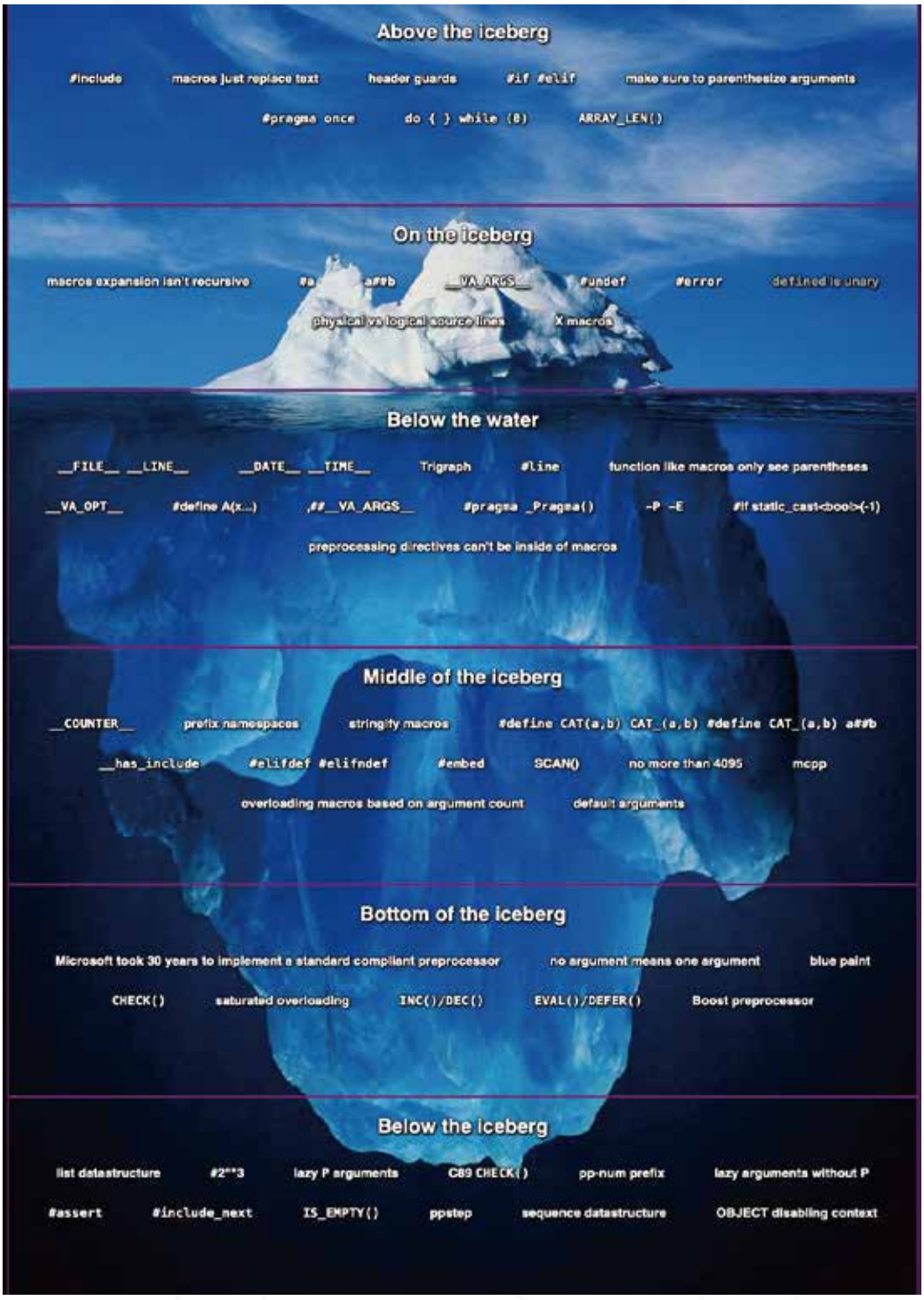
FT_Preprocessor

Summary:
This document contains the exercises of preprocessor

Chapter I

Preamble



Contents

Chapter II
Introduction

This project aims to broaden your insight of preprocessor by using C. You will implement macro expansion, line control, and conditional com- pila- tion to your program serve multi environments.

Chapter III
General rules

Your project must be written in C.

Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a 0 during the evaluation.

All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.

We encourage you to create test programs for your project even though this work won't have to be submitted and won't be graded. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.

Submit your work to your assigned git repository. Only the work in the git repository will be graded.

Chapter IV

Exercise 00 : Hello Preprocessor

Function name	hello_preprocessor
Prototype	void hello_preprocessor();
Turn-in directory	ex00/
Files to turn in	hello_preprocessor.c
Allow Keyword	__FUNCTION__, printf()

Time to say hello to your old friend! Even if you never knew....

Write a function for say hello to preprocessor.
You must use only printf() with %s specifier, and __FUNCTION__.

Your function will be tested by following main:

```c
void hello_preprocessor();

int main() {
	hello_preprocessor();
	return (0);
}
```

Greetings!

```
❯ ./a.out
hello_preprocessor
```

Start from searching about pre-defined MACROS like __FILE__, __LINE__, __FUNCTION__ !

Chapter V

Exercise 01: FT Get Filename

Function name	ft_get_filename
Prototype	char* ft_get_filename(const char * path);
Parameters	path : path for search
Return value	filename: correct behavior null : an error occurred
Turn-in directory	ex01/
Files to turn in	ft_get_filename.c
Allow Keyword	#ifndef, #elif, defined, #endif and c libraries

Implement a function which returns filename from whole path (Including extensions like .txt).
Note that Windows and Unix-based Operating Systems have a different file path syntax. Your function must be written with Preprocessor and able to get the same filename from the different OS.

WINDOWS :

```
C:Users\su\Desktop\42.txt
```

UNIX :

```
/users/su/Desktop/42.txt
```

Your function should works like example below:

```
// in window
int main() {
  char* filename_                    :("C:\\Program files\\Downloads\\index.html");
  printf("%s\n", filename_in_window);
  // index.html
}
// in mac
int main() {
  char* filename_in_mac = ft_get_filename("/usr/share/doc/bash/article.pdf");

  printf("%s\n", filename_in_mac);
  // article.pdf
}
```

char *filename_in_window
filename_in_window

Requirments

- You must implement with Preprocessor only in .c file.
- ONLY Windows and macOS is considered. In any other OS, the function should return NULL.
- Don't interpret special characters except the OS's file path delimiter.
- If the end of path is a delimiter, return NULL.
- If no delimiter specified in path, return NULL.
- In any other case, path ends with a filename.
- You should allocate your string, of course.
- !!! How Preprocessor detects OS?

Do you know about Conditional compilation?

Chapter VI

Exercise 02 : FT_HOSTCMP

Macro name	FT_HOSTCMP
Parameters	A : value of host B : value of network
Turn-in directory	ex02/
Files to turn in	ft_hostcmp.h
Allow Function	none

Aligning bytes in computer's memory, can also be an 'architecture depending' job. This time, by implementing FT_HOSTCMP, you will figure out how it works.

Implement a Macro which expands to 0 if passed argument A and B is equal, or non-0 value in any other case in ft_hostcmp.h.
You should consider endian and an architecture of processor (32 bit systems or else) .

- While Preprocessing, you should detect endian with FT_USE_LITTLE_ENDIAN, and FT_USE_BIG_ENDIAN.
- While Preprocessing, you should detect an architecture with FT_USE_32, FT_USE_64.

Think about the difference of Big Endian and Little Endian.

Your Macro should be able to work with the main below:

```c
#define FT_USE_LITTLE_ENDIAN    1
#define FT_USE_32_BIT           1
#include "ft_hostcmp.h"

int main() {
    printf("%d\n", FT_HOSTCMP(8080, htonl(8080))); // print "1"
    return (0);
}
```

How about using zero-function including your own? Is it a good idea?