Universidad La Salle

Informe Especificación Léxica

Pertenece a:

- Angela Milagros Quispe Huanca

Docente:

- MSc. Vicente Enrique Machaca Arceda

October 25, 2024

Contents

1 Introducción

El presente informe presenta la justificaci´on, los objetivos y la propuesta del nuevo lenguaje de programaci´on MAYC++. Este lenguaje surge de la necesidad de mejorar las herramientas disponibles para los desarrolladores de software en un entorno digital en constante evoluci´on. En un contexto donde la eficiencia, la claridad y la estandarizaci´on del c´odigo son fundamentales, MAYC++ ofrece una soluci´on innovadora: transformar autom´aticamente todo el c´odigo C++ a may´usculas. Esta caracter´ıstica distintiva facilita la legibilidad y uniformidad del c´odigo, permitiendo a los equipos de desarrollo mantener un estilo consistente en sus proyectos. Al aprovechar la potencia y flexibilidad de C++, combinado con la ventaja de una presentaci´on en may´usculas, MAYC++ se posiciona como una herramienta poderosa para el desarrollo de software moderno y de alta calidad.

1.1 Justificación

La creaci´on de MAYC++ responde a la necesidad de mejorar la claridad y la consistencia en la escritura de c´odigo en proyectos de desarrollo de software. Aunque C++ es conocido por su eficiencia y control a nivel del sistema, la legibilidad puede ser un desaf´ıo, especialmente en proyectos de gran escala con m´ultiples colaboradores. MAYC++ introduce una soluci´on innovadora al estandarizar todo el c´odigo en may´usculas, facilitando la lectura y asegurando que el c´odigo sea visualmente consistente. En un contexto donde los desarrolladores buscan lenguajes que ofrezcan tanto rendimiento como claridad, MAYC++ destaca al combinar la potencia de C++ con una presentaci´on uniforme que mejora la legibilidad del c´odigo. Este enfoque no solo facilita el mantenimiento y la revisi´on del c´odigo, sino que tambi´en ayuda a evitar errores comunes que surgen de la falta de estandarizaci´on en la escritura del c´odigo. Con la creciente demanda de herramientas que equilibren eficiencia y facilidad de uso, MAYC++ se posiciona como una alternativa que aprovecha las fortalezas de C++ mientras proporciona un formato visualmente claro y consistente. Al utilizar MAYC++, los desarrolladores pueden crear aplicaciones robustas y de alto rendimiento con menos esfuerzo y mayor claridad, optimizando tanto el tiempo de desarrollo como la calidad del c´odigo.

1.2 Objetivos

- Proporcionar a los programadores una herramienta que permita escribir c´odigo C++ en may´usculas de manera r´apida y eficiente, mejorando la legibilidad y la uniformidad en proyectos colaborativos. • Simplificar la adopci´on de C++ mediante la estandarizaci´on del c´odigo en may´usculas, lo que facilita la lectura y comprensi´on del c´odigo, especialmente para desarrolladores que desean mejorar su habilidad en C++ sin perder claridad. 3 • Ofrecer un entorno de desarrollo potente y vers´atil que sea adecuado para una amplia gama de aplicaciones, desde desarrollo

de sistemas hasta aplicaciones web y cient´ıficas, aprovechando la claridad visual y el rendimiento optimizado de MAYC++.

2 Propuesta

1. Código en mayúsculas por defecto, que mejora la visibilidad y la claridad del código, facilitando la lectura y revisión por parte de los desarrolladores.
2. Tipado estático estricto, aprovechando la seguridad que proporciona la verificación de tipos en tiempo de compilación, reduciendo así los errores en ejecución.
3. Compatibilidad total con bibliotecas y funcionalidades de C++, permitiendo el uso del amplio conjunto de recursos disponibles para este lenguaje, manteniendo la eficiencia y el rendimiento.
4. Integración transparente con código existente en C++, facilitando la adopción de mayc++ en proyectos ya desarrollados y permitiendo la cohabitación de diferentes estilos de código en un mismo proyecto.
5. Soporte para múltiples paradigmas de programación, como la programación orientada a objetos y la programación funcional, garantizando que los desarrolladores puedan utilizar el estilo que mejor se adapte a sus necesidades.

2.1 Especificacion Lexica

Esta especificación léxica proporciona una visión general de los componentes básicos presentes en el código fuente de MAYC++, un lenguaje diseñado para trabajar exclusivamente en mayúsculas.

2.1.1 Definicion de los Comentarios

Los comentarios en este lenguaje de programación pueden ser de dos tipos: comentarios de una sola línea y comentarios multilínea.

- Comentarios de una sola línea: Se definen con "//". Todo lo que sigue después de "//" en la misma línea se considera un comentario y es ignorado por el compilador.

  Expresión regular: "// .*"

- Comentarios multilínea: Se definen con "/*" para iniciar y "*/" para terminar. Todo el texto entre "/*" y "*/" se considera un comentario y es ignorado por el compilador.

  Expresión regular:"/* .* */"

2.1.2 Definicion de los Identificadores

Los identificadores son secuencias de letras, dígitos y guiones bajos que comienzan con una letra o un guión bajo. No pueden ser palabras clave ni literales.
Expresión regular: $([a-zA-Z]|_)([a-zA-Z]|[0-9]|_)*$

2.1.3 Definición de las Palabras Clave

Las palabras clave son palabras reservadas que tienen un significado especial en el lenguaje y no pueden ser utilizadas como identificadores. En MAYC++, las palabras clave están en mayúsculas y son las siguientes:

IF
ELSE
WHILE
DO
FOR
RETURN
MAIN

IN

FLOAT
STRING
CHAR
INT
DOUBLE
BOOLEAN: FALSE, TRUE

2.1.4 Definicion de los Literales

Los literales pueden ser números enteros, números decimales, caracteres entre comillas simples

- Números enteros: Secuencia de uno o más dígitos del conjunto [0-9].
  Expresión regular:[0-9]+
- Números decimales: Secuencia de uno o más dígitos seguidos por un punto decimal y luego una secuencia de uno o más dígitos.
  Expresión regular:[0-9]+.[0-9]+
- Un solo carácter o más entre comillas simples, como 'a', 'hola', etc.
  Expresión regular: " ' .*' "

2.1.5 Definicion de los Operadores

Los operadores son símbolos utilizados para realizar operaciones en las expresiones.
Algunos ejemplos comunes incluyen:

- Operadores aritmeticos: $+, -, *, /$
- Operadores de comparación (¿, ¡, ==, !=)
- Operadores lógicos &&, ||, !, etc.
- Lista de operadores: $+, -, *, /, \%, >, <, >=, <=, ==, != , \&\&, ||, !, =, ++, --, <<, >>, :$

2.2 Expresiones Regulares

TYPE(TOKEN)	EXPRESIÓN REGULAR
t_int	"INT"
t_float	"FLOAT"
t_bool	"BOOL"
t_char	"CHAR"
t_string	"STRING"
t_double	"DOUBLE"
ID	"$([A-Z]|\_)([A-Z]|[0-9]|\_)*$"
NUM	[0-9]+
LITERAL	":*"
RETURN	"RETURN"
IF	"IF"
WHILE	"WHILE"
COUT	"COUT"
co_R	"<<"
COMA	","
pyc	";"
par_L	"("
par_R	")"
lla_L	"{"
lla_R	"}"
o_equ	"="
o_2eq	"=="
o_may	">"
o_men	"<"
o_dif	"!="
o_sum	"+"
o_res	"-"
o_mul	""
o_div	"/"
T_MAIN	"MAIN"
T_ELSE	"ELSE"

2.3 CÓDIGO GENERAL

```
import ply.lex as lex
import csv
import re

# ANALIZADOR LEXICO
# Lista de Tokens
tokens = (
    'identificador', 'num', 'bool', 'texto', 'lla_L', 'lla_R', 'pyc',
    'dosp', 'punto', 'coma', 'dif', 'o_Ma', 'o_me', 'o_mayeq', 'o_meneq',
    'Mmin', 'Mmas', 'co_L', 'co_R', 'o_sum', 'o_mul', 'o_and',
    'o_div', 'o_res', 'o_equ', 'o_2eq', 'o_dif', 'par_L', 'par_R', 'while', 'for
    'if', 'else', 'do', 'break', 'return', 'case', 't_int', 't_double',
    't_char', 't_float', 't_string', 'boool',
    'com_lineal', 'com_mult', 'lit', 'main', 'cout', 'cin', 'endl'
)

# Expresiones regulares para cada token
t_identificador = r'[a-zA-Z][a-zA-Z0-9]*'
t_num = r'\d+'
t_bool = r'[VF]'
t_texto = r'\".*\"'
t_lla_L = r'\{'
t_lla_R = r'\}'
t_pyc = r';'
t_dosp = r':'
t_punto = r'\.'
t_coma = r','
t_dif = r'!='
t_o_Ma = r'>'
t_o_me = r'<'
t_o_mayeq = r'>='
t_o_meneq = r'<='
t_Mmin = r'--'
t_Mmas = r'\+\+'
t_co_L = r'<<'
t_co_R = r'>>'
t_o_sum = r'\+'
t_o_mul = r'\.\.'
t_o_and = r'&&'
t_o_div = r'/'
t_o_res = r'-'
```

```
t_o_equ = r'='
t_o_dif = r'!='
t_o_2eq = r'=='
t_par_L = r'\('
t_par_R = r'\)'
t_while = r'WHILE'
t_for = r'FOR'
t_if = r'IF'
t_else = r'ELSE'
t_do = r'DO'
t_break = r'BREAK'
t_return = r'RETURN'
t_case = r'CASE'
t_t_int = r'INT'
t_t_double = r'DOUBLE'
t_t_char = r'CHAR'
t_t_float = r'FLOAT'
t_t_string = r'STRING'
t_boool = r'BOOL'
t_com_lineal = r'//.*'
t_com_mult = r'/\*.*\*/'
t_lit = r'\'[^\']*\''
t_main = r'MAIN'
t_cout = r'COUT'
t_cin = r'CIN'
t_endl = r'ENDL'

t_ignore = ' \t'

def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

def t_error(t):
    print(f"Illegal character '{t.value[0]}'")
    t.lexer.skip(1)

# Construir el lexer
lexer = lex.lex()

def tokenize_code(input_code):
    # Darle la entrada al lexer
    lexer.input(input_code)

    # Lista para almacenar los tokens como diccionarios
    tokens_list = []
```

```
    # Tokenize
    while True:
        tok = lexer.token()
        if not tok:
            break  # No hay m s entrada
        tokens_list.append({'type': tok.type, 'value': tok.value, 'line': tok.lin

    return tokens_list

def write_tokens_to_file(tokens_list):
    # Escribir los tokens en Input.txt para procesamiento adicional si es necesa
    with open("Input.txt", "w") as f:
        for token in tokens_list:
            f.write(f"{token['type']} ")
        f.write("$\n")  # A adir $ al final despu s de todos los tokens

    # Escribir los detalles completos de cada token en Detalles.txt
    with open("Detalles.txt", "w") as f:
        for token in tokens_list:
            f.write(f"Type: {token['type']}, Value: {token['value']}, Line: {tok

def generate_dot_file(tokens_list):
    # Abrir el archivo .dot para escribir el grafo
    with open("graph.dot", "w") as f:
        f.write("digraph G {\n")

        # Escribir los nodos
        for i, token in enumerate(tokens_list):
            f.write(f'  node{i} [label="{token["type"]}: {token["value"]}"];\n')

        # Escribir las aristas
        for i in range(len(tokens_list) - 1):
            f.write(f'  node{i} -> node{i+1};\n')

        f.write("}")

# Leer el contenido del archivo de c digo fuente
with open('source_code.txt', 'r') as file:
    data = file.read()

# Obtener la lista de tokens
tokens_list = tokenize_code(data)

# Escribir tokens en archivos
write_tokens_to_file(tokens_list)
```

```
# Generar archivo .dot
generate_dot_file(tokens_list)
```