



Instituto Tecnológico de Costa Rica

Escuela de Computación

Compiladores e intérpretes

Etapas 1

Profesor:

Kirstein Gätjens Soto

Estudiantes:

David Acuña López – 2020426228

Deylan Sandoval – 2020234274

Cartago, Costa Rica

28 de abril del 2025, I semestre

Índice

Concurso de nombres	3
Definición del lenguaje	4
Índice de pruebas	33
Listado de palabras reservadas	35
Algoritmos de conversión	36
Gramática actualizada	37
Tabla de presencia	49
Listado de errores y recuperación	50
Autómatas	51

Concurso de nombres

Propuesta del nombre: Notch Engine

Logo:



Extensión: .ne

Descripción: El nombre fue escogido en homenaje a Markus Persson, también conocido como Notch, creador y padre fundador de Minecraft como un título independiente, autoría de Mojang, compañía que le perteneció desde 2009 hasta 2014. El logo representa un bloque similar a los que se encuentran en el videojuego, con líneas transversales que lo atraviesan en todas sus caras, simulando los procesos y el transporte de información que se da a lo interno una vez se programa en Notch Engine.

Definición del lenguaje

1. Estructura del título del programa: **WorldName <id>**:

Se escoge porque es el inicio del programa y da el nombre del programa, al igual que el worldname

```
$$ "Primer programa" es el nombre del nuevo programa que se va a crear
WorldName Primer programa:
```

2. Sección de constantes: **Bedrock**

Se escoge porque es un bloque que no se puede cambiar ni mover

```
Bedrock
obsidian ghastr valor_constante = 3.14;
```

3. Sección de tipos: **ResourcePack**

Se escoge porque los tipos también son los recursos para el programa

```
ResourcePack    $$ sección de tipos
Anvil Edad -> Stack ;
Anvil Nombre -> Spider ;
```

4. Sección de variables: **Inventory**

Se escoge porque los recursos que se guardan pueden estarse cambiando

```
ResourcePack    $$ sección de tipos
Anvil Edad -> Stack ;
Anvil Nombre -> Spider ;

Inventory        $$ sección de variable
Edad MiEdad = 455 , SuEdad = 20 , J ;
Nombre MiNombre = "Juan";
Rune MiLetra = 'K', suLetra = 'A' ;
Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
chest Juan = {"Juan",99,1111111111}, Pedro = {"peter",15,122223333};
torch estapresente = off;
```

5. Sección de prototipos: **Recipe**

Se escoge porque es como la formula o base para crear después un objeto

```
Recipe          $$ sección de prototipos

spell ImprimaN(Stack :: N);
```

6. Sección de rutinas: **CraftingTable**

Se escoge porque es donde se puede poner los recibe que ahora si hagan el objeto

```
Recipe      $$ sección de prototipos
spell ImprimaN(Stack :: N);

CraftingTable  $$ sección de rutinas

spell ImprimaN;  $$ como hay un prototipo los encabezados se pueden poner o no poner.
Inventory
Stack K = 0;
PolloCrudo
  repeater K<N Craft
  PolloCrudo
    dropperStack(K);
    soulSand K;
  PolloAsado;
PolloAsado;
```

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

Se escoge porque es el lugar donde todo empieza

```
$$aquí se inicia el programa
SpawnPoint
```

8. Sistema de asignación de constantes: **Obsidian** <tipo> <id> <value>

Se escoge porque cuando se coloca una obsidiana va ser muy difícil que cambie

```
Bedrock
obsidian ghast valor_constante = 3.14;
```

9. Sistema de asignación de tipos: **Anvil** <id> -> <tipo>

Se escoge porque es como colocar los objetos de herramientas

```
ResourcePack      $$ sección de tipos
Anvil Edad -> Stack ;
Anvil Nombre -> Spider ;
```

10. Sistema de declaración de variables: <tipo> id = <lit> , id = <lit>

Se escoge porque es una asignación común en los nuevos lenguajes

```
ResourcePack      $$ sección de tipos
    Anvil Edad -> Stack ;
    Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
    Edad MiEdad = 455 , SuEdad = 20 , J ;
    Nombre MiNombre = "Juan";
    Rune MiLetra = 'K', suLetra = 'A' ;
    Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
    Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
    Entity
        Spider Nombre;
        Edad   Vejez;
        Stack   Cedula;
    chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
    torch estapresente = off;
```

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

Se escoge porque el stack una cantidad entera de objetos

```
ResourcePack      $$ sección de tipos
    Anvil Edad -> Stack ;
    Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
    Edad MiEdad = 455 , SuEdad = 20 , J ;
    Nombre MiNombre = "Juan";
    Rune MiLetra = 'K', suLetra = 'A' ;
    Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
    Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
    Entity
        Spider Nombre;
        Edad   Vejez;
        Stack   Cedula;
    chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
    torch estapresente = off;
```

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

Se escoge porque la runa siempre es un solo simbolo

```
ResourcePack      $$ sección de tipos
    Anvil Edad -> Stack ;
    Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
    Edad MiEdad = 455 , SuEdad = 20 , J ;
    Nombre MiNombre = "Juan";
    Rune MiLetra = 'K', suLetra = 'A' ;
    Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
    Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
    Entity
        Spider Nombre;
        Edad   Vejez;
        Stack   Cedula;
    chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
    torch estapresente = off;
```

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

Se escoge porque una arana pueden unirse cosas, en este caso caracteres

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad  Vejez;
    Stack Cedula;
  chest Juan = {:"Juan",99,111111111:}, Pedro = {:"peter",15,122223333:} ;
  torch estapresente = off;
```

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

Se escoge porque es un objeto que esta en dos estados on y off

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad  Vejez;
    Stack Cedula;
  chest Juan = {:"Juan",99,111111111:}, Pedro = {:"peter",15,122223333:} ;
  torch estapresente = off;
```

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

Se escoge porque es un lugar donde se pueden guardar datos

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad  Vejez;
    Stack Cedula;
  chest Juan = {:"Juan",99,111111111:}, Pedro = {:"peter",15,122223333:} ;
  torch estapresente = off;
```

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

Se escoge porque un libro se puede decir que es igual a un archivo de texto

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
```

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

Se escoge porque puede andar flotando por el mundo

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghash flotante = -3,14;
```

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

Se escoge porque es un lugar donde se puede ordenar/colocar cosas

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghash flotante = -3,14;
```

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

Se escoge porque un registro se puede decir que es una entidad


```

ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;

```

20. Literales booleanas: **On/Off**

Se escoge porque son los dos estados de la lampara

```

ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;

```

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

Se escoge porque un delimitador común es { } y para diferenciar de los otros tipos se agrega :

```

ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;

```

22. Literales de archivos: {/ /}

Se escoge porque un delimitador común es { } y para diferenciar de los otros tipos se agrega /

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {:"Juan",99,111111111:}, Pedro = {:"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;
```

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

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {:"Juan",99,111111111:}, Pedro = {:"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;
```

24. Literales de enteros: 5,-5

Se escoge porque es la forma normal de hacerlo

```

ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;

```

25. Literales de caracteres: 'K'

Se escoge porque en los nuevos lenguajes los caracteres se representan con una comilla sencilla

```

ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;

```

26. Literales de strings: "Hola Mundo"

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

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;
```

27. Literales de arreglos: []

Se escoge porque es igual a la nomenclatura en java

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune MiLetra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {"Juan",99,111111111:}, Pedro = {"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;
```

28. Literales de registros: {<id>: <value>, <id>: <value>, ...}

Se escoge porque un delimitador común es {} y con una estructura comun a los lenguajes modernos

```
ResourcePack      $$ sección de tipos
  Anvil Edad -> Stack ;
  Anvil Nombre -> Spider ;

Inventory          $$ sección de variable
  Edad MiEdad = 455 , SuEdad = 20 , J ;
  Nombre MiNombre = "Juan";
  Rune Milettra = 'K', suLetra = 'A' ;
  Shelf Rune vocales[5] = ['a','e','i','o','u'], tri[3] = ['a','b','c'] , consonantes[22], matriz[10][10][10];
  Shelf Stack Mat3d[2][3][4] = [[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
  Entity
    Spider Nombre;
    Edad Vejez;
    Stack Cedula;
  chest Juan = {:"Juan",99,111111111:}, Pedro = {:"peter",15,122223333:} ;
  torch estapresente = off;
  book txt = {/ "Archivo.txt", 'L' /}
  ghastr flotante = -3,14;
```

29. Sistema de acceso arreglos: [#]

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

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

Shelf arreglo = [ 1 , 2 , 3 ];

$$El valor sera: 2
stack resultado = arreglo[1];
```

30. Sistema de acceso strings: [#]

Se escoge porque es una forma común en los lenguajes modernos

```
$$Operaciones sobre strings
spider palabra = "Hola"
rune caracter = palabra[1]
```

31. Sistema de acceso registros: '@ (e.g.: registro@campo)

Se escoge porque @ es un carácter común y que no se usan en otro lado

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

Entity{
  Spider Nombre;
  Edad Vejez;
  Stack Cedula;
}

$$EL valor sera: 2020352842
stack valor := estudiante@carnet.
```

32. Asignación y Familia: '=' (i.e: =, +=, -=, *=, /=, %=)

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

```
$$Asignacion y familia

$con = se asigana el valor dado
stack num1 = 4;
stack 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: + - * % //

Se escoge porque son la nomenclatura básica de símbolos en Python

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

34. Incremento y Decremento: **soulsand**, **magma**

Se escoge porque soulsand te baja la velocidad y el magma en el agua te hace subir

```

$$incremento y drecremento
stack num1 = 4;

$$El valor de num_incremento va a ser 5
stack num_incremento = magma num1;

$$El valor de num_decremetno va a ser 3
stack num_decremetno = soulsand num1;

```

35. Operaciones básicas sobre caracteres: **isEngraved**, **isInscribed**, **etchUp**, **etchDown**

Se escoge porque es una forma de fácil entendimiento y esta relacionado con minecraft

```

$$Operaciones sobre caracteres

$$El valor va a ser Falso
torch esdigito = isEngraved 'k';

$$El valor va ser verdadero
torch esAlpha = isInscribed 'k';

$$El valor va a ser falso
torch esmayuscula = etchUp 'k';

$$El valor va a ser verdadero
torch esminuscula = etchDown 'k';|

```

36. Operaciones lógicas solicitadas: **and**, **or**, **not**, **xor**

Se escoge porque son los nombres con los operadores con letras

```
$$Operación lógica y
torch variable1 = on;
torch variable2 = off;
torch operacion_y = variable1 and variable2; $$off

$$Operación lógica o
torch variable1 = on;
torch variable2 = off;
torch operacion_o = variable1 or variable2; $$on

$$Operación lógica no
torch variable = on;
torch operacion_no = not variable; $$off

$$Operación lógica xor
torch variable1 = on;
torch variable2 = off;
torch operacion_xor = variable1 xor variable2; $$on
```

37. Operaciones de Strings solicitadas: **bind, #, from ##,##, seek**

Se escoge porque es una forma fácil de entender que hacen las operaciones:

concatenar, largo, cortar, recortar, buscar. Cortar y Recortar son ternarias y usan doble # para evitar ambigüedad con el length


```
$$Operaciones sobre strings
spider palabra = "Hola"
rune character = palabra[1]

$$El valor sera: "Hola mundo"
spider concatenar = "Hola " bind "mundo";

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

$$El valor sera: "la mu"
spider cortar = "Hola mundo" from [2,7];

$$El valor sera: "Ho"
spider recortar = "Hola mundo" ## 2;

$$El valor sera: 2
stack encontrar = "Hola mundo" seek 'l';

$$El valor sera: 'l'
spider recortar = "Hola mundo" ## 2;
```

38. Operaciones de conjuntos solicitadas: **add, drop, items, feed, map, kill**

Se escoge porque es una forma fácil de entender que hacen las operaciones:
agregar, eliminar, unión, intersección, pertenece, vacío

```

$$Operaciones sobre conjuntos

chest conjunto1 = {:'a', 'b', 'c', 'd', 'e' :};
chest conjunto2 = {:'d', 'e', 'f', 'g', 'h' :};

$$El valor sera: {:'a', 'b', 'c', 'd', 'e', 'f' :}
chest agrega = conjunto1 add 'f'

$$El valor sera:{:'a', 'b', 'd', 'e' :};
chest borrar = conjunto1 drop 'c'

$$El valor sera: {:'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h' :}
chest union = conjunto1 items conjunto2

$$El valor sera:{:'d', 'e' :}
chest interseccion = conjunto1 feed conjunto2

$$El valor sera: Encendido
torch pertenecer = conjunto1 map 'c'

$$El valor sera: Apagado
torch vacio = kill conjunto1

```

39. Operaciones de archivos solicitadas: **unlock, lock, craft, gather, forge**

Se escoge porque es una forma fácil de entender que hacen las operaciones:

abrir, cerrar, crear, leer, escribir

```

$$Operaciones sobre archivos

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

$$Abre el archivo
unlock archivo

$$extrae el contenido del archivo
spider contenido = gather archivo

$$Escribe texto en el archivo
archivo forge "Hola mundo"
|
$$Crea un nuevo archivo
craft "Nuevo.txt"

$$Cierra el archivo
lock archivo

```

40. Operaciones de números flotantes: :+, :-, :*, :%, ://

Se escoge porque porque es la nomenclatura básica en Python, y se usa : para diferenciarlo de los enterops

```

$$Operaciones aritmeticas sobre flotantes

Ghast num1 = :4.0;
Ghast num2 = :3.0;
Ghast suma = num1 :+ num2;
Ghast resta = num1 :- num2;
Ghast multiplicacion = num1 :* num2;
Ghast residuo = num1 :% num2;
Ghast division = num1 :// num2;|

```

41. Operaciones de comparación solicitadas: <, >, <=, >=, is, isNot

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

```

$$operaciones de comparacion

$$el valor sera Encendido
torch mayorQue = 2>2

$$el valor sera Apagado
torch menorQue = 2<2

$$el valor sera Encendido
torch menorIgualQue = 2<=2

$$el valor sera Encendido
torch mayorIgualQue = 2>=2

$$el valor sera Encendido
torch IgualQue = 2 is 2

$$el valor sera Apagado
torch diferenteQue = 2 isnot 2

```

42. Manejo de Bloques de más de una instrucción: **PolloCrudo PolloAsado**

Se escoge porque al inicio está crudo y, al final del proceso, cocido o asado

```

Recipe          $$ sección de prototipos

Ritual ImprimaN(Stack :: N);

CraftingTable    $$ sección de rutinas

Ritual ImprimaN;  $$ como hay un prototipo los encabezados se pueden poner o no poner.
Inventory
Stack K = 0;
PolloCrudo
  repeater K<N Craft
  PolloCrudo
    dropperStack(K);
    soulSand K;
  PolloAsado;
PolloAsado;

```

43. Instrucción while: **repeater <cond> craft <instrucción>**

Se escoge porque un repetidor manda señales de redstone constantemente Ojo con el craft que es parte de la instrucción

```

Recipe          $$ sección de prototipos

Ritual ImprimaN(Stack :: N);

CraftingTable    $$ sección de rutinas

Ritual ImprimaN;  $$ como hay un prototipo los encabezados se pueden poner o no poner.
Inventory
Stack K = 0;
PolloCrudo
    repeater K<N Craft
        PolloCrudo
            dropperStack(K);
            soulSand K;
        PolloAsado;
PolloAsado;

```

44. Instrucción if-then-else: **target <cond> craft hit <inst> miss <inst>**

Se escoge porque simular a cómo el bloque objetivo evalúa si se dio en el centro o no. Recuerden que el hit y el miss pueden ir en orden inverso y ambos son opcionales, pero al menos debe haber uno.

```

$$Se utiliza while  break  if-then-else

stack k = 0;
torch hacer = on;

repeater hacer craft
    PolloCrudo
        target (contador isNot 1) craft
            hit
                PolloCrudo
                    contador -= 1
                PolloAsado;
            miss
                creeper;
    PolloAsado;

```

45. Instrucción switch: **jukebox <condition> craft , disc <case> : , silence**

Se escoge porque simular a la caja tocadiscos que tiene varias opciones para seleccionar canciones. Le agregué el craft y cambié el default por silence.

Recuerden que el silence puede ir en cualquier lugar y que es obligatorio usar el polloCrudo y polloAsado

```
$$Se utiliza for y continue y swith

walk stack i = 0 to i < 10; magma i
  PolloCrudo
  jukebox (i) craft
  PolloCrudo
  disc I = 5 :
    PolloCrudo
    creepper;
    PolloAsado;

    silence;
    PolloCrudo
    enderPearl
    PolloAsado;

  PolloAsado

PolloAsado
```

46. Instrucción Repeat-until: **spawner <instrucciones> exhausted <cond>;**

Se escoge porque similar al generador de monstruos que spawnea criaturas hasta que se cumpla una condición

\$\$Se utiliza Repeat-until

```
stack contador = 0;
torch hacer = on;
spawnner
|
|   PolloCrudo
|   |   target (contador isNot 1) craft
|   |   |   hit
|   |   |   |   PolloCrudo
|   |   |   |   |   contador -= 1
|   |   |   |   |   PolloAsado;
|   |   |   |   |   miss
|   |   |   |   |   |   hacer = off;
|   |   |   |   |   |
|   |   |   |   |   PolloAsado;
|   |   |   |   |   exhausted hacer;
```

47. Instrucción For: **walk VAR set <exp> to <exp> step <exp> craft <instrucción>**

Se escoge porque caminar es repetir un a cantidad de pasos

\$\$Se utiliza for y continue y swicth

```
walk stack i = 0 to i < 10; magma i
|
|   PolloCrudo
|   |   jukebox (i) craft
|   |   |   PolloCrudo
|   |   |   |   disc I = 5 :
|   |   |   |   |   PolloCrudo
|   |   |   |   |   |   creepper;
|   |   |   |   |   |   PolloAsado;
|   |   |   |   |   silence;
|   |   |   |   |   |   PolloCrudo
|   |   |   |   |   |   |   enderPearl
|   |   |   |   |   |   |   PolloAsado;
|   |   |   |   |   PolloAsado
|   |   PolloAsado
```

48. Instrucción With: **with** <Referencia a Record> **craft** <instrucción>

Se escoge porque enemigo que de casualidad tiene un nombre similar a with

49. Instrucción break: **creeper**

Se escoge porque puede explotar y salir de donde esta

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

stack k = 0;
torch hacer = on;

repeater hacer craft
| PolloCrudo
| | target (contador isNot 1) craft
| | | hit
| | | | PolloCrudo
| | | | | contador -= 1
| | | | PolloAsado;
| | | miss
| | | | creeper;
| PolloAsado;
```

50. Instrucción continue: **enderPearl**

Se escoge porque saltar el resto de una iteración es similar a teletransportarse a Ender Pearl


```
$$Se utiliza for y continue y swith

walk stack i = 0 to i < 10; magma i
  PolloCrudo
  jukebox (i) craft
    PolloCrudo
    disc I = 5 :
      PolloCrudo
      creepper;
      PolloAsado;

      silence;
      PolloCrudo
      enderPearl
      PolloAsado;

    PolloAsado
  PolloAsado
```

51. Instrucción Halt: **ragequit**

Se escoge porque ragequit transmite la idea de terminar repentinamente la ejecución, como abandonar el juego por enojo

\$\$Se utiliza for y continue y swith

```
walk stack i = 0 to i < 10; magma i
  PolloCrudo
    jukebox (i) craft
      PolloCrudo
        disc I = 5 :
          PolloCrudo
            ragequit;
          PolloAsado;
        silence;
      PolloCrudo
        enderPearl
        PolloAsado;
    PolloAsado
  PolloAsado
```

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

52. Encabezado de funciones: **Spell** <id>(<parameters>) -> <tipo>

```
Recipe          $$ sección de prototipos
spell ImprimaN(Stack :: N);
```

53. Encabezado de procedimientos: **Ritual** <id>(<parameters>)

Se escoge porque es un procedimiento que solo invoca

```
Recipe          $$ sección de prototipos
Ritual ImprimaN(Stack :: N);
```

Manejo de parámetros formales: (<type> :: <name>, <name>; <type> ref <name>; ...)

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

```
Recipe          $$ sección de prototipos
Ritual ImprimaN(Stack :: N);
```

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

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

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

55. Instrucción return: **respawn**

Se escoge porque como devolver cuando se termina

```
Recipe          $$ sección de prototipos  
  
spell ImprimaN(Stack :: N);  
  
CraftingTable   $$ sección de rutinas  
  
spell ImprimaN;  $$ como hay un prototipo los encabezados se pueden poner o no poner.  
Inventory  
Stack K = 0;  
PolloCrudo  
    repeater K<N Craft  
    PolloCrudo  
        dropperStack(K);  
        soulSand K;  
    PolloAsado;  
PolloAsado;  
respawn K
```

56. Operación de size of: **chunk <exp> o <tipo>**

Se escoge porque un chunk es como la medida para el mundo

```
$$ chunk (sizeof)  
stack tamanoChar = chunk (rune); //1 que se refiere un byte
```

57. Sistema de coerción de tipos: **<exp> >> <tipo>**

Se escoge porque es una forma intuitiva de hacerlo

```

Ritual mesesEnAño()
  PolloCrudo
    spider ano = hopperSpider("Cuantos años");
    stack meses = ano>>stack * 12
    dropperStack(meses)

  PolloAsado

```

58. Manejo de la entrada estándar: **x = hopper<TipoBásico>()**

Se escoge porque una Hopper recibe objetos

```

Ritual mesesEnAño()
  PolloCrudo
    spider ano = hopperSpider("Cuantos años");
    stack meses = ano>>stack * 12
    dropperStack(meses)

  PolloAsado

```

59. Manejo de la salida estándar: **dropper<tipoBásico>(dato)**

Se escoge porque un objeto que suelta cosas

```

Ritual mesesEnAño()
  PolloCrudo
    spider ano = hopperSpider("Cuantos años");
    stack meses = ano>>stack * 12
    dropperStack(meses)

  PolloAsado

```

60. Terminador o separador de instrucciones - Instrucción nula: ;

Se escoge porque es la nomenclatura común en los lenguajes

```

$$ "Primer programa" es el nombre del nuevo programa que se va a crear
WorldName Primer programa:

$$aquí se inicia el programa
SpawnPoint

    $* Aquí se llama función que ha sido creada
    recibe 2 números enteros *$
    $$también se usa el manejo de parámetros reales con 1,2
    sumar(1,2);

$$ Aquí se representa donde termina el programa
world save;

```

61. Todo programa se debe cerrar con un: **worldSave**

Se escoge porque cuando sales del mundo en minecraft lo guardas

```

$$ "Primer programa" es el nombre del nuevo programa que se va a crear
WorldName Primer programa:

$$aquí se inicia el programa
SpawnPoint

    $* Aquí se llama función que ha sido creada
    recibe 2 números enteros *$
    $$también se usa el manejo de parámetros reales con 1,2
    sumar(1,2);

$$ Aquí se representa donde termina el programa
world save;

```

62. Comentario de Bloque: **\$* comentario *\$**

Se escoge porque un es un símbolo fácil de escribir y diferenciar

```

$$ "Primer programa" es el nombre del nuevo programa que se va a crear
WorldName Primer programa:

$$aqui se inicia el programa
SpawnPoint

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

$$ Aqui se representa donde termina el programa
world save;|

```

63. Comentario de Línea: **\$\$ comentario**

Se escoge porque un es un símbolo fácil de escribir y diferenciar

```

$$ "Primer programa" es el nombre del nuevo programa que se va a crear
WorldName Primer programa:

$$aqui se inicia el programa
SpawnPoint

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

$$ Aqui se representa donde termina el programa
world save;|

```

64. 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;
```

65. 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
```

66. 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
```

67. 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

Listado de palabras reservadas

Elementos:

WorldName Bedrock ResourcePack Inventory Recipe Crafting
Table SpawnPoint Obsidian Anvil Stack Rune Spider
Torch Chest Book Ghast Shelf Entity soulsand magm
a isEngraved isInscribed etchUp etchDown bind from
except seek add drop items feed map biom kill
unlock lock craft gather forge expand PolloCrudo
PolloAsado repeater craft target hit miss jukebox di
sc silence spawner walk set wither creeper enderPe
arl ragequit Spell Ritual respawn chunk hopper drop
per worldSave magma soulsand

Operadores binarios:

= += *= -= %= /=
+ - * % //
%+ %- %* %/ %%
> < >= <= is isNot

Operadores ternarios

Comentarios

\$\$ \$* *\$

Algoritmos de conversión

Tipo Stack (int)

Spider -> Stack: El string tiene que contener solo caracteres que sean un dígito los crea en un número entero, y no es válido devuelve un error

Ej:

```
Ingrese un String: 12  
Dato como entero: 12
```

Rune -> Stack: devuelve el valor ascii del carácter ingresado

Ej:

```
Ingrese un booleano: r  
Dato como entero: 1
```

Torck -> Stack: Si el dato ingresado es 0, devuelve un cero, cualquier otro dato será un 1

Ej:

```
Ingrese un Char: a  
Dato como entero: 97
```

Gramática actualizada

<inicio> ::= WORLDNAME <identificador> <seccion>
 <punto_entrada> <final>

<seccion> ::= BEDROCK <sistema_asignacion_constante>

<seccion> ::= RESOURCEPACK <sistema_asignacion_tipos>

<seccion> ::= INVENTORY <sistema_asignacion_variables>

<seccion> ::= RECIPE <encabezado>

<seccion> ::= CRAFTINGTABLE <instrucciones>

<punto_entrada> ::= SPAWNPOINT <bloque>

<punto_entrada> ::= ϵ

<sistema_asignacion_constante> ::= OBSIDIAN <tipo> <identificador>
 <tipo>

<sistema_asignacion_tipos> ::= ANVIL <identificador> -> <tipo>

<sistema_asignacion_variables> ::= <tipo> <identificador> <asignacion>
 <literal>

<tipo> ::= STACK

<tipo> ::= RUNE

<tipo> ::= SPIDER

<tipo> ::= TORCH

<tipo> ::= CHEST

<tipo> ::= BOOK

<tipo> ::= GHAST

<tipo> ::= SHELF

<tipo> ::= ENTITY

<literal_booleano> ::= ON

<literal_booleano> ::= OFF

<literal_flotante> ::= LITERAL_FLOTANTE

<literal_entero> ::= LITERAL_ENTERO

<literal_caracter> ::= LITERAL_CARACTER

<literal_string> ::= LITERAL_STRING <acceso_string>

<literal_arreglo> ::= LITERAL_ARREGLO <acceso_arreglo>

<literal_registro> ::= LITERAL_REGISTRO <acceso_registro>

<acceso_arreglo> ::= <identificador> <indices>

<acceso_arreglo> ::=

<indices> ::= [<literal_entero>] <mas_indices>

<mas_indices> ::= <separador> [<literal_entero>]
<mas_indices>

<mas_indices> ::= ϵ

<acceso_string> ::= ϵ

<acceso_string> ::= <identificador> [<literal_entero>]

<acceso_registro> ::=

<acceso_registro> ::= <identificador> @ <identificador>

<asignacion> ::= =

<asignacion> ::= <operador> =

<operador> ::= +

<operador> ::= -

<operador> ::= *

<operador> ::= /

<operador> ::= %

<operacion_incre_decre> ::= SOULSAND

<operacion_incre_decre> ::= MAGMA

<operacion_caracter> ::= ISENGRAVED

<operacion_caracter> ::= ISINSCRIBED

<operacion_caracter> ::= ETCHUP

<operacion_caracter> ::= ETCHDOWN

<operacion_logica> ::= AND

<operacion_logica> ::= OR

<operacion_logica> ::= NOT

<operacion_logica> ::= XOR

<operacion_string> ::= BIND

<operacion_string> ::= FROM

<operacion_string> ::= EXCEPT

<operacion_string> ::= SEEK

<operacion_flotante> ::= % <operador>

<operacion_comparacion> ::= <

<operacion_comparacion> ::= >

<operacion_comparacion> ::= <=

<operacion_comparacion> ::= >=

<operacion_comparacion> ::= IS

<operacion_comparacion> ::= ISNOT

<bloque> ::= POLLOCRUDO <instrucciones> POLLOASADO
<instrucciones>

<ciclo> ::= REPEATER <condicion> CRAFT <instrucciones>

<if_then_else> ::= TARGET <condicion> CRAFT <instrucciones>
<if_then_else2>

<if_then_else2> ::= MISS <instrucciones>

<switch> ::= JUKEBOX <condicion> CRAFT POLLOCRUDO
<casos_switch> SILENCE : <caso> POLLOASADO

<casos_switch> ::= DISK <caso> :

<caso> ::= <instrucciones>

<caso> ::= <condicion>

<ciclo> ::= SPAWNER <instrucciones> EXHAUSTED <literal_entero>
TO <literal_entero> <ciclo_for_tipo>

<ciclo> ::= WALK <variable> SET <instrucciones>

<ciclo_for_tipo> ::= STEP <literal_entero> CRAFT

<ciclo_for_tipo> ::= CRAFT <instrucciones> <instrucciones>

<with> ::= WITHER <referencia_record> CRAFT

<break_continue_halt> ::= BREAK <terminador>

<break_continue_halt> ::= CONTINUE <terminador>

$\langle \text{break_continue_halt} \rangle ::= \text{HALT } \langle \text{terminador} \rangle \langle \text{parametros} \rangle \rightarrow \langle \text{tipo} \rangle$

$\langle \text{encabezado} \rangle ::= \text{SPELL } \langle \text{identificador} \rangle (\langle \text{parametros} \rangle)$

$\langle \text{encabezado} \rangle ::= \text{RITUAL } \langle \text{identificador} \rangle ($

$\langle \text{parametros_formales} \rangle ::= (\langle \text{lista_parametros_formales} \rangle)$

$\langle \text{parametros} \rangle ::= \langle \text{parametros_formales} \rangle$

$\langle \text{parametros} \rangle ::= \langle \text{parametros_reales} \rangle$

$\langle \text{lista_parametros_formales} \rangle ::= \langle \text{parametro_formal} \rangle \langle \text{lista_parametros_formales2} \rangle$

$\langle \text{lista_parametros_formales2} \rangle ::= \langle \text{separador} \rangle \langle \text{lista_parametros_formales} \rangle$

$\langle \text{lista_parametros_formales2} \rangle ::= \varepsilon$

$\langle \text{lista_parametros_formales} \rangle ::= \varepsilon$

$\langle \text{parametro_formal} \rangle ::= \langle \text{identificador} \rangle$

$\langle \text{parametro_formal} \rangle ::= \langle \text{tipo} \rangle \langle \text{parametro_formal_tipo} \rangle$

<parametro_formal_tipo> ::= <identificador>

<parametro_formal_tipo> ::= REF <identificador>

<parametros_reales> ::= (<lista_parametros_reales>)

<lista_parametros_reales> ::= <parametro_real> <mas_parametros_reales>

<lista_parametros_reales> ::= ϵ

<mas_parametros_reales> ::= <separador> <lista_parametros_reales>

<parametro_real> ::= <identificador>

<parametro_real> ::= <literal_booleano>

<parametro_real> ::= ϵ

<parametro_real> ::= <literal_flotante>

<parametro_real> ::= <literal_entero>

<parametro_real> ::= <literal_caracter>

<parametro_real> ::= <literal_string>

<parametro_real> ::= <literal_arreglo>

<parametro_real> ::= <literal_registro>

<return> ::= RESPAWN

<size_of> ::= CHUNK <size_of_tipo>

<size_of_tipo> ::= <tipo>

<size_of_tipo> ::= <expresion>

<size_of_tipo> ::= <variable>

<cohercion> ::= <expresion> >> <tipo>)

<entrada_estandar> ::= HOPPER <tipo> (<valor>)

<salida_estandar> ::= DROPPER <tipo> (

<terminador> ::= ;

<final> ::= WORLDSAVE

<identificador> ::= IDENTIFICADOR

<valor> ::= VALOR

<literal> ::= LITERAL

<separador> ::= ,

<instrucciones> ::= <instruccion><instruccion>

<instruccion> ::= <ciclo>

<instruccion> ::= <switch>

<instruccion> ::= <with>

<instruccion> ::= <break_continue_halt>

<instruccion> ::= <return>

<instruccion> ::= <entrada_estandar>

<instruccion> ::= <salida_estandar>

<instruccion> ::= <sistema_asignacion_constante>

<instruccion> ::= <sistema_asignacion_tipos>

<instruccion> ::= <sistema_asignacion_variables>

<instruccion> ::= <operacion_en_caracter>

<instruccion> ::= <operacion_en_string>

<instruccion> ::= <operacion_en_flotante>

<instruccion> ::= <if_then_else>

<instruccion> ::= <operacion_comparacion>

<instruccion> ::= <operacion_size_of>

<instruccion> ::= <cohercion>

<instruccion> ::= ϵ

<condicion> ::= CONDICION <mas_condicion>

<mas_condicion> ::= <operacion_logica> CONDICION <mas_condicion>

<mas_condicion> ::= ϵ

<expresion> ::= <termino> <operador> <termino>

<termino> ::= <tipo>

<variable> ::= VARIABLE <operacion>

<referencia_record> ::= REFERENCIA

<operacion> ::= <operacion_incre_decre>

<operacion_en_caracter> ::= <expresion> . <operacion_caracter>

<operacion_en_string> ::= <expresion> <operacion_string> <expresion>

<operacion_en_flotante> ::= <literal_flotante> <operacion_flotante>
<literal_flotante>

<operacion_size_of> ::= <size_of>

Tabla de precedencia

Esta es la tabla de precedencia, aquí se explica como orden que deben de seguir las palabras o símbolos que hay en el lenguaje sobre operaciones y demás, ayuda a entender como el compilador elije que orden ejecutarlas en caso de la presencia de mas de una en el mismo lugar

Nivel	Operadores	Descripción	Asociatividad
1	() .	paréntesis	Izquierda
2	+ - not	Signo (unario), negación lógica	Derecha
	[] @	Acceso a un elemento (unarios): puntero y dirección	Derecha
	magma souldan	Incremento y decremento	Derecha
	(cast) sizeof	Conversión de tipo (casting) y tamaño de un elemento	Derecha
3	* // %	Producto, división, módulo	Izquierda
4	+ -	Suma y resta	Izquierda
5	< <= >= >	Comparaciones de superioridad e inferioridad	Izquierda
6	is isNot	Comparaciones,de igualdad	Izquierda
7	and	And logico	Izquierda
8	or	Or logico	Izquierda
9	= += -= *= //= %=	Asignaciones	Derecha
10	,	Coma	Izquierda

Listado de errores y recuperación

-1. Carácter monstruo: Carácter no reconocido por el autómata.

Solución: Se reporta el error y se avanza al siguiente carácter.

-2. Comentario que nunca cierre: No se encuentra el símbolo de cierre (*\$).

Solución: Se sigue leyendo hasta el final del documento y se reporta un error si no se cierra.

-3. Número mal formado: Secuencia numérica que contiene caracteres inválidos (ejemplo 12a3).

Solución: Se consume hasta un separador válido (espacio, salto de línea) y se reporta el error.

-4. Token desconocido: Lexema que no corresponde a ningún estado de aceptación.

Solución: Se consume hasta el próximo separador y se reporta el error.

-5. Operador inválido: Secuencia de símbolos que no forma un operador permitido (ejemplo **, %%).

Solución: Se analiza solo el primer símbolo como operador válido y se reporta el error.

-6. Identificador inválido: Identificador que comienza con un número (ejemplo 4variable).

Solución: Se lee hasta un separador y se reporta el error.

-7. Asignación incorrecta: Uso incorrecto del signo igual, como =; o =-.

Solución: Se reporta un mensaje de error si no se sigue la secuencia del automata

-8. Terminador mal formado: Se esperaba un ; o : y aparece otro carácter.

Solución: Se reporta el error y se continúa leyendo los siguientes lexemas.

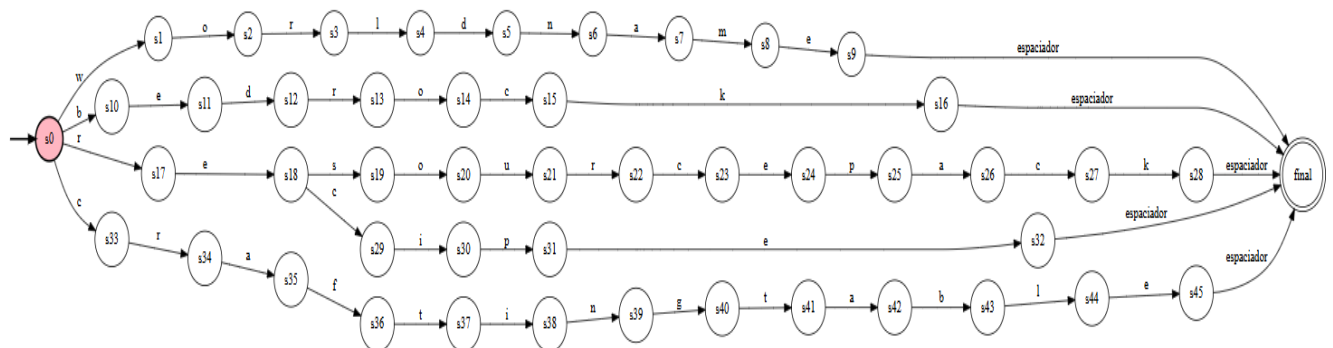
Autómatas

El autómata del proyecto es demasiado grande por lo que hemos tomado la decisión de segmentarlo por parte. Nuestro estado inicial va a ser s0, de ahí se va expandiendo a diferentes estados. **Una nota importante** es que más adelante se explicara los errores o identificadores ya que si se explica en el mismo autómata podría ser confuso. Por cada imagen de autómatas se explicará que se hizo, secciones utilizadas y las palabras reservadas representadas. **Otra segunda nota** es que no se está tomando en esta segmentación el orden de la numeración de los estados para poder representar cada sección o grupo de tokens, sin embargo en este orden el autómata irá creciendo

Estructura del título del programa y secciones

Palabras:

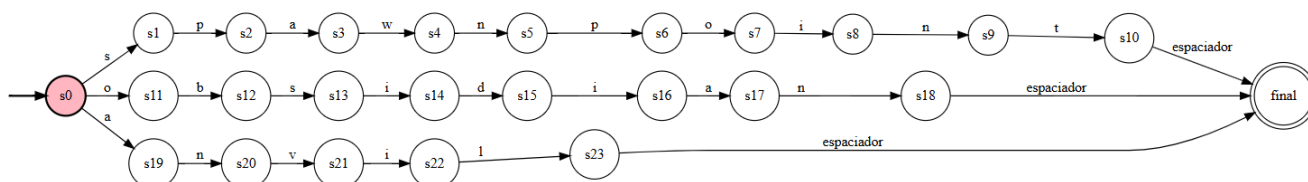
- worldname
- bedrock
- resourcepack
- récipe
- craftingtable



Puntos de entrada del sistema y sistemas de asignación

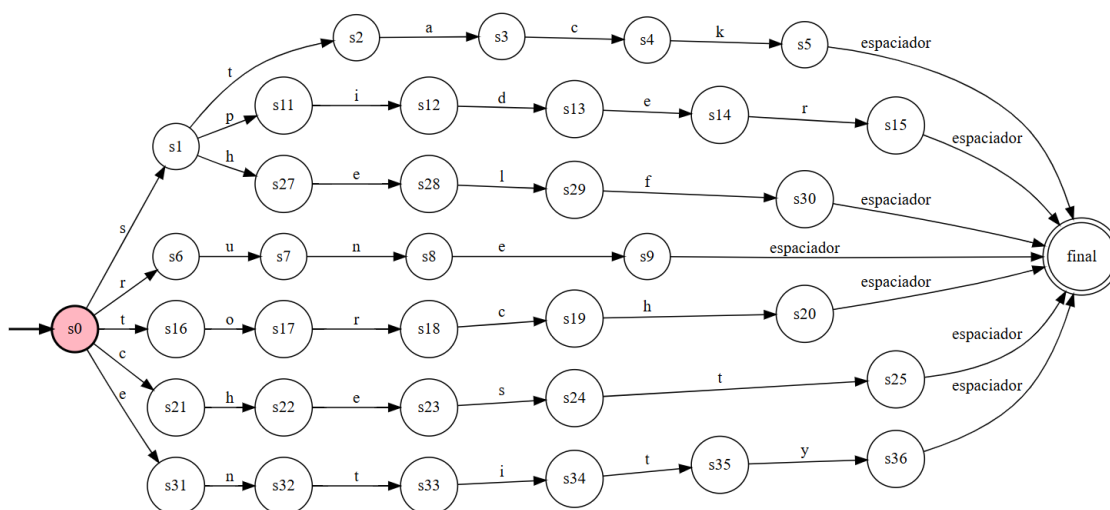
Palabras:

- spawnpoint
- obsidian
- anvil



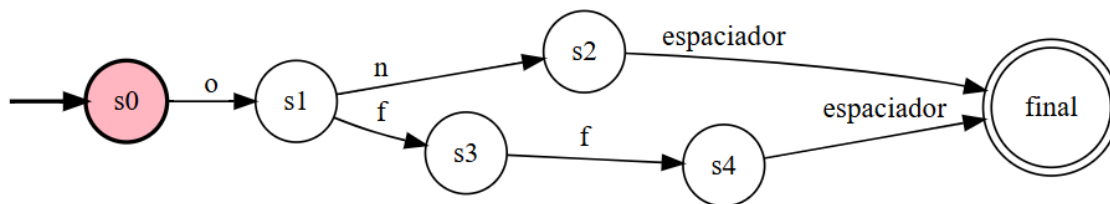
Tipos

- palabras:
- stack
- rune
- spider
- torch
- chest
- shelf
- entity

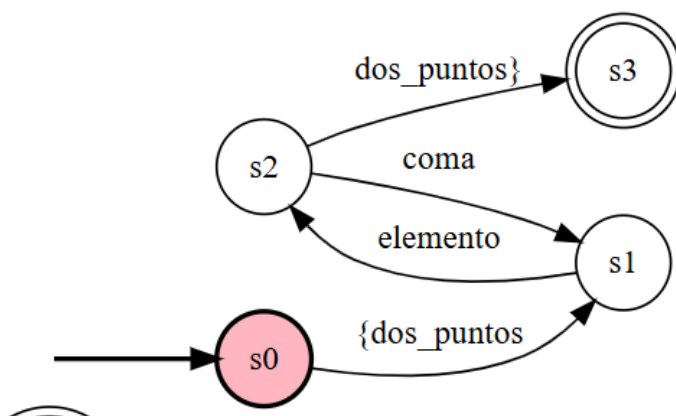


Literales

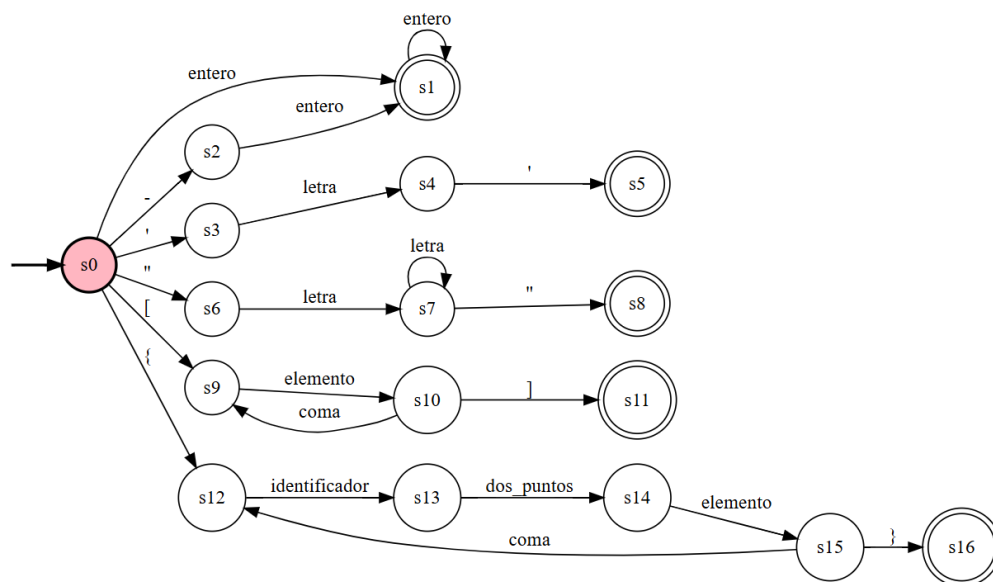
Literal booleana



Literal de conjunto



Literal de enteros, char, string, arreglos y registros

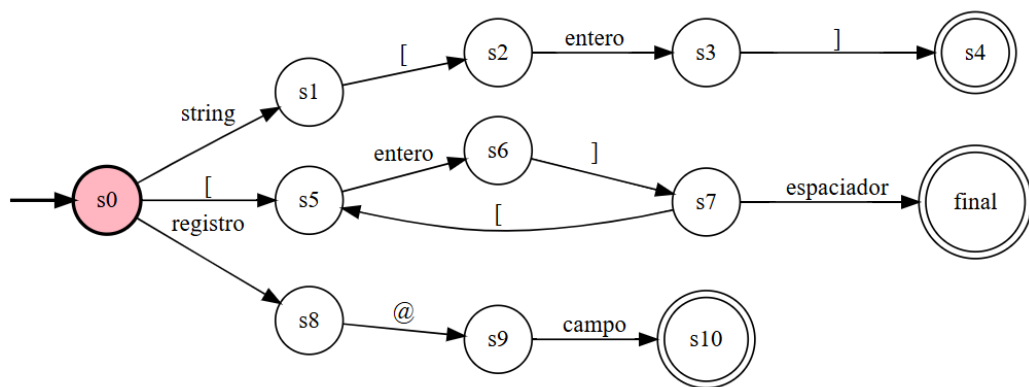


Sistema de acceso

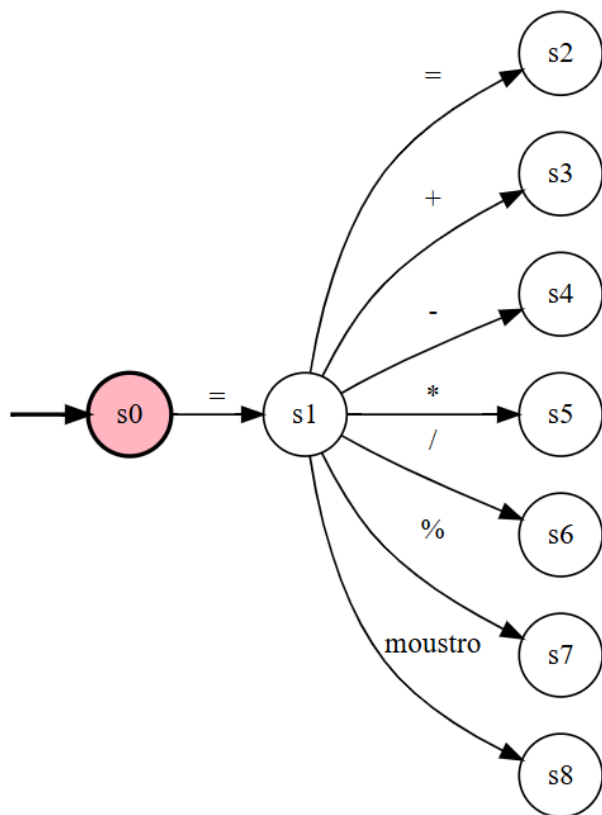
Sistema de acceso de arreglos

Sistema de acceso de string

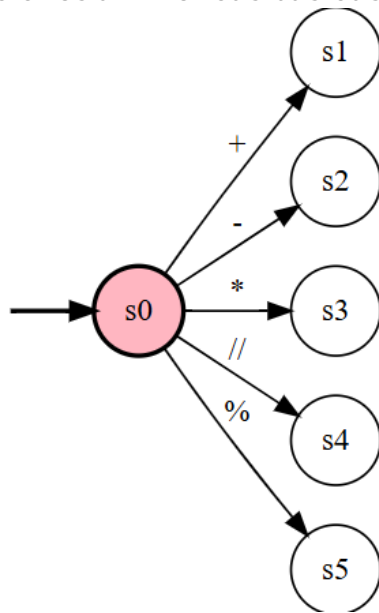
Sistema de acceso de registros



Asignación y familia

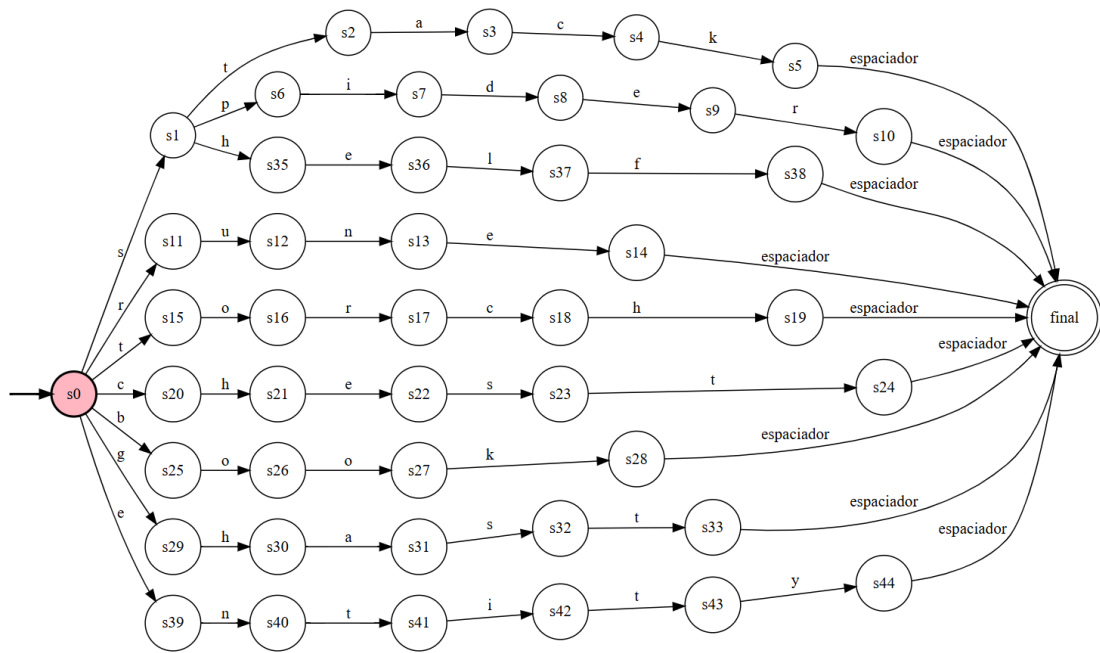


Operaciones aritméticas básicas de enteros



Tipos

- Stack
- Rune
- Spider
- Torch
- Chest
- Book
- Ghast
- Shelf
- Entity



Instrucciones del analizador sintáctico

Para poder correr el archivo llamado **parcer.py**, cambiar la ruta en `inicializar_parcer` por la ruta del archivo código fuente que se queira probar. En este caso ya hay un archivo de prueba llamado **prueba.ne** por lo que nada mas debe de copiar la ruta de ese archivo y correr el archivo **parcer.py**

```
# Ejemplo de uso
if __name__ == "__main__":
    parser = Parser()
    if parser.inicializar_parser("/prueba.ne"):
        parser.parse()
        parser.finalizar_parser()
```

Este código se encuentra al final del archivo