**Para-C:** **C-based Coding designed to be simple and fast with new syntax and functionality**

LUNA KLATZER, HTBLA Leonding

LORENZ HOLZBAUER, HTBLA Leonding

Content

[1. Introduction 3](#_Toc71660358)

[2. Base structure 4](#_Toc71660359)

[2.1 File structure 4](#_Toc71660360)

[2.1.1 Para-C modules and files 4](#_Toc71660361)

[2.1.2 Importing PARA-modules 4](#_Toc71660362)

[2.1.3 Importing C-modules 4](#_Toc71660363)

[2.2 Core Syntax 4](#_Toc71660364)

[2.3 Built-in Macros and defining Macros 4](#_Toc71660365)

[2.4 Structs and Type-def 4](#_Toc71660366)

[2.5 Exceptions 4](#_Toc71660367)

[3. Compiler and logical Structure of Para-C 4](#_Toc71660368)

[4. Usage-examples of Para-C 4](#_Toc71660369)

[4.1 Using C-Code inside Para-C 4](#_Toc71660370)

[4.1.1 Using Standard C-code 4](#_Toc71660371)

[4.1.2 Using C-libraries 4](#_Toc71660372)

[4.1.3 Restrictions of Para-C 4](#_Toc71660373)

[Footnotes 5](#_Toc71660374)

# Introduction

Para-C (From Greek Origin: Beside C[[1]](#endnote-1)) is a new programming language designed to integrate C and serve as a helper and simplifier to write better C code. Including adding more features, like new built-in Macros and functions, OOP-structures, more straightforward array and malloc-handling, expanded data types and simplified functions. That means it will not create its independent structure, but lay onto C and have a simpler syntax and structure comparable to Python and TypeScript.

The compiler therefore will take the Para-C code that will include C and Para Syntax and compile it either down to simple C and add all required modules and libraries, or directly compile an executable program that can be used. The importing and module system will lay on the `\_\_parac\_\_.h` language file, which will define import paths of files, constants, macros and functions, which are used throughout the compiled code. That means the compiler will fetch all required data and insert them into the file which can then be modified as well if the user intends to do so.

That means that programming in Para-C will be similar but simpler and well looking due to the simplifications, additional structures and helper functions. The Compiler will already look for certain issues and also try to optimise Para-C code, but if more complex code is used, the C-compiler will be used to detect issues. This is because Para-C is not intended to replace the C-compiler and check/validate C-code, but only in its separate modules, structures and syntax. It will provide simple checking, through the fact of using the classic compiler structure:

Lexical analyser (Tokenizer) -> syntax analyser -> semantic analyser (Logical and correctness checks) -> Code Generator and optimiser

This means syntax issues, logic errors or general errors will be detected and logged. The C-code compiler will do the rest of the job, which will be preferably GCC.

Syntax-wise Para-C will still lay onto C to avoid causing issues with more compiler code that would be required for a new syntax that can’t be easily integrated into the C-syntax. So newer structures won’t look so new, and possibly similar to C# or C++, like data-types, one-liners and getters etc.

Furthermore, syntax warnings might also be reported, as a help/motivator to avoid causing inconsistent writing and style. Still, in the end, the compiler will only create files with the standard C formatting and ignore any formatting done in the used Para-C file since after tokenisation the source code will be ignored by the compiler.

However, to keep the Para-C code clean and well-looking, the Compiler will include more advanced syntax- and style checking. Including possibly harder conventions, that will improve on the loose C-conventions, which are more open to writing code. That means Para-C will introduce more conventions regarding naming, type declarations, formatting, commenting and will likely also adopt a few Python conventions[[2]](#endnote-2) and integrate some of the ideas in the Zen of Python[[3]](#endnote-3)(Since the Compiler is also written in Cython).

Note: We intend Para-C not to be a widely „optimised“ or „production-ready“ programming language. It is solely a free-time project designed for learning and testing purposes, which we do not intend for anything other than that.

# Base structure

The structure of Para-C will closely lean to the C-Structure (C11-Standard) but still have its independent system apart from it. That means it will include its own: Structure for its Parac-modules and C-modules (*See Section 2.1 File Structure*), Import-structure, Built-in macros[[4]](#endnote-4), Built-in functions[[5]](#endnote-5) (which will partly replace the C-functions for easier handling) and OOP-Structure using GObject[[6]](#endnote-6) and system for integrating C-code and Macro definitions.

Still, the language will serve the main purpose of helping and providing features for C-code, meaning that the structure will be converted into C-code.

## File structure

The file structure in Para-C is similar to C. It will work mainly around the Para-C compiler folder with the executable and C-compiler materials and libraries that Para-C supports. (*See Section 4.1 Using C-Code inside Para-C)*

### Para-C modules and files

### Importing PARA-modules

### Importing C-modules

## Core Syntax

## Built-in Macros and defining Macros

## Structs and Type-def

## Exceptions

# Compiler and logical Structure of Para-C

# Usage-examples of Para-C

## Using C-Code inside Para-C

### Using Standard C-code

### Using C-libraries

### Restrictions of Para-C

Endnotes

1. Para meaning and origin: [[here]](https://en.wiktionary.org/wiki/%CF%80%CE%B1%CF%81%CE%AC#Preposition) [↑](#endnote-ref-1)
2. PEP8 – Conventions of Python: [[here]](https://www.python.org/dev/peps/pep-0008/) [↑](#endnote-ref-2)
3. PEP20 – Zen of python: [[here]](https://www.python.org/dev/peps/pep-0020/) [↑](#endnote-ref-3)
4. List of Pre-defined macros in C: [[here]](https://gcc.gnu.org/onlinedocs/cpp/Predefined-Macros.html) [↑](#endnote-ref-4)
5. List of Built-in C-functions: [[here]](https://www.tutorialspoint.com/ansi_c/c_function_references.htm) [↑](#endnote-ref-5)
6. Introduction to GObject: [[here]](https://www.freedesktop.org/software/gstreamer-sdk/data/docs/latest/gobject/howto-gobject.html) [↑](#endnote-ref-6)