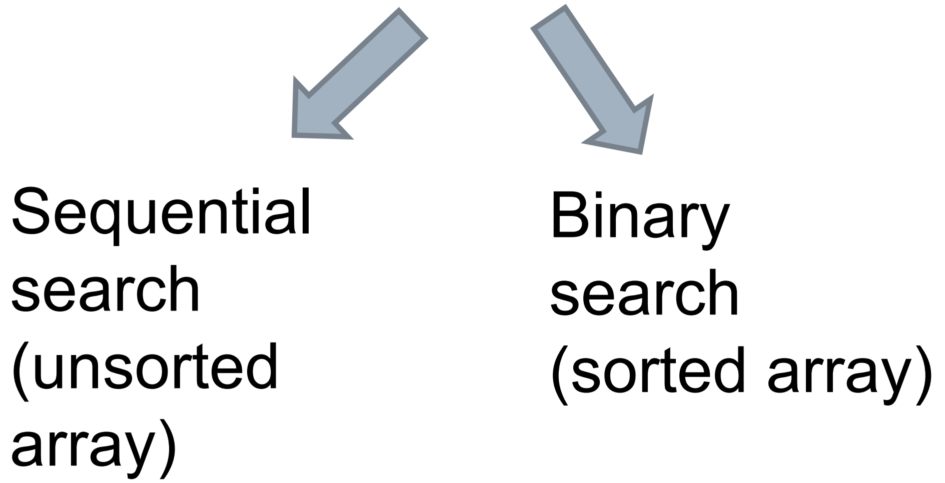
Computer Engineering 12
Project 2

Xiang Li
xli8@scu.edu

A Question?

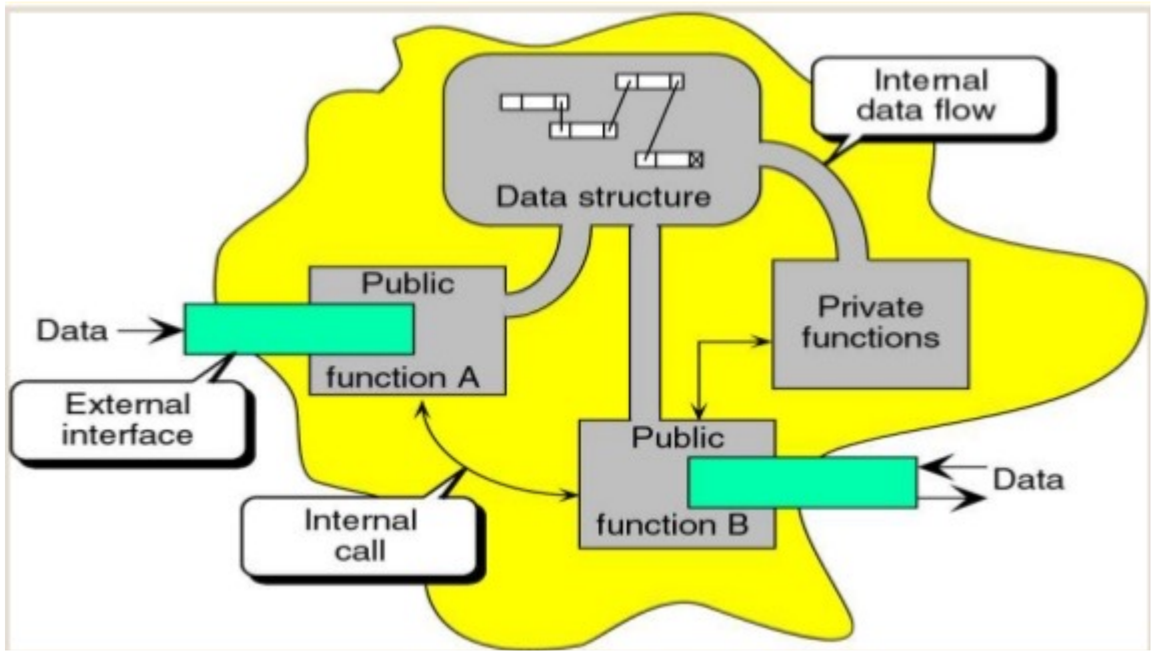
- Assume your boss (or customer) doesn't care about the underlying data structure at all. It does not matter whether you use sequential search or binary search.
- He/she only needs a function that can help him/her identify whether a specific element is in your array, and if yes, where it is.
- char *findElement(?, char *elt)


Sequential search (unsorted array)
Binary search (sorted array)

Abstract Data Type

Encapsulate the data and the operation on the data, and then hide them from the user.

1. Declaration of data
2. Declaration of operations
3. Encapsulation of data and operations.



Abstract Data Type

- An abstract data type is a data type whose implementation has been hidden or abstracted away
- We cannot see the implementation. Instead we have to use a set of functions called the “interface”
- We say the implementation is kept private and the interface is public

Data Structure & ADT

- Data Structure:
  - Things to care - how to organize data in memory or disk
  - Examples: array; linked list; tree; graph
- ADT:
  - Things to care
    - what data
    - what operations can be done on these data
  - Things not to care
    - how to organize data in memory or disk (It can be done through any data structure)

- ***In the following weeks, we'll implement the same ADT by organizing data in different ways.***

LAB 2

A New ADT: Set

A set: an unordered collection of distinct elements.

Order doesn't matter

No repetition

Example: Students registered for a class is an example of a set. We care who's in it, not really the order in which they signed up.

We are going to build a SET ADT in Lab 2

Who Needs a Set

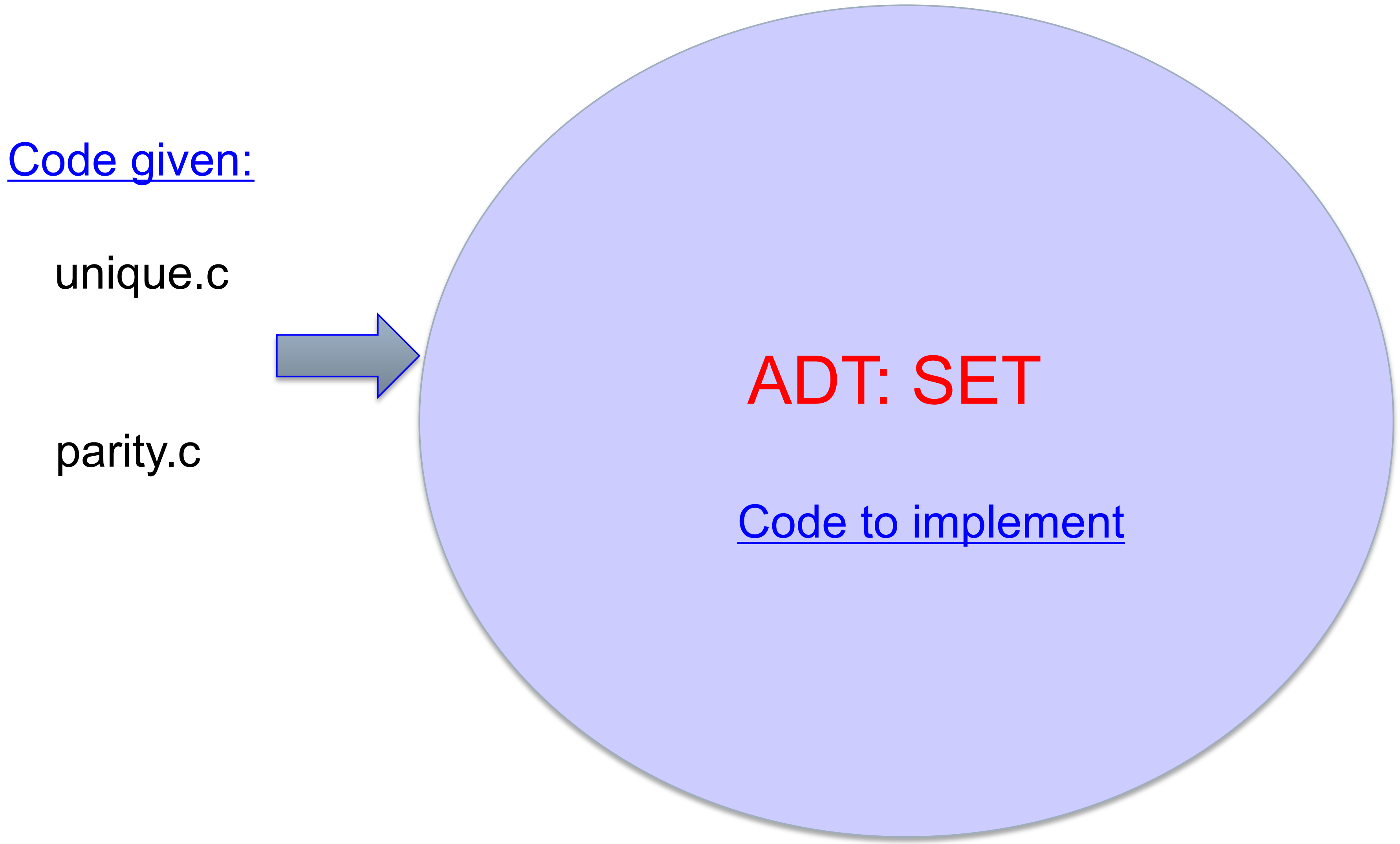
There are two example applications that need a set

1. Application 1 (unique.c) with two input files and the second one is optional. Read in all the **distinct** words in a given file and insert these words in a set.
   1) Print out the total amount of unique words;
   2) If the second file is given then all words in the second file are deleted from the set; and the count is printed;
   3) Print out each of these unique words.

2. Application 2: use a set to maintain a collection of words that occur an odd number of times in a given file. (parity.c)

Both of these two applications need a set to store the words occurring in a given file!

ADT: SET - An Illustration

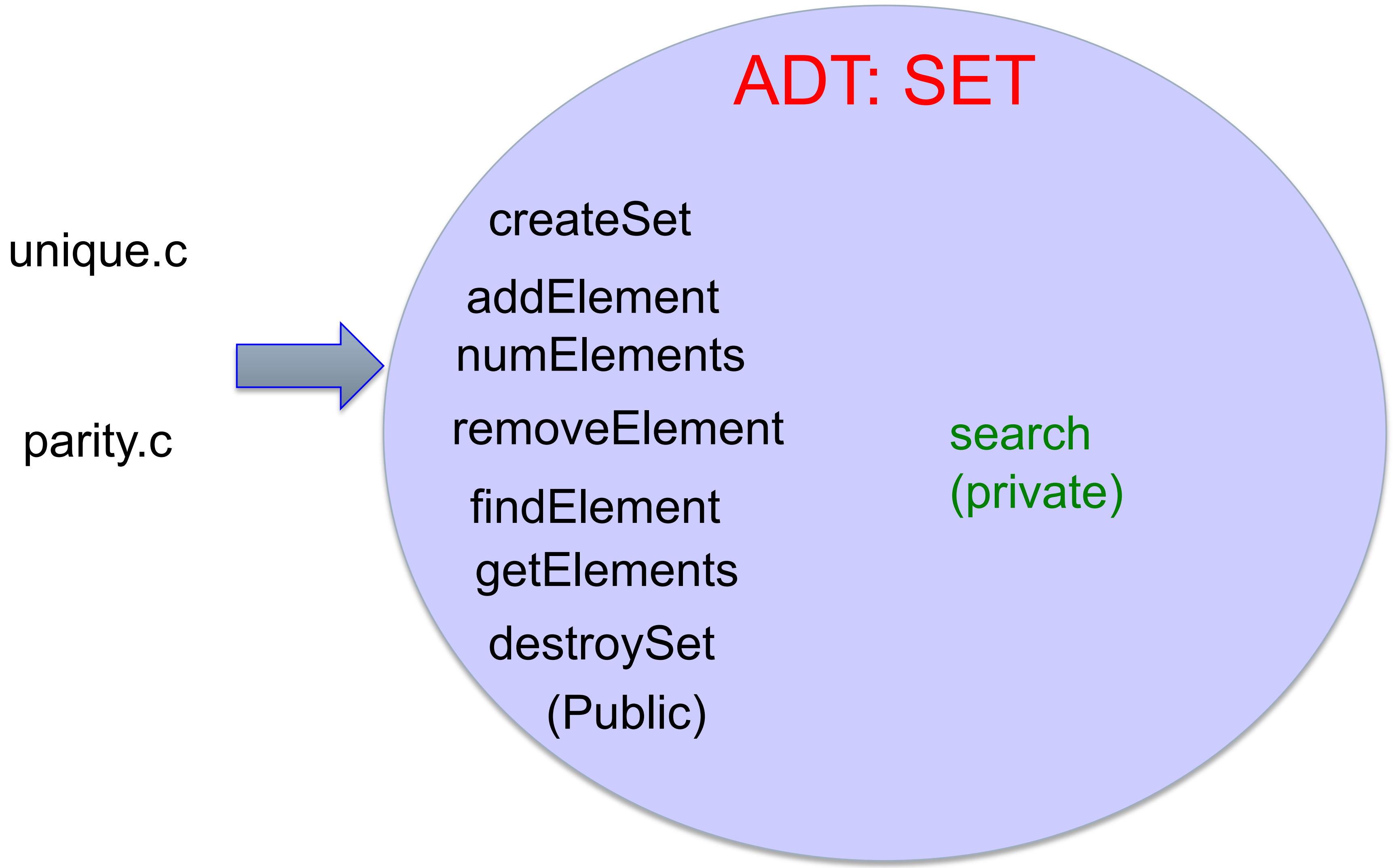


What Interfaces to Provide

The outside program (i.e. applications) should be able to

1. Create a set
2. Insert an element into the set
3. Count how many elements are currently in this set
4. Delete an element from the set
5. Search whether a specific element is in the set
6. Allocate and return an array of elements in the set pointed to by a pointer
7. Destroy the set

ADT: SET - An Illustration



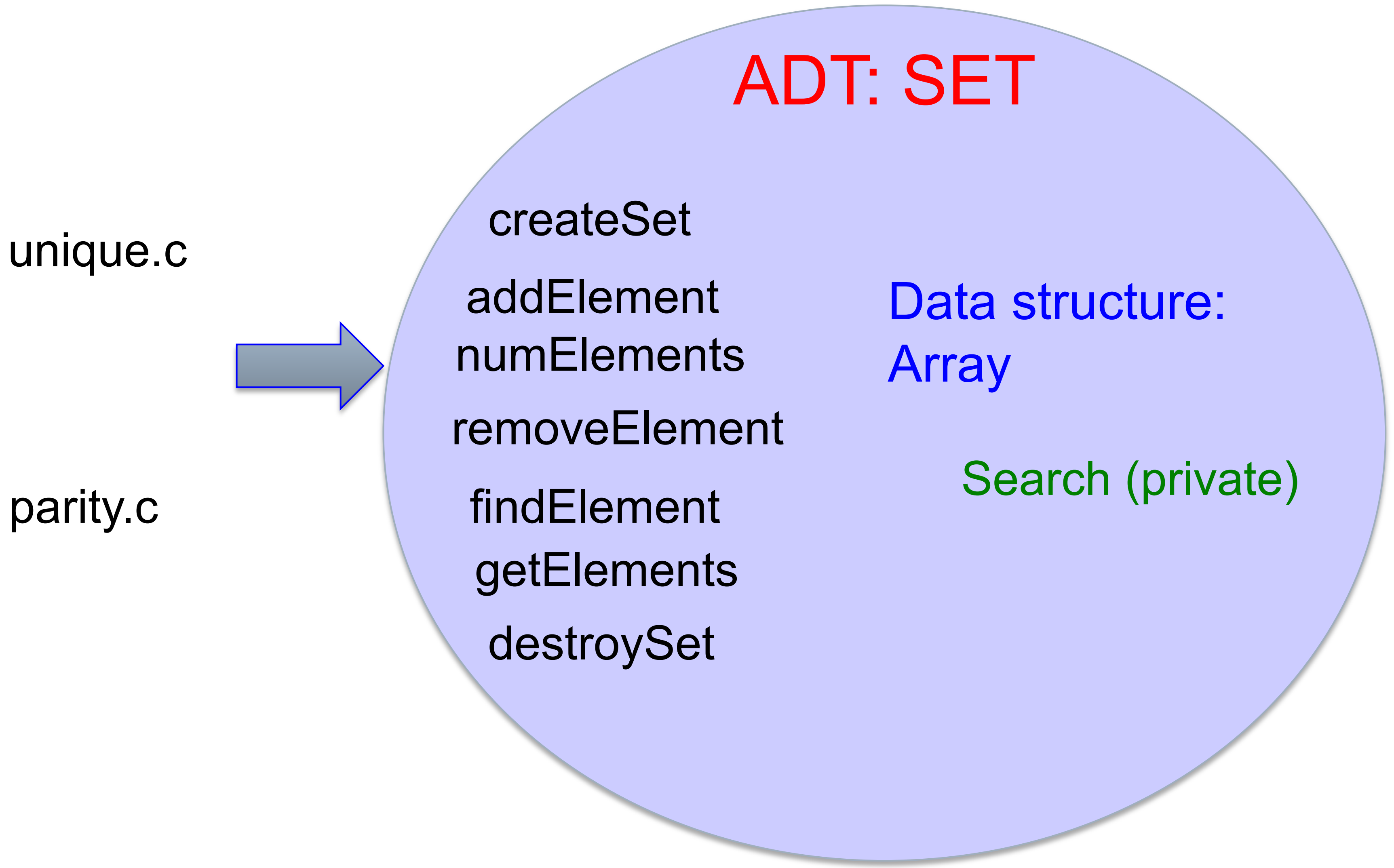
Implementing a SET

- How to store elements of a set in memory? Data structure

Here are some options: array, linked list, tree, graph, …

=> Array !

ADT: SET - An Illustration



ADT - SET Assignment

- You will be given the set.h which defines the following interfaces for this ADT
  - SET *createSet(int maxElts);
  - void addElement(SET *sp, char *elt);
  - int numElements(SET *sp);
  - void removeElement(SET *sp, char *elt);
  - char *findElement(SET *sp, char *elt);
  - char **getElements(SET *sp);
  - void destroySet(SET *sp);

- This SET handles strings!
- You are implementing just this ADT … not the test programs or the header
- Two week project
- First week: you will finish an unsorted implementation

ADT - SET Assignment

- main program: unique.c, parity.c
- your implementations: unsorted.c, sorted.c
- you must test **all two** programs against unsorted.c in the first week
- you must finish the sorted implementation for the second week
- the TAs will help you if you don’t quite understand, e.g., how to work with the different files

Implementing a SET Using Array

- Pre-knowledge:
  - Store string elements in an array
  - Dynamically sized array
  - Structure

A Simpler Case - int

In the assignment, you are required to implement the set for strings.

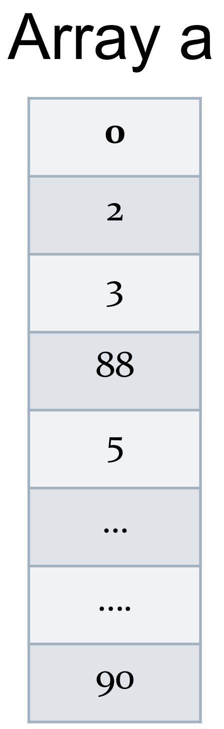
However, strings are relatively hard to understand;

So let’s use a simpler case – int as an example! Then figure out how to implement the case of string.

Pre-knowledge 1 - Store string elements in an array

- How to Store Integers in an Array

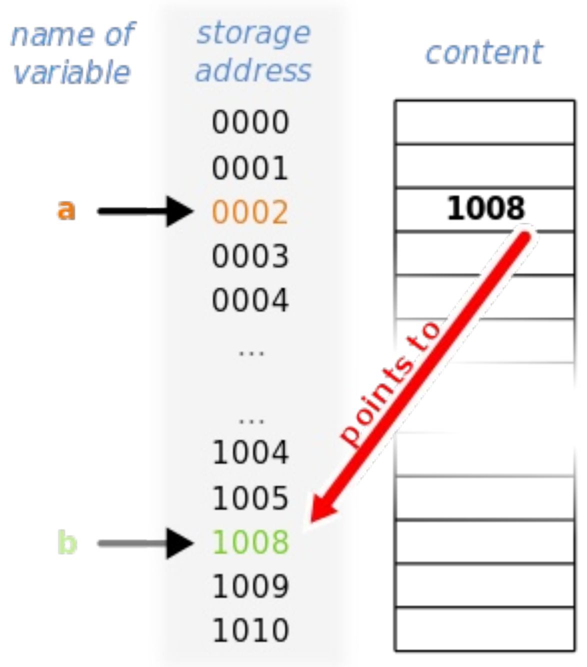
int a[100];



How to store strings in an array in C?

Pointer

In computer science, a pointer is a programming language object, whose value refers to (or "points to") another value stored elsewhere in the computer memory using its memory address. A pointer references a location in memory, and obtaining the value stored at that location is known as dereferencing the pointer.

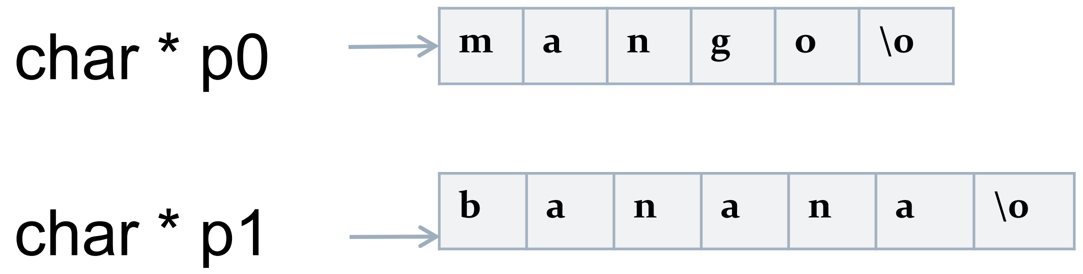


int b = 3; (b is allocated at address 1008)
int* a = &b; (let pointer a point to the address of b)

a =? *a = ?

String in C – Character Array

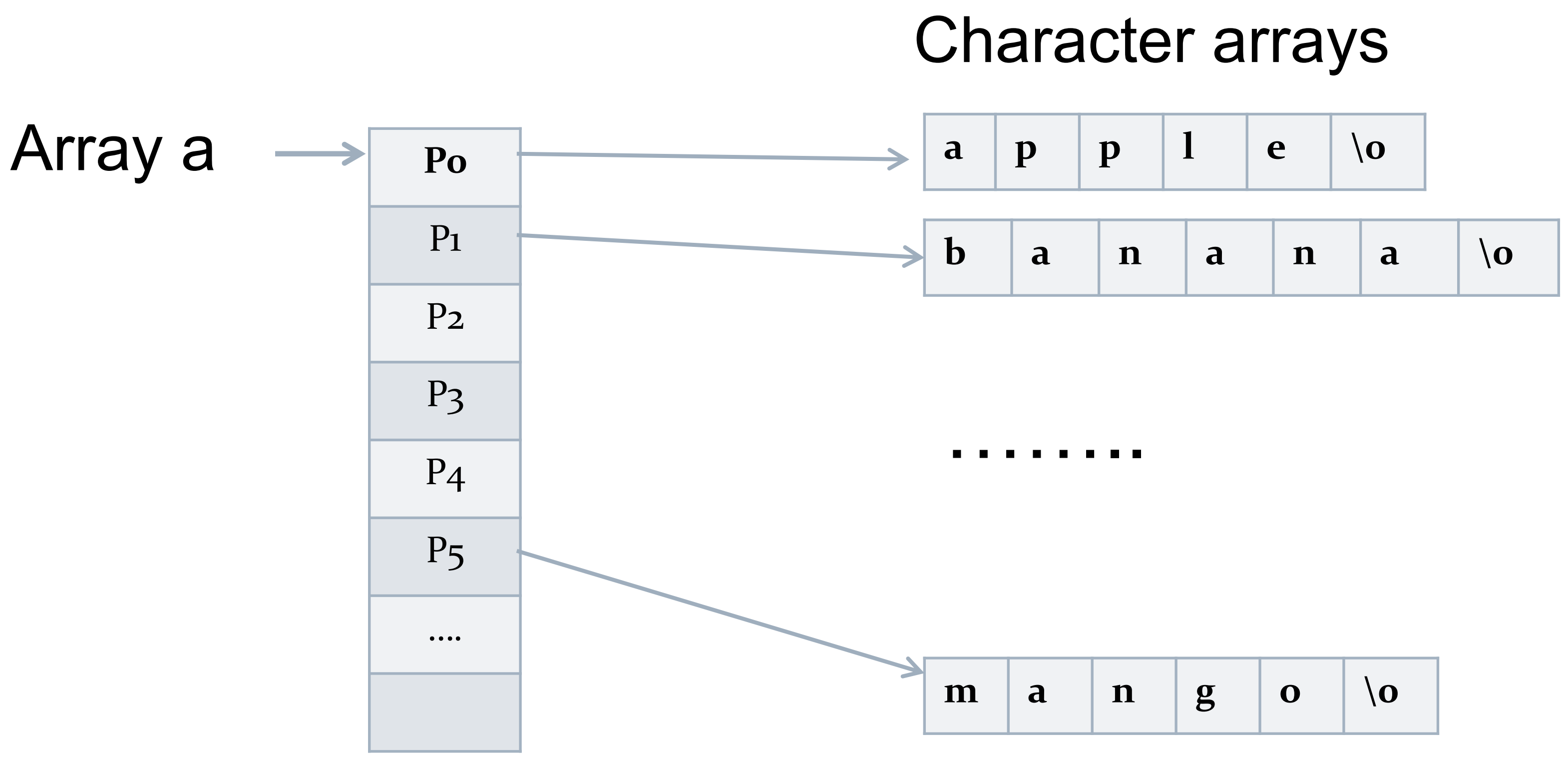
Character Array



Refer to the character array through a character pointer: **char** *

e.g. the pointer name is p0. The value of p0 is 1008, which is the address of the character array of “mango”.

How to Store Strings in an Array



Use array a to store character pointers: **char** *

How to declare array a? Char* a[100] ;

The First Function

```
SET *createSet(int maxElts);
```

↑

The maximum number
of elements in the SET

- Note that createSet() requires a parameter that indicates the max # of elements that we will need to store.
- **Problem:** In our project, array length is not known in advance and will be passed as a parameter maxElts.
  - If they say 10,000 and then use 10,001 … It's not your problem, **they** broke the contract.
- How to implement a dynamically sized array?

Pre-knowledge 2 - Dynamically Sized Array

- In C, how do we implement a dynamically sized array?
  - we implement a **statically sized** array (that doesn't change) by doing: int a[10];
  - Can we do int a[n]?

Dynamically Sized Array

- In C we implement a dynamically allocated array by using a pointer:
  - ❑ int *a;
  - ❑ This initially points to garbage, until you do a memory allocation using a function that allocates memory …

➢ We then need to explicitly allocate the memory using malloc()

- ❑ a = malloc(sizeof(**int**)*n);
- ❑ assert(a!=NULL)

The **assert.h** header file of the C Standard Library provides a macro called **assert** which can be used to verify assumptions made by the program and print a diagnostic message if this assumption is false.

Assert

- An *assertion* specifies that a program satisfies certain conditions at particular points in its execution.

- Common uses
  - Pointers are not NULL.
  - Indices and size values are non-negative and less than a known limit.

- Ex:
  - p is a pointer: assert(p != NULL);
  - we want to access the i_{th} element of an array a, which has size of n: assert(?)

- Make sure to test the condition before the usage of the variable.

How about String?

A dynamically allocated array of strings

```
char **a;
a = malloc(sizeof(char *)*n);
assert(a!=NULL);
```

Implementing a SET

- We will implement our set using an array. We need therefore to keep track of three things:
  - ❑ 1. the address of the array
  - ❑ 2. the length of the array
  - ❑ 3. the number of elements in the array
- What's the difference between 2 and 3?
- How to track three things together?

  => structure!

Pre-knowledge 3 - Structure in C

- Concept
  - A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling.
  - E.g. struct mystruct {

```
int a;
int b;
char c;
};
```

  - For students who are not familiar with structures, please check our textbook “The C Programming Language” Chapter 6.

Implementing a String SET

- Recall that we need to track three things as
  - 1. the address of the array
  - 2. the length of the array
  - 3. the number of elements in the array

- Let’s construct the structure
  - struct set {
    - int count;
    - int length;
    - char **data;

    };

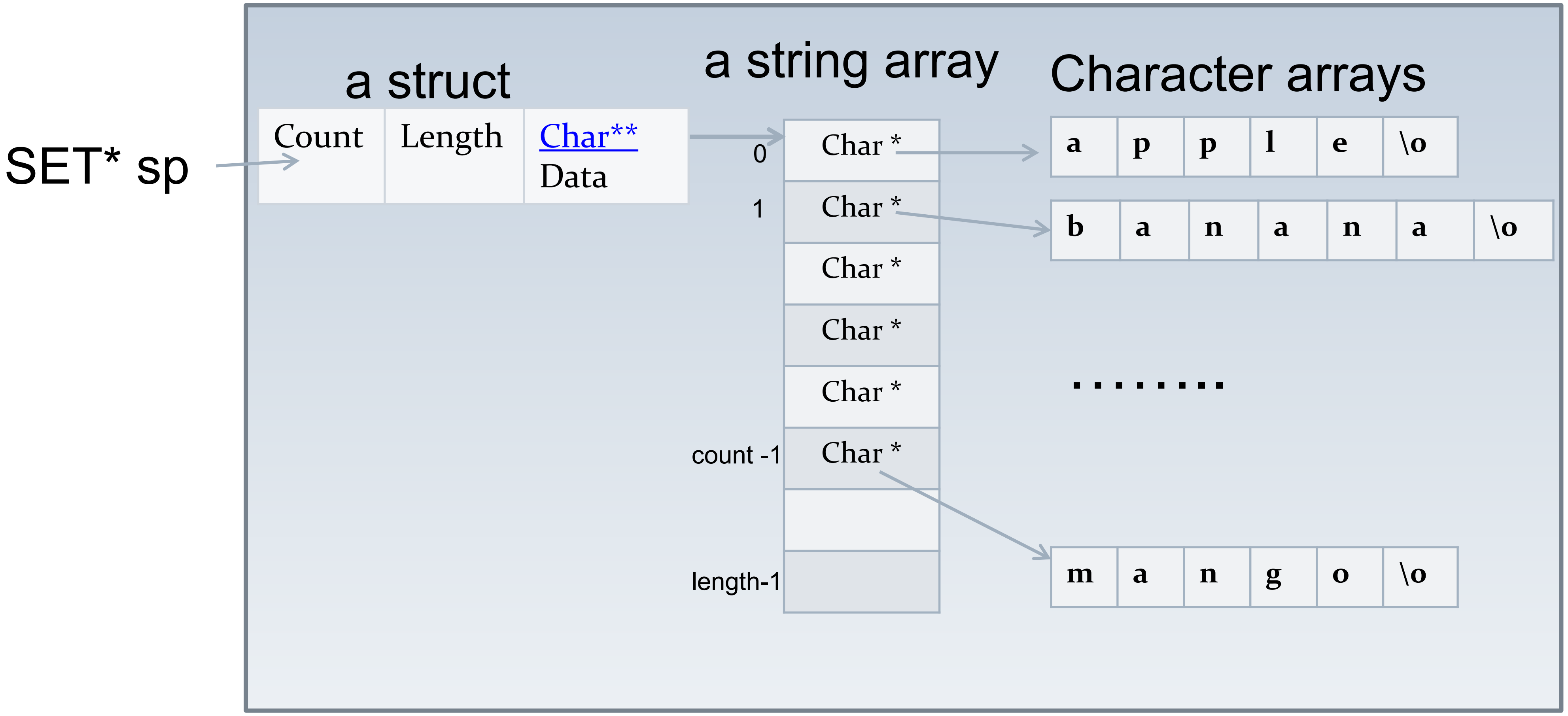
Implementing a String SET

- SET.c //file name

```
#include <assert.h> // for error detection
#include   "set.h"  // public interface
typedef struct set {
    int count;
    int length;
    char **data;
} SET;
```

Refer to set.h for the declaration of all the pubic functions.

The Data Structure of a SET



How to initialize this data structure?

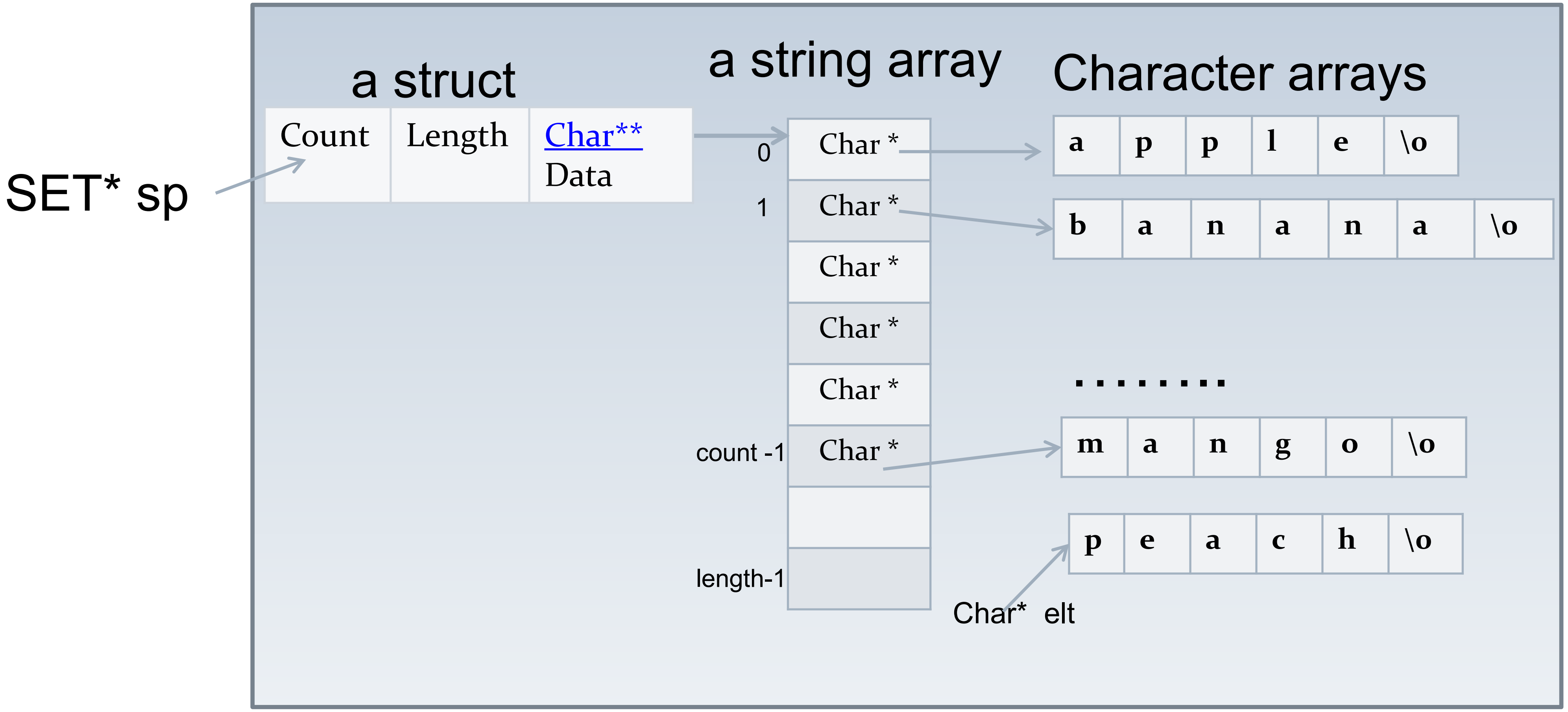
Function 1: createSET

```
// draw memory state diagram alongside the instructions
SET *createSet(int maxElts)
{
    SET *sp;

    sp = malloc(sizeof(SET));
    assert(sp != NULL);
    sp->count = 0;
    sp->length = maxElts;
    sp->data = malloc(sizeof(char*)*maxElts);
    assert(sp ->data!= NULL);

    return sp;
}
```

Add an element



How to add an element to this data structure?

Function 2: addElement

- addElement(SET *sp, char* elt) … Pay attention to some special cases
  - If the element number already equals to the array maximum length, cannot add
  - If you can find the new element in the SET, cannot add
    - How to know if the element is in the SET?
      - char *findElement(SET *sp, char *elt); - public interface
      - static int search(SET *sp, char *elt, bool *found) - private function

Implement addElement()

- Try 1: sp->data[sp->count] = elt;
  - show why this doesn’t work … it points to one word (whatever was last in the buffer)
  - only copies pointer, not contents
  - buffer will always change and therefore sp->data[] will also change.

Implement addElement()

- Try 2: strcpy(sp->data[sp->count], elt);

- Why this is wrong …
  - segfault! …
  - copies contents into unallocated memory.

Implement addElement()

- Try 3:
  - sp->data[sp->count] =
    - malloc((strlen(elt) +1)* sizeof(char));
  - strcpy(sp->data[sp->count], elt);

Implement addElement()

- TRY 4:
  - sp->data[sp->count] = strdup(elt); //allocates memory using malloc, copies the string, and returns a pointer to the copy

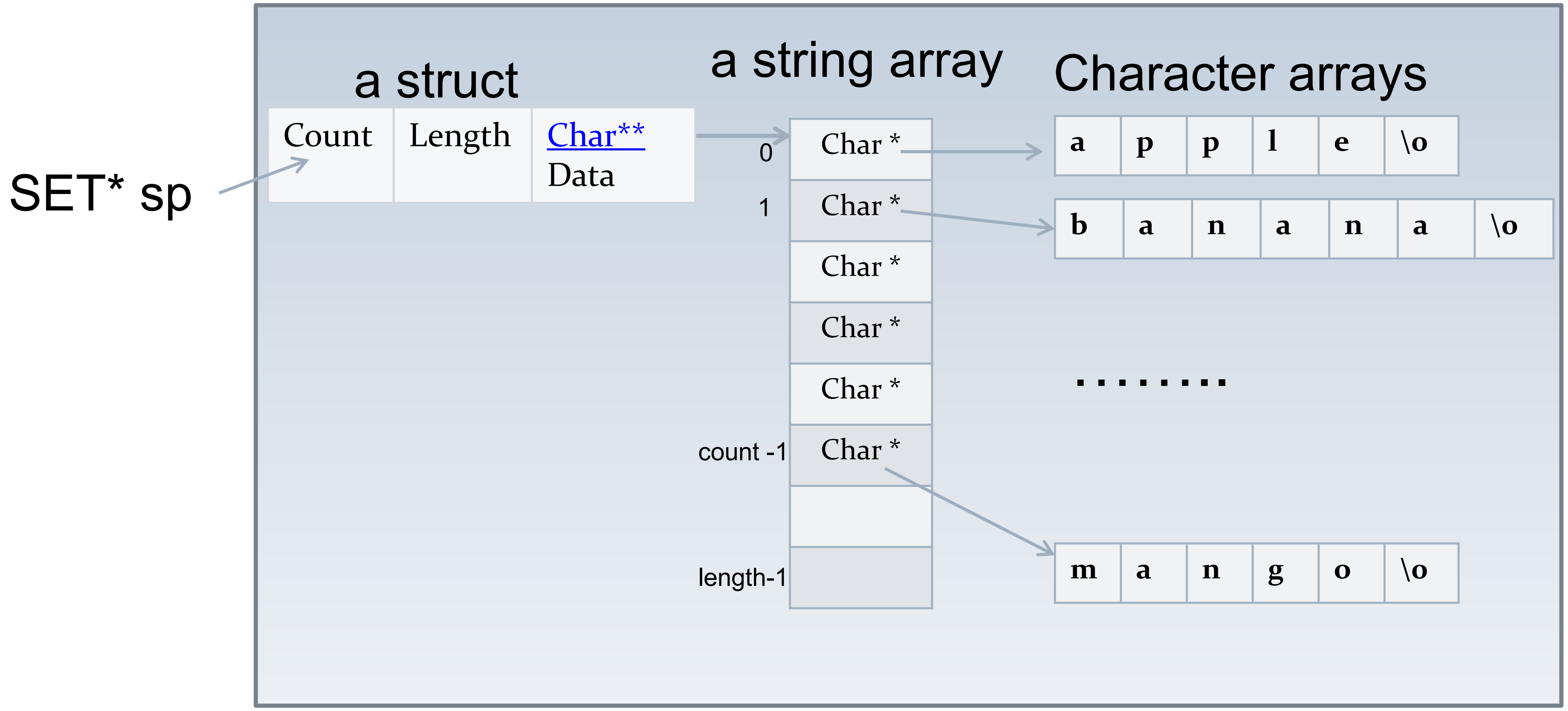
- To find out more
  - $man strdup

- RTFM = ... read the fun manual

Function 3: destroySet()

- What if you don't free malloc'd memory?
  - memory leak
- When removing something, remember that you will need to do so in reverse order.

Destroy the SET



How to destroy this data structure?

Implement destroySet()

```
void destroySet(SET *sp)
```

Other Functions

- numElements(SET *sp) … Done
- findElement(…) … Verify whether an element already exists
- removeElement(…) .. Identify the element to remove first

Private Function: Search

- Implement a "private" function for our own use, which we won't make public. Search() … which returns the index if it's found in the set.

```
static int search(SET *sp, char *elt, bool *found)
```

- In C, static functions are functions that are only visible to other functions in the same file. We mark it "static" to keep it local to the file, so no other code can call it. Good practice, but not absolutely necessary.
- This function is not in set.h
- This function will be different for the sorted and unsorted cases

Private Function: Search

- How to find out whether an element exist in an array?

[Array Search!!]

Week 1 of Project 2

- Implementing a SET using unsorted array!

- What search to use for an unsorted array?

Sequential Search!

Week 2 of Project 2

- Implementing a SET using sorted array!

- What search to use for a sorted array?

Binary Search!

PROJECT 2 - PART 2: IN THE SORTED CASE

What are the major differences?

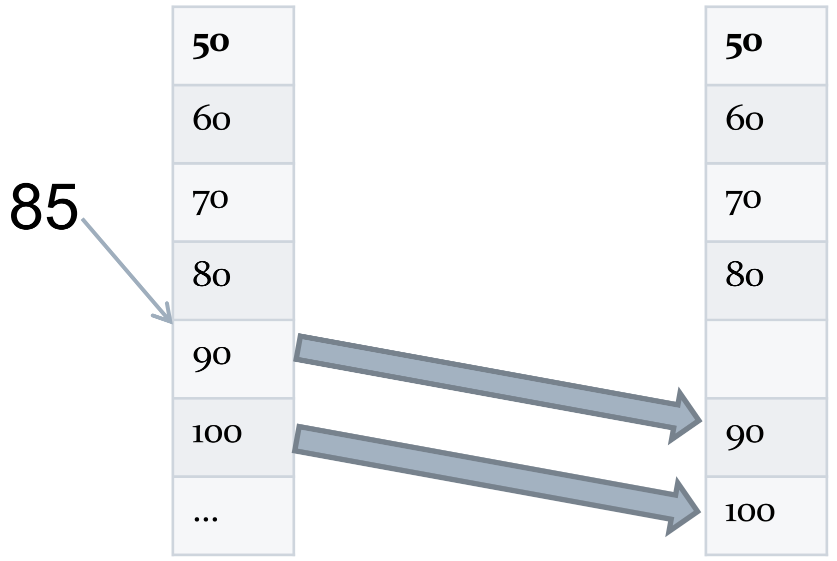
Interfaces:

```
SET *createSet(int maxElts);
void destroySet(SET *sp);
int numElements(SET *sp);
bool findElement(SET *sp, char *elt);
bool addElement(SET *sp, char *elt);
bool removeElement(SET *sp, char *elt);
```

Private function

```
static int search(SET *sp, char *elt, bool *found)
```

Add an Element

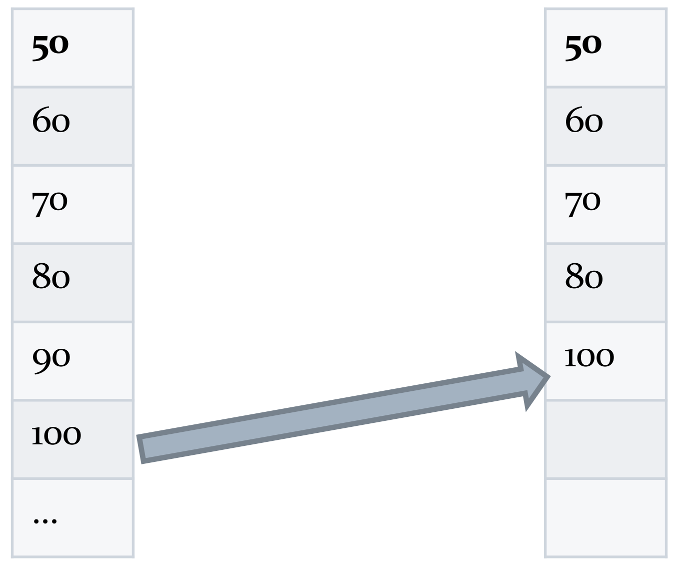


To shift all elements from slot idx **DOWN** one slot we start at the “bottom” (adding at idx).

```
for (i=sp->count; i> idx; i--)

    Sp->data[i] = sp->data[i-1];
```

Remove an Element



To shift all elements from slot idx **UP** one slot we start at the“top” (deleting at idx).

```
for (i= idx +1; i<sp->count; i++)

    Sp->data[i-1] = sp->data[i];
```

Insert & Remove an Element

- To shift up, you start at top, to shift down, you start at the bottom.

- What is the worst-case bigO runtime of insert (not include searching)?
  - ❑ O(n)

- What is the worst-case bigO runtime of deletion?
  - ❑ O(n)

- What is the bigO runtime of search?
  - ❑ O(log(n))

How to Do Search in the Sorted Case?

Re-examine Binary Search

- Check your notes. As we wrote it, bsearch only returns whether the element is present.
- However, we do need the location of the element, since
  - Removing an element requires knowing where the element is.
  - Also, adding an element requires keeping a list in sorted order (so could benefit from knowing where element *should* be).
- So we want to modify bsearch to "return" two things:
  - whether the elt is present
  - where it is/should go

```
int bsearch(int a[], int n, int x, bool found) ?
```

Binary Search: Code

- The given binary search is for integer case, you have to work out your solution for the string case.

- This is the last help on the project.

- If you were paying attention you need only write about 15 lines of original code.

Basic Operations	Unsorted Array	Sorted Array
Search/Find		
Insert		
Delete		
Min/Max		

Basic Operations	Unsorted Array	Sorted Array
Search/Find	O(n)	O(log n)
Insert	O(n)	O(n)
Delete	O(n)	O(n)
Min/Max	O(n)	O(1)