Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI3641 – Lenguajes de Programación 1

Trimestre: Septiembre - Diciembre 2023

Profesor: Ricardo Monascal

Estudiante: Junior Miguel Lara Torres, Carnet: 17-10303

Parcial 1 (25 pts)

- "En algunas preguntas, se usarán las constantes X, Y y Z. Estas constantes debe obtenerlas de los últimos tres números de su carnet."

Caso particular, 17-10303 entonces $X = 3,\ \ Y = 0,\ \ Z = 3$.

- "En aquellas preguntas donde se le pida implementar un programa, mantenga su código en un repositorio git remoto (preferiblemente Github) y coloque un enlace al mismo en lugar de su respuesta. Todo su código debe ser legible y estar debidamente documentado."

Todos los códigos, este propio documento, documento oficial del enunciado de examen 1 serán cargados en:

https://github.com/JMLTUnderCode/Programming/tree/main/USB_Language_Programation

- **1er Pregunta:**

El lenguaje escogido basado por mi nombre “Junior” es JavaScript.

(a) De una breve descripción del lenguaje escogido.

i. Diga qué tipo de alcances y asociaciones posee, argumentando las ventajas y desventajas de la decisión tomada por los diseñadores del lenguaje, en el contexto de sus usuarios objetivos.

```
function ka(a) {
    var k = 2;

    function ro() {
        // ...
    }

    var to = 3;
}

ro();
console.log(k, a, to);
```

Para el caso de JavaScript (JS) se tiene un alcance estático, conocido como léxico y en inglés “Lexical Scope”, además de esto JS es un lenguaje de programación de scripts orientado a objetos lo que da pie a entender que todo el JS es un objeto, en especial las funciones son de “primera clase”, pues pueden ser asignadas a variables, pasadas como argumentos a otras funciones, etc. Por otra parte, teniendo en cuenta la definición de asociación profunda la cual crea clausuras, en este caso, cuando invocas o pasas como argumento a la función la cual utiliza el entorno en el momento en que se creó, podemos decir entonces que, como JS tiene alcance estático, se resuelven las variables en el ámbito/entorno donde se definen, eso quiere decir que JS posee asociación profunda. En el pequeño ejemplo de código vemos que al llamar la función “ro” o bien sea mostrar por pantalla k, a, o to arroja “ReferenceError” debido a las razones explicadas. Otro dato importante de JS es que es un lenguaje interpretado que se interpreta línea por línea en tiempo de ejecución.

Ya entendiendo que al tener alcance estático más una asociación profunda como ventaja general es que para el público WebDeveloper se tiene una transparencia en los entornos de programación dado la naturaleza de los conceptos, ahora como desventaja claro está la falta de dinamismo en casos particulares, sin embargo, a nivel de dinamismo tenemos que al ser un lenguaje interpretado se tiene mayor flexibilidad, pero como consecuencia se pierde rapidez.

ii. Diga qué tipo de módulos ofrece (de tenerlos) y las diferentes formas de importar y exportar nombres.

Los módulos en JS primeramente poseen extensión “.js”, estos permiten almacenar código reutilizable al cual es importado al archivo principal del programa, es decir es la forma de dividir código en varios archivos para luego ser reutilizado.

MyModule.js

```
export const name = "Junior";
export function profe() {
  console.log("Sia Sia");
}
```

La característica principal de este archivo modulo es que todo aquello que se vaya a “exportar” del archivo modulo, para tu archivo principal digamos, debe llevar usar la keyword reservada “export” antes como se muestra en el ejemplo. Puedes exportar funciones, var, let, const y clases. No se puede usar export dentro de una función ya que deben ser elementos de nivel superior.

MyModule.js

```
const name = "Junior";
function profe() {
  console.log("Sia Sia");
}
export{name, profe};
```

Una forma más conveniente de exportar es usar una sola declaración de exportación al final del módulo, con una sintaxis de lista con los elementos a exportar separada por comas entre llaves.

Ya luego para importar esas constantes, clases, funciones, y otros objetos definidos, en tu script se debe usar la keyword reservada “import” seguida de una lista separada por comas de las características a importar entre llaves, seguida de la keyword “from” y a continuación la ruta al archivo del módulo.

main.js

```
import {name, profe} from "./modules/MyModule.js";
// More code ...
```

Por otro lado, cuando se tiene conflicto de nombres en las funciones entonces podemos usar la keyword reservada “as” para tener la libertad de cambiar el nombre de la constante, función, etc. Tenemos dos formas (imágenes).

MyModule.js

```
const name = "Junior";
function profe(){
  console.log("Sia Sia");
}
export {name, profe};
```

main.js

```
import {
  name  as MiNombre,
  profe as profeBeLike
} from "./modules/MyModule.js"
// more code ...
```

MyModule.js

```
const name = "Junior";
function profe(){
  console.log("Sia Sia");
}
export {name as MyN, profe as Sia};
```

main.js

```
import {MyN, Sia} from "./MyModule.js"
// more code ...
```

Podemos encontrar la característica de crear un objeto de tipo modulo, es decir una forma que facilite el uso de las n funciones u objetos definidas dentro del modulo es permitiendo crear con el módulo un propio objeto con el símbolo "*" indicando que todas las características exportables serán vinculadas a "LP1" para luego mediante el operador punto (.) ser usadas.

```
main.js
import * as LP1 from "./MyModule.js";
const n = LP1.name;
LP1.profe();
// more code ...
```

iii. Diga si el lenguaje ofrece la posibilidad de crear aliases, sobrecarga y polimorfismo. En caso afirmativo, dé algunos ejemplos.

En el ámbito de crear alias al tener un objeto como se muestra en la imagen con nombre y edad, si se quiere desestructurar el objeto para obtener las propiedades nombre y edad podemos hacer "const {name, age} = infoUser;" teniendo en cuenta que los nombres deben coincidir con los del objeto.

```
main.js
const infoUser = {
  name: "Junior Lara",
  age: 24
};
const {name, age} = infoUser;
console.log(name); // Junior Lara
console.log(age);  // 24
```

```
main.js
const infoUser = {
  name: "Junior Lara",
  age: 24
};
const {name: myN, age: myA} = infoUser;
console.log(myN); // Junior Lara
console.log(myA); // 24
```

Ahora cuando se quiera cambiar el alias de las propiedades de ese objeto para comodidad nuestro tenemos el método siguiente, el cual usa el operador dos puntos ":" seguido del nombre a elección. En el ejemplo se cambia nombre por myN y edad por myA.

Para el caso de las sobrecargas en JS se tiene una peculiaridad y no es como en otros lenguajes que, al crear dos funciones del mismo nombre, pero con diferentes argumentos entonces el lenguaje identifica al momento de llamar cuál de las instancias es la correcta, en JS se puede invocar una función con cualquier combinación de argumentos. Es decir, puedo declararla con un X parámetros, pero al momento de invocarla puedo usar una combinación no necesariamente X, puede ser mayor o menor que X. En caso de no coincidir los parámetros con los argumentos utilizados, no es considerado ningún error en el lenguaje, en este caso el intérprete realizara lo necesario/posible para adaptarse a como fue invocada. Cuando esto pasa se tienen dos casos:

Si faltan parámetros, su valor será "undefined". Como se muestra en la imagen podemos llamar a la función de dos formas y según sea el caso, el argumento “klk” toma el valor “undefined” haciendo que se cumple la regla “undefined || {cosa} = {cosa}”.

main.js

```
function ayChamo(klk){
  var seme = klk || "elarroz";
  console.log(seme);
}
ayChamo("quemo"); // quemo
ayChamo();        // elarroz
```

Se puede acceder a los parámetros restantes a través de "arguments" en caso de sobrar, el cual es un objeto que está disponible dentro de la función y contiene todos esos parámetros que se le han pasado a la función cuando fue invocada. El programa de la imagen muestra los elementos almacenados en objeto y la cantidad.

main.js

```
function kakaroto(){
  console.log(arguments);
  console.log(arguments.length);
}
kakaroto();
kakaroto(69);
kakaroto("kokunnn", "ball", 12+1);
```

Ahora el hablar de sobrecarga para operadores tenemos el operador “+” que tiene 3 funciones diferentes, para el caso de sumar números, para el caso de determinar el signo de un número, para el caso de concatenar strings. Además de eso el operador “+” consta con la propiedad de convertir tipos, imagen de ejemplo. Y otro operador es el “*” usado para multiplicar números como en todo lenguaje, sin embargo, también es usado al momento de importar convertir un archivo module.js en un objeto.

main.js

```
console.log(10+10); // 20
console.log(+7);    // 7
console.log("Hi"+"world") // "Hi world"

// Conversor de tipos
console.log("1" + "3"); // "12+1" >:)
console.log("1" + 3);   // "12+1" >:)
console.log(+"1" + 3);  // 12+1   >:)
```

Para el polimorfismo tenemos que una sintaxis especial mediante la keyword reservada “extends” para las clases en JS, creando así bajo el mismo nombre de la clase padre otros submetodos.

main.js

```
class A {
    area(x, y) {
        console.log(x * y);
    }
}
class B extends A {
    area(a, b) {
        super.area(a, b);
        console.log('Nais');
    }
}
let ob = new B();
let output = ob.area(10, 30);
// 300
// Nais
```

iv. Diga qué herramientas ofrece a potenciales desarrolladores, como: compiladores, intérpretes, debuggers, profilers, frameworks, etc.

Como JS es un lenguaje interpretado es razonable pensar que no posee compiladores, sin embargo, al realizar investigaciones podemos encontrar "compiladores" que realmente son "transpiler" otro tipo de compilador, tenemos TypeScript, Babel, Flow, CoffeScript Compiler, Reason, ClojureScript, PureScript, Sucrase, Shadow-cljs y SWC.

Para los intérpretes tenemos a SpiderMonkey el primer interprete creado por Brendan Eich, todos los navegadores basados en JS funcionan como intérpretes, y uno conocido actualmente como la base es el interprete V8, lo tiene la mayoría de los navegadores, Node.js, Deno y V8.NET, también JavaScriptCore, JScript.NET, Tamarin y Nashorn.

En la sección de debuggers tenemos a uun top 7 mas conocidos en la actualidad JS Bin, Airbrake, Firefox JavaScript Debugger, ESlint, Raygun, Rollbar, GlitchTip, entre muchos otros claro.

Para los profilers tenemos al principal JS-Profiler con pagina web https://js-profiler.com/ para checar y verificar el rendimiento de las rutinas mas usadas en los scripts.

Entre los frameworks tenemos a Vue.js, React, AngularJS, Node.js, jQuery, Meteor, Ember.js, Next.js, Polymer, Svelte, Angular, Express.js, JavaScript Library, Knockout y Socket.IO.

(b) Implemente los siguientes programas en el lenguaje escogido:

i. Dada una cadena de caracteres w y un entero no–negativo k, calcular la rotación de k posiciones de la cadena w. Utilice la siguiente fórmula como referencia:

$$rotar(w,k) = \left\{ \begin{array}{ll} w & \text{si } k = 0 \vee |w| = 0 \\ rotar(x +\!\!+ [a], k-1) & \text{si } k > 0 \wedge w = ax \wedge a \text{es un caracter} \end{array} \right\}$$

En la ffórmula anterior, el operador ++ corresponde a la concatenación de cadena de caracteres.

ii. Dada una matriz cuadrada A (cuya dimensión es $N \times N$), calcular el producto $A \times A^T$ (donde A^T es la transpuesta de A).
Recordemos que la multiplicación de dos matrices cuadradas, A y B (de tamaño $N \times N$), viene dada por la siguiente formula:

$$\forall i \in [1..N], j \in [1..N] : (A \times B)_{i,j} = \sum_{k \in [1..N]} A_{i,k} \times B_{k,j}$$

A su vez, la transpuesta de una matriz cuadrada A (de tamaño $N \times N$), es aquella que tiene los valores intercambiados respecto a sus coordenadas. Esto es: $\forall i \in [1..N], j \in [1..N] : A^T_{i,j} = A_{j,i}$

<u>Ambos códigos cargados al Git descrito en pagina 1,</u>

- **2er Pregunta:**

  Considere el siguiente programa escrito en pseudo-código.

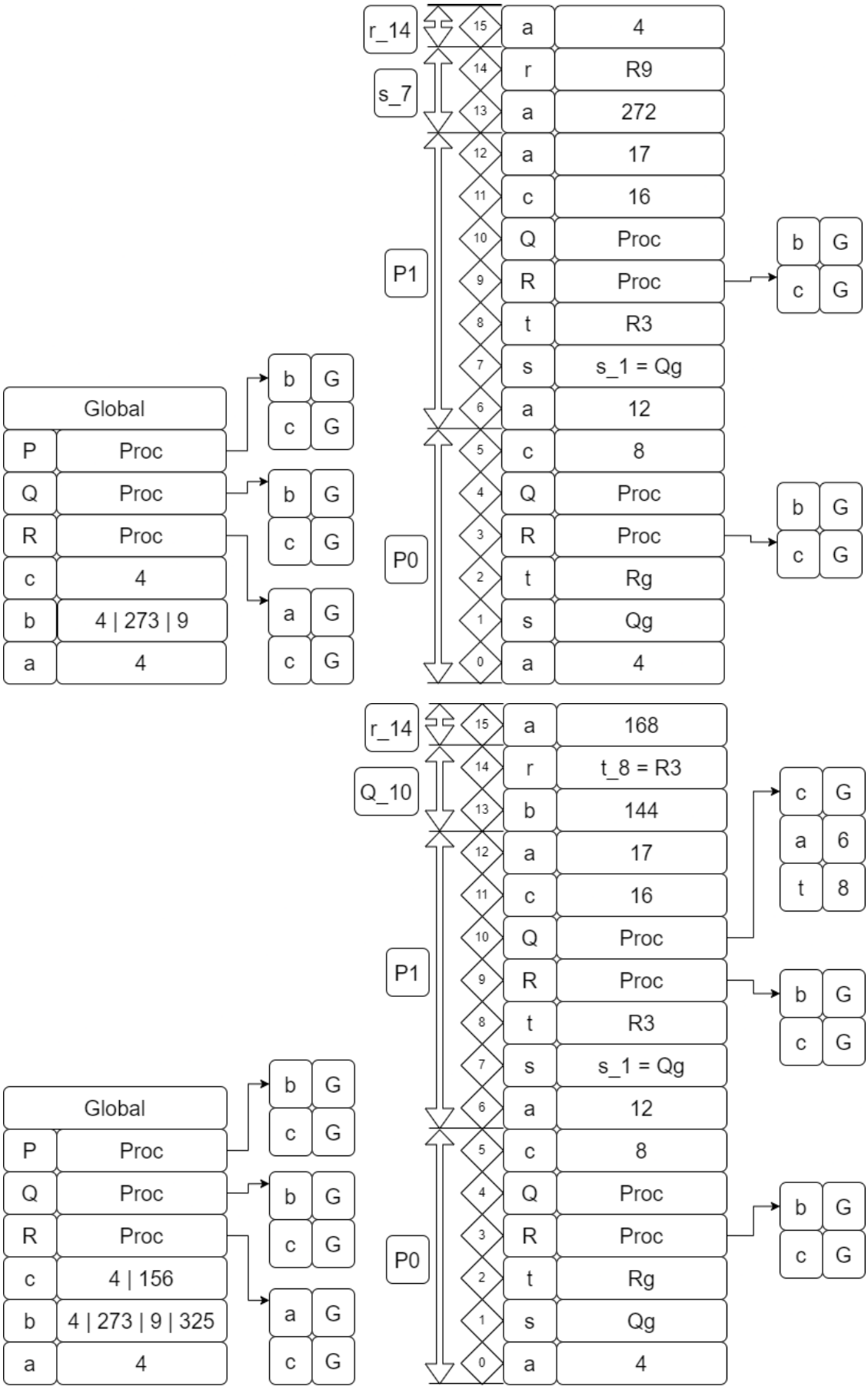
- Note que deberá reemplazar los valores para X, Y y Z como fue explicado en los párrafos de introducción del examen.
  $X = 3,\ Y = 0,\ Z = 3$

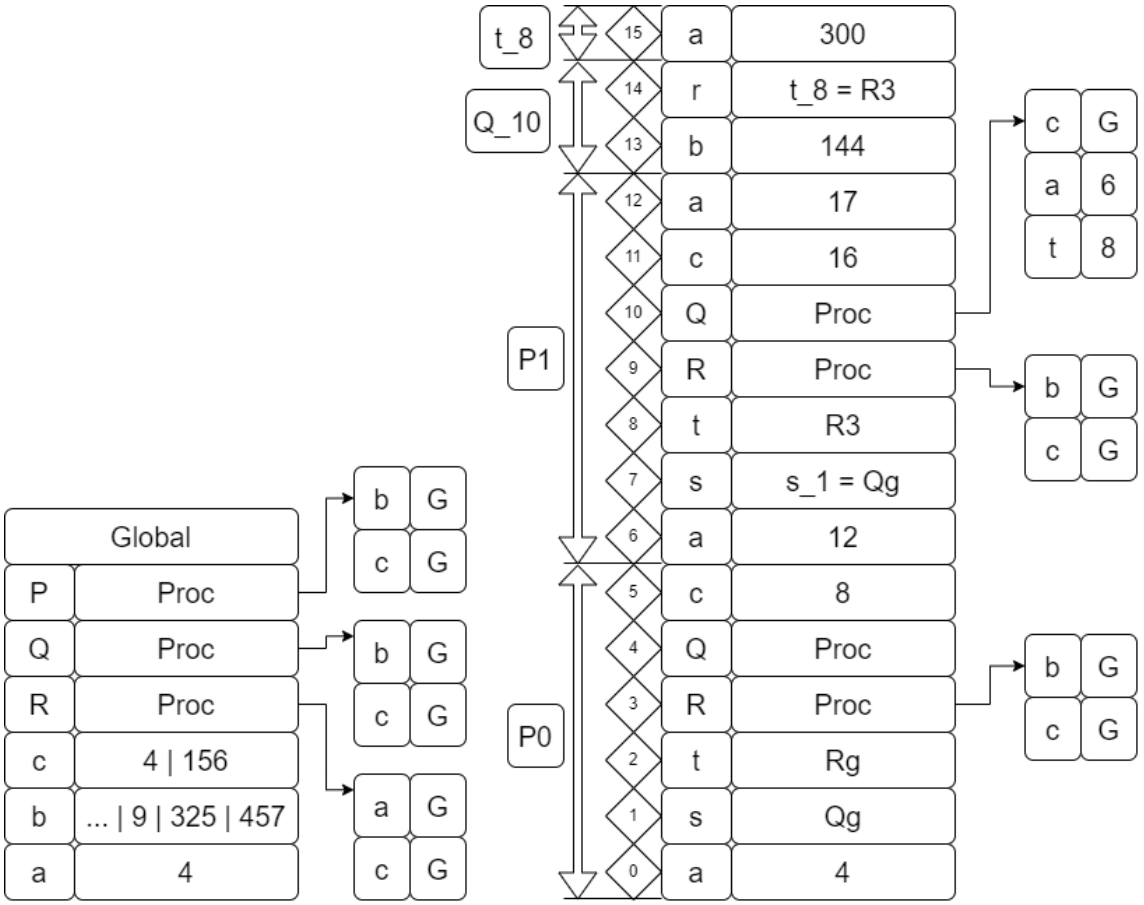
- Recuerde mostrar los pasos de su ejecución (por lo menos al nivel de cada nuevo marco de pila creado).

  Diga qué imprime el programa en cuestión;

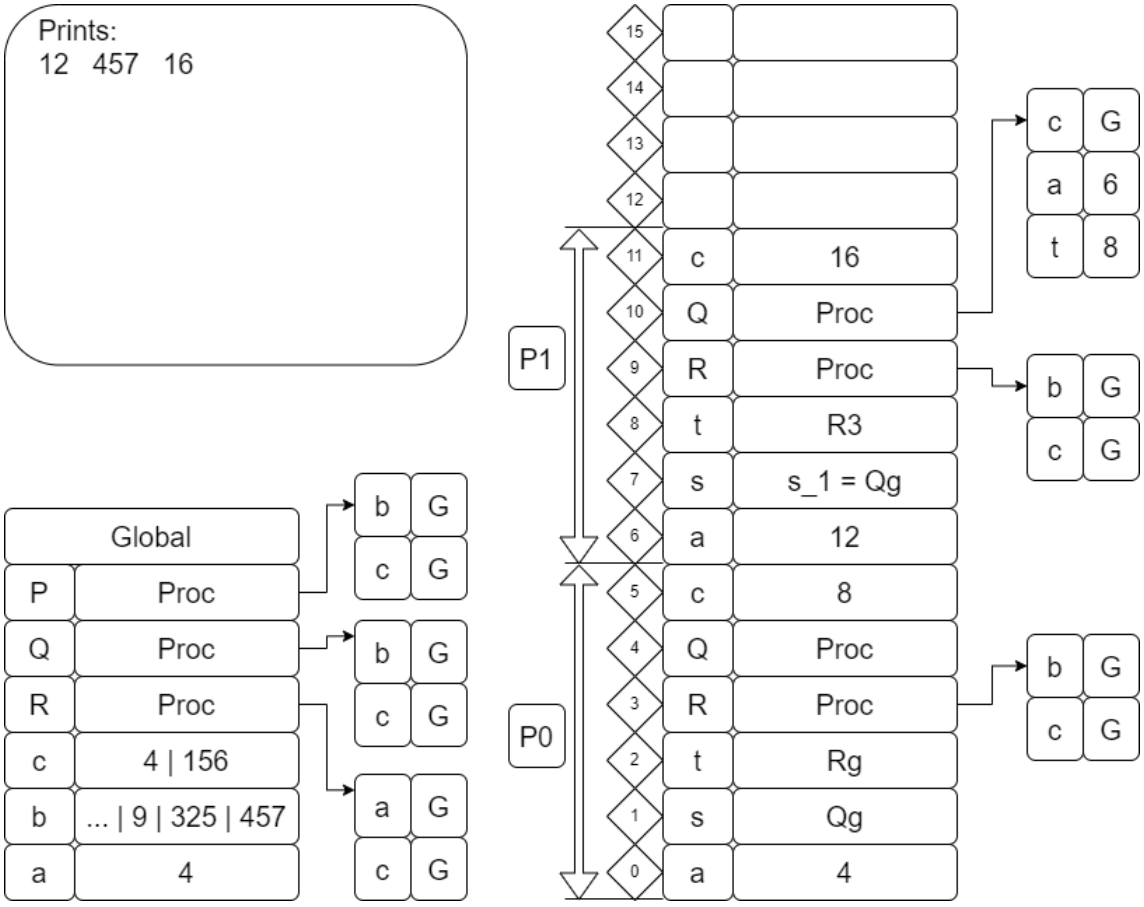
```
int a = 4, b = 4, c = 4;

sub R (int b) {
    a := b + c - 1
}

sub Q (int a, sub r) {
    b := a + 1
    r(c)
}
sub P(int a, sub s, sub t) {

    sub R(int a) {
        b := c + a + 1
    }

    sub Q (int b, sub r) {
        c := a + b
        r(c + a)
        t(c + b)
    }

    int c := a + b

    if (a < 8) {
        P(a + 8, s, R)
    } else {
        int a := c + 1
        s(c * a, R)
        Q(c * b, t)
    }

    print(a, b, c)
}
P(a, Q, R);
print(a, b, c)
```

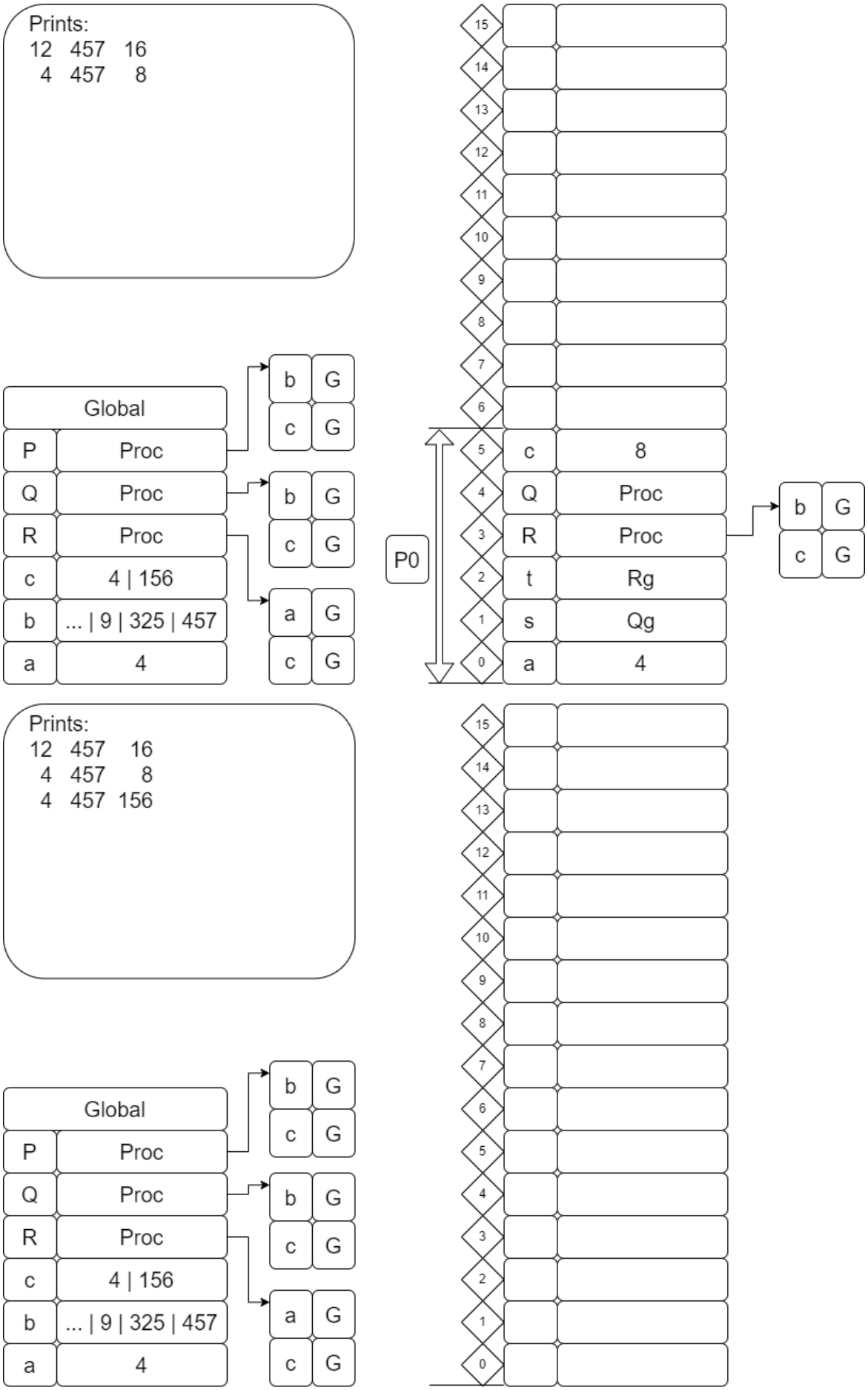
(a) Alcance estático y asociación profunda



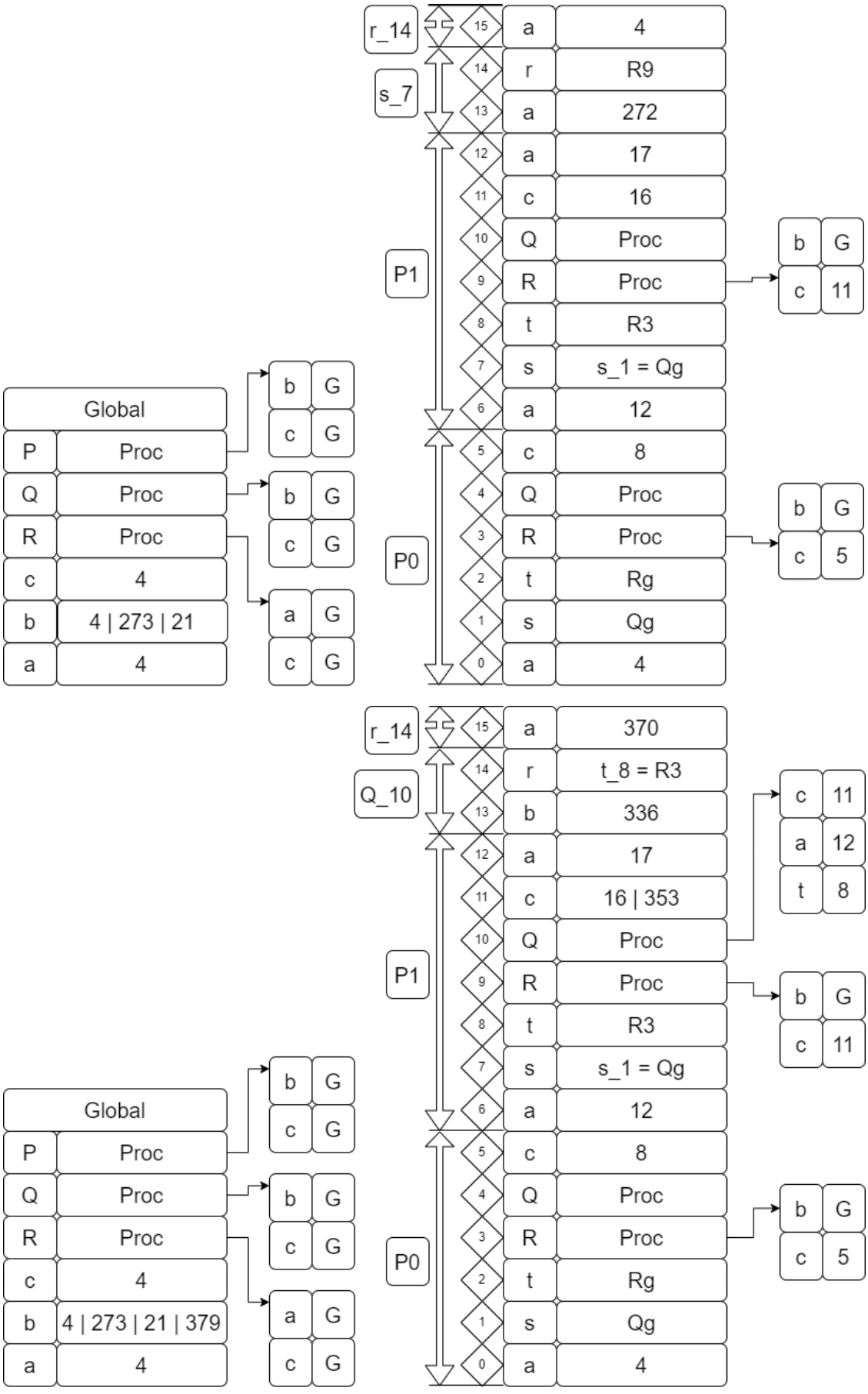


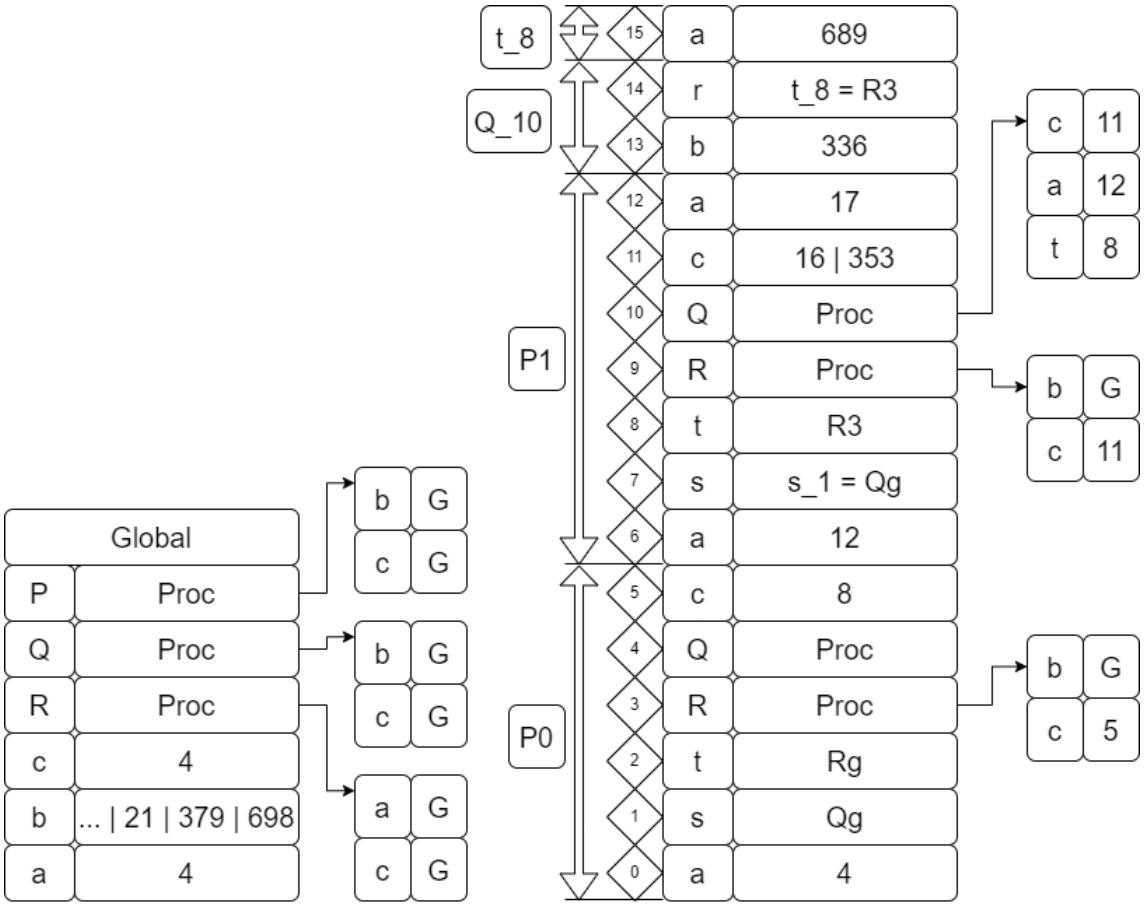
Prints:
12 457 16



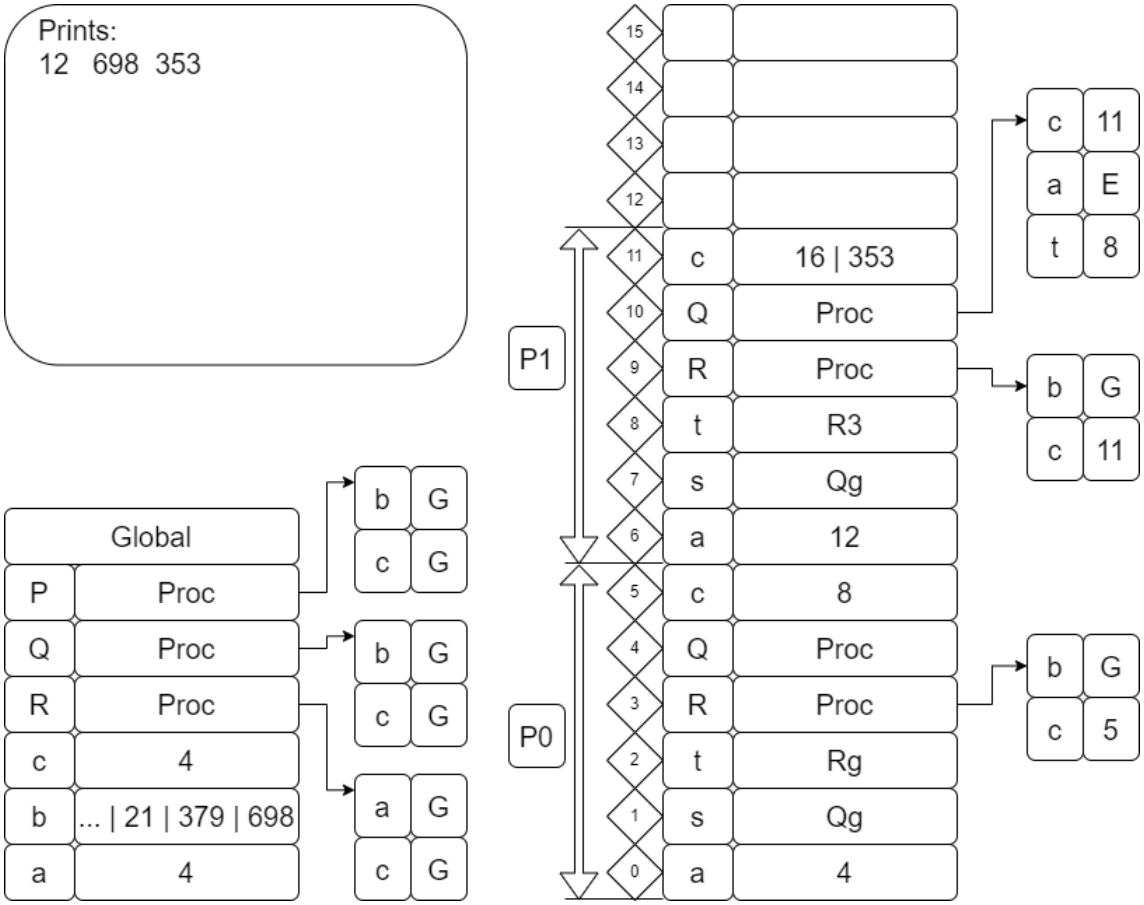

Prints:
12 457 16
4 457 8
Global
P Proc
Q Proc
R Proc
c 4 | 156
b ... | 9 | 325 | 457
a 4
b G
c G
b G
c G
a G
c G
15
14
13
12
11
10
9
8
7
6
5 c 8
4 Q Proc
3 R Proc
2 t Rg
1 s Qg
0 a 4
P0
b G
c G
Prints:
12 457 16
4 457 8
4 457 156
Global
P Proc
Q Proc
R Proc
c 4 | 156
b ... | 9 | 325 | 457
a 4
b G
c G
b G
c G
a G
c G
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

(b) Alcance dinámico y asociación profunda

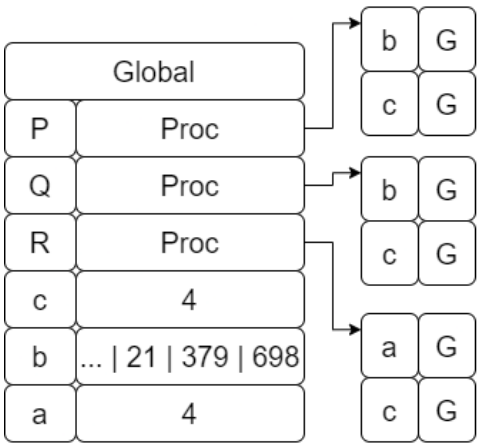


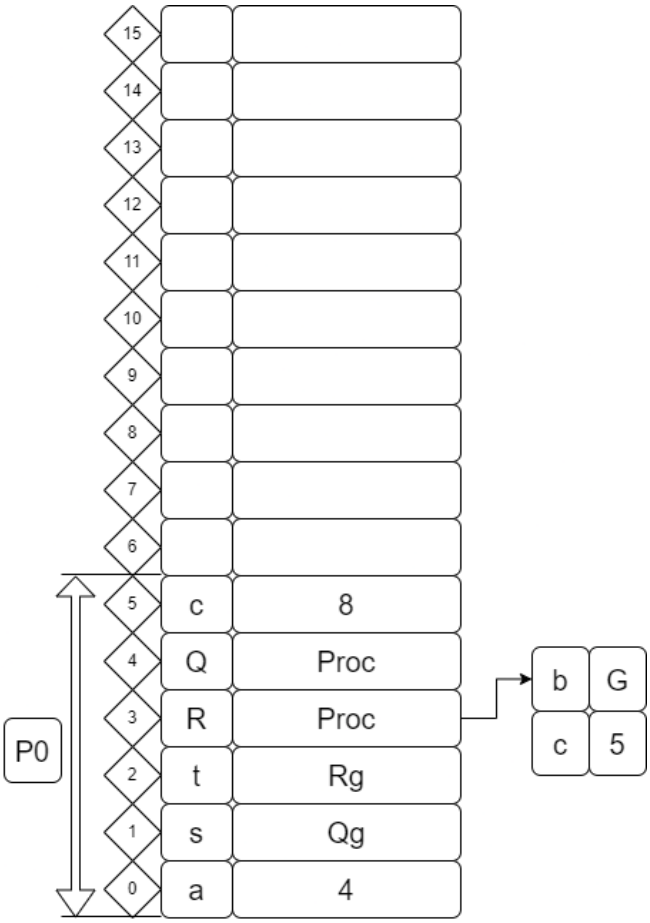


Prints:
12 698 353

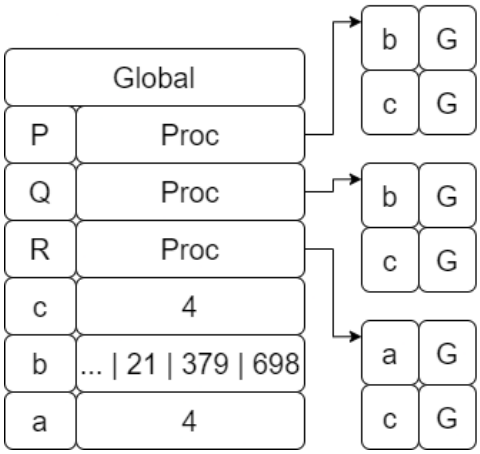


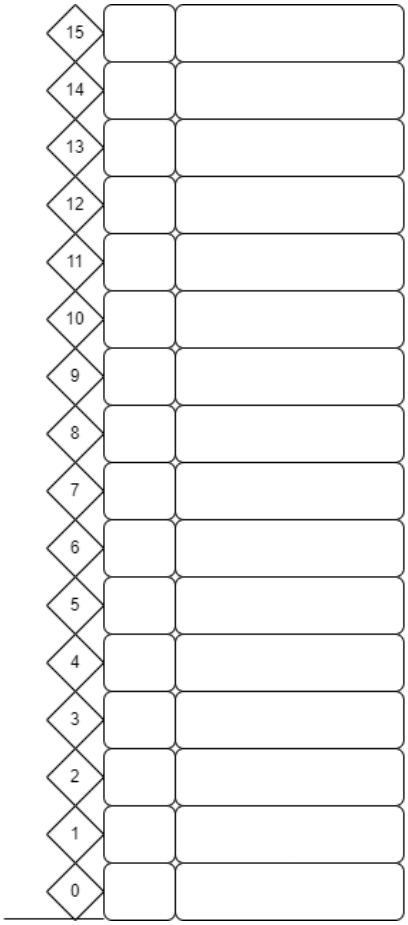
Prints:
12 698 353
4 698 8



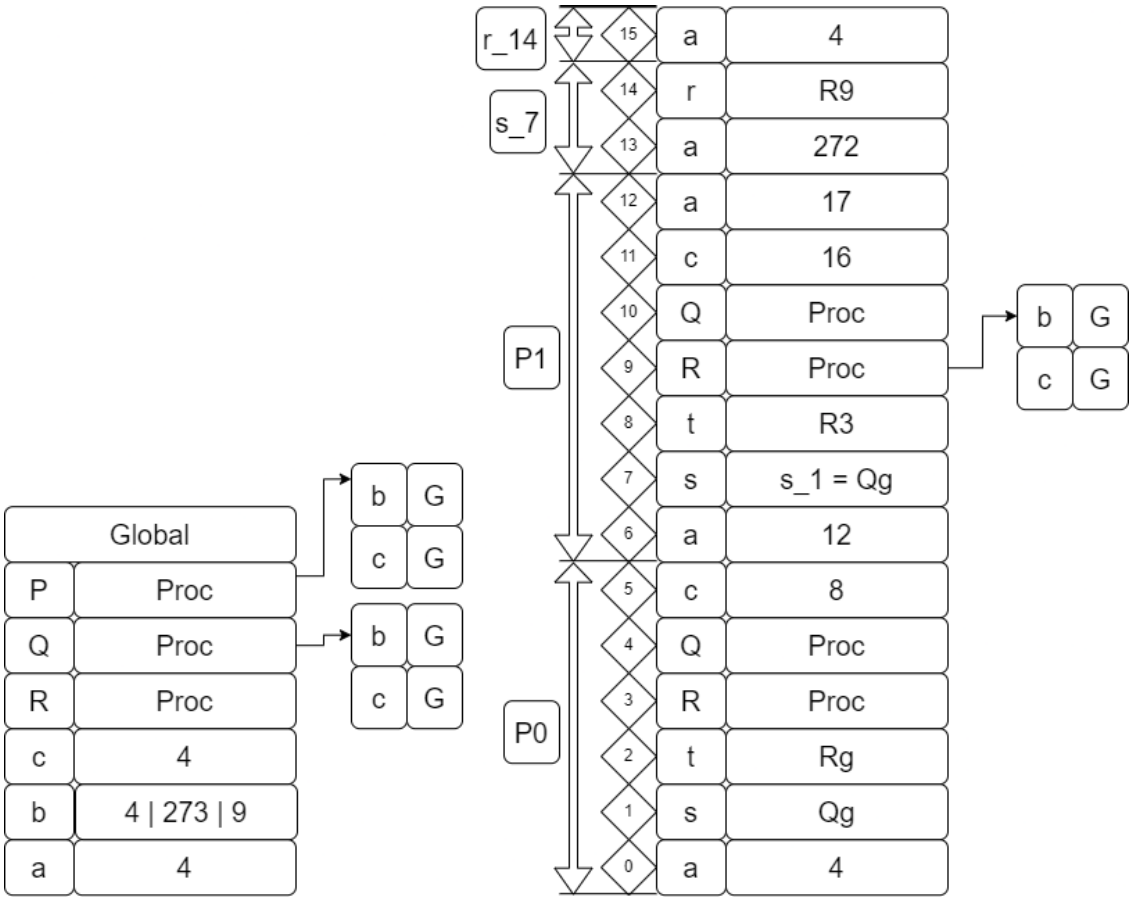


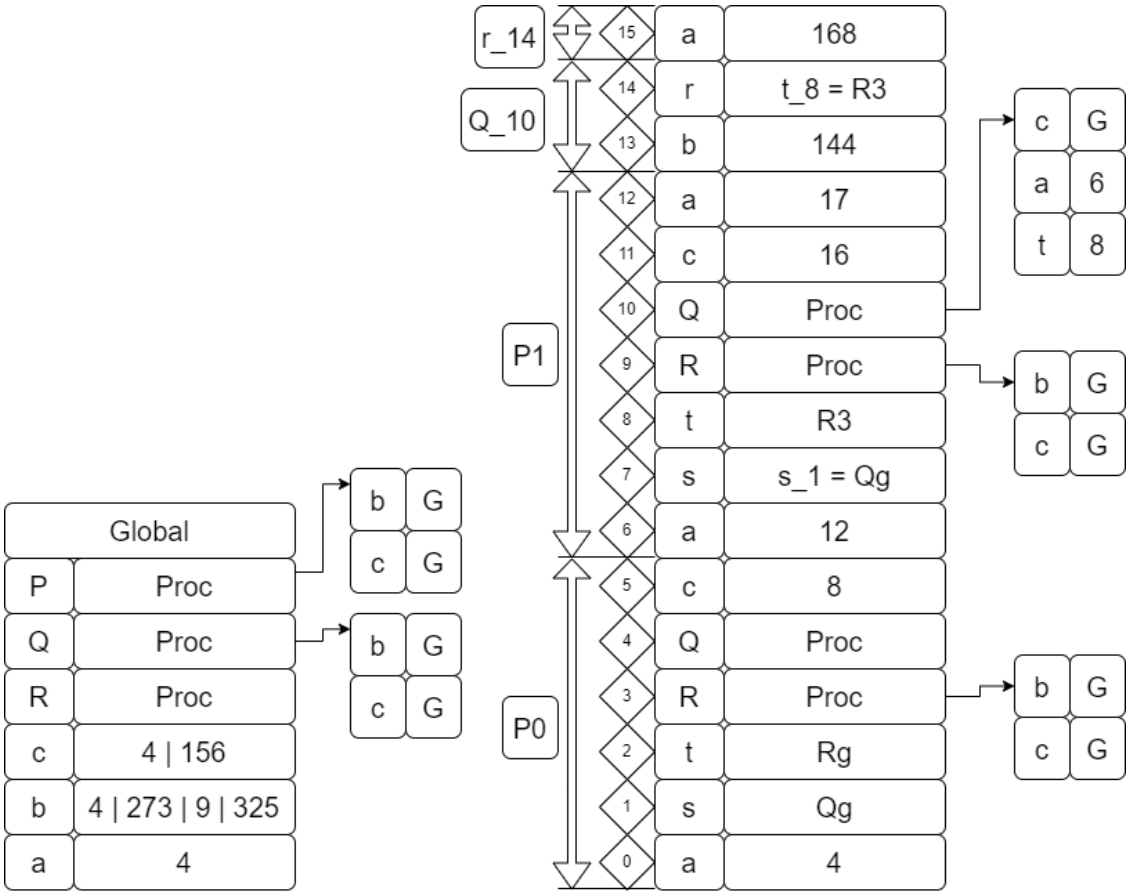
Prints:
12 698 353
4 698 8
4 698 4

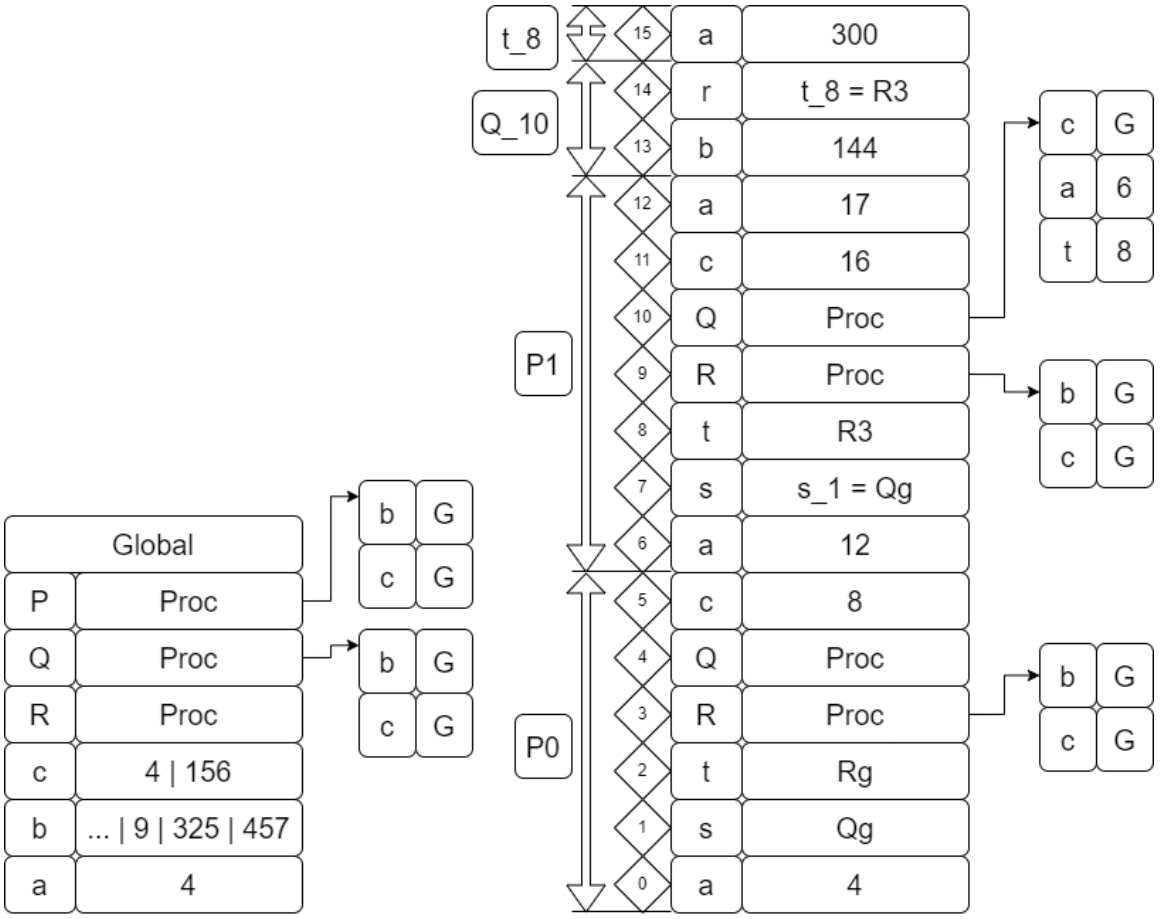




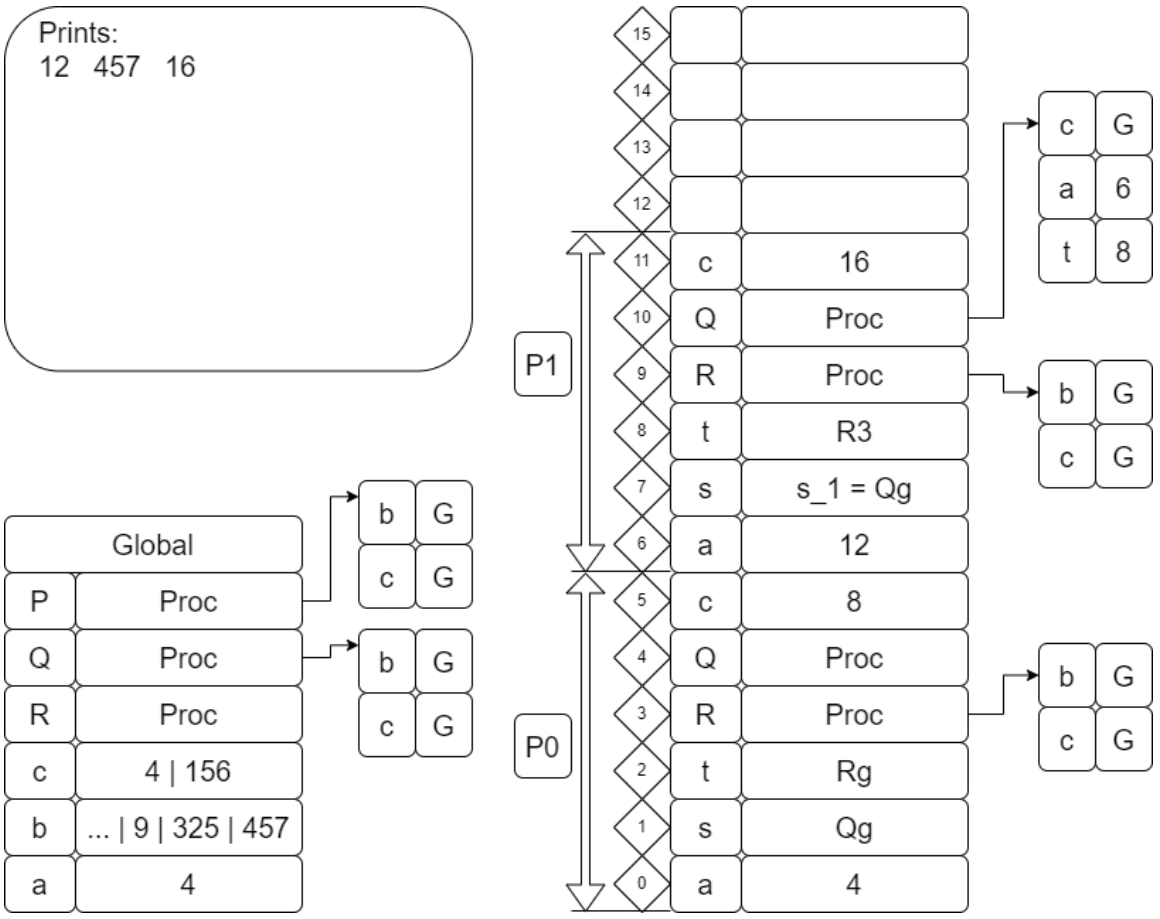
(c) Alcance estático y asociación superficial



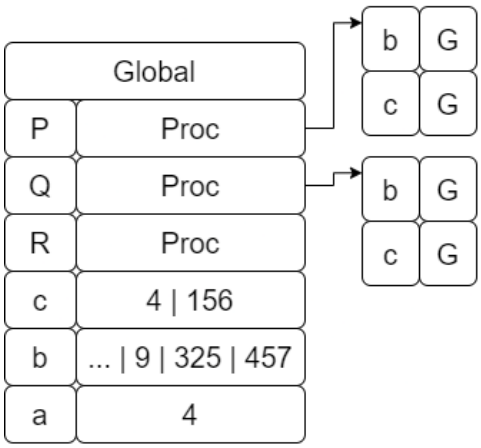


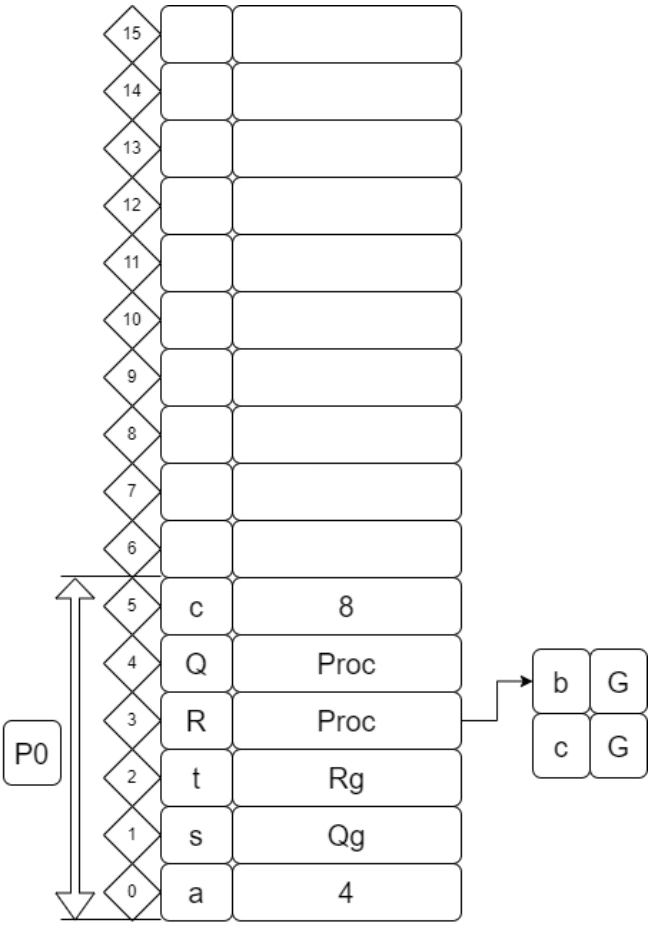


Prints:
12 457 16

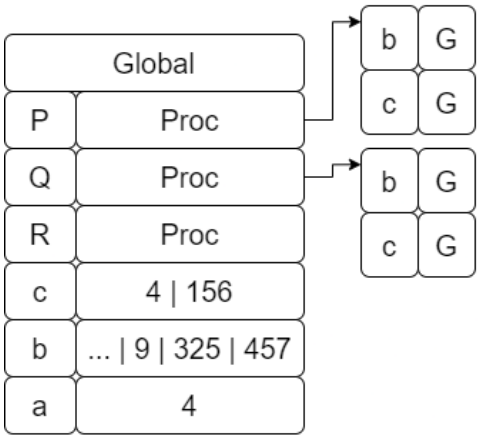


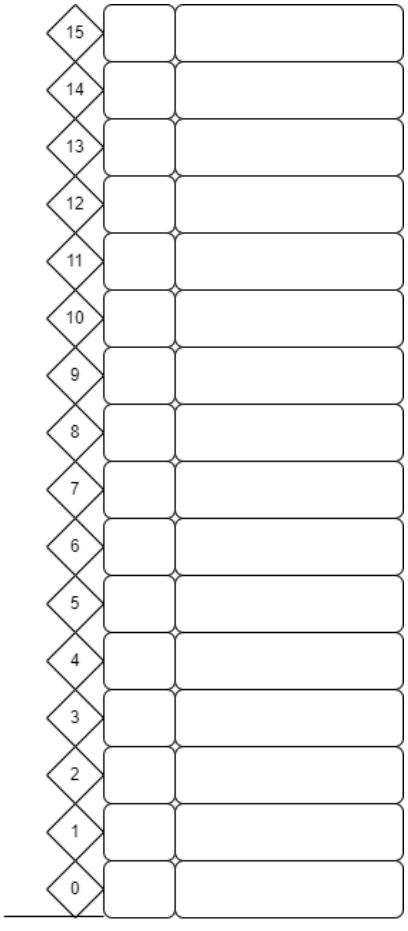
Prints:
12 457 16
4 457 8



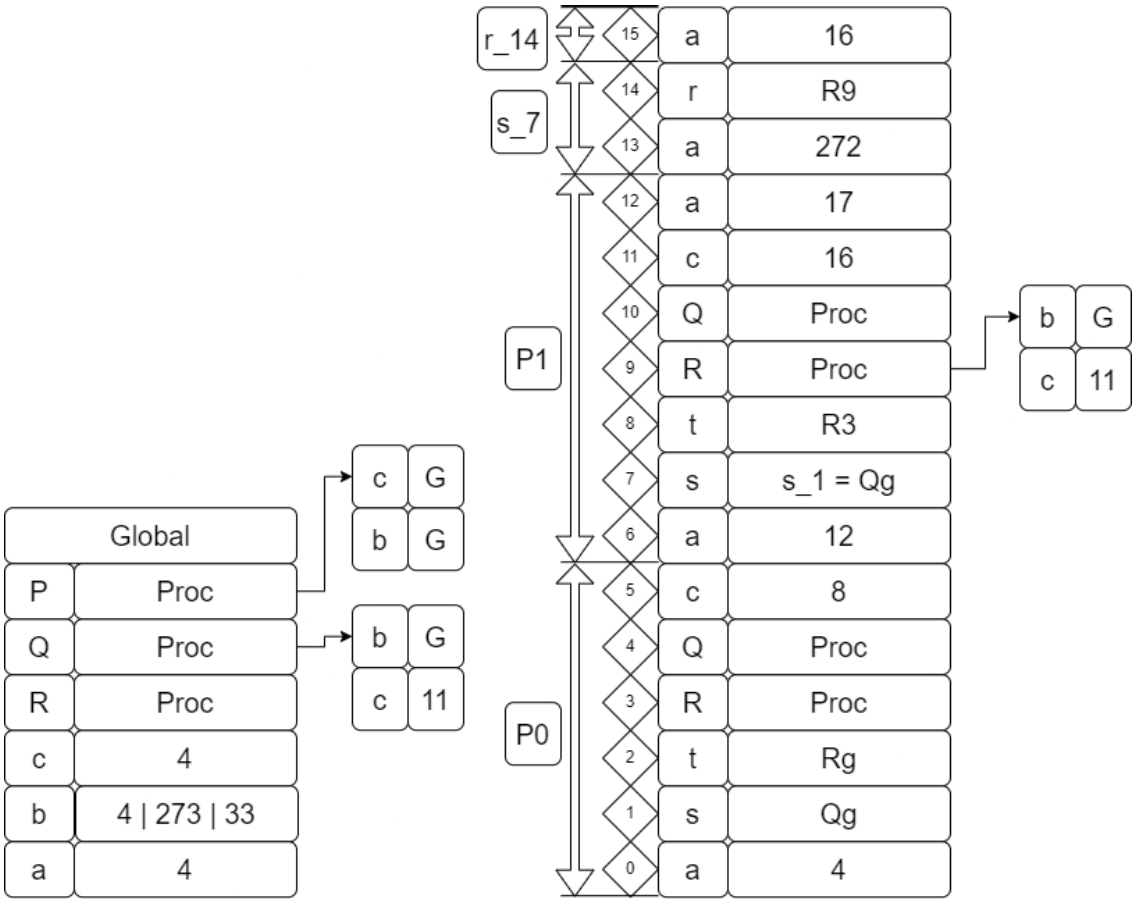


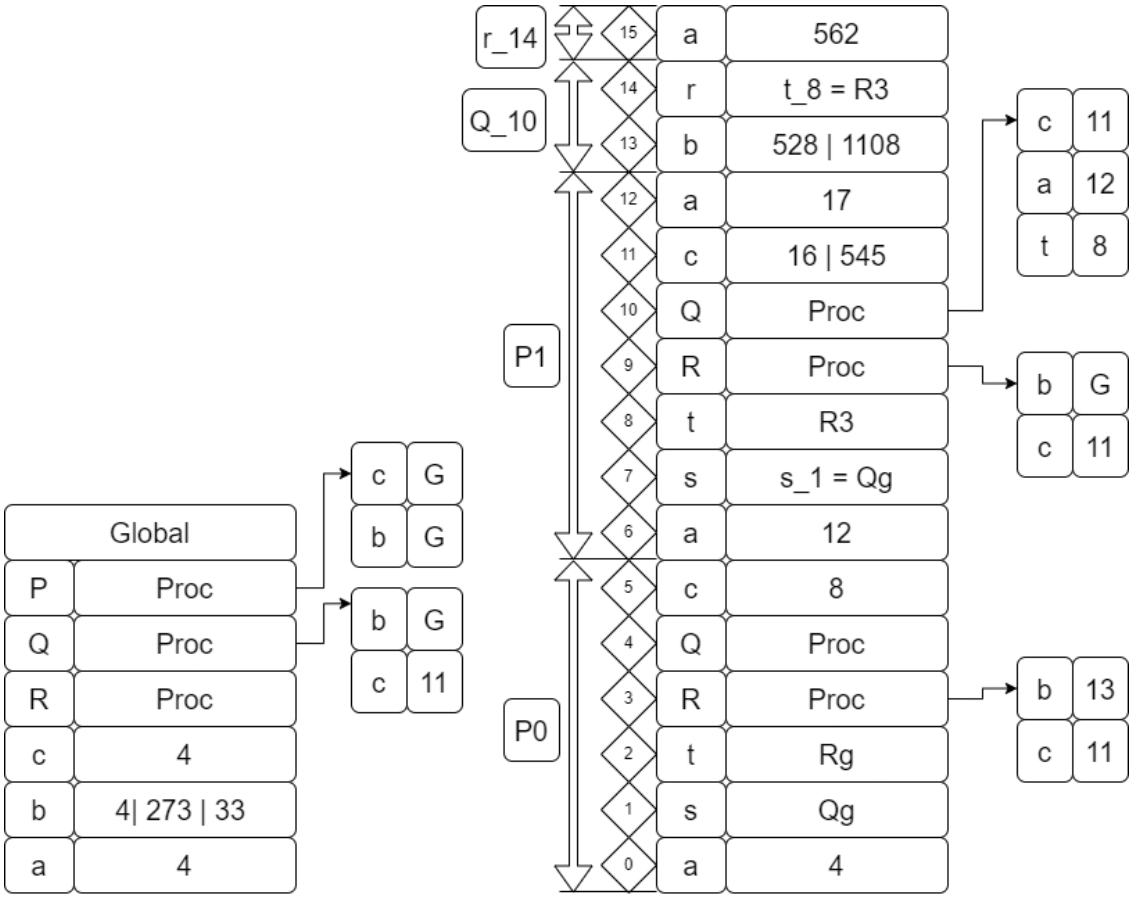
Prints:
12 457 16
4 457 8
4 457 156

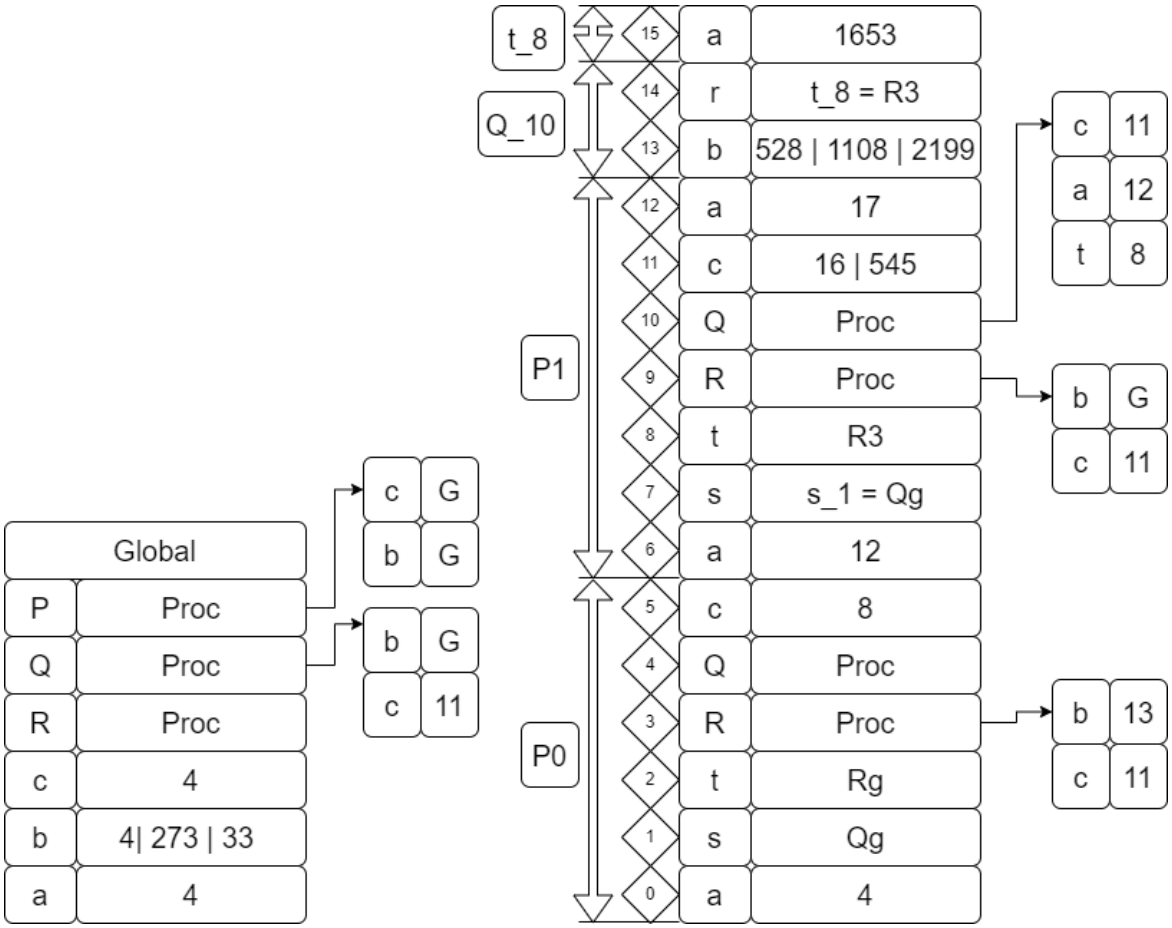




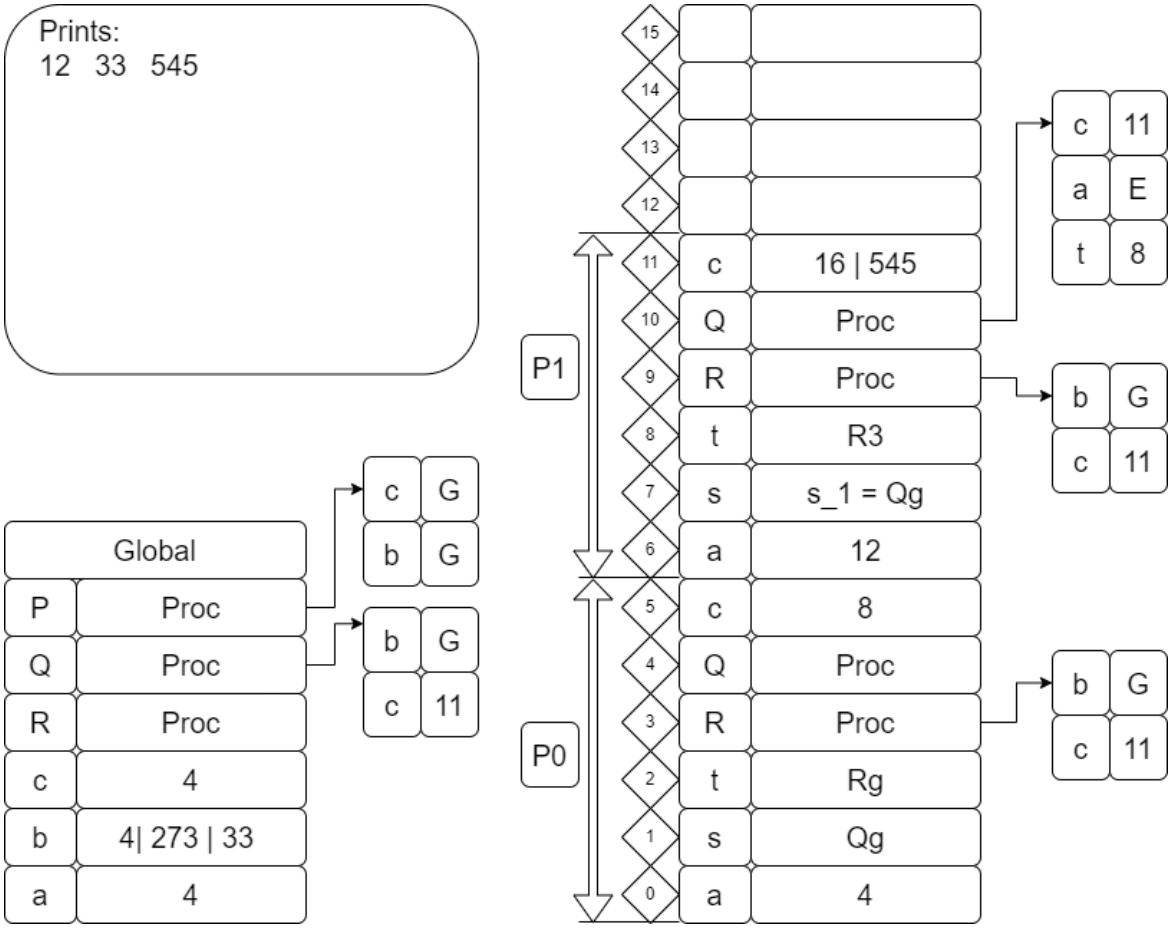
(d) Alcance dinámico y asociación superficial

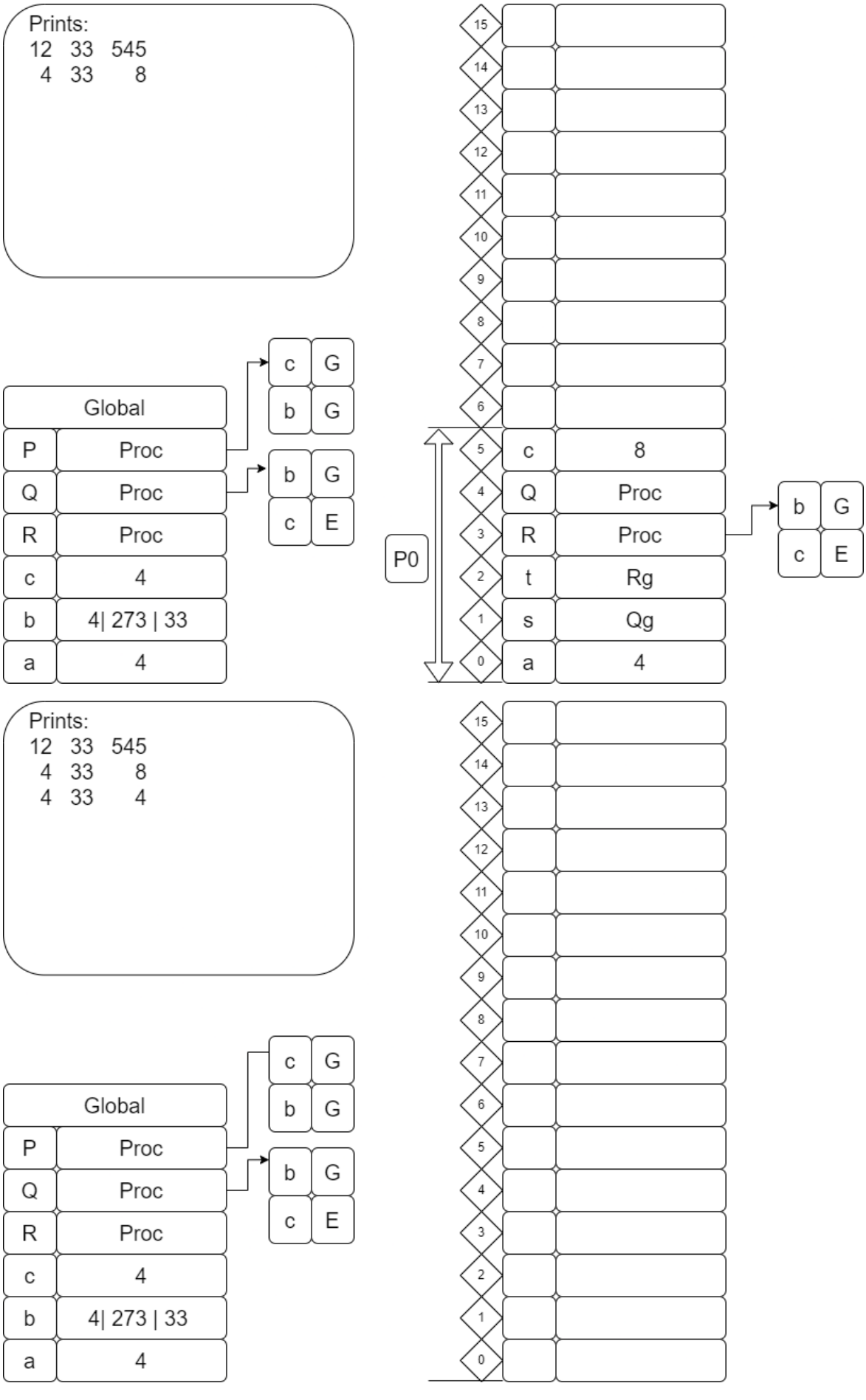






Prints:
12 33 545




Prints:
12 33 545
4 33 8
Global
P Proc
Q Proc
R Proc
c 4
b 4| 273 | 33
a 4
c G
b G
b G
c E
15
14
13
12
11
10
9
8
7
6
5 c 8
4 Q Proc
3 R Proc
2 t Rg
1 s Qg
0 a 4
P0
b G
c E
Prints:
12 33 545
4 33 8
4 33 4
Global
P Proc
Q Proc
R Proc
c 4
b 4| 273 | 33
a 4
c G
b G
b G
c E
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

- **3er Pregunta:**

- **4er Pregunta:**

- **5er Pregunta:**

- **Pregunta Extra:**