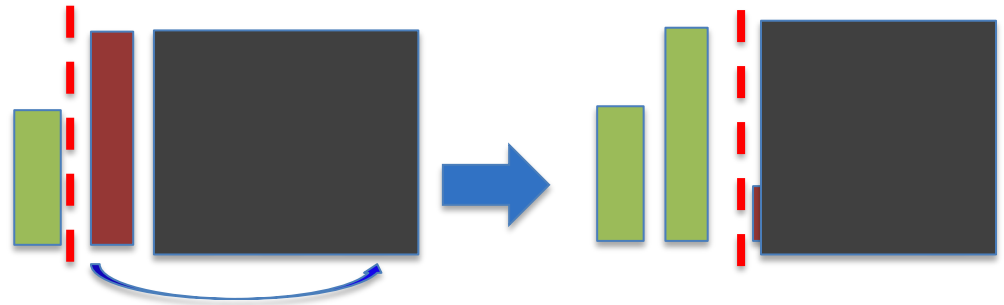
In this algorithm, we process element-by-element, *keeping all processed elements in sorted order*.

At first we note that:

- Nothing needs to do when processing at $A[0]$

Next step= ?

Insertion Sort: understanding



Then:

- Consider element $A[1]$
- Think of **the left** of $A[1]$ as an array $A[0..0]$
- Note that that left array is sorted
- Now, insert $A[1]$ to the left (and hence expand the left) so that the new left array $A[0..1]$ is sorted

Next step= ?

Insertion Sort: understanding (n=5)

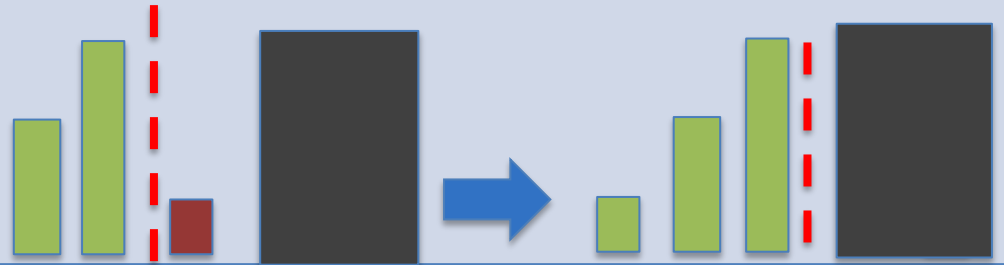
Round 1: consider $A[1]$

- $A[0..0]$ is sorted
- insert $A[1]$ to the left so that $A[0..1]$ is sorted



Round 2: consider $A[2]$

- $A[0..1]$ is sorted
- insert $A[2]$ to the left so that $A[0..2]$ is sorted



Round i: consider $A[4]$

- $A[0..4]$ is sorted
- insert $A[i]$ to the left so that $A[0..4]$ is sorted

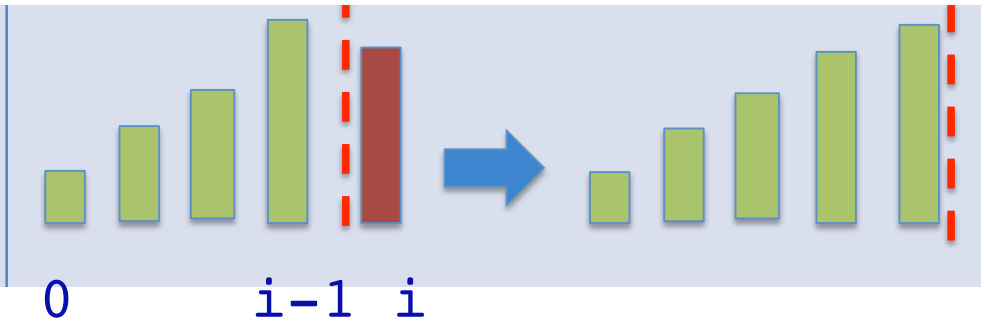


Insertion Sort: Algorithm

Process $A[1]$ in round 1, ..., $A[n-1]$ in round $n-1$

Round i : consider $A[i]$

- $A[0..i-1]$ is sorted
- insert $A[i]$ to the left so that $A[0..i]$ is sorted



How to: insert $A[i]$ to the left?

```
void ins_sort(int A[], int n) {
```

Ex 7.2

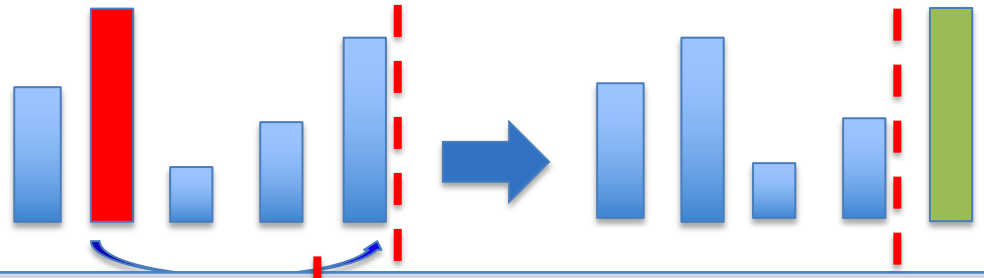
Modify so that the array of values is sorted into decreasing order.

```
/* insertion sort: sorting elements A[0] to A[n-1]
    into non-decreasing order */
/* assume that A[0] to A[n-1] have valid values */
1  for (i= 1; i<n ; i++) {
2      /* swap A[i] left into correct position */
3      for (j= i-1; j>=0 && A[j+1]<A[j]; j--) {
4          /* not there yet */
5          int_swap( &A[j] , &A[j+1] );
6      }
7  }
/* and that's all there is to it */
```

Selection Sort: understanding (n=5)

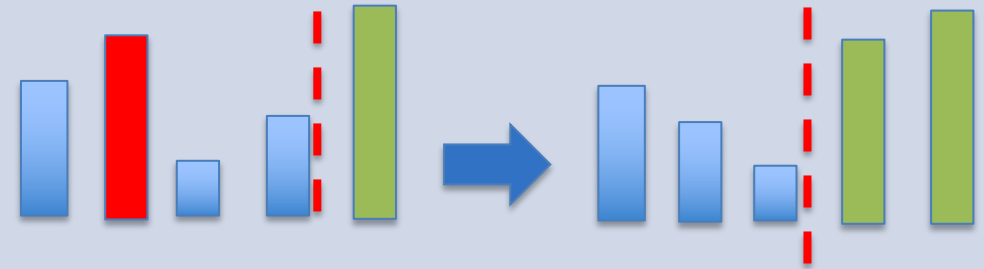
Round 1: consider $A[0 \dots 4]$

- *determine position of the largest*
- *swap with the last, ie. $A[4]$*



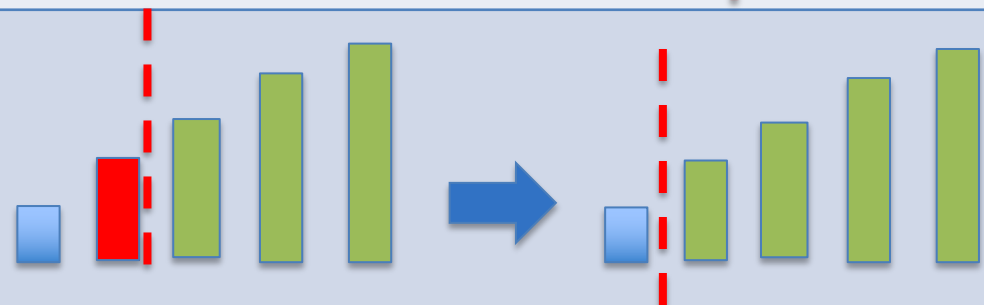
Round 2: consider $A[0 \dots 3]$

- *determine position of the largest*
- *swap with the last, ie. $A[3]$*



Round 4: consider $A[0 \dots 1]$

- *determine position of the largest*
- *swap with the last, ie. $A[1]$*

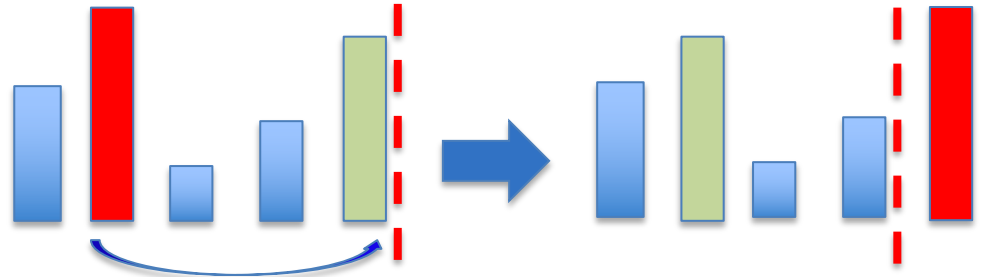


Ex 7.6: Write non-recursive function

First round: examining $A[0..n-1]$, last round: examining $A[0..1]$

Round ? : consider $A[0..i]$

- *determine position of the largest*
 - *swap with the last, ie.*
- `A[i]`



```
void sel_sort(int A[], int n) {
```

}