**MANUAL TÉCNICO**

**Elizabeth Carreño Alvarez**

**Carlos Hernán Gómez Gómez**

**Universidad Nacional De Colombia**

**Administración**

**Administración De Sistemas Informáticos**

**Manizales**

**2021**

[**RESUMEN**](#_tmye4sx39oc3) **3**

[**INTRODUCCIÓN**](#_286cyhxgpbfc) **4**

[**PASOS PARA EL DESARROLLO DE LA CH-MÁQUINA**](#_frml1ki9y3nj) **5**

[**ELABORACIÓN DEL CH-MÁQUINA**](#_ss965epwu2b) **5**

## 

## 

## RESUMEN

Esta aplicación tiene como objetivo principal la simulación de un sistema operativo para un computador de funcionamiento básico al cual se denominará CH-MÁQUINA. Para esto se cargan unos archivos, los cuales se verifica si tiene la sintaxis correcta definida en el lenguaje ch, esto para poder ejecutar y posteriormente mostrar los resultados arrojados por el algoritmo.

El CH-MÁQUINA es un ambiente de simulación diseñado gráficamente como un computador ficticio, que comprende un espacio determinado de memoria, el núcleo del sistema operativo (que para el caso de simulación no importa su contenido), procesador, componentes como monitor e impresora que permiten visualizar los resultados que se obtienen y otros elementos que también apoyan el proceso de visualización de la ejecución de un programa.

Para facilitar el uso de esta aplicación, el código del lenguaje empleado implica poseer conocimientos básicos de programación, así como de la arquitectura del computador, con el fin de comprender cómo funciona cada proceso. La CH-MÁQUINA permite conocer si existen errores en caso de haberlos con el fin de depurarlos, y así proceder a realizar la carga definitiva del programa en memoria para su ejecución.

## 

## 

## INTRODUCCIÓN

El CH-MÁQUINA posee cualidades que permiten la simulación del funcionamiento de un sistema operativo ficticio, como lo son la ejecución de programas (sólo programas con extensión .ch), tiene un espacio para la memoria que es manejado para programas, se presentan las diferentes instrucciones que están siendo ejecutadas para cada programa, es posible conocer los valores de los registros de los programas que se encuentran cargados en la memoria, el valor que va adquiriendo el acumulador y la instrucción que está en ejecución.

Cuenta con opciones de visualización de los resultados entregados durante la ejecución del programa que simulan una pantalla y una impresora.

Python es un intérprete que cuenta con una amplia cantidad de librerías, lo cual nos permite diversas herramientas para poder crear nuestros proyectos.

Con el fin de poder ejecutar la aplicación CH-MÁQUINA, se hace necesario contar con una serie de herramientas que permiten acceder al código fuente.

1- Configurar el lenguaje Python en el pc.

2- Instalación de IDEs (Entorno de desarrollo integrado).

## 

## PASOS PARA EL DESARROLLO DE LA CH-MÁQUINA

• Creación de carpetas para desarrollo

Durante el desarrollo de la aplicación el primer paso que se tuvo en cuenta fue la creación de unas carpetas que serían utilizadas para guardar los diferentes módulos o archivos que se generarán en el desarrollo.

Nota: Es necesario tener instalado el Python en la computadora donde vamos a desarrollar el CH-MÁQUINA y también donde lo vayamos a correr es este caso se está trabajando con Python 3.

•Instalación de los IDE’s

“Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI)”.

## ELABORACIÓN DEL CH-MÁQUINA

En el archivo app.py se tiene la interfaz en una ventana que permite ingresar los valores de kernel y memoria con sus respectivos métodos para que funcione el simulador. El simulador se encuentra en otra ventana donde permite la carga y ejecución de programas ch con sus respectivos métodos de carga de archivos .ch, validación de sintaxis y ejecución.

from tkinter import \*

# libreria para poder abrir y guardar archivos

from tkinter import ttk

from tkinter import filedialog

#from pagina\_principal import ventana\_principal

#Python Imaging Library (PIL) es una librería gratuita que permite la edición de imágenes directamente desde Python. Soporta una variedad de formatos, incluídos los más utilizados como GIF, JPEG y PNG. Una gran parte del código está escrito en C

from PIL import Image, ImageTk

from tkinter import messagebox

import memoria\_kernel

#root de la app

ventana = Tk()

ventana.title("Datos memoria y kernel")

#ventana.config(bg="white")

ventana.geometry("500x200")

global z

z = 1 #ultimo digito de la cedula 1053835141

global instrucciones\_archivo

instrucciones\_archivo = []

global memoria\_principal

memoria\_principal = [100]

global kernel

kernel=10\*z+9

global memoria

memoria=100

global errores

errores = []

#dicccionario de variables

global variables

variables = {}

global etiquetas

etiquetas = {}

global lista\_rutas

lista\_rutas = []

#inserta el sistema operativo en la memoria principal

i = 0

def insertar\_kernel(memoria\_principal, kernel):

global i

if i < 1:# borra un nuevo kernel

i += 1

memoria\_principal.append("ACOMULADOR")

for i in range(1, int(kernel)+1):

memoria\_principal.append("SISTEMA OPERATIVO")

#print(memoria\_principal)

#verifica si al establecer el kernel hay espacio o no para cambiarlo por ese valor

def verificar\_memoria\_kernel(memoria, kernel):

print(memoria)

if (int(memoria.isalpha())==True or not(memoria) or int(kernel.isalpha())==True or not(kernel)): #comprueba si el dato ingresado es un digito o esta vacio el campo que se debe ingresar

messagebox.askokcancel(message="Datos incorrectos, el dato ingresado es obligatorio y debe ser un digito")

elif (int(kernel) > 0 and int(memoria) > 0 and int(memoria) <= 9999 and int(kernel) < int(memoria)):

insertar\_kernel(memoria\_principal, kernel)

memoria\_kernel.insertar\_memoria\_y\_kernel(memoria, kernel)

msgbox = messagebox.askquestion(message="Datos correctos!!")

if msgbox == 'yes':

ventana.destroy()

else:

messagebox.askokcancel(message="Datos incorrectos, la memoria debe ser menor a 9999 posiciones y el tamaño del kernel debe ser menor al tamaño de la memoria")

#titulo para la entrada de datos

Label(text="Ingresa los tamaños de la memoria y el kernel").grid(row=1, column=2)

#creamos input de memoria

Label(text="Memoria").grid(row=2, column=2, sticky='nw')

entrada\_memoria = StringVar()

entrada\_memoria.set(100)

entradaM = Entry(ventana, textvariable=entrada\_memoria)

entradaM.grid(row=2, column=3)

boton\_memoria\_kernel = Button(ventana, text="Validar Memoria y Kernel", command=lambda: verificar\_memoria\_kernel((entrada\_memoria.get()),(entrada\_kernel.get()))).grid(row=4, column=2)

#creamos input de kernel

Label(text="Kernel").grid(row=3, column=2, sticky='nw')

entrada\_kernel = StringVar()

entrada\_kernel.set(kernel) #10\*z+9 z=1

entradaK = Entry(ventana, textvariable=entrada\_kernel)

entradaK.grid(row=3, column=3)

#carga ventana 1

ventana.mainloop()

#ventana 2

ventana\_principal = Tk()

ventana\_principal.geometry("3800x3000")

#realizamos el menu de opciones

menubar = Menu(ventana\_principal)

filemenu = Menu(menubar, tearoff=0)

filemenu.add\_command(label="Abrir", command=lambda: abrir\_archivo())

menubar.add\_cascade(label="Archivo", menu=filemenu)

menubar.add\_command(label="Ejecute", command=lambda: verificar\_sintaxis(instrucciones\_archivo))

menubar.add\_cascade(label="Muestre memoria")

menubar.add\_cascade(label="Pausa")

menubar.add\_cascade(label="Paso a paso")

menubar.add\_cascade(label="Salir")

ventana\_principal.configure(menu=menubar)

#creamos input de acomulador

Label(ventana\_principal,text="Acomulador").grid(row=3, column=0, sticky='nw')

entrada\_acomulador = IntVar()

entradaA = Entry(ventana\_principal, textvariable=entrada\_acomulador)

entradaA.grid(row=3, column=1)

#carga de instrucciones de los archivos

treeview\_archivos = ttk.Treeview(ventana\_principal, height=10, columns=2)

scrollbar = ttk.Scrollbar(ventana\_principal, orient = "vertical", command=treeview\_archivos.yview )

scrollbar.grid(row=6, column=1, sticky="nse")

treeview\_archivos.configure(yscrollcommand=scrollbar.set)

treeview\_archivos.grid(row=6, column=1, sticky='nsew')

#caja de texto para las variables

texto\_variables = Text(ventana\_principal,width=20, height=20)

texto\_variables.grid(row=6, column=2)

#imagen del computador

computer\_image = ImageTk.PhotoImage(Image.open("./img/computadora.png"))

computer\_image\_label = Label(ventana\_principal, image=computer\_image)

computer\_image\_label.grid(row=1, column=3, padx=10, pady=10)

boton\_kernel\_usuario = Button(ventana\_principal, text="Modo Usuario").grid(row=2, column=3)

#mostrar lo del pc

carga\_datos\_pantalla = Listbox(ventana\_principal, width=20, height=20)

scroll = Scrollbar(ventana\_principal, orient = VERTICAL)

#Creo un menu desplegable para establecer modo kernel o modo usuario

opciones=["kernel","usuario"]#Estas seran las opciones

var=StringVar()#es la variable que se encontrara como primera opcion

var.set(opciones[0])

#mostrar instrucciones paso a paso

treeview\_archivos\_paso\_a\_paso = ttk.Treeview(ventana\_principal, columns=2, height=10)

scrollbar = ttk.Scrollbar(ventana\_principal, orient=VERTICAL, command=treeview\_archivos\_paso\_a\_paso.yview)

scrollbar.grid(row=6, column=4, sticky="ns", padx="7")

treeview\_archivos\_paso\_a\_paso.configure(yscrollcommand=scrollbar.set)

treeview\_archivos\_paso\_a\_paso.grid(row=6, column=4, sticky="nsew")

#metodo para cargar los archivos en la caja de texto con scroll

def abrir\_archivo():

global instrucciones\_archivo

archivo\_abierto=filedialog.askopenfilename(initialdir="/Documents/ch-maquina/programs",

title="Seleccione archivo",filetypes=(("ch files","\*.ch"),

("all files","\*.\*")))

if archivo\_abierto!='':

archivo=open(archivo\_abierto, "r", )

instrucciones\_archivo = archivo.readlines()#Conversión del archivo de texto en una lista por renglones

archivo.close()

for i in range(0, len(instrucciones\_archivo)):

treeview\_archivos.insert("" , 'end', text="00" + str(i+1), values= (instrucciones\_archivo[i],))

#verificar sintaxis

def verificar\_sintaxis(instrucciones\_archivo):

archivo = instrucciones\_archivo

palabra = []

for instruccion in archivo:

print(instruccion)

instruccion = instruccion.strip("\n")

palabra = instruccion.split(" ")

if(palabra[0] == "cargue"):

funcion\_error\_cargue\_almacene\_multiplique\_sume\_reste(palabra)

elif(palabra[0] == "almacene"):

funcion\_error\_cargue\_almacene\_multiplique\_sume\_reste(palabra)

elif(palabra[0] == "nueva"):

funcion\_error\_nueva(palabra)

elif(palabra[0] == "lea"):

funcion\_error(palabra)

elif(palabra[0] == "sume"):

funcion\_error\_cargue\_almacene\_multiplique\_sume\_reste(palabra)

elif(palabra[0] == "reste"):

funcion\_error\_cargue\_almacene\_multiplique\_sume\_reste(palabra)

elif(palabra[0] == "multiplique"):

funcion\_error\_cargue\_almacene\_multiplique\_sume\_reste(palabra)

elif(palabra[0] == "divida"):

funcion\_error\_divida\_modulo(palabra)

elif(palabra[0] == "potencia"):

funcion\_error\_potencia(palabra)

elif(palabra[0] == "modulo"):

funcion\_error\_divida\_modulo(palabra)

elif(palabra[0] == "concatene"):

funcion\_error\_concatene\_extraiga(palabra)

elif(palabra[0] == "elimine"):

funcion\_error(palabra)

elif(palabra[0] == "extraiga"):

funcion\_error\_concatene\_extraiga(palabra)

elif(palabra[0] == "Y"):

funcion\_error\_y\_o\_no(palabra)

elif(palabra[0] == "O"):

funcion\_error\_y\_o\_no(palabra)

elif(palabra[0] == "NO"):

funcion\_error\_y\_o\_no(palabra)

elif(palabra[0] == "muestre"):

funcion\_error(palabra)

elif(palabra[0] == "imprima"):

funcion\_error(palabra)

elif(palabra[0] == "vaya"):

funcion\_error\_vaya(palabra)

elif(palabra[0] == "vayasi"):

funcion\_error\_vaya\_si(palabra)

elif(palabra[0] == "etiqueta"):

funcion\_error\_etiqueta(palabra)

elif(palabra[0] == "retorne"):

funcion\_error\_retorne(palabra)

else:

funcion\_error\_comentario(palabra)

#verficar sintaxis de las operaciones lea, muestra, imprima, elimine

def funcion\_error(palabra):

if(len(palabra)>2):

errores.append("Error, se estan utilizando mas de 2 operandos en la operacion "+ palabra[0])

elif(palabra[1] not in variables):

errores.append("Error, la variable " + palabra[1] + " no ha sido asignada")

print(errores)

#error comentarios o operaciones no declaradas

def funcion\_error\_comentario(palabra):

comentario = ['//', ' ']

if(palabra[0] in comentario):

errores.append("Error, no es una operación valida " + palabra[1])

#error nueva

def funcion\_error\_nueva(palabra):

global variables

tipos\_datos = ["C", "I", "R", "L"]

if(len(palabra)>4):

errores.append("Error, se estan utilizando mas de 4 operandos en la operacion "+ palabra[0])

elif((palabra[2]=="I") and (palabra[3].isdigit()==False)):

errores.append("Error, el valor de inicializacion " + palabra[3] + " no es correcto, en el tipo de dato " + palabra[2])

elif((palabra[2]=="L") and not(palabra[3] == "1" or palabra[3] == "0")):

errores.append("Error en el el valor de inicializacion " + palabra[3] + " no es correcto, en el tipo de dato " + palabra[2])

elif((palabra[2]=="C") and (palabra[3].isalpha()==False)):

errores.append("Error en el el valor de inicializacion " + palabra[3] + " no es correcto, en el tipo de dato " + palabra[2])

elif((palabra[2]=="R") and (palabra[3].isdecimal()==False)):

errores.append("Error en el el valor de inicializacion " + palabra[3] + " no es correcto, en el tipo de dato " + palabra[2])

elif(palabra[1] not in variables):

funcion\_variable\_nombre\_valido\_variable(palabra)

elif(palabra[2] not in tipos\_datos):

errores.append("Error, el tipo de dato especificado " + + palabra[2] + " no ha sido declarado" )

#verifica que el nombre no sea una operacion

def funcion\_variable\_nombre\_valido\_variable(palabra):

palabras\_operaciones = ["cargue", "almacene","nueva", "lea", "sume", "reste", "multiplique", "divida", "potencia", "modulo", "concatene", "elimine", "extraiga", "Y", "O", "muestre", "vaya", "vayasi", "etiqueta", "retorne"]

if(palabra[1] in palabras\_operaciones):

errores.append("Error, el nombre de la variable " + palabra[1] +" no es válido" )

else:

variables[palabra[1]] = { 'tipo': palabra[2], 'valor': palabra[3] }

#validan los errores de las funciones cargue, almacene, multiplique, sume, reste

def funcion\_error\_cargue\_almacene\_multiplique\_sume\_reste(palabra):

if(len(palabra)>2):

errores.append("Error, se estan utilizando mas de 2 operandos en la operacion "+ palabra[0])

elif(palabra[1] not in variables):

errores.append("Error, la variable " + palabra[1] + " no ha sido asignada")

elif((variables[palabra[1]]['tipo'] == "L") or (variables[palabra[1]]['tipo'] == "C")):

errores.append("Error, la variable " + palabra[1] + " no se puede ejecutar en la operacion " + palabra[0] + " porque su tipo de dato es " + variables[palabra[1]]['tipo'])

print(errores)

#validan los errores de las funciones divida, modulo

def funcion\_error\_divida\_modulo(palabra):

if(len(palabra)>2):

errores.append("Error, se estan utilizando mas de 2 operandos en la operacion "+ palabra[0])

elif(palabra[1] not in variables):

errores.append("Error, la variable " + palabra[1] + " no ha sido asignada")

elif((variables[palabra[1]]['tipo'] == "L") or (variables[palabra[1]]['tipo'] == "C")):

errores.append("Error, la variable " + palabra[1] + " no se puede ejecutar en la operacion " + palabra[0] + " porque su tipo de dato es " + variables[palabra[1]]['tipo'])

elif(variables[palabra[1]]['valor'] == "0"):

errores.append("Error, la operacion "+ palabra[0] + " no permite dividir entre " + variables[palabra[1]]['valor'] )

#valida error de la potencia

def funcion\_error\_potencia(palabra):

if(len(palabra)>2):

errores.append("Error, se estan utilizando mas de 2 operandos en la operacion "+ palabra[0])

elif(palabra[1] not in variables):

errores.append("Error, la variable " + palabra[1] + " no ha sido asignada")

elif((variables[palabra[1]]['tipo'] == "L") or (variables[palabra[1]]['tipo'] == "C")):

errores.append("Error, la variable " + palabra[1] + " no se puede ejecutar en la operacion " + palabra[0] + " porque su tipo de dato es " + variables[palabra[1]]['tipo'])

elif((variables[palabra[1]]['valor'].isdigit()==False) or not(variables[palabra[1]]['valor'] <= 0)):

errores.append("Error, la operacion "+ palabra[0] + " no permite elevar a una portencia " + variables[palabra[1]]['valor'] )

#error concatene

def funcion\_error\_concatene\_extraiga(palabra):

if(len(palabra)>2):

errores.append("Error, se estan utilizando mas de 2 operandos en la operacion "+ palabra[0])

elif(palabra[1] not in variables):

errores.append("Error, la variable " + palabra[1] + " no ha sido asignada")

elif((variables[palabra[1]]['tipo'] != "C")):

errores.append("Error, la variable " + palabra[1] + " no se puede ejecutar en la operacion " + palabra[0] + " porque su tipo de dato es " + variables[palabra[1]]['tipo'])

elif((palabra[1].isalpha())==False):

errores.append("Error, el operando no es alfanumerico "+ palabra[0])

#error y\_o

def funcion\_error\_y\_o\_no(palabra):

if(len(palabra)>4):

errores.append("Error, se estan utilizando mas de 4 operandos en la operacion " + palabra[0])

elif(palabra[1] not in variables):

errores.append("Error, la variable " + palabra[1] + " no ha sido asignada")

elif((variables[palabra[1]]['tipo'] != "L")):

errores.append("Error, la variable " + palabra[1] + " no se puede ejecutar en la operacion " + palabra[0] + " porque su tipo de dato es " + variables[palabra[1]]['tipo'])

#error etiqueta

def funcion\_error\_etiqueta(palabra):

if(len(palabra)>3):

errores.append("Error, se estan utilizando mas de 2 operandos en la operacion "+ palabra[0])

elif(palabra[2].isdigit()==False):

errores.append("Error, el valor " + palabra[2] + " no es válido para la operación " + palabra[0] )

elif(palabra[1] not in etiquetas):

funcion\_variable\_nombre\_valido\_etiqueta(palabra)

#verifica que el nombre no sea una operacion

def funcion\_variable\_nombre\_valido\_etiqueta(palabra):

palabras\_operaciones = ["cargue", "almacene","nueva", "lea", "sume", "reste", "multiplique", "divida", "potencia", "modulo", "concatene", "elimine", "extraiga", "Y", "O", "muestre", "vaya", "vayasi", "etiqueta", "retorne"]

if(palabra[1] in palabras\_operaciones):

errores.append("Error, el nombre de la de etiqueta " + palabra[1] +" no es válido" )

else:

etiquetas[palabra[1]] = { 'valor': palabra[2] }

#error\_retorne

def funcion\_error\_retorne(palabra):

if(len(palabra)>2):

errores.append("Error, se estan utilizando mas de 2 operandos en la operacion "+ palabra[0])

elif(palabra[1].isdigit()==False):

errores.append("Error, el valor " + palabra[1] +" no es válido para la operación " + palabra[0])

#error\_vaya

def funcion\_error\_vaya(palabra):

if(len(palabra)>2):

errores.append("Error, se estan utilizando mas de 2 operandos en la operacion "+ palabra[0])

elif(palabra[1] not in etiquetas):

errores.append("Error, la etiqueta " + palabra[1] + " no ha sido asignada")

elif(etiquetas[palabra[1]]['valor'].isdigit()==False):

errores.append("Error, el valor " + etiqueta[palabra[1]]['valor'] + " no es válido para la operación " + palabra[0] )

#error\_vaya\_si

def funcion\_error\_vaya\_si(palabra):

if(len(palabra)>3):

errores.append("Error, se estan utilizando mas de 3 operandos en la operacion "+ palabra[0])

elif(palabra[1] not in etiquetas):

errores.append("Error, la etiqueta " + palabra[1] + " no ha sido asignada")

elif(palabra[2] not in etiquetas):

errores.append("Error, la etiqueta " + palabra[2] + " no ha sido asignada")

elif(etiquetas[palabra[1]]['valor'].isdigit()==False):

errores.append("Error, el valor " + etiqueta[palabra[1]]['valor'] + " no es válido para la operación " + palabra[0] )

elif(etiquetas[palabra[2]]['valor'].isdigit()==False):

errores.append("Error, el valor " + etiqueta[palabra[2]]['valor'] + " no es válido para la operación " + palabra[0] )

ventana\_principal.mainloop()