Lenguaje de Programación:

“COSMOS”

1. Estructura General:

En el lenguaje “Cosmos”, se define inicialmente su estructura de la siguiente forma:

```
#Integral Sistema () {

}
```

2. Declaración de Variables

- #Integral: Este tipo de dato, acepta cualquier tipo de números enteros sin contar los decimales.

```
#Integral A = 1;
```

- #Caracter: Este tipo de dato acepta cualquier carácter, está debe ir entre comillas dobles.

```
#Caracter Identificador = “Mensaje”;
```

- #Decision: Este tipo de dato, representa un valor de verdad. Solamente Verdadero o falso.

```
#Decision variable;

#Decision variable = verdadero;

#Decision variable = falso;
```

- #Decimal: Este tipo de dato, acepta números con punto decimal.

```
#Decimal A= 7.23;
```

-#Faltante: Este tipo de dato no es un dato en sí mismo ya que representa la ausencia de datos y es utilizada cuando no se esperan datos.

```
#Faltante Sistema () {

}
```

- Identificador: Una letra seguida de una letra o más letras o ninguna.

3. Palabras reservadas

- #Mostrar: Nos imprime los mensajes en pantalla, a los cuales se les considera como carácter de formato.

```
#Mostrar(“Hola mundo”);
```

Para mostrar una variable:

```
#Mostrar(nombre_variable);
```

- #Leer: Acepta varios tipos de datos, los cuales son enviados directamente del teclado y son almacenados en la memoria.

```
#Leer(variable);
```

Para leer más de una variable:

```
#Leer(variable1, variable2);
```

- Freno: Usado para terminar la ejecución de un bucle o ciclo.

```
#Integral Sistema(){

     #Integral i = 1;

     Mientras(i <=10){
       Si (i == 6){
       Freno;
     }
     #Mostrar(i);
       i++;
    }
#Regresar 0;
}
```

-#Pordefecto: Se usa en una estructura de control **Cambio** para manejar escenarios que no se han cubierto en las etiquetas **#Escenario**.

```
#Integral Sistema () {

     #Integral num;
     #Mostrar(“Ingrese un numero del 1 al 2: ”);
     #Leer(num);
```

```
Cambio(num){
     #Escenario 1:
     #Mostrar(“Lunes”);
     Freno;
     #Escenario 2:
     #Mostrar(“Martes”);

  #Pordefecto:

     #Mostrar(“Opcion no valida”);
     Freno;
}
#Regresar 0;
}
```

-**#Escenario:** Es una palabra clave utilizada en una estructura switch para comparar el valor de una expresión con una serie de escenarios y ejecutar el bloque de código correspondiente al primer escenario que coincide.

```
     #Integral Sistema () {
         #Integral num;
         #Mostrar(“Ingrese un numero del 1 al 2: ”);
         #Leer(num);

     Cambio(num){
         #Escenario 1:
         #Mostrar(“Lunes”);
         Freno;
         #Escenario 2:
         #Mostrar(“Martes”);
         #Pordefecto:
         #Mostrar(“Opcion no valida”);
         Freno;
         }
     #Regresar 0;
     }
```

-**#Regresar**: Se usa dentro de una función para devolver un valor específico al llamador de la función.

```
     #Integral sum(#Integral a, #Integral b){

     #Regresar a+b;
     }
```

3.1 Delimitadores

Punto y coma (;): El punto y coma es utilizado como un delimitador de la sentencia de código, es decir, indica el fin de una sentencia. Ejemplo:

```
#Integral numero = 10;
```

Aquí se le asigna a la variable número el valor 10 y el punto y coma que le sigue significa que la sentencia está completa osea terminada.

3.2 Comentario

Comentario(-/ y \-) : Este se utiliza cuando el programador quiere aclarar o comentar algo sobre el código, este no afecta al funcionamiento del mismo. Para convertir una parte de código en comentario se tiene que empezar con "-/" y se tiene que terminar con "\-".
Ejemplo:

```
-/Este es un ejemplo\-
```

4. Operadores

4.1 Asignación:

Nota: Los dos números pueden ser dos variables que contengan números.

Operador igual (=): Este operador es utilizado para asignar un valor a una variable.

```
x = 10;
```

4.2 Aritméticos:

Operador suma (+): Este operador devuelve la suma de dos números. Ejemplo:

```
#Integral a = 5; #Integral b = 3; #Integral C; -
         a+b = C; -/El resultado es 8\-
```

Operador resta (-): Este operador devuelve la resta de dos números. Ejemplo:

```
#Integral a = 5; #Integral b = 3; #Integral C;
        a-b = C;  -/El resultado es 2\-
```

Operador multiplicación (*): Este operador devuelve la multiplicación de dos números. Ejemplo:

```
#Integral a = 5; #Integral b = 3; #Integral C;
        a*b = C; -/El resultado es 15\-
```

Operador división (/): Este operador devuelve la división de dos números. Ejemplo:

```
#Integral a = 5; #Integral b = 3; #Integral C;
```

```
a/b = C; -/El resultado es 1.66\-
```

Operador modulo (%): Este operador devuelve el residuo de la división de dos números. Ejemplo:

```
#Integral a = 5; #Integral b = 3; #Integral C;
        a%b = C; -/El resultado es 2\-
```

4.3 Igualdad y Relacionales:

4.3.1 Igualdad

Operador igual a (==): Este operador devuelve verdadero si los dos valores son iguales. Ejemplo:

```
si(q == r){ -/ cuando q es igual a r \-

#Mostrar("Este es un ejemplo");

 }
```

Operador distinto de (!=): Este operador devuelve verdadero si los dos valores son distintos.

```
si(q != r){ -/cuando es diferente a r\-
#Mostrar("Este es un ejemplo");
}
```

4.3.2 Relacionales

- Operadores

Operador mayor que (>): Este operador devuelve verdadero cuando el valor de la izquierda es mayor que el de la derecha.

```
Para (#Integral i = 10 hasta i > 1 decremento  1) {
       #Mostrar(i);  -/ Mostraría  10, 9, 8, 7, 6, 5, 4, 3, 2 \-
         }
```

Operador menor que (<): Este operador devuelve verdadero cuando el valor de la izquierda es menor que el de la derecha.

```
Para (#Integral i = 1 hasta i < 10 incremento  1) {
          #Mostrar(i); -/ Mostraría 1, 2, 3, 4, 5 , 6, 7, 8, 9 \-
             }
```

Operador mayor que (>=): Este operador devuelve verdadero cuando el valor de la izquierda es mayor o igual que el de la derecha.

```
Para (#Integral i = 8 hasta i  >= 1 decremento  1) {
          #Mostrar(i);    -/ Mostraría  8, 7, 6, 5, 4, 3, 2, 1 \-
         }
```

Operador menor que (<=): Este operador devuelve verdadero cuando el valor de la izquierda es menor o igual que el de la derecha.

```
Para (#Integral i <= 1 hasta i = 8 incremento  1) {
          #Mostrar(i);  -/ Mostraría 1, 2, 3, 4, 5 , 6, 7, 8 \-
         }
```

4.3.3 Lógicos

Operador y (&&): Este operador devuelve verdadero cuando ambos operandos son ciertos.

```
#Integral a, b;
Si(a=0 && b=3) {
#Mostrar("es verdadero si se cumple a y b ");
}
```

Operador o (||): Este operador devuelve verdadero cuando al menos un operando es cierto.

```
#Integral a, b;
Si(a=0 || b=3) {
      #Mostrar("es verdadero si se cumple a o b ");
}
```

5. Sentencias de Control

-**Si**: Procesa un conjunto de sentencias basadas en el resultado de evaluar las expresiones de condición.

```
#Integral a, b;
Si ( a > b){
    #Mostrar("A es mayor a B");
}
```

-**Si Sino**: Define el valor que se asigna si la condición es falsa.

```
#Integral a, b;
Si ( a > b){
    #Mostrar("A es mayor a B");
} Sino { #Mostrar("B es mayor a A");
}
```

-**Cambio**: Es una estructura de control de flujo que permite seleccionar un camino de ejecución entre múltiples opciones basado en el valor de una variable o expresión.

```
#Integral Sistema () {
     #Integral num;
     #Mostrar(“Ingrese un numero del 1 al 2: ”);
     #Leer(num);

     Cambio(num){
          #Escenario 1:
          #Mostrar(“Lunes”);
          Freno;

          #Escenario 2:
          #Mostrar(“Martes”);

          #Pordefecto:
          #Mostrar(“Opcion no valida”);
          Freno;
}
#Regresar 0;
}
```

6. Ciclos

- Para: Este es un ciclo utilizado para repetir un conjunto de instrucciones cierta cantidad de veces, para que se repitan deben de cumplir. La estructura del ciclo es la siguiente.

```
Para (Tipo_dato variable operando número hasta variable operando
numero incremento #Integral);

Para (#Integral i < 10 hasta i = 1 incremento  1)  {
#Mostrar(“Esto es un ejemplo”) };
-/ muestra en consola una lista de números del 10 al 1\-

Para (#Integral i < 10 hasta i = 1 decremento 1)  {#Mostrar(“Esto es
un ejemplo”) };
-/ muestra en consola una lista de números del 1 al 10\-
}
```

- **Mientras**: Permite repetir una instrucción hasta que una expresión especificada sea falsa.

```
#Mostrar(“Ingresar un numero que sea menor o igual a 18”);

       #Leer(numero);
```

```
        mientras (numero > 0 || numero < 18) {

        #Mostrar(“El numero ingresado no es correcto);
        #Mostrar(”Ingresar nuevamente un numero que sea menor o igual
        a 18”);

        #Leer(numero);

      }

#Mostrar(”El número se ingreso satisfactoriamente”, numero);
```

- **Hacer Mientras**: Permite repetir una instrucción o una instrucción compuesta hasta que una expresión especificada sea falsa. Ejemplo:

Estructura:

```
Hacer {
    } Mientras (variable operador número)

Hacer {
      #Mostrar(“Este es un ejemplo”)
    } Mientras (x<8)
```

7. Validaciones

7.1 Excepciones:

- Intentar: nos ayuda a evaluar las expresiones que se coloquen, ya si sucede algún inconveniente o excepción mientras se está ejecutando el programa da un aviso sobre dicha excepción.

- Atrapar: “atrapa” dichas excepciones o inconvenientes encontrados mientras esté en ejecución, también realiza una operación de recuperación que le indiquemos con anterioridad.

```
#Integral x;
     Intentar {
           x= 7/0;
     }

     Atrapar (Excepcion e){
           #Mostrar (“El valor es indefinido”);
     }
```

8. Modificadores de Acceso

- **Publico**: De una clase son accesibles desde cualquier lugar en el código, ya sea dentro o fuera de la clase.

```
Clase ejemplo{
     Publico:
     #Integral I;
     #Faltante imprimir(){
     #Mostrar(“#Mostrar: “, I);
}
```

- **Privado**: De una clase sólo son accesibles desde dentro de la clase. Es decir, solo los métodos y las funciones de la clase pueden acceder a los miembros Privado.

```
Clase ejemplo2{
     Privado:
     #Integral y;
     #Faltante imprimir(){
     #Mostrar(“#Mostrar: “, y);
}
```

- Protegido: De una clase sólo son accesibles desde dentro de la clase y también aplican a las clases derivadas, es decir sólo dentro de la clase, no fuera de ella.

```
 Clase ejemplo3 {
       Protegido:
       #Integral y;
       #Faltante imprimir(){
       #Mostrar(“#Mostrar: “, y);
 }
```

9. Clases y Atributos

-**Clase**: Encapsula un conjunto de variables y funciones que operan en esas variables. Las clases proporcionan una manera de organizar y modular el código en unidades lógicas y coherentes.

```
Clase Persona {
     Publico:
     #Caracter nombre;
     #Integral edad;

#Faltante #MostrarDatos(){
     #Mostrar(“Nombre:”, nombre);
     #Mostrar(“Edad:”,edad);
     }
};

 Sistema () {
    Persona persona1;
    persona1.nombre = “Juan”;
```

```
    persona1.edad = 25;
    persona1.#MostrarDatos();

#Regresar 0;
}
```

10. Métodos

- #Establecer: provee, da o “establece” un valor, el cual es miembro de una clase privada.

- #Obtener: recibe u “obtiene” el valor establecido anteriormente, el cual pertenece a una clase privada.

Ejemplo general:

```
Clase Vehículo {
      Privado:
      #Caracter nombre;

      Publico:
      #Faltante #establecerNombre(#Caracter n) {
      nombre = n;
      }

      #Faltante #MostrarNombre(){
      #Mostrar(“Nombre del vehiculo: ”, nombre);
      }

}

#Integral Sistema(){
      Vehiculo v;
      v.#obtenerNombre(“Automovil”);
      v.#MostrarNombre();

      #Regresar 0;
}
```

-Procedimiento: Es un conjunto de instrucciones que realizan una tarea específica y pueden ser llamadas en cualquier parte del programa. El procedimiento tiene un nombre y puede recibir argumentos y devolver un valor como resultado.

```
#Decimal area_circulo(#Decimal radio);

      #Integral Sistema () {

#Decimal radio;
#Mostrar(“Introduce el radio del circulo: “);
#Leer(radio);
```

```
#Decimal area = area_circulo(radio);
#Mostrar(“El area del circulo es: “);
#Regresar 0;
    }

#Decimal area_circulo(#Decimal radio){
#Decimal area = 3.1416 * radio * radio;
#Regresar area;
    }
```

-Metodos: Es una función que está definida dentro de una clase y puede ser invocada por los objetos de esa clase.

```
Clase Vehículo {
      Privado:
      #Caracter nombre;

      Publico:
      #Faltante #establecerNombre(#Caracter n) {
      nombre = n;
      }

      #Faltante #MostrarNombre(){
      #Mostrar(“Nombre del vehiculo: ”, nombre);
      }

}

#Integral Sistema(){
      Vehiculo v;
      v.#obtenerNombre(“Automovil”);
      v.#MostrarNombre();

      #Regresar 0;
}
```

11. Impresión de información a Consola

12. Estructura de datos

-Vector: Un vector es una estructura de datos utilizada para guardar datos que son del mismo tipo. La estructura básica del vector en cosmos es:

```
#tipo_de_dato<Integral> nombre_del_vector
```

```
#Integral<3> nombre;      -/para diferenciar el vector se utiliza la siguiente estructura.\-
```

Aquí el vector tiene 4 espacios que corresponde a 0,1,2,3,4 que para poder asignar cada dato a un espacio, se estructura de la siguiente manera:

```
 nombre<0> = 10; -/aquí se está asignando al espacio 0 en el vector nombre el valor de 10;
```

-Tabla: Es una estructura de datos que almacena un conjunto de valores del mismo tipo en una disposición rectangular bidimensional, es decir, en filas y columnas. Se puede acceder a los elementos individuales de la matriz utilizando un índice de fila y un índice de columna.

```
#Integral Sistema(){

     #Integral Tabla<2><3> = {{1,2,3},{4,5,6}};

     #Mostrar(“Elemento en la fila 0 y columna 1:”;
     #Mostrar(Tabla<0><1>);

 -/ cambiar el valor del elemento en la fila 1 y columna 2\-

  Tabla<1><2> = 10;

}
```

-Pilas: Las pilas es una estructura de datos que almacena datos, su principal características es que utiliza el método UEPS el cual significa que el ultimo que entra es el primero en salir. en cosmos la pila se declara con “UEPS” La estructura es la siguiente:

```
UEPS[tipo_de_dato] nombre_pila;
UEPS[Integral] ejemplo;
```

Para poder operar las pilas utilizamos los siguientes comandos.

```
ejemplo.ingresar(9); -/En este caso se ingresa el numero 9 al tope de la pila-\
ejemplo.ingresar(8); -/En este caso se ingresa el numero 8 al tope de la pila-\
ejemplo.quitar; -/En este se esta quitando el numero del tope de la pila-\
ejemplo.mostrar; -/En este caso imprime el numero que esta al tope de la pila-\
Mientras(ejemplo.vacio){ mostrar(“la pila esta vacia”); } -/En este el ciclo es verdadero cuando la pila esta vacía.-\
```

-**Turno**: Es una estructura de datos que representa una colección de elementos en la que los elementos se insertan al final y se eliminan del principio. Es decir, el primer elemento que entra el turno es el primero que se elimina.

```
   #Integral Sistema () {

Cola<Integral> miCola;

-/Agregar elementos a la Turno -\

miCola.ingresar(10);
miCola.ingresar(20);
miCola.ingresar(30);

-/Mostrar el primer elemento del Turno-\

Mostrar(“Primer elemento: “, miCola.frente()); -/miCola.frente() indica el inicio del Turno-\

-/Eliminar el primer elemento del Turno-\

miCola.quitar();

-/Eliminar el primer elemento del Turno-\
Mostrar(“Cantidad de elementos en la cola: “,micola.tamaño());

}
```