



Tecnológico de Monterrey

MANUAL DE USUARIO Alebrije

Profesores:
Elda G. Quiroga, M.C.
Dr. Héctor Ceballos, PhD

Gerardo Galan Garzafox

A00821196

24 de Noviembre del 2021

Indice

Indice	2
Introduccion	3
Instalacion	4
Explicacion	6
Variables	6
Tipos de dato	6
Asignacion	7
Alcance	8
Estatutos de Interacción	8
Lectura	8
Escritura	9
Expresiones aritmética, lógica y relacional	9
Control de flujo	10
Decisiones	10
Ciclo Condicional	11
Ciclo de Rango	12
Modulos	13
Funciones void	13
Funciones con retorno	14
Arreglos	15
Funciones predefinidas	16
Estadistica	16
Mean	16
Median	17
Mode	17
Variance	18
Plotxy	18
Regression	19
Asistencia	20
Len	20
Min	21
Max	21
Video ejemplo	22

Introduccion

Alebrije es un lenguaje de programación compilado para gente que se está introduciendo al mundo de la programación. Este lenguaje cuenta con las mismas necesidades básicas que casi todos los lenguajes de programación ofrecen: manejo de variables; tipos de datos; entradas y salidas; control de flujo; programación modular; estructura de datos simples(arreglos); además cuenta con funciones para cálculos básicos de estadística.

Instalacion

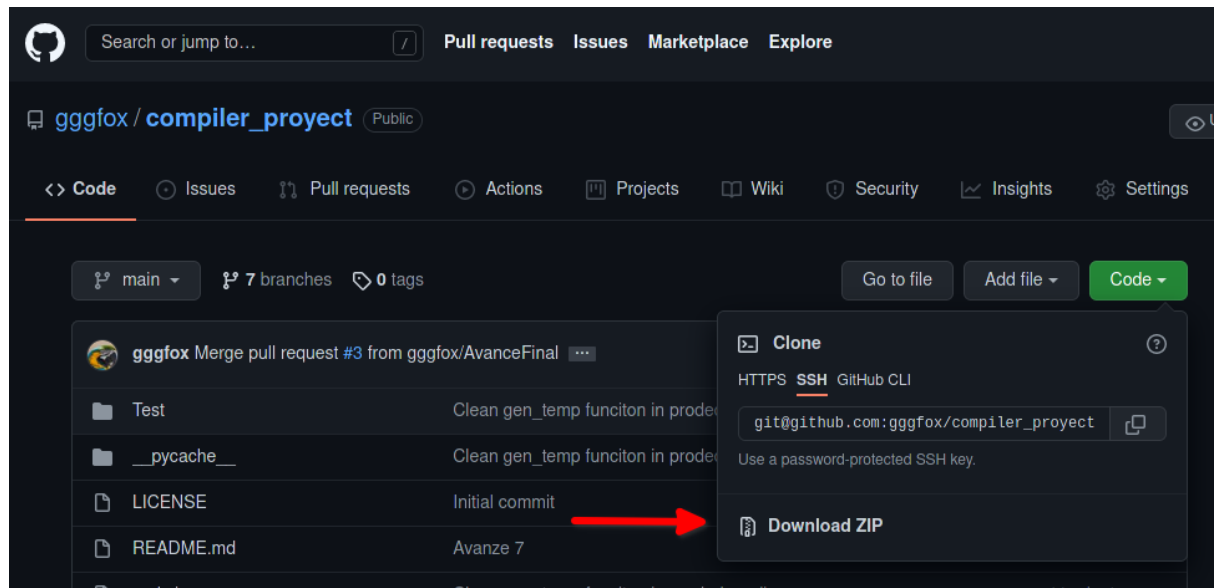
- El 1er paso para empezar a programar en alebrije es tener instalada en tu computadora una versión del lenguaje de programación python, versión 3 en adelante (<https://www.python.org/downloads/>) y el manejador de paquetes de python pip (<https://pypi.org/project/pip/>) .

- El 2do paso es instalar los paquetes de python necesarios para correr alebrije, esto se logra ejecutando el siguiente comando en una terminal.

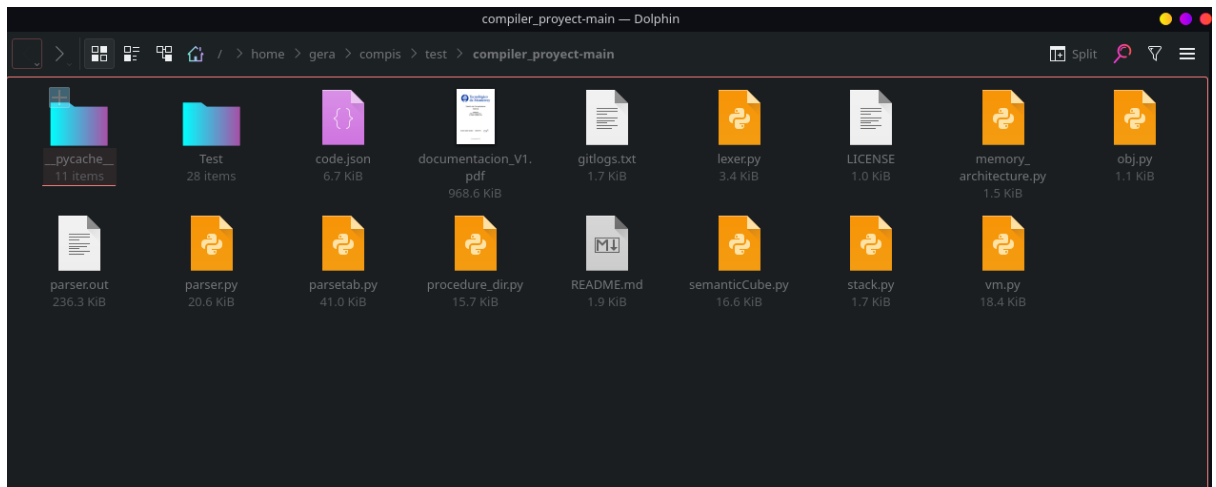
```
pip install scipy numpy statistics ply matplotlib tabulate
```

- El 3er paso es descargar un ZIP del proyecto de la siguiente pagina:

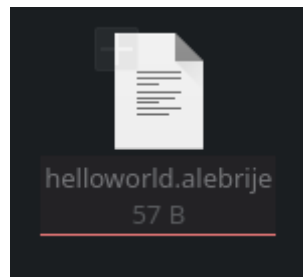
https://github.com/gggfox/compiler_proyect



- El 4to paso es únicamente descomprimir el archivo ZIP en una carpeta, el archivo descomprimido se vería algo así.



- El 5to paso consiste en crear un archivo de texto llamado helloworld.alebrije dentro de la carpeta que descomprimimos.



- Dentro de este archivo escribir el siguiente bloque de código, que consiste en el típico programa inicial de cualquier lenguaje de programación que imprime en pantalla un “hello world”.

```
program HelloWorld;

main(){
    write("hello world");
}
```

- El 6to paso es escribir el siguiente comando en la terminal, en el mismo directorio del proyecto.

```
python parser.py helloworld.alebrije
```

Si todo salió bien te debería de verse así:

```
~/compis/test/compiler_project-main > git P mai?8 python parser.py helloworld.alebrije
hello world
```

Explicacion

Como vimos en la introducción lo mínimo que necesitamos para correr un programa en alebrije es en primer lugar usar la palabra clave program seguida por un nombre de programa que empiece con letras (minuscula o mayuscula), seguida por más letras o números, por último un punto y coma. Luego se debe de tener la palabra clave main() con un par de corchetes {}, dentro de esos puede haber varias cosas, en el caso de este ejemplo hay un write que es una función del sistema que imprime en pantalla expresiones o strings.

```
program HelloWorld;  
  
main(){  
    write("hello world");  
}
```

Variables

Tipos de dato

Alebrije cuenta con 5 tipos de datos int,float, bool, char y string

Tipo de dato	Ejemplos de valores posibles	
int	5	100
float	3.14	0.7
bool	true	false
char	"a"	"r"
string	"hola"	"world"

La manera en la que se declaran variables es la siguiente.

```
program TipoDeDato;

main(){
    vars{
        int: entero, numero;
        float: decimal;
        bool: booleano;
        char: caracter;
        string: cadenaDeCaracteres;
    }
    write("hello world");
}
```

Como se puede observar en el ejemplo previo se escribe un la palabra clave vars justo debajo del main. Dentro de vars se pone el tipo de dato seguido por 2 puntos luego el nombre de la variable que es definido por el usuario seguido mas comas y variables o un punto y coma. Una vez que se termina de declarar las variables de un tipo de dato se pueden declarar las de otro tipo de dato, o del mismo, siempre y cuando estén dentro de los corchetes de vars.

Asignacion

La manera en la que se le asigna un valor a una variable es la siguiente.

```
program Asignacion;

main(){
    vars{
        int: entero, numero;
        float: decimal;
        bool:t;
        char: caracter;
        string: cadenaDeCaracteres;
    }

    entero = 100;
    decimal = 0.7;
    cadenaDeCaracteres = "hello world";
    write(cadenaDeCaracteres);
}
```

Al momento de ejecutar este programa la salida en consola es `hello world`

Como se puede observar la variable debe de ir del lado izquierdo seguido por un igual, el valor a asignar, por último un punto y coma. Cabe resaltar que el valor a asignar debe de ser uno que sea aceptado por el tipo de dato.

Alcance

Como podemos observar en el siguiente ejemplo se puede tener un vars como el que se tiene en main, debajo de program. Sin embargo la diferencia es que el alcance de la variable entera glob, es global, esto significa que no se tiene que volver a declarar en main u otras funciones, pero si se le puede asignar un valor, esto nos permite utilizar la variable a través de múltiples funciones.

```
program Alcance;
vars{
  int: glob;
}

main(){
  vars{
    int: entero, numero;
  }
  entero = 100;
  glob = 3;
}
```

Estatutos de Interacción

El lenguaje Alebrije cuenta con funciones predefinidas para lectura de entradas del usuario y para escritura en consola.

Lectura

Para la lectura se utiliza la palabra clave read() dentro de los parenthesis debe de tener una o más variables separadas por coma. Todas estas deben de estar previamente definidas y deben de recibir una entrada por parte del usuario que sea aceptable por el tipo de dato de la variable.

```
program Lectura;
vars{
  int: glob;
}

main(){
  vars{
    int: entero, numero;
  }
  read(numero);
  read(glob, entero);
}
```

Escritura

Para la escritura en consola se utiliza la función predefinida con palabra clave `write()`, dentro de los paréntesis puede haber strings constantes, expresiones aritméticas o variables.

```
program Escritura;

main(){
    vars{
        int: entero, numero;
    }
    read(numero);
    write("resultado ", numero, "= ", 2+2);
}
```

Resultado de ejecución del programa con número = 4

```
>4 ~/compis/test/compiler_proyect-main > git P mai?8 python parser.py escritura.alebrije
resultado 4= 4
```

Expresiones aritmética, lógica y relacional

Logica	Aritmetica	Relacional
>	^	and
<	*	or
>=	/	
<=	%	Asignación Aritmética
==	+	+=
!=	-	-=
		*=
		/=

El lenguaje Alebrije permite la evaluación de expresiones aritméticas, lógicas y relacionales, además de la asignación aritmética. Como podemos observar en el siguiente ejemplo se sigue una un orden de prioridad donde los símbolos aritméticos tienen más prioridad que los lógicos y estos que los relacionales. También dentro de los símbolos aritméticos existe una prioridad similar a PEMDAS.


```
program Aritmetica;
```

```
main(){
    vars{
        int: entero;
        bool: b;
    }
    entero = 2 * (5 - 2) + 8;
    b = entero <= (100/2);
    entero += 1;
    write(b, "\n");
    write(entero);
}
```

```
❏ > ~/compis/test/compiler_proyect-main > git P mai?8 python parser.py aritmetica.alebrije
True
15
```

Control de flujo

Decisiones

Es posible controlar el flujo del programa a través del uso de las palabras clave `if()` y `else`. Dentro de los parenthesis del `if()` debe de ir una expresión lógica (que termine en verdadero o falso), en caso de ser verdadero se ejecuta lo que esté dentro de los corchetes{} del `if()`, si no se salta el `if()` y se va al `else` en caso de que exista. La palabra clave `else` es opcional, la manera en la que funciona es que se corren en caso de que la expresión lógica del `if()` sea falsa.

```
program Decisiones;
```

```
main(){
    vars{
        int: num1,num2;
    }
    num1 = 10;
    num2 = 9;
    if(num1 == num2 + 1){
        write("entro al if");
    }else{
        write("entro al else");
    }
}
```

```
❏ > ~/compis/test/compiler_proyect-main > git P mai?8 python parser.py decisiones.alebrije
entro al if
```

Como se mencionó en la explicación de decisiones, también es posible escribir un estatuto de decisión if() sin else.

```
program Decisiones;

main(){
    vars{
        int: num1,num2;
    }
    num1 = 10;
    num2 = 9;
    if(num1 == num2 + 1){
        write("entro al if");
    }
}
```

Ciclo Condicional

Para el uso de ciclos condicionales se requiere la palabra clave while() dentro de sus parenthesis debe de haber alguna expresión sea verdadera o falsa al evaluarse. Este ciclo va a leer la condición, ejecutar lo que esté dentro de los corchetes{} y regresar a evaluar la expresión, por lo tanto hay que recordar que necesitamos alguna manera de cambiar la condición a falso para salir del ciclo. En el siguiente ejemplo se utiliza una variable de control llamada num1 a la cual se le suma 1 en cada iteración del ciclo para hasta que sobrepase el valor de num2 + 1.

```
program CicloCondicional;

main(){
    vars{
        int: num1, num2, suma;
    }
    suma = 0;
    num1 = 0;
    num2 = 100;
    while(num1 < num2 + 1){
        suma = suma + num1;
        num1 += 1;
    }
    write(suma);
}
```

```
5050 > ~/compis/test/compiler_proyect-main > git P mai?8 python parser.py ciclo_condicional.alebrije
```

Como se puede observar este programa calculó la suma de todos los números enteros positivos menores a 101, lo que nos regresa un valor de 5050.


Ciclo de Rango

El ciclo de rango for es muy similar al ciclo condicional while la principal diferencia es que la variable de control en este caso num1, se le suma un 1 al final de cada iteración de manera automática. También se le asigna un valor a num1 en la misma línea del for.

```
program CicloRango;

main(){
    vars{
        int: num1, suma;
    }
    suma = 0;

    for num1 = (2/2 - 1) to (100 + 1){
        suma += num1;
    }
    write(suma);
}
```



Como podemos observar el programa nos arroja el mismo resultado que con el ejemplo que utilizamos en el ciclo condicional while.

Modulos

Alebrije soporta la programación por medio de módulos primero debe de realizar la declaración de uno antes de hacerle una llamada. Los módulos o funciones pueden regresar un valor de cualquiera de los tipos de datos soportados por Alebrije (int, float, bool, char, string) o puede no regresar ningún tipo de valor (void). Las funciones pueden tener o no tener parámetros en su declaración pero cuando se le llama a esa función el número de argumentos en la llamada debe de igualar al número de parámetros y los tipos de datos con los que fue declarada deben de coincidir.

Funciones void

Dentro de alebrije se soporta el uso de funciones sin un valor de retorno. Como vemos la primero de declarar una función con la palabra clave function seguido por el tipo de dato void para indicar que no devuelve ningún valor, seguido por su nombre y los parámetros dentro de los parenthesis, donde se escribe el tipo de datos seguido por 2 puntos y el nombre del parámetro. En la función main se hace un llamado a la función y se le pasa como argumento una variable de tipo string que es la que pide la función dentro de sus parámetros.

```
program Void;

function void hello(string: name){
    write("hello " + name, "\n");
}

main(){
    vars{ string: nombre; }
    nombre = "julio";
    hello(nombre);
    hello("maria");
}
```

```
❏ > ~/compis/test/compiler_proyect-main > git P mai?8 python parser.py void.alebrije
hello julio
hello maria
```

En este ejemplo podemos observar que se le puede mandar una variable a las funciones, constantes o expresiones como argumentos siempre y cuando cumplan con el tipo de datos solicitado por los parámetros.

Funciones con retorno

Las funciones que se declaran con un tipo de dato diferente a `void` deben de regresar el tipo de dato que se les asignó. Como se demuestra en el siguiente ejemplo esto se logra con la palabra clave `return()` la cual puede tener una expresión dentro de los parenthesis como es el caso de nuestro ejemplo el cual regresa el parámetro número más 2. Al momento de llamar a la función en el ejemplo mandamos una expresión aritmética, esta se resuelve antes de llamar a la función, resulta en 14, después se manda ese 14 a la función y nos regresa un 16, el cual le asignamos a la variable entera num.

```
program Retorno;

function int suma2(int: numero){
    return(numero + 2);
}

main(){
    vars{
        int: num;
    }
    num = 2;
    num = suma2((10/5) + (num*6));
    write(num);
}
```

```
16 > ~/compis/test/compiler_proyect-main > git P mai?8 > python parser.py retorno.alebrije
```

Arreglos

El lenguaje Alebrije soporta el uso de elementos no atómicos en la forma de arreglos. Igual que las variables y las funciones estos deben de declararse antes de llamarse, la manera en la que se declaran es muy similar a las variables solo que se colocan brackets y un número entero dentro de estos después del nombre, resultando en algo similar a esto: "var[20]". Al momento de hacer llamadas a la casilla de un arreglo se debe de poner una expresión que resulte en un número entero dentro de los brackets además ese número entero resultante debe de estar dentro del rango permitido por el arreglo, de lo contrario habrá un error.

```
program Arreglos;

main(){
    vars{
        int: numeros[5], a,b;
    }
    a = 2;
    b = 2;
    numeros[0] = 3;
    numeros[1] = 10;
    numeros[a] = numeros[0] + numeros[1];
    numeros[numeros[0]] = numeros[a] * 2;
    numeros[4] = numeros[numeros[0]] - 6;
    b = 4;
    write(numeros[b]);
}
```

```
20  > ~/compis/test/compiler_proyect-main > git  P mai?8  python parser.py arreglos.alebrije
```

Funciones predefinidas

Estadística

Alebrije es un lenguaje que tiene varias funciones predefinidas para realizar estadística básica. Todas estas funciones reciben arreglos como argumentos.

Funciones de estadística	
mean(arreglo)	median(arreglo)
mode(arreglo)	variance(arreglo)
plotxy(arreglo1, arreglo2)	regression(arreglo1, arreglo2)

Mean

Esta función obtiene el promedio de un arreglo, la suma de todos sus números dividida entre la cantidad de números.

```
program Mean;
  vars{
    int: numeros[100], i;
  }

  main(){
    for i = 0 to 100{
      numeros[i] = i;
    }

    write(mean(numeros));
  }
```

```
50  > ~/compis/test/compiler_proyect-main > git P mai?8 > python parser.py mean.alebrije
```

Median

La función median obtiene la mediana de un arreglo de datos, el que está a la mitad de los datos, en caso de tener 2 números medios saca el promedio entre estos 2.

```
program Median;
  vars{
    int: i, numeros[100];
  }

  main(){
    for i = 1 to 100{
      numeros[i- 1] = (i)*2;
    }

    write(median(numeros));
  }
```

```
100  > ~/compis/test/compiler_proyect-main > git P mai?8 python parser.py median.alebrije
```

Mode

La función mode obtiene el número que más se repite en un arreglo.

```
program Mode;
  vars{
    int: numeros[100], i;
  }

  main(){
    for i = 0 to 20{
      numeros[i] = 22;
    }
    for i = 20 to 100{
      numeros[i] = i;
    }

    write(mode(numeros));
  }
```

```
22  > ~/compis/test/compiler_proyect-main > git P mai?8 python parser.py mode.alebrije
```


Variance

La función variance obtiene la varianza de un arreglo, en este ejemplo sacamos su raíz cuadrada al elevarla a 0.5 lo que nos da 5 como resultado que es la medida en la que nuestros datos que se apartan del promedio, que es 50 para este ejemplo.

```
program Variance;
  vars{
    int: i, numeros[100];
  }

  main(){
    for i = 0 to 100{
      if (i%2 ==0){
        numeros[i] = 45;
      }else{
        numeros[i] = 55;
      }
    }

    write(variance(numeros)^(0.5));
  }
```

```
5.0  > ~/compis/test/compiler_proyect-main > git  mai?8  python parser.py variance.alebrije
```

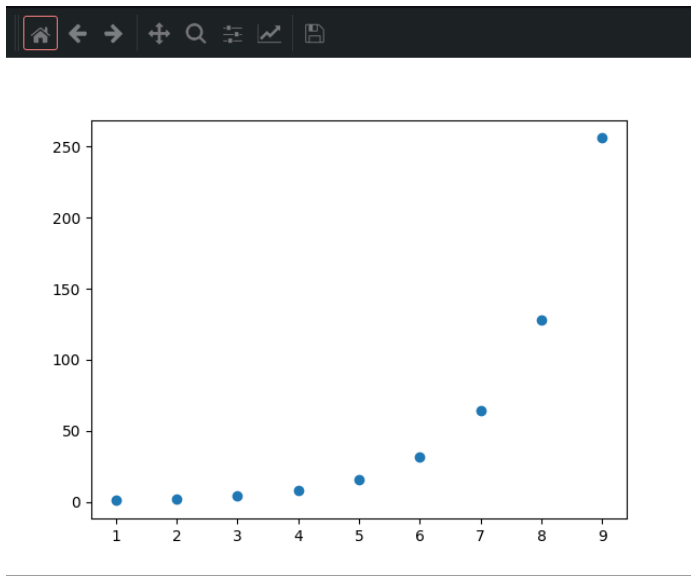
Plotxy

Esta es una función void predefinida no regresa nada, imprimir un gráfico como el que se muestra debajo del código de ejemplo. Toma como argumentos a 2 arreglos, uno para el eje x y otro para el eje y.

```
program PlotXY;
  vars{
    int: i, numeros[10], numeros2[10];
  }

  main(){
    for i = 0 to 10{
      numeros[i] = i;
      numeros2[i] = (2^i)/2;
    }

    plotxy(numeros, numeros2);
  }
```



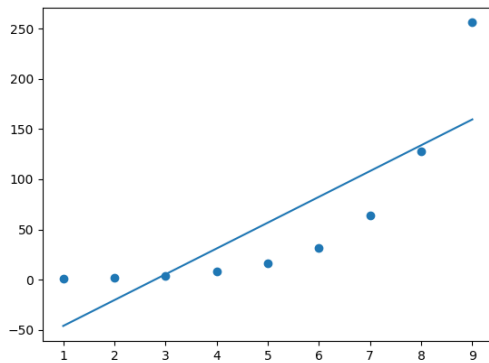
Regression

Esta es una función predefinida que realiza una regresión lineal simple. Al igual que plotxy recibe como argumentos a 2 arreglos.

```
program Regression;
  vars{
    int: i, numeros[10], numeros2[10];
  }

  main(){
    for i = 0 to 10{
      numeros[i] = i;
      numeros2[i] = (2^i)/2;
    }

    regression(numeros, numeros2);
  }
```



Asistencia

Funciones de asistencia
len(arreglo)
min(arreglo)
max(arreglo)

Len

Esta función regresa la dimensión con la que se realizó la declaración de un arreglo.

```
program Tam;
  vars{
    int: numeros[6],tamano;
  }

  main(){
    numeros[0] = -1;
    numeros[1] = 1;
    numeros[2] = 0;
    numeros[3] = 10;
    numeros[4] = -2;
    numeros[5] = 12;
    tamano = len(numeros);
    write(tamano);
  }
```

```
6 > ~/compis/test/compiler_proyect-main > git P mai?8 > python parser.py tamano.alebrije
```

Min

Obtiene el número con el menor valor de todo un arreglo.

```
program Menor;
  vars{
    int: numeros[6],menor;
  }

  main(){
    numeros[0] = -1;
    numeros[1] = 1;
    numeros[2] = 0;
    numeros[3] = 10;
    numeros[4] = -2;
    numeros[5] = 12;

    menor = min(numeros);
    write(menor);
  }
```

```
~ > ~/compis/test/compiler_proyect-main > git P mai?8 python parser.py min.alebrije
-2
```

Max

Encuentra al numero con el valor mayor de todo un arreglo.

```
program Mayor;
  vars{
    int: numeros[6],mayor;
  }

  main(){
    numeros[0] = -1;
    numeros[1] = 1;
    numeros[2] = 0;
    numeros[3] = 10;
    numeros[4] = -2;
    numeros[5] = 12;

    mayor = max(numeros);
    write(mayor);
  }
```

```
~ > ~/compis/test/compiler_proyect-main > git P mai?8 python parser.py max.alebrije
12
```

Video ejemplo

<https://www.youtube.com/watch?v=Poa7Y-VB09Y>