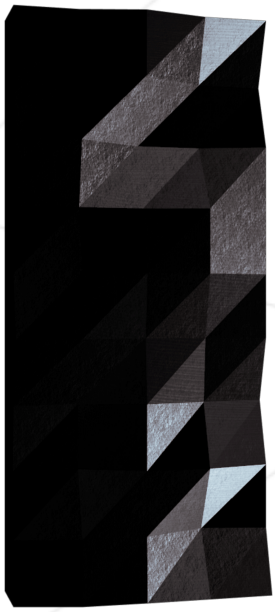

Starfleet - Day 00

Arrays and Strings

Staff 42 pedago@42.fr

Summary: This document is the day00's subject for the Starfleet Piscine.

Contents

Chapter I

General rules

- Every instructions goes here regarding your piscine
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- The exercises must be done in order. The evaluation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The subject can be modified up to 4 hours before the final turn-in time.
- You will NOT be graded by a program, unless explictly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.
- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

Chapter II

Day-specific rules

- During this piscine, we use a little script named `compile`. It compiles your exercises of the day, example:

```
$> compile nameExercise.c
$> ./nameExercise
hello!
```

  You can download the script on the intra page of the day.

  You can follow these commands to add the script into your executables:

```
$> mkdir ~/.bin
$> mv ~/Downloads/compile ~/.bin/.
$> chmod +x ~/.bin/compile
```

  Then, in your `~/.zshrc` (if you use zsh) or `~/.bashrc` (bash), add the following command:

```
export PATH=$PATH:$HOME/.bin
```

  Restaure your shell (using `reset` command), You can now use it!

- For each exercise, you must turn-in a file named `bigo` describing the time and space complexity of your algorithm as below. You can add to it any additional explanations that you will find useful.

```
$> cat bigo
O(n) time, with n the number of elements in the array.
O(1) space.
$>
```

- Your work must be written in `C`. You are allowed to use all functions from standard libraries.

- For each exercise, you must provide a file named main.c with all the tests required to attest that your functions are working as expected.

Chapter III

Exercise 00: Bad day

	Exercise 00
Exercise 00: Bad day	
Turn-in directory : $ex00/$	
Files to turn in : `searchPrice.c main.c header.h bigo`	
Allowed functions : `all`	
Notes : `n/a`	

The database of an auction website has been `broken` and no longer works. The person in charge of the database has gone on vacation and is unreachable. Little stroke of luck, you know the existance of a large file which contains, for each auction object, its name and its price.

However, a big auction has just been launched and thousands of buyers are waiting on the website, your mission will be to make the customers wait!

There are around `100.000` works of art in a file named "art.txt":

- `Mona Lisa -> 2.000.000.000`
- `Guernica -> 550.000.000`
- `...`

Given an array of the following structure:

```
struct s_art {
        char    *name;
        int     price;
};
```

Implement a simple function that `searches` in the array the price of a work of art, given its name as parameter:

```
int  searchPrice(struct s_art **arts, char *name);
```

If the name isn't in the array, just return -1.

Examples:

- Call the function `searchPrice(arts, "Mona Lisa")`
  The return value must be => 2000000000
- Call the function `searchPrice(arts, "I dont exist")`
  The return value must be => -1

What are the time and space complexities of your algorithm? remember to add it in the 'bigo' file!

Chapter IV

Exercise 01: Complaints

	Exercise 01
Exercise 01: Complaints	
Turn-in directory : *ex*01/	
Files to turn in : `sortArts.c main.c header.h bigo`	
Allowed functions : `all`	
Notes : `n/a`	

Customers are complaining, the website takes too long to give a result.
Your first intuition is that the search would be a lot faster in a `sorted` array.

Implement a function to sort the array based on the name of the paintings :

```
void sortArts(struct s_art **arts);
```

The sort must be `case-sensitive` and in `ascending` order.

`Your function must sort the array in less than 2 minutes.`

Chapter V

Exercise 02: Smart search

	Exercise 02
Exercise 02: Smart search	
Turn-in directory : $ex02/$	
Files to turn in : `searchPriceV2.c main.c header.h bigo`	
Allowed functions : `all`	
Notes : `n/a`	

Now that you have a sorted array, you should be able to implement a `better` search algorithm.

Implement a simple function that searches in the `sorted` array for the price of an art, given as parameters the array `arts` with `n` its length, and `name`, the name of the search art:

```
int searchPrice(struct s_art **arts, int n, char *name);
```

If the name isn't in the array, just return `-1`.

`Iterating through all the elements of the array is not the right solution ;)`

Chapter VI

Exercise 03: More complaints

	Exercise 03
Exercise 03: More complaints	
Turn-in directory : $ex03/$	
Files to turn in : `searchPriceV3.c main.c header.h bigo`	
Allowed functions : `all`	
Notes : `n/a`	

Customers are still complaining. Indeed, there are slightly too many elements in the array and, for people with slow connections, it still takes `too much` time to search through it...

You must improve that!

Instead of a huge array, you must work with an `hash table`.

Here are some informations in order to implement the hash table.

You have to implement the 4 following functions:

- `hash(input)` : Create a hash product from the input.
- `dictInit(capacity)` : Initialize the hash table given the capacity of the array.
- `dictInsert(dict, key, value)` : Insert an item in the hash table given his key and value.
- `dictSearch(dict, key)` : Find an element in the hash table given the key, if not found, return NULL.

Given the following structures:

```
struct  s_art {
  char   *name;
  int    price;
};

struct s_item {
  char          *key; //here the key will be the name
  struct s_art  *value;
  struct s_item *next;
};

struct s_dict {
  struct s_item **items;
  int           capacity; //the capacity (allocated size) of array 'items'
};
```

`Note`: There is multiple way to handle `colision`, however, we decide to let you implement a collision using a linked list.

The functions defined above must be declared as follows:

```
size_t hash(char *input);

struct s_dict *dictInit(int capacity);

int dictInsert(struct s_dict *dict, char *key, struct s_art *value);

struct s_art *dictSearch(struct s_dict *dict, char *key);
```

You have then to implement the following function that will call the `dictSearch` to search in the hashTable a price of an art, given its name as parameter:

```
int  searchPrice(struct s_dict *dict, char *name);
```

You won't be evaluated on the hash function and its associated number of collisions, so we advise you to create your own simple hash function!

Chapter VII

Exercise 04: Homemade cookies

	Exercise 04
Exercise 04: Homemade cookies	
Turn-in directory : $ex04/$	
Files to turn in : `howManyJesus.c main.c header.h bigo`	
Allowed functions : `all`	
Notes : `n/a`	

After `saving` the day at the auction company, you are getting ready for well deserved `vacation`. But not too fast, your `grandma` just called you and she needs you!

She would like to save one of her favorite books - `THE BIBLE` - on her computer, clearly letting you know that she really counts on you (and that fresh baked cookies will be ready for you at her house).

However, her computer runs on `WINDOWS-95` and everything still works with `floppy disk` (brrr). Easy peasy, you download her book and then try to put it on the disk: the memory space is `insufficient`. Craving for Granny's delicious cookies, you have no choice ... You have to `compress` the file ...

Before getting to work, you wonder `how often` the word `'jesus'` appears in the Bible. Indeed, how many times?

Implement the following function using the `Rabin-Karp` method:
given a long string (`bible`) and a substring (`jesus`), find how many time the substring occur:

```
int howManyJesus(char *bible, char *jesus);
```

Examples:

- Call the function `howManyJesus(bible, "God")`
  The return value must be => 4121

- Call the function `howManyJesus(book, "Lord")`
  The return value must be => 1068

`The function will be case-sensitive.`

Chapter VIII

Exercise 05: Get down to it

	Exercise 05
Exercise 05: Get down to it	
Turn-in directory : $ex05/$	
Files to turn in : `compress.c main.c header.h bigo`	
Allowed functions : `all`	
Notes : `n/a`	

Hmmm you start to smell `granny's cookies`, but not yet! You have to compress the book before ...

To compress the file:

- `We provide you some of the most used words that are inside an hash table.`
- `Add a header containing all the substrings.`
- `Replace all the occurence of these substring that you find in the book with the char '@' followed by its index in the dictionnary.`

For instance:

```
entry:

    book: "The first second was alright,
    but the second second was tough."

    hashtable contains: ["first", "second"]

compression: (using cat -e)

    "<first,second>The @^A @^B was alright,
    but the @^B @^B was tough."

    (where '^A' and '^B' are soh and stx respectively, value 1 and 2 on the ascii table)
```

Your function should have the following prototype, with `book` the string that contains the book to compress, and `dict`, the hash table:

```
char *compress(char *book, struct s_dict *dict);
```

Note:

- We don't provide you an hash table, you have to implement one that is the `same` as the exercise `03`, except that this time, it stores `integers` instead of `pointers`. So you have to update the function in consequence, take a look at the prototype in `header.h`.
- To create the header, you have to go through `every` element in the hash table.
- There will always be `less` than `255` words.

Chapter IX

Exercise 06: Take a load off

	Exercise 06
Exercise 06: Take a load off	
Turn-in directory : $ex06/$	
Files to turn in : `decompress.c main.c header.h bigo`	
Allowed functions : `all`	
Notes : `n/a`	

But wait, how is granny going to `open` her book?

Your mission is to `decompress` the book.

Implement a function that decompresses the string of characters received as parameters:

```
char *decompress(char *cBook);
```

But for this time you're gonna have to implement a `dynamic string`! (you will see it's `fantastic`!)

Here are some information for implementing the dynamic string.

Implement the 2 following functions:

- `stringInit()` : Initialize the dynamic string.
- `stringAppend(dyString, str)` : append the str to the dynamic string.

Given the following structures:

```
struct s_string {
  char *content;
  int length; //the current length of 'content'
  int capacity; //the allocated size of 'content'
};
```

The functions defined above must be declared as follows:

```
struct s_string *stringInit(void); //success: return new allocated pointer, fail: return NULL
int        stringAppend(struct s_string *dyString, char *str); //success: return 1, fail: return 0
```

Examples of use:

```
struct s_string *s = stringInit();

stringAppend(s, "Hello");
printf("%s\n", s->content); //'Hello'

stringAppend(s, " World!");
printf("%s\n", s->content); //'Hello World!'
```

Your 'stringAppend' function must run in constant amortized time.

Chapter X

Exercise 07: Why is my picture sideways?

	Exercise 07
Exercise 07: Why is my picture sideways?	
Turn-in directory : *ex*07/	
Files to turn in : `rotate.c main.c header.h bigo`	
Allowed functions : `all`	
Notes : `n/a`	

You are now so close to eat Granny's cookies!

But wait, Granny needs your help just one more time. She wants to send a `picture` stored on her computer, the problem is that the picture is sideways!

Given a matrix of integer `picture` of size `n`,
implement a function, to rotate the picture clockwise (90 degrees to the right):

```
void rotate(int **picture, int n);
```

The picture will always be a square of size `n`.

With the main file in the exercise resources, the output should be as in the example below.

```
$> compile rotate.c
$> ./rotate unicorn.pgm > output.pgm
$> diff rotated.unicorn.pgm output.pgm
$>
```

You can actually see the images in finder! :)

Chapter XI

Exercise 08: Syntactic Corrector

	Exercise 08
Exercise 08: Syntactic Corrector	
Turn-in directory : *ex*08/	
Files to turn in : `whatWasThat.c main.c header.h bigo`	
Allowed functions : `all`	
Notes : `n/a`	

`Grandpa` comes to you (while you eat Granny's cookies) and let's say that he is pretty rusty.

When he talk to you, you do `not understand` what he is saying, he may be talking about his `will`, you need to figure it out!

Suddenly, a `crazy` idea comes to your `mind`, create a `Syntactic Corrector`.

Implement a function that takes a `word` and a `dictionary` as parameters, and returns the dictionary word that most closely resembles the word.

To create this algorithm, you will first have to implement `3 functions` which modify the string:

- `replace` : `replace a character` in the string at a given position. The range of possible characters is `a to z` (minuscule).
- `delete` : `delete a character` in the string at a given position.
- `add` : `add a character` in the string at a given position. The range of possible characters is `a to z` (minuscule).

`Note`: all of these functions are `allocating a new string`.

The functions defined above must be declared as follows:

```
char *replace(char *word, int pos, char c);

char *delete(char *word, int pos);

char *add(char *word, int pos, char c);
```

Examples using these functions:

```
replace("hello", 1, 'a');   //return "hallo"

replace("hello", 100, 'a'); //return NULL

delete("hello", 1);         //return "hllo"

add("hello", 1, "a");       //return "haello"

add("hello", 0, "a");       //return "ahello"
```

The algorithm you must create has to `combine` these operations and find in the dictionary if one other word matches.

You must return the word in the dictionnary which requires the `minimum` number of `combinations`.

You can combine at least `DEPTH` operations at the same time (where `DEPTH` is a define in the header file).

```
#define DEPTH 3
```

The function must be prototyped as follow:

```
char *whatWasThat(char *word, char** dictionary);
```

If there is no word alike, return `NULL`.

An example:

```
print("%s\n", whatWasThat("hello", ["halo", "hocket", "dojo"])); //"halo"
```

In the above example, it will return `"halo"` because you can get it in `2` operations: one `replace` ('a' by 'e') and one `delete` ('l' at position 2). `"dojo"` will be in 4.

Other examples:

- Papy says: `"Ack in my day we weren't going at the bar like you, we were working!"`

```
$> #Closest word of "ack" in the dict ["attack", "back", "clack"] ?
$> ./whatWasThat "ack"     "attack" "back" "clack"
back
```

- Papy says: `"Get off my damn lawn you whippesnapars!"`

```
$> #Closest word of "whippesnapars" in the dict ["gingersnap", "snapshooter", "whippersnappers"] ?
$> ./whatWasThat "whippesnapars"     "gingersnap" "snapshooter" "whippersnappers"
whippersnappers
```