Instituto Tecnológico de Costa Rica

Escuela de Computación

Compiladores e intérpretes

Etapa cero

Profesor:
Kirstein Gätjens Soto

Estudiantes:
David Acuña López – 2020426228
Deylan Sandoval – 2020234274

Cartago, Costa Rica

10 de marzo del 2025, I semestre

Índice

Concurso de nombres

Propuesta del nombre: CraftCode

Logo:

Extensión: .mcr

Descripción: La creación del lenguaje de programación CraftCode se basó principalmente en el mundo de Minecraft tomando como referencia sus elementos. Cada instrucción y estructura de control utiliza palabras clave relacionadas con el juego, lo que facilita su comprensión para los jugadores familiarizados con este universo. Además, CraftCode mantiene una sintaxis sencilla y flexible, permitiendo a los programadores realizar operaciones básicas y avanzadas con una lógica similar a la de los lenguajes de programación tradicionales, pero con un enfoque temático y creativo.

Definición del lenguaje

1. Estructura del título del programa: **minecraft**

1. Se escoge porque es el centro que crea todo como crear el juego o crear el programa

```
/* "Primer programa" es el nombre del nuevo programa que se va
a crear */
minecraft "Primer programa";
```

2. Sección de constantes: **casa**

1. Se escoge porque en la casa se guardan todas las cosas se van a estar usando constantemente

```
casa
    obsidiana exactcoord valor_constante = 3.14;
```

3. Sección de tipos: **mobs**

1. Se escoge porque los mobs son todos los tipos de creaturas que existen

```
mobs
    posicion => coord;
```

4. Sección de variables: **aldea**

1. Se escoge porque los recursos y las formas de las aldeas son variables

```
aldea
    coord entero = -123;
    pluma caracter = 'K';
    libro string = "Hola Mundo";
    lampara boolenaTrue = Encendido;
    lampara booleanFalse = Apagado;
    inventario conjunto = {: 'a', 'e', 'i', 'o', 'u' :};
    librero archivo = {/ "Archivo.txt" , 'L' /};
    exactcoord flotante = -3.45;
    cofre arreglo = [ 1 , 2 , 3 ];
    shulker registros = {* "hola", 8, 'k' *};
```

5. Sección de prototipos: **soporte**

1. Se escoge porque el prototipo es la base de la armadura, pero no hace nada sin la armadura

```
soporte
    encantar coord sumar(coord num1,coord num2);
```

6. Sección de rutinas: **armadura**

1. Se escoge porque es lo que pone en el soporte y este si puede ser utilizada

```
armadura
    encantar coord sumar(coord num1,coord num2)
  puerta
        coord suma = num1 + num2;
        crafteo suma;
    muro;
```

7. Punto de entrada del programa: **madera**

1. Se escoge porque la madera es lo primero que se busca cuando empieza un mundo

```
//aqui se inicia el programa
madera
```

8. Sistema de asignación de constantes: **obsidiana**

1. Se escoge porque cuando se coloca una obsidiana va ser muy difícil que cambie
2. obsidiana <tipo> id = <literal>

```
casa
    obsidiana exactcoord valor_constante = 3.14;
```

9. Sistema de asignación de tipos: **=>**

1. Se escoge porque es una variable de la adignacion =
2. id => <tipo>

```
mobs
    posicion => coord;
```

10. Sistema de declaración de variables: **=**

1. Se escoge porque es una asignación común en los nuevos lenguajes
2. <tipo> id = <inicialización>

```
coord entero = -123;
```

11. Tipo de dato entero: **coord**

1. Se escoge porque una coordenada es un numero entero en el juego

```
coord entero = -123;
```

12. Tipo de dato caracter: **pluma**

1. Se escoge porque la pluma es lo que se utiliza para escribir en libros y un string necesita caracteres para formarse

```
pluma caracter = 'K';
```

13. Tipo de dato string: **libro**

1. Se escoge porque un libro contine lo que escribe una pluma y un string contiene los caracteres

```
libro string = "Hola Mundo";
```

14. Tipo de dato booleano: **lampara**

1. Se escoge porque es un objeto que esta en dos estados encendido o apagado

```
lampara boolenaTrue = Encendido;
lampara booleanFalse = Apagado;
```

15. Tipo de dato conjunto: **inventario**

1. Se escoge porque es un lugar donde se pueden guardar datos

```
inventario conjunto = {: ‘a’, ‘e’, ‘i’, ‘o’, ‘u’ :};
```

16. Tipo de dato archivo de texto: **librero**

1. Se escoge porque un librero contiene varios libros y un archivos contiene varios strings

```
librero archivo = {/ "Archivo.txt" , 'L' /};
```

17. Tipo de datos números flotantes: **exactcoord**

1. Se escoge porque para dar una coordenada mas exacta de debe dar con decimales

```
exactcoord flotante = -3.45;
```

18. Tipo de dato arreglos: **cofre**

1. Se escoge porque en un cofre se pueden guardar datos

```
cofre arreglo = [ 1 , 2 , 3 ];
```

19. Tipo de dato registros: **shulker**

1. Se escoge porque es la mejor manera de guardar y transportar datos

```
shulker registros = {* ”hola”, 8, ‘k’ *};
```

20. Literales booleanas: **Encendido** - **Apagado**

1. Se escoge porque son los dos estados de la lampara

```
lampara boolenaTrue = Encendido;
lampara booleanFalse = Apagado;
```

21. Literales de conjuntos: **{: :}**

1. Se escoge porque un delimitador común es { } y para diferenciar de los otros tipos se agrega :
2. {: ‘a’, ‘e’, ‘i’, ‘o’, ‘u’ :}

```
inventario conjunto = {: ‘a’, ‘e’, ‘i’, ‘o’, ‘u’ :};
```

22. Literales de archivos: **{/ /}**

1. Se escoge porque un delimitador común es { } y para diferenciar de los otros tipos se agrega /
2. {/ "Archivo.txt" , 'L' /}.

```
librero archivo = {/ "Archivo.txt" , 'L' /};
```

23. Literales de números flotantes: **-3.45**

1. Se escoge porque es una forma convencional de redactar números, con – para negativos y . para diferenciar cuando se varios parámetros

```
exactcoord flotante = -3.45;
```

24. Literales de enteros: **-123, 0xF4EC**

1. Se escoge porque es la forma normal de hacerlo

```
coord entero = -123;
```

25. Literales de caracteres: **'K'**

1. Se escoge porque en los nuevos lenguajes los caracteres se represetan con una comilla sencilla

```
pluma caracter = 'K';
```

26. Literales de strings: **"Hola Mundo"**

1. Se escoge porque la forma común en todos los lenguajes se hace con comillas dobles

```
libro string = "Hola Mundo";
```

27. Literales de arreglos: **[ ]**

1. Se escoge porque es igual a la nomenclatura en java
2. [1 , 2 , 3]

```
cofre arreglo = [ 1 , 2 , 3 ];
```

28. Literales de registros: **{* *}**

1. Se escoge porque un delimitador común es { } y para diferenciar de los otros tipos se agrega *
2. {* ”hola”, 8, ‘k’ *}

```
shulker registros = {* ”hola”, 8, ‘k’ *};
```

29. Sistema de acceso arreglos: **[#]**

1. Se escoge porque es la nomenclatura común en los lenguajes alto nivel

```
///Acceso a una posicion en un arreglo

cofre arreglo = [ 1 , 2 , 3 ];

///El valor sera: 2
coord resultado = arreglo[1];
```

30. Sistema de acceso strings: **$¢**

1. Se escoge porque $ representa el string y ¢ porque representa el carácter

```
//El valor sera: 'l'
libro recortar =  "Hola mundo" $¢  2;
```

31. Sistema de acceso registros: **@**

1. Se escoge porque es una forma común de acceder a los caracteres

```
///Acceso a una campo en un registro

estudiante => sulker;
estudiante@carnet := 2020352842;

///EL valor sera: 2020352842
valor := estudiante@carnet.
```

32. Asignación y Familia: **= += -= *= %= /=**

1. Se escogen porque es la forma similar a Python y de fácil entendimiento

```
//Asignacion y familia

//con = se asigana el valor dado
coord num1 = 4;
coord num2 = 2;

//el valor de num1 va a ser: 6
num1 += num2;

//el valor de num1 va a ser:2
num1 -= num2;

//el valor de num1 va a ser: 4
num1 *= num2;

//el valor de num1 va a ser: 0
num1 %=  num2;

//el valor de num1 va a ser: 2
num1 //=  num2;
```

33. Operaciones aritméticas básicas de enteros: **+ - * % / //**

1. Se escoge porque son la nomenclatura básica de símbolos en Python

```
//Operaciones aritmeticas sobre enteros
coord num1 = 4;
coord num2 = 2;
coord suma = num1 + num2;
coord resta = num1 - num2;
coord multiplicacion = num1 * num2;
coord residuo = num1 % num2;
coord divisionEntera = num1 // num2;
exactcoord division =  num1 / num2;
```

34. Incremento y Decremento: **++ --**

1. Se escoge porque en la mayoría de lenguajes usa esta nomenclatura y es de fácil entendimiento

```
///incremento y drecremento
coord num1 = 4;

///El valor de num_incremento va a ser 5
coord num_incremento = num1++;

///El valor de num_decremetno va a ser 3
coord num_decremetno = num1--;
```

35. Operaciones básicas sobre caracteres: **₡# - ₡â - ₡Ã - ₡ã**

1. Se escoge porque ₡ representa caracteres, con # para representar un numero, â con la flecha para saber que también cuenta las mayúsculas y Ã – ã para saber su es mayúscula o minúscula

```
///Operaciones sobre caracteres

///El valor va a ser Falso
lampara esdigito = ₡# 'k';

///El valor va ser verdadero
lampara esAlpha = ₡â  'k';

///El valor va a ser falso
lampara esmayusucla = ₡Ã 'k';

///El valor va a ser verdadero
lampara esminuscula = ₡ã 'k';
```

36. Operaciones lógicas solicitadas: **!! ¡ !& !@**

1. Se escoge porque ! por simplicidad y facilidad de recordar y de utilizar

2. Además tiene una estructura sintáctica agradable

```
///Operación lógica y
lampara variable1 = Encendido;
lampara variable2 = Apagado;
lampara operacion_y = variable1 !! variable2;  ///Apagado
```

```
///Operación lógica o
lampara variable1 = Encendido;
lampara variable2 = Apagado;
lampara operacion_o = variable1 !¡ variable2;  ///Encendido
```

```
///Operación lógica no
lampara variable = Encendido;
lampara operacion_no = !&variable; ///Apagado
```

```
///Operación lógica xor
lampara variable1 = Encendido;
lampara variable2 = Apagado;
lampara operacion_xor = variable1 !@ variable2;  ///Encendido
```

37. Operaciones de Strings solicitadas: **$+ - $# - $>< - $< - $@**

1. Se escoge porque $ presenta un string con:
    i. + para saber se concatenan como varios lenguajes
    ii. – porque representa que se va a borrar el carácter
    iii. # representa el un numero y largo devuelve un numero
    iv. >< porque cortar devuelve el string entre dos posiciones
    v. < porque devuelve el string antes de una posición

```
//Operaciones sobre strings


//El valor sera: "Hola mundo"
libro concatenar = "Hola " $+ "mundo";

//El valor sera:4
coord largo = $# "Hola";

//El valor sera: "la mu"
libro cortar =  "Hola mundo" $>< [2,7];

//El valor sera: "Ho"
libro recortar =  "Hola mundo" $< 2;

//El valor sera: 2
coord encontrar = "Hola mundo" $@ 'l';
```

38. Operaciones de conjuntos solicitadas: agrega = **©+ - ©- - ©µ - ©¡ - ©Þ - ©ε**

1. Se escoge porque © representa un conjunto con:
    i. + para agregar un datos
    ii. – para borrar un dato
    iii. µ para dar un conjunto con todos los datos
    iv. ¡ para representar los datos que están en los dos conjutos
    v. Þ representa si un datos esta en un conjunto
    vi. ε como épsilon representa vacio

```
///Operaciones sobre conjuntos

inventario conjunto1 = {: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ :};
inventario conjunto2 = {: ‘d’, ‘e’, ‘f’, ‘g’, ‘h’ :};


///El valor sera: {: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’ :}
inventario agrega = conjunto1 @+ 'f'

///El valor sera:{: ‘a’, ‘b’, ‘d’, ‘e’ :};
inventario borrar = conjunto1 @- 'c'

///El valor sera: {: ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’ :}
inventario union = conjunto1 @µ conjunto2

///El valor sera:{:  ‘d’, ‘e’ :}
inventario interseccion = conjunto1 @¡ conjunto2

///El valor sera: Encendido
lampara pertenecer = conjunto1 @Þ 'c'

///El valor sera: Apagado
lampara vacio = @ε conjunto1
```

39. Operaciones de archivos solicitadas: **ƒo – ƒc - ƒw – ƒr - ƒn - ƒa**

1. Se escoge porque ƒ representa un file con :
    i. o para dar open
    ii. c para dar close
    iii. w para dar write
    iv. r para dar read
    v. n para dar new
    vi. a para dar asociar

```
///Operaciones sobre archivos


librero archivo = {/ "Archivo.txt" , 'L' /};
librero archivo2 = {/ "Archivo2.txt" , 'L' /};

///Abre el archivo
fo archivo
fo archivo2

///extrae el contenido del archivo
libro contenido = fr archivo

///Escribre texto en el archivo
archivo fe  "Hola mundo"

///Crea un nuevo archivo
fn  "Nuevo.txt"

///Asocia 2 archivos
archivo fa archivo2



///Cierra el archivo
fc archivo
fc archivo2
```

40. Operaciones de números flotantes: **+ - * % / (SELECCIONADO)**

1. Se escoge porque porque es la nomenclatura básica en Python

```
///Operaciones aritmeticas sobre flotantes

exactcoord num1 = 4.0;
exactcoord num2 = 3.0;
exactcoord suma = num1 + num2;
exactcoord resta = num1 - num2;
exactcoord multiplicacion = num1 * num2;
exactcoord residuo = num1 % num2;
exactcoord division =  num1 / num2;
```

41. Operaciones de comparación solicitadas: **<= >= == ><**

1. Se escoge porque es la nomenclatura común en los lenguajes alto nivel

```
//operaciones de comparacion

///el valor sera Encendido
lampara mayorQue = 2>2

///el valor sera Apagado
lampara menorQue = 2<2

///el valor sera Encendido
lampara menorIgualQue = 2<=2

///el valor sera Encendido
lampara mayorIgualQue = 2>=2

///el valor sera Encendido
lampara IgualQue = 2==2

///el valor sera Apagado
lampara diferenteQue = 2!=2
```

42. Manejo de Bloques de más de una instrucción: **puerta** - **muro**

1. Se escoge porque la puerta abre para entrar a un lugar y el muro detiene el acceso

```
armadura
    encantar coord sumar(coord num1,coord num2)
  puerta
        coord suma = num1 + num2;
        crafteo suma;
    muro;
```

43. Instrucción while: **nether**

1. Se escoge porque el nether es un lugar al que se entra y se puede quedar ahí todo el tiempo hasta que se decide salir

```
//Se utiliza while  break  if-then-else


coord contador = 0;
lampara hacer = Encendido;
nether (lampara)
    puerta
        palanca (contador != 1) antorcha_redstone
            puerta
                contador -= 1
            muro;
        trampa
            bedrock;

    muro;
```

44. Instrucción if-then-else: **palanca – actorcha_resdtone - trampa**

1. Se escoge porque la palanca si se activa enciende la antorcha y la trampa se activa con solo pasar por encima


```
//Se utiliza while  break  if-then-else


coord contador = 0;
lampara hacer = Encendido;
nether (lampara)
    puerta
        palanca (contador != 1) antorcha_redstone
            puerta
                contador -= 1
            muro;
        trampa
            bedrock;

    muro;
```


45. Instrucción switch: **gancho - hilo - sensor**

1. Se escoge porque un gancho es un sistema que activa un función pero necesita el hilo para que sea activado y se usa el sensor para que se active siempre

```
///Se utiliza for y continue y swicth

repetidor (coord i = 0; i < 10; i++)
    puerta
        gancho (i)
            puerta
                hilo 2;
                    puerta
                        bedrock;
                    muro;

                sensor;
                    puerta
                        slime
                    muro;

            muro

    muro
```

46. Instrucción Repeat-until: **activo** - **observador**

1. Se escoge porque un observador activar una activa una actividad, por lo que si se cumple la condición en el observador de entra a activo

```
///Se utiliza Repeat-until


coord contador = 0;
lampara hacer = Encendido;
activo
    puerta
        palanca (contador <= 1) antorcha_redstone
            hacer = Apagado;
        trampa
            puerta
                contador -= 1
            muro;


    muro;
observador hacer;
```

47. Instrucción For: **repetidor**

1. Se escoge porque un repetidor ayuda a repretir una acción varias veces en un ciclo igual

```
///Se utiliza for y continue y swicth

repetidor (coord i = 0; i < 10; i++)
    puerta
        gancho (i)
            puerta
                hilo 2;
                    puerta
                        bedrock;
                    muro;

                sensor;
                    puerta
                        slime
                    muro;

            muro

    muro
```

48. Instrucción With: **yunque**

1. Se escoge porque el yunque permite la manipulación de las herramientas

49. Instrucción break: **bedrock**

1. Se escoge porque la bedrock puede tener cualquier acción

```
//Se utiliza while  break  if-then-else


coord contador = 0;
lampara hacer = Encendido;
nether (lampara)
    puerta
        palanca (contador != 1) antorcha_redstone
            puerta
                contador -= 1
            muro;
        trampa
            bedrock;

    muro;
```

50. Instrucción continue: **slime**

1. Se escoge porque si caes en slime puede saltar y en un continue salta la acción
2.

```
///Se utiliza for y continue y swicth

repetidor (coord i = 0; i < 10; i++)
    puerta
        gancho (i)
            puerta
                hilo 2;
                    puerta
                        bedrock;
                    muro;

                sensor;
                    puerta
                        slime
                    muro;

            muro

    muro
```

51. Instrucción Halt: **barrera**

1. Se escoge porque una barrera invisible puede tener donde se puede jugar en minecraft

```
/// Instrucción halt
repetidor (coord i = 0; i < 10; i++)
    puerta
        gancho (i)
            puerta
                hilo 2;
                    puerta
                        bedrock;
                        barrera; /// halt, se detiene el programa
                    muro;

                sensor;
                    puerta
                        slime
                    muro;

            muro
    muro
```

Este es el mismo ejemplo que el anterior pero con la instrucción halt en medio que detiene todo el programa.

52. Encabezado de funciones: **encantar**

1. Se escoge porque es una acción en el juego que sabemos que vamos a dar algo y nos va a retornar un resultado
2. encantar <tipo> id (<params>)

```
soporte
    encantar coord sumar(coord num1,coord num2);
```

53. Encabezado de procedimientos: **redstone**

1. Se escoge porque con la redstone se pueden hacer sistema que no devuelven un valor

```
///se utiliza encabezado de procedimientos, manejo de entrada
y salida estandar, y cohersion de tipos


redstone mesesEnAno()
  puerta
      libro ano = tolva("Cuantos años");
      coord meses = horno(libro,coord,ano) * 12
      cartel(meses)

    muro;
```

54. Manejo de parámetros formales: **(<tipo> id ,id . <tipo> id)**

1. Se escoge porque es la nomenclatura común en los lenguajes de alto nivel

```
soporte
    encantar coord sumar(coord num1,coord num2);
```

55. Manejo de parámetros reales: **(5,A,4,B)**

1. Se escoge porque es la nomenclatura común en los lenguajes de alto nivel

```
/* Aqui se llama funcion que ha sido creada
 recibe 2 numeros enteros */
 //tambien se usa el manejo de parametros reales con 1,2
 sumar(1,2);
```

56. Instrucción return: **crafteo**

1. Se escoge porque hacer nuevos objetos se hacer por medio de crafteo y devuelven el objeto nuevo

```
armadura
    encantar coord sumar(coord num1,coord num2)
  puerta
        coord suma = num1 + num2;
        crafteo suma;
    muro;
```

57. Operación de size of: **stack**

1. Se escoge porque un stack nos da información de cuantos objetos hay un stack

```
/// stack (sizeof)
coord tamanoChar = stack(pluma); //1 que se refiere un byte
```

58. Sistema de coherción de tipos: **horno**

1. Se escoge porque en el horno objetos pueden ser convertidos en otros
2. horno (<tiposrc>,<tipodest>, literal)

```
///se utiliza encabezado de procedimientos, manejo de entrada
y salida estandar, y cohersion de tipos


redstone mesesEnAno()
  puerta
        libro ano = tolva("Cuantos años");
        coord meses = horno(libro,coord,ano) * 12
        cartel(meses)

    muro;
```

59. Manejo de la entrada estándar: **tolva**

1. Se escoge porque una tolva recibe objetos

```
///se utiliza encabezado de procedimientos, manejo de entrada
y salida estandar, y cohersion de tipos


redstone mesesEnAno()
  puerta
        libro ano = tolva("Cuantos años");
        coord meses = horno(libro,coord,ano) * 12
        cartel(meses)

    muro;
```

60. Manejo de la salida estándar: **cartel**

1. Se escoge porque es un lugar donde se puede dejar información para que los demás lo vean

```
///se utiliza encabezado de procedimientos, manejo de entrada
y salida estandar, y cohersion de tipos


redstone mesesEnAno()
  puerta
      libro ano = tolva("Cuantos años");
      coord meses = horno(libro,coord,ano) * 12
      cartel(meses)

    muro;
```

61. Terminador o separador de instrucciones - Instrucción nula: **;**

1. Se escoge porque es la nomenclatura común en los lenguajes

```
armadura
    encantar coord sumar(coord num1,coord num2)
  puerta
      coord suma = num1 + num2;
      crafteo suma;
    muro;
```

62. Todo programa se debe cerrar con un: **end**

1. Se escoge porque el end es el final del juego

```
// Aqui se representa donde termina el programa
end;
```

63. Comentario de Bloque: **///**

1. Se escoge porque un comentario común es // pero para diferencia con la division entera se agrega un / mas

```
/* Aqui se llama funcion que ha sido creada
 recibe 2 numeros enteros */
 ///tambien se usa el manejo de parametros reales con 1,2
 sumar(1,2);
```

64. Comentario de Línea: **/* */**

1. Se escoge porque es la nomenclatura común en java

```
/* Aqui se llama funcion que ha sido creada
 recibe 2 numeros enteros */
 ///tambien se usa el manejo de parametros reales con 1,2
 sumar(1,2);
```

65. Tipo creativo: temperatura

- Este tipo de dato sirve para manipular información de temperaturas en grados celcius, Fahrenheit y kelvin.

```
/* tipo creativo temperatura,  12 grados celsius*/
temperatura varTemp = 12C;
```

66. Literal para el tipo creativo:
    - Las literales para el tipo creativos son números enteros normales, simplemente se le añade una C, F o K según el tipo de temperatura que sea.

```
/* tipo creativo temperatura,  12 grados celsius*/
temperatura varTemp1 = 12C;
temperatura varTemp2 = 12F;
temperatura varTemp3 = 12K;
```

Operaciones

Transformar: transf<entrada><salida>(valor)

- Se utiliza el transf seguido de las letras en mayúscula de la temperatura de entrada y la temperatura de salida
- Transforma un tipo de temperatura a otra. Es importante mencionar que no es lo mismo que simplemente escribir el otro tipo de temperatura en la misma variable ya que no se haría la conversión
- Este es un ejemplo de Celsius a Kelvin

```
/* operacion transformar del tipo creativo
   de celcius a kelvin*/
temperatura varTemp = 12C;
temperatura varTemp1 = transfCK(varTemp); /// 285K
```

Ajustar: ajus<entrada><salida>(<valor1>, <valor2>)

- Se utiliza el ajus seguido de las letras en mayúscula de la temperatura de entrada y la temperatura de salida
- Este permite sumar dos tipos de temperaturas distintas

```
/* operacion ajustar del tipo creativo
   de celcius a kelvin*/
temperatura varTemp = 12C;
temperatura varTemp1 = ajusCK(varTemp, 12K); /// 297K
```

Reducir: red<temp entrada><temp salida>(<exp1>, <exp2>)

- Se utiliza el red seguido de las letras en mayúscula de la temperatura de entrada y la temperatura de salida
- Este permite saber la diferencia entre dos tipos de temperaturas

```
/* operacion reducir del tipo creativo
   de celcius a kelvin*/
temperatura varTemp = 12C;
temperatura varTemp1 = redCK(varTemp, 12K); /// 273K
```

Comparación: comp<entrada><salida>(<valor1>, <valor2>)

- Se utiliza el comp seguido de las letras en mayúscula de la temperatura de entrada y la temperatura de salida
- Este permite comparar dos tipos de temperaturas para saber si son iguales.

```
/* operacion comparación del tipo creativo
   de celcius a kelvin*/
temperatura varTemp = 12C;
temperatura varTemp1 = compCK(varTemp, 285K); /// Encendido
```

67. Dos Operaciones adicionales sobre el tipo entero:

1) **Promedio de temperaturas:** promtemp<entrada>(<valor1>, <valor1>, ...)
   - Este nos permite sacar un promedio de un tipo de temperatura
   - Las temperaturas deben de ser de un mismo tipo

```
/* promedio del tipo creativo
   de celcius a kelvin*/
temperatura varTemp = promtempC(12C, 34C, 22C, 54C); /// 30.5C
```

2) **Diferencia absoluta:** difabs<entrada><salida>(<valor1>, <valor2>)
    - Este nos permite sacar un promedio de un tipo de temperatura
    - Las temperaturas deben de ser de un mismo tipo

```
/* diferencia absoluta del tipo creativo
   de celcius a kelvin*/
temperatura varTemp = difabsCC(20C, 23C); /// 3C
```

68. Dos Instrucciones que no sean fácilmente implementables con lo que ya tenga el lenguaje:
    1) **Sensación térmica:** termtemp<entrada>(<dato1>, <dato2>, <factor>)

    2) **Interpolación de temperaturas:** interptemp<entrada>(<dato1>, <dato2>, <factor>)

        - Permite interpolar una temperatura entre dos puntos y un factor de interpolación entre 0 y 1

```
/* Interpolación del tipo creativo
   de celcius*/
temperatura varTemp = intertemp(20C, 30C, 0.5); /// 25C
```

Índice de pruebas

- test-mcr-01-Programa inicial
- test-mcr-02-Seccion de constantes
- test-mcr-03-Seccion de tipos
- test-mcr-04-Seccion de variables
- test-mcr-05-Prototipo
- test-mcr-06-Funcion
- test-mcr-07-Arreglo
- test-mcr-08-Operaciones de string
- test-mcr-09-Registro
- test-mcr-10-Asignacion y familia
- test-mcr-11-Operaciones enteros
- test-mcr-12-Incremento y decrement

- test-mcr-13-Operaciones caracteres
- test-mcr-14-Operaciones conjuntos
- test-mcr-15-Operaciones archivos
- test-mcr-16-Operaciones flotantes
- test-mcr-17-Operaciones comparacion
- test-mcr-18-While con if-else
- test-mcr-19-For con switch
- test-mcr-20-Repeat until
- test-mcr-21-Procedimiento
- test-mcr-22-Creativo
- test-mcr-23-Logica
- test-mcr-24-Halt-Sizeof

Gramática

Estructura del título del programa

1. <inicio> ::= <inst_inicio>

<inst_inicio> ::= minecraft <identificador> <terminador>

Sección constantes

2. <seccion_constantes> ::= casa <sistema_asignacion>

Sección de tipos

3. <seccion_tipos> ::= mobs <sistema_asignacion>

Sección de variables

4. <seccion_variables> ::= aldea <declaracion_variable>

Sección de prototipos

5. <seccion_prototipos> ::= soporte <encabezado_funcion>

Sección de rutinas

6. <seccion_rutinas> ::= armadura <instrucciones>

Punto de entrada del programa

7. <punto_entrada> ::= madera <programa>

Sistema de asignación de constantes

8. <sistema_asignacion> ::= obsidiana <tipo> <identificador> <asignacion> <literal> <terminador>

Sistema de asignación de tipos

9. <sistema_asignacion> ::= <identificador> => <tipo> <terminador>

Sistema de declaración de variables

10. <declaracion_variable> ::= <tipo> <identificador> <asignacion> <literal> <terminador>

Tipo de dato entero

11. <tipo> ::= coord

Tipo de dato caracter

12. <tipo> ::= pluma

Tipo de dato string

13. <tipo> ::= libro

Tipo de dato booleano

14. <tipo> ::= lampara

Tipo de dato conjunto

15. <tipo> ::= inventario

Tipo de dato archivo de texto

16. <tipo> ::= librero

Tipo de datos números flotantes

17. <tipo> ::= exactcoord

Tipo de dato arreglos

18. <tipo> ::= cofre

Tipo de dato registros

19. <tipo> ::= shulker

Literales booleanas

20. <booleano> ::= Encendido
21. <booleano> ::= Apagado

Literales de números flotantes

22. <literal> ::= <literal_flotante>
23. <literal_flotante> ::= _numero_flotante <terminador>

Literales de enteros

24. <literal> ::= <literal_entero>
25. <literal_entero> ::= _numero_entero <terminador>

Literales de caracteres

26. <literal> ::= <literal_caracter>
27. <literal_caracter> ::= _caracter <terminador>

Literales de strings

28. <literal> ::= <literal_string>
29. <literal_string> ::= " _string " <terminador>

Literales de arreglos

30. <literal> ::= <literal_arreglo>
31. <literal_arreglo> ::= [_arreglo] <terminador>

Literales de registros

32. <literal> ::= <literal_registro>
33. <literal_registro> ::= {* _registro *} <terminador>

Sistema de acceso arreglos

34. <sistema_arreglo> ::= _identificador "[" <indice> "]"

Sistema de acceso strings

35. <sistema_string> ::= _string "$₡" <indice>

Sistema de acceso registros

36. <sistema_registros> ::= _registro "@" <identificador>

Asignación y Familia

37. <asignacion> ::= =
38. <asignacion> ::= +=
39. <asignacion> ::= -=
40. <asignacion> ::= *=
41. <asignacion> ::= %=
42. <asignacion> ::= /=

Operaciones aritméticas básicas de enteros

43. <operacion_aritmetica> ::= "+"
44. <operacion_aritmetica> ::= "-"
45. <operacion_aritmetica> ::= "*"
46. <operacion_aritmetica> ::= "%"
47. <operacion_aritmetica> ::= "/"
48. <operacion_aritmetica> ::= "//"

Incremento y Decremento

49. <operacion_incre_decre> ::= "++"
50. <operacion_incre_decre> ::= "--"

Operaciones básicas sobre caracteres

51. <operacion_careacter> ::= ₡#
52. <operacion_careacter> ::= ₡â
53. <operacion_careacter> ::= ₡Ã
54. <operacion_careacter> ::= ₡ã

Operaciones lógicas solicitadas

55. <operacion_logica> ::= "¡¡"
56. <operacion_logica> ::= "¡!"

57. <operacion_logica> ::= "¡&"
58. <operacion_logica> ::= "¡@"

Operaciones de Strings solicitadas

59. <operacion_string> ::= "$+"
60. <operacion_string> ::= "$#"
61. <operacion_string> ::= "$><"
62. <operacion_string> ::= "$<"
63. <operacion_string> ::= "$@"

Operaciones de conjuntos solicitadas

64. <operacion_conjunto> ::= "©+"
65. <operacion_conjunto> ::= "©-"
66. <operacion_conjunto> ::= "©µ"
67. <operacion_conjunto> ::= "©¡"
68. <operacion_conjunto> ::= "©Þ"
69. <operacion_conjunto> ::= "©ε"

Operaciones de comparación solicitadas

70. <operacion_comparacion> ::= "<="
71. <operacion_comparacion> ::= ">="
72. <operacion_comparacion> ::= "=="
73. <operacion_comparacion> ::= "><"

Manejo de Bloques de más de una instrucción

74. <bloque> ::= puerta <instrucciones> muro

Instrucción while

75. <ciclo> ::= nether "(" <expresion> ")" <bloque>

Instrucción if-then-else

76. <instruccion_if> ::= palanca <expresion> antorcha_redstone <bloque> <else_opcional>
77. <else_opcional> ::= trampa <bloque>
78. <else_opcional> ::= ε

Instrucción switch

79. <instruccion_switch> ::= gancho "(" <expresion> ")" <bloque_switch>
80. <bloque_switch> ::= puerta <lista_casos> <caso_default> muro
81. <instruccion_switch> ::= gancho "(" <expresion> ")" <bloque_switch>
82. <bloque_switch> ::= puerta <lista_casos> <caso_default> muro
83. <lista_casos> ::= <caso> <lista_casos2>
84. <lista_casos2> ::= <caso> <lista_casos2>
85. <lista_casos2> ::= ε
86. <caso> ::= hilo <expresion> <bloque>
87. <caso_default> ::= sensor <bloque>
88. <caso_default> ::= ε

Instrucción Repeat-until

89. <ciclo> ::= activo <bloque> observador "(" <expresion> ")"

Instrucción For

90. <ciclo> ::= repetidor "(" <asignacion_for> ";" <expresion> ";" <actualizacion> ")" <bloque>

Instrucción With

91. <instruccion_with> ::= yunque "(" <expresion> ")" <bloque>

Instrucción break

92. <instruccion_break> ::= bedrock

Instrucción continue

93. <instruccion_continue> ::= slime

Instrucción Halt

94. <instruccion_halt> ::= barrera

Encabezado de funciones

95. <encabezado_funcion> ::= encantar <tipo> <identificador> "(" <params> ")"

Encabezado de procedimientos

96. <encabezado_procedimiento> ::= redstone <identificador> "(" <params> ")" <bloque>

Manejo de parámetros formales

97. <params> ::= <param> <params2>
98. <params2> ::= "," <param> <params2>
99. <params2> ::= ε
100. <param> ::= <tipo> <identificador>

Manejo de parámetros reales

101. <paramsr> ::= <paramr> <paramsr2>
102. <paramsr2> ::= "," <paramr> <paramsr2>
103. <paramsr2> ::= ε
104. <paramr> ::= <tipo> <identificador>

Instrucción return

105. <instruccion_return> ::= crafteo

Operación de size of

106. <operacion_sizeof> ::= stack "(" <tipo> ")"

Sistema de coherción de tipos

107. <sistema_cohercion> ::= horno "(" <tipo> "," <tipo> "," <literal> ")"

Manejo de la entrada estándar

108. <entrada_estandar> ::= tolva "(" <mensaje> ")"

Manejo de la salida estándar

109. <salida_estandar> ::= cartel "(" <expresion> ")"

Terminador o separador de instrucciones - Instrucción nula

110. <terminador> ::= ;

Todo programa se debe cerrar con un

111. <end> ::= end <terminador>

Comentario de Bloque

112. <comentario_linea> ::= "/*" <texto> "*/"

Comentario de Línea

113. <comentario_bloque> ::= "///" <texto>

Tipo dato creativo

114. <tipo> ::= temperatura

Otros

115. <mensaje> ::= <literal_string>
116. <indice> ::= _numero_entero
117. <identificador> ::= <identificador>
118. <asignacion_for> ::= <tipo> <identificador> <asignacion> <literal>