Manual de Usuario

Este manual ayuda sobre como ejecutar el compilador alojado en el repositorio de GitHub.

1. Tener instalado SO Linux/Ubuntu, ya que sobre este se trabajará.

```
alexander@alexander-VirtualBox:~$ cat /proc/version
Linux version 5.4.0-53-generic (buildd@lcy01-amd64-007) (gcc version 9.3.0 (Ubuntu 9.3.0-1
7ubuntu1~20.04)) #59-Ubuntu SMP Wed Oct 21 09:38:44 UTC 2020
```

   Se trabaja en un procesador de 64 bits.
   Versión linux: 5.4.0-53-generic.
   Versión Ubuntu: 20.04.
2. Instalar Git.
   - Para instalar este se debe ejecutar los siguientes comandos:
     sudo apt-get update

```
alexander@alexander-VirtualBox:~$ apt-get update
```

   - Se coloca el código:
     sudo apt install git

```
alexander@alexander-VirtualBox:~$ sudo apt install git
[sudo] contraseña para alexander:
```

     Deberás colocar tu contraseña (si es el dado caso que tienes) para poder validar la instalación.

3. Instalar Elixir.
   - Añadir el repositorio de Erlang:
     wget https://packages.erlang-solutions.com/erlang-solutions_2.0_all.deb && sudo dpkg -i erlang-solutions_2.0_all.deb

```
alexander@alexander-VirtualBox:~$ wget https://packages.erlang-solutions.com/erlang-soluti
ons_2.0_all.deb && sudo dpkg -i erlang-solutions_2.0_all.deb
```

   - Ejecutar
     sudo apt-get update

```
alexander@alexander-VirtualBox:~$ apt-get update
```

   - Instale la plataforma Erlang / OTP y todas sus aplicaciones:
     sudo apt-get install esl-erlang

```
alexander@alexander-VirtualBox:~$ sudo apt-get install esl-erlang
```

4. Instalar GCC.

- Instalación gcc.

  sudo apt-get install gcc

```
alexander@alexander-VirtualBox:~$ sudo apt-get install gcc
```

5. Instalar make.
   - Ejecutar

     sudo apt-get update

```
alexander@alexander-VirtualBox:~$ apt-get update
```

   - Instale make.

     sudo apt-get install make

```
alexander@alexander-VirtualBox:~$ sudo apt-get install make
```

6. Instalar bash
   - Ejecutar

     sudo apt-get update

```
alexander@alexander-VirtualBox:~$ apt-get update
```

   - Instale bash.

     sudo apt-get install bash

```
alexander@alexander-VirtualBox:~$ sudo apt-get install bash
```

7. Instalar lenguaje ensamblador.
   - Ejecutar

     sudo apt-get install nasm build-essential

```
alexander@alexander-VirtualBox:~$ sudo apt-get install nasm build-essential
[sudo] contraseña para alexander:
```

   Deberás colocar tu contraseña (si es el dado caso que tienes) para poder validar la instalación.

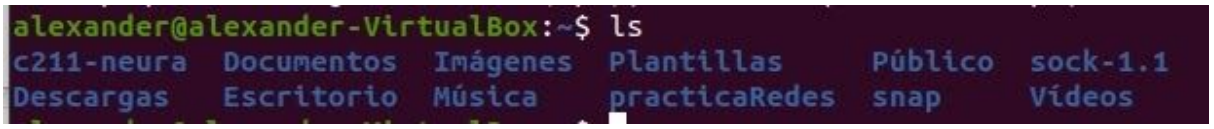
8. Acceso al Repositorio.
   - Ejecutar

     git clone https://github.com/hiphoox/c211-neura.git

```
alexander@alexander-VirtualBox:~$ git clone https://github.com/hiphoox/c211-neura.git
```

```
alexander@alexander-VirtualBox:~$ git clone https://github.com/hiphoox/c211-neura.git
Clonando en 'c211-neura'...
Username for 'https://github.com': AlexanderReyesMarin
Password for 'https://AlexanderReyesMarin@github.com':
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 1), reused 9 (delta 1), pack-reused 0
Desempaquetando objetos: 100% (9/9), 1.17 KiB | 399.00 KiB/s, listo.
```

Anexar tu nombre de usuario y contraseña GITHUB para validar la clonación del repositorio correspondiente.



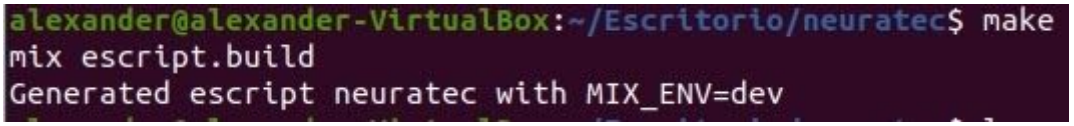
vemos los archivos y directorios que tenemos dentro del directorio en el que estamos, en este caso en home.

Podemos ubicar la carpeta de nombre *c211-neura* o *neuratec* (puede variar el nombre pero puede ser cualquiera de las dos) en cualquier directorio dentro de nuestro espacio de trabajo.

9. Carpeta neuratec.

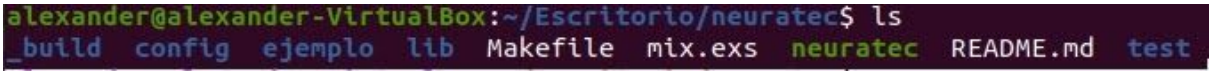
   Ya estamos en la Carpeta *c211-neura* o *neuratec* y revisamos su contenido con el comando *ls*

Escribimos make en la terminal y nos genera el ejecutable.
make: con este se ejecuta mix escript.build



Códigos similares que son aplicables en este caso ‘mix run’, ‘mix compile’

Se genera el ejecutable neuratec.



Si al realizar el paso anterior genera errores, se tendrá que acceder manualmente al archivo mix.exs y se tendrá que cambiar a nuestra versión de elixir.

10. Ejecución de neuratec.

Dentro de la carpeta ejemplo existe un programa en C: return_2.c, es el que vamos a necesitar para ejecutar nuestro ejecutable neuratec.

./neuratec ./ejemplo/return_2.c

```
alexander@alexander-VirtualBox:~/Escritorio/neuratec$ ./neuratec ./ejemplo/return_2.c
Compiling file: ./ejemplo/return_2.c
```

```
Salida sanitizante: ["int", "main(){return", "2;}"]

Salida lexer: [
  :int_keyword,
  :main_keyword,
  :open_paren,
  :close_paren,
  :open_brace,
  :return_keyword,
  {:constant, 2},
  :semicolon,
  :close_brace
]

Salida parser: %AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: nil,
        node_name: :constant,
        right_node: nil,
        value: 2
      },
      node_name: :return,
      right_node: nil,
      value: nil
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

```
Salida Generador de codigo:

.text
.p2align 4

.globl  main
.type    main, @function
main:
    movl    $2, %eax
    ret
```

OTRO EJEMPLO:

Debemos tener en cuenta la ruta de la carpeta valid, dentro de esta carpeta tenemos las demás pruebas válidas a probar.

cd test/stage_1/valid

El contenido de esta carpeta es:

```
alexander@alexander-VirtualBox:~/Escritorio/neuratec/test/stage_1/valid$ ls
Makefile       multi_digit.s  no_newlines.c  return_0.s  spaces.c     tabulador.s
multi_digit.c  newlines.c     no_newlines.s  return_2.c  spaces.s
multi_digitc   newlines.s     return_0.c     return_2.s  tabulador.c
```

Usaremos para nuestro siguiente ejecución a return_0.c

./neuratec ./test/stage_1/valid/return_0.c

```
alexander@alexander-VirtualBox:~/Escritorio/neuratec$ ./neuratec ./test/stage_1/valid/return_0.c
Compiling file: ./test/stage_1/valid/return_0.c
```

```
Salida sanitizante: ["int", "main()", "{", "return", "0;", "}"]

Salida lexer: [
  :int_keyword,
  :main_keyword,
  :open_paren,
  :close_paren,
  :open_brace,
  :return_keyword,
  {:constant, 0},
  :semicolon,
  :close_brace
]

Salida parser: %AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: nil,
        node_name: :constant,
        right_node: nil,
        value: 0
      },
      node_name: :return,
      right_node: nil,
      value: nil
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

```
Salida Generador de codigo:

.text
.p2align 4

.globl  main
.type    main, @function
main:
    movl    $0, %eax
    ret
```

11. Pruebas automatizadas.

    Debemos ir a la carpeta valid, dentro de esta carpeta tenemos un make que al momento de escribir *make ejecutar,* se ejecutan los programas en c y se crea un ensamblador de los mismos.

    cd test/stage_1/valid

    make ejecutar

```
alexander@alexander-VirtualBox:~/Escritorio/neuratec$ cd test/stage_1/valid
alexander@alexander-VirtualBox:~/Escritorio/neuratec/test/stage_1/valid$ make ejecutar
gcc multi_digit.c
gcc multi_digit.c -o multi_digit
gcc -S -O3 -fno-asynchronous-unwind-tables multi_digit.c
gcc newlines.c
gcc newlines.c -o newlines
gcc -S -O3 -fno-asynchronous-unwind-tables newlines.c
gcc no_newlines.c
gcc no_newlines.c -o no_newlines
gcc -S -O3 -fno-asynchronous-unwind-tables no_newlines.c
gcc return_0.c
gcc return_0.c -o return_0
gcc -S -O3 -fno-asynchronous-unwind-tables return_0.c
gcc return_2.c
gcc return_2.c -o return_2
gcc -S -O3 -fno-asynchronous-unwind-tables return_2.c
gcc spaces.c
gcc spaces.c -o spaces
gcc -S -O3 -fno-asynchronous-unwind-tables spaces.c
gcc tabulador.c
gcc tabulador.c -o tabulador
gcc -S -O3 -fno-asynchronous-unwind-tables tabulador.c
```

Volvemos a la carpeta test

cd ../

cd ../

En la carpeta test, dentro de esta carpeta tenemos un make que al momento de escribir *make correr_pruebas,* verifican los programas de las carpetas valid e invalid.

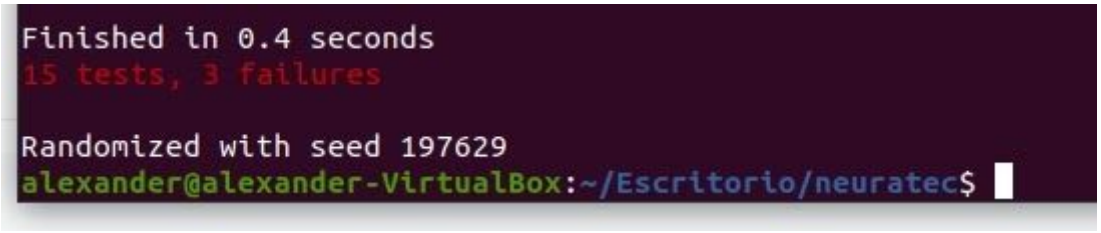
```
alexander@alexander-VirtualBox:~/Escritorio/neuratec/test$ make correr_pruebas
./test_compiler.sh /path/to/your/compiler 1
==================================================
STAGE 1
====================Valid Programs====================
tabulador.....................................OK
newlines......................................OK
multi_digit...................................OK
no_newlines...................................OK
return_0......................................OK
spaces........................................OK
return_2......................................OK
====================Invalid Programs==================
boolean.......................................OK
doble_punto...................................OK
missing_paren.................................OK
missing_retval................................OK
no_brace......................................OK
no_semicolon..................................OK
no_space......................................OK
nt_missing....................................OK
wrong_case....................................OK
====================Stage 1 Summary=================
16 successes, 0 failures
====================TOTAL SUMMARY====================
16 successes, 0 failures
```

12. MIX TEST.

Debemos estar en el directorio de neuratec y ejecutamos el siguiente comando.

mix test

Ejecuta las pruebas de un proyecto, pero solo hasta el lexer.



Para salir de la línea de comandos o terminal solo escribir:

exit