RALLY

Proyecto guiado por:

Profesor. Hector Gibrán Ceballos

Profesora. Elda Quiroga

Autores:

Alan Zavala Lévaro A01338448

Paola Villarreal García A00821971

Junio 2021

Aprende a programar en RALLY

“Hello World”

Para empezar a programar en RALLY debes declarar tu programa cómo…

program *nombre :*

Este nombre puede ser el que tú quieras, solo debe empezar en minúscula. Te recomendamos que el nombre tenga sentido con lo que el programa realizará.

Ya que le pusiste nombre hay que declarar la clase principal llamada *main* y su función principal llamada *init*...

```
main {
        init {
        }
{
```

Esta clase y función son necesarias para que el programa funcione correctamente, ya que ellas se encargan de hacer la ejecución de todo el código escrito, es lo primero que se realiza y tiene el control de qué y cómo se hace el código, así que, siempre en cualquier programa que realices deben estar declaradas.

¡Ya tienes la base! Ahora escribe tu primer “Hello World” en pantalla usando el estatuto *write*.

```
program helloWorld :
        main {
                init {
                        write(“Hello World!”);
                }
        }
end
```

No olvides poner al final la palabra *end* para indicar que tu programa ha terminado.

Variables

Las variables sirven para asignarles diferentes tipos de valores y poder usarlas para distintas expresiones aritméticas.

Primero deben ser declaradas, pero su uso depende de en qué parte del programa se declaren:

- Si se declaran dentro de la clase *main* y antes de la declaración de cualquier función, estas podrán usarse en todas las funciones de esa clase, por lo cual, serían variables globales.

- Si se declaran dentro de las funciones, entonces sólo podrán ser usadas dentro de esa función, por lo cual, serían variables locales

Para poder declarar una o varias variables se debe usar la palabra reservada *var* y enseguida escribir el tipo de variable, estas pueden ser:

- *int* para guardar valores enteros
- *float* para guardar valores decimales
- *char* para guardar un carácter, este carácter debe estar entre comillas sencillas

También existen los objetos, pero de ellos hablaremos más adelante.

El nombre de tu variables puede ser el que tu quieras, mientras que no empiece con mayúscula o número, e igualmente no puedes hacer uso de ningún carácter diferente a letras o números, solamente de guión bajo. Por ejemplo, nombres correctos de variables pueden ser: *nombre*, *dato1*, *apellidoMaterno*, *num_tel*. Algo importante que debes saber es que el nombre de las variable no puede repetirse, deben ser ids únicos.

Ejemplo:

```
program variables :
        main {
                var int i, j;
                    float k;
                init {
                        var char c;
                }
        }
end
```

No olvides cerrar cada línea con el punto y coma (;).

Sin embargo… un programa no puede contener solo declaración de variables, ¡Hay que hacer uso de ellas!

Puedes hacer asignaciones y operaciones aritméticas sencillas, cómo: sumas, restas, multiplicaciones y divisiones.

Ejemplo:

```
program variables :
    main {
        var int i, j;
            float k;
        init {
            var char c;
            i = 3;
            j = i;
            k = j * (4.3 + i) / j;
            c = 'a';
        }
    }
end
```

Como puedes ver, puedes asignar números, variables y hasta expresiones a otra variable, solo debes de tener cuidado que tengan congruencia los tipos de cada uno, es decir, que combines correctamente estos, si no, tendrás un error.

---

* Al final de este documento podrás encontrar una tabla con todas las operaciones que las variables pueden realizar

Estructuras de datos

Un tipo de variables muy importantes que hay que mencionar son las de array y matriz.

Array

Las variables de tipo array se podrían ver como una lista de valores, su declaración es muy similar a como ya vimos, solo se debe agregar un número que represente cuántos valores espera tener esa lista.

Ejemplo:

```
var int a[5];
```

Esto quiere decir que la variable 'a' tiene 5 espacios disponibles para asignarle valores.

Matriz

Las variables de tipo matriz funcionan igual que array, solo que en lugar de ser solo una lista aquí se contiene filas y columnas.

Ejemplo:

```
var float b[2][3]
```

El primer valor representa el número de filas y el segundo valor representa el número de columnas

- Indexar

Indexar es acceder a una de las casillas para asignarle o leer su valor. Para ambos tipos de variables su indexación empieza en cero, por lo que, en la variable array 'a' podemos entrar a las casillas 0, 1, 2, 3 y 4, y en la variable 'b' podemos entrar a las filas 0, 1, y columnas 0, 1, 2.

Ejemplo:

```
program estructuras :
    main {
        var int i[8], j[5][5];
        init {
            i[3] = 3;
            j[2][4] = i[3];
```

```
        }
    }
end
```

Siempre que se quiera asignar o leer un array o matriz se debe poner a qué casilla se desea acceder.

No es necesario llenar de valores todas las casillas, pero como consejo, es mejor siempre hacer estas variables del tamaño justo que necesitas, si no ¡desperdiciarás memoria!

Por último, puedes acceder a una posición no sólo con constantes, sino también con variables, expresión y hasta indexaciones de otras arrays o matrices, pero ten cuidado, siempre debe indexarse con un valor de tipo *int* y a una posición existente.

Ejemplo:

```
program estructuras :
    main {
        var int i[8], j[5][5];
        init {
            var int a, b;
            a = 2;
            b = 3;
            i[a*b] = 3;
            j[i[6]][a+2] = 8;
        }
    }
end
```

Estructuras de decisión

Condiciones

A veces en los programas necesitamos que se haga una operación solo si se cumple una condición, para esto tenemos el estatuto de decisiones *if*.

Ejemplo:

```
program condicionales :
    main {
        init {
            var int dato1, dato2;
            read(dato1, dato 2);
            if (dato1 < dato2 and dato1 != 0) then {
                write("El dato 1 es menor que el dato 2");
            }
        }
    }
end
```

¡Hay dos cosas nuevas que notar!

Lo primero es el estatuto *read*, este funciona para leer lo que se escriba en pantalla, esto quiere decir que, se le asignará a la variable el número que se teclee.

Lo segundo es que ahora no solo tenemos operaciones aritméticas sino también operaciones relacionales y lógicas.

- Los relacionales sirven para hacer comparaciones entre dos valores, existen {>, >=, <, <=, ==, !=}.
- Las lógicas sirven para evaluar si una expresión es falsa o verdadera (valores booleanos), existen {and, or}

Es importante tener en cuenta que nuestros condicionales siempre evaluarán expresiones relacionales o lógicas.

---

* Al final de este documento podrás encontrar una tabla de evaluación de los valores booleanos

Pero ahora, en el ejemplo anterior, ¿qué pasará si nuestro dato 1 es mayor que el dato 2? No hará nada, lo mejor sería que escriba otro texto, para esto usamos *else*.

Ejemplo:

```
program condicionales :
    main {
        init {
            var int dato1, dato2;
            read(dato1, dato2);
            if (dato1 < dato2) then {
                write("El dato 1 es menor que el dato 2");
            } else {
                write("El dato 2 es menor que el dato 1")
            }
        }
    }
end
```

Sin embargo, si ambos datos son iguales, entraría a la condición de *else* y eso estaría incorrecto, por lo cual, usamos *elif*.

```
program condicionales :
    main {
        init {
            var int dato1, dato2;
            read(dato1, dato2);
            if (dato1 < dato2) then {
                write("El dato 1 es menor que el dato 2");
            }
            elif (dato1 == dato2) then {
                write ("Los dos datos son iguales");
            } else {
```

```
                write("El dato 2 es menor que el dato 1")
            }
        }
    }
end
```

Podemos ver que este tipo de condicionales, siempre necesita empezar con un *if*, y puede o no contenter condicionales *elif* y *else*.

Los condicionales *elif* pueden repetirse, si necesitas uno, dos o más de 10, puedes escribirlos. Al contrario, un *else* sólo puede estar una vez y debe ser siempre el último.

También, puedes encadenar condicionales, esto quiere decir que un *if* puede estar dentro de otro *if*, y de otro, y de otro, y de otro… cuántos se necesiten.

Estructuras cíclicas

Así como con las condiciones queríamos que una operación se hiciera solo si se cumple una condición, ahora podemos necesitar que una operación se haga muchas veces, para esto tenemos los estatutos *while* y *for*.

While

Ejemplo:

```
program ciclos:
    main {
        init {
            var int dato1, dato2;
            read(dato1, dato 2);
            while (dato1 < dato2) do {
                write(dato1);
                dato1 = dato1 + 1;
            }
        }
    }
end
```

For

Ejemplo:

```
program ciclos:
    main {
        init {
            var int dato1, dato2;
            dato2 = 10;
            for(dato1 = 0) until (dato1 < dato2) do {
                write(dato1);
                dato1 = dato1 + 1;
```

```
                    }
                }
            }
        end
```

No olvides aumentar la variable correspondiente, sino ¡tu programa se ciclara!

Y así como los encadenamientos de condicionales, aquí también puedes encadenar ciclos y no sólo eso, puedes agregar condiciones dentro de los ciclos.

Entonces, puedes encontrarte con un *if*, dentro de un *while*, dentro de un *for*, dentro de otro *if* y mucho más.

Funciones

Las funciones son un fragmento de código que se especializa en algo. Como ya vimos, nosotros tenemos una función definida llamada *init*, sin embargo, puedes crear más funciones ¡ya viste todo lo que puede ir dentro de ellas!

Estas nuevas funciones deben ser creadas dentro de la clase *main*, pero fuera de la función *init*. Y el acceso a ellas podrá ser desde donde tú quieras siempre y cuando la función a la que deseas acceder ya haya sido declarada antes de la llamada.

Es importante saber que aquí no existe el encadenamiento, así que no puedes tener funciones dentro de otras.

Para definir las funciones necesitamos la palabra reservada *funct* y enseguida el tipo de la función, esta puede ser:

- *void*, si no necesita regresar ningún valor
- *int*, *float* o *char*, si se necesita regresar un valor de esos tipos

Después se les asigna un nombre, las reglas son las mismas que como los nombres de variables: no pueden repetirse, ni empezar con mayúscula o número. Un consejo, es que el nombre que le pongas a la función tenga sentido con lo que se hará en ella, para una mejor comprensión.

Ejemplo:

```
program funciones:
    main {
        funct void escribirHelloWorld () {
            write(“Hello World”);
        }
        init {
            call escribirHelloWorld();
        }
    }
end
```

Como puedes ver, para acceder a la función se tuvo que escribir la palabra reservada *call* y luego el nombre la función. Esta forma de llamada solo se hace porque la función es de tipo *void*, de lo contrario, no necesitas usar la palabra reservada.

Si la función llegara a ser de tipo *int*, *float* o *char*, necesita la llamada estar dentro de una expresión u otro estatuto, ya que esa función tendrá un *return*, lo que quiere decir que regresará un valor al punto donde se hace la llamada.

Ejemplo:

```
program funciones:
    main {
        funct float calcularSuma() {
            var float resultado;
            resultado = 5.6 + 7.8;
            return resultado;
        }
        init {
            write("El resultado es: ", calcularSuma())
        }
    }
end
```

Siempre que sea una función de tipo diferente a void debe tener un *return* y este solo puede existir al final de ella. Además un *return* solo regresar variables.

Parámetros

En las funciones existen los parámetros, estos sirven para mandarle variables, constantes o posiciones de un array y que dentro de la función haga uso de ellos. Estos pueden enviarse a todos los tipos de funciones y pueden ser desde una, hasta las que se necesiten.

Para enviarlas, en la llamada debes poner solo el nombre de las variables dentro de los paréntesis y separadas por una coma, o bien puedes enviar constantes. Para recibirlas, en la declaración de la función se debe poner el tipo de la variable y un nombre dentro de los paréntesis y separadas por una coma.

Ejemplo

```
program funciones:
    main {
```

```
        funct void escribirDatos (int d1, float d2) {
            write(d1, d2);
        }
        init {
            var int dato1;
            dato1 = 7;
            call escribirDatos(dato1, 8.3);
        }
    }
end
```

Como puedes ver, el nombre que pongas en los parámetros de la función pueden no ser igual al nombre de la variable, sin embargo, hay que tener cuidado que el tipo de cada uno coincida y que el número de variables que se están enviado sea el mismo número de parámetros que espera recibir la función.

Ejemplo:

```
program funciones:
    main {
        funct float calcularSuma(float a, float b) {
            var float resultado;
            resultado = a + b;
            return resultado;
        }
        init {
            write("El resultado es: ", calcularSuma(3.4, 5.6))
        }
    }
end
```

- Recursividad

En las funciones existe la recursividad, esto es especificar un proceso basado en su propia definición, se puede ver como una forma diferente de hacer un ciclo.

Normalmente son funciones de un tipo diferente a *void* y con parámetros; dentro de esta función se tiene una llamada a ¡la misma función! así que debemos encargarnos de planificar el momento en que dejan de llamarse a sí mismas o tendremos una función recursiva infinita.

Ejemplo:

```
program factorial :

    main {

        funct int factorial(int n) {

            var int nF;

            if (n >= 1) then {

                nF = n * factorial(n - 1);

            } else {

                nF = 1;

            }

            return nF;

        }


        init {

            var int num, fact;

            read(num);

            fact = factorial(num);

            write("El factorial de:", num, "es: ", fact);

        }

    }

end
```

Clases

Las clases son abstracciones de objetos de la vida real y sirven para ejemplificarlos con código.

En nuestro programa ya tenemos una clase, la *main* y asi como ella puede tener variables globales y funciones, podemos declarar más clases que también pueden contener variables globales y funciones (solo que aquí se les llama atributos y métodos). No es necesario que cada una tenga la función *init*, esa solo la lleva la de *main*.

Estas clases se deben declarar antes de la de *main*. Debe escribirse primero la palabra reservada *class* y enseguida un nombre, el cual debe ser único, también puede contener letras y guión bajo, pero a diferencia de funciones y variables, esta debe comenzar con mayúscula.

Ejemplo:

```
program clases :
        class Perro {
                var int edad;
                funct void setEdad(int e) {
                        edad = e;
                }
        }
        main {
                init {
                        write("Hola");
                }
        }
end
```

Para hacer uso de estas clases, se pueden declarar variables de tipo objeto, ¿qué quiere decir esto? que esta variable puede acceder a cualquier atributo o método de la clase por la que se declaró. Pero funciona igual, si se declara global puedes acceder a ellas donde sea y si se declara local solo en esa función podrás usarla.

Ejemplo:

```
program clases :
        class Perro {
                var int edad;
                funct void setEdad(int e) {
                        edad = e;
                }
        }
        main {
                init {
                        Perro perro1, perro2;
                        perro1.edad = 5;
                        perro2.setEdad(7);
                        write(perro1.edad + perro2.edad);
                }
        }
end
```

Puedes ver, que casi todo es igual a como ya hemos visto, solo que es necesario agregar la variable antes de cada llamada.

Algo importante que notar es cómo existen dos instancias de la misma clase, una es por parte de “perro1” y otra por parte de “perro2”, pertenecen a una sola clase, pero cada uno tiene una edad diferente.

Programa

¡Listo! Ya conoces todo lo que puedes hacer con el lenguaje RALLY

Lo único que falta es ¿Cómo compilas tus propios programas? Muy fácil.

Deberás escribir tu código, guardarlo en un archivo .txt, el cual debe estar en la misma carpeta donde se encuentre instalado el programa, y desde cualquier consola que uses accedes a la carpeta, corres con python el archivo Compiler.py y luego escribes el nombre del archivo incluyendo su extensión.

Por último, aquí te dejamos un ejemplo completo que contiene todo lo visto en este manual.

Esperamos que hayas aprendido mucho.

```
program registroAlumno :
  class Escuela {
    var char carrera;
      int id;
      float promedio;
    funct void setPromedio(float p) {
      promedio = p;
    }
    funct char getCarrera() {
      return carrera;
    }
    funct int getID() {
      return id;
    }
  }
  main {
    var Escuela alumno1, alumno2;
    funct void promedioAlumnos() {
      var int i;
```

```
    float prom;
  i = 1;


  while(i < 3) do {
    write("Promedio del alumno ", i);
    read(prom);
    if (i == 1) then {
      call alumno1.setPromedio(prom);
    }
    else {
      call alumno2.setPromedio(prom);
    }
    i = i + 1;
  }
}
funct int getID(int alumno) {
  var int id;
  write("ID del alumno ", alumno);
  read(id);
  return id;
}
init {
  var char escuela[2];
    int i;
  write("Registro de 2 alumnos");
  from (i = 0) until (i < 2) do {
    write("A que escuela pertenece, escribe la letra de la opcion");
    write("A- Ingieneria");
    write("B- Ciencias Sociales");
```

```
    write("C- Quimica");
    write("D- Licenciatura");
    read(escuela[i]);

    if (escuela[i] != 'A' and escuela[i] != 'B' and escuela[i] != 'C' and escuela[i] != 'D') then {
      write("Error en la opcion, escriba una correcta");
    }
    else {
      i = i + 1;
    }
}
alumno1.carrera = escuela[0];
alumno2.carrera = escuela[1];
alumno1.id = getID(1);
alumno2.id = getID(2);
call promedioAlumnos();
if (alumno1.getCarrera() == 'A') then {
    write("Alumno 1 pertenece a la carrera de Ingieneria");
}
elif (alumno1.getCarrera() == 'B') then {
    write("Alumno 1 pertenece a la carrera de Ciencias Sociales");
}
elif (alumno1.getCarrera() == 'C') then {
    write("Alumno 1 pertenece a la carrera de Quimica");
}
else {
    write("Alumno 1 pertenece a la carrera de Licenciatura");
}
write("tiene un id de ", alumno1.getID(), "y un promedio de ", alumno1.promedio);
```

```
        write("Alumno 2 pertenece a la carrera ", alumno2.getCarrera(), "tiene un id de ",
    alumno2.getID(), "y un promedio de ", alumno2.promedio);

      }

   }

end
```

---

* La indentación no es necesaria, pero se sugiere para mantener un mejor orden

Anexos

Tabla de operaciones

Operador	Operando 1	Operando 2	Resultante
+	int	int	int
	int	float	float
	int	char	error
	float	float	float
	float	int	float
	float	char	error
	char	char	error
	char	int	error
	char	float	error
-	int	int	int
	int	float	float
	int	char	error
	float	float	float
	float	int	float
	float	char	error
	char	char	error
	char	int	error
	char	float	error
*	int	int	int
	int	float	float
	int	char	error
	float	float	float
	float	int	float
	float	char	error

	char	char	error
	char	int	error
	char	float	error
/	int	int	int
	int	float	float
	int	char	error
	float	float	float
	float	int	float
	float	char	error
	char	char	error
	char	int	error
	char	float	error
>	int	int	bool
	int	float	bool
	int	char	error
	float	float	bool
	float	int	bool
	float	char	error
	char	char	bool
	char	int	error
	char	float	error
>=	int	int	bool
	int	float	bool
	int	char	error
	float	float	bool
	float	int	bool
	float	char	error
	char	char	bool

	char	int	error
	char	float	error
<	int	int	bool
	int	float	bool
	int	char	error
	float	float	bool
	float	int	bool
	float	char	error
	char	char	bool
	char	int	error
	char	float	error
<=	int	int	bool
	int	float	bool
	int	char	error
	float	float	bool
	float	int	bool
	float	char	error
	char	char	bool
	char	int	error
	char	float	error
!=	int	int	bool
	int	float	bool
	int	char	error
	float	float	bool
	float	int	bool
	float	char	error
	char	char	bool
	char	int	error

	char	float	error
==	int	int	bool
	int	float	bool
	int	char	error
	float	float	bool
	float	int	bool
	float	char	error
	char	char	bool
	char	int	error
	char	float	error
and	int	int	error
	int	float	error
	int	char	error
	int	bool	error
	float	float	error
	float	int	error
	float	char	error
	float	bool	error
	char	char	error
	char	int	error
	char	float	error
	char	bool	error
	bool	bool	bool
or	int	int	error
	int	float	error
	int	char	error
	int	bool	error
	float	float	error

	float	int	error
	float	char	error
	float	bool	error
	char	char	error
	char	int	error
	char	float	error
	char	bool	error
	bool	bool	bool

Tabla de evaluación de valores booleanos

Operador	Operando 1	Operando 2	Resultante
and	true	true	true
	true	false	false
	false	true	false
	false	false	false
or	true	true	true
	true	false	true
	false	true	true
	false	false	false