Universidad Nacional Autónoma de México

Facultad de Ciencias

Compiladores

Tarea 6:

Autores:

Escamilla Soto Cristopher Alejandro
Montiel Manriquez Ricardo

3 de Junio del 2022

1. Considera la siguiente implementación de *insertion sort*

```
void sort(int[] v, int n) {
    int i = 1;
    while (i<n) {
        int k = v[i];
        int j = i-1;
        while (j>=0 && v[j]>k) {
            v[j+1] = v[j];
            j = j-1;
        }
        v[j+1] =k;
        i = i+1;
    }
}
```

Obtener el código de tres direcciones para este programa.

Obs: Los índices de los arreglos deben ser obtenidos por medio de desplazamientos considerando que un entero es de tamaño 4.

1	i = 1
2	if False i <n goto (16)
3	T1 = 4 * i
4	k = v[T1]
5	T2 = i-1
6	j = T2
7	T3 = 4*j
8	if False (j>= 0 && v[T3] >k) goto (15)
9	T4 = j+1
10	T5=4*T4
11	T6 = 4*j
12	v[T5]=v[T6]
13	j=j-1
14	if(j>=0 && v[T3] goto (9)
15	if(i<n) goto (3)
16	T8 = 4*j
17	T9 = T8+4
18	v[T9]= k
19	i=i+1

2. Describe qué son los records de activación y su uso en la ejecución de código. Además explica cómo funcionan los records de activación en presencia de recursión.

   Obs: Puedes consultar el extracto del libro del dragón que se anexa a la tarea o usar cualquier otra referencia. No olvides incluir una lista con las referencias que consultaste.

   Los records de activación son estructuras de datos almacenadas en la pila del control, nos sirven para guardar información acerca del estado de la maquina, estos pueden ser el valor del contador del programa y los registros donde ocurren las llamadas. Estos records de activación son generados durante las llamadas de procedimiento.

   Los datos que pueden aparecer en un récord de activación que por lo general son: valores temporales, datos locales, el estado guardado de la maquina antes de la llamada, un enlace al control apuntando al récord de activación de quien esta llamando, un espacio para el valor que se regresa y finalmente los parámetros de la función que llama.

   Su uso en la ejecución de código es de la siguiente manera, cuando el control llega a la primera llamada se activa la función y su récord de activación entra en la pila, cuando control vuelve de esa activación se hace pop al récord dejando solo el record de la llamada.

   Los records de activación frente a la recesión funcionan de manera en la que podemos tener varios records de activación en la pila al mismo tiempo. Por ejemplo, si tenemos una llamada a algo puede que inicien dos records de activación que están en la pila y se van de ella durante la vida de nuestra llamada, dejando solamente el récord de activación de nuestra llamada.

   Y bien para las funciones recursivas la asignacion de espacio no puede ser estatica dado que para entradas grandes podriamos asignar menor espacio y el programa nos genere un error, por lo que se usa alamacenamiento dinamico y este se adapta a las necesidades de la una funcion particular que se encuentre activa. Las funciones recursivas tienen tantos records de activacion como llamadas recursivas, por lo que se proporciona una pila y la primera llamada recursiva es la primera en entrar a la pila y por lo tanto su resultado sera el ultimo en salir

3. Considera el siguiente código:

```
int mul(int x, int y) {
if (x) return y - (0 - mul(x - 1, y)); else return 0; }
```

   Proporciona una traducción a RTL suponiendo que ya se han seleccionado las instrucciones.

   Deberás mostrar las traducciones intermedias de subexpresiones del código.

Supongamos que x #1 | que y #2 | y que 0 #3

```
L0 mul(#1 #2)
    entry
    exit Lf

L1 ubranch eq #1 -> L2, L9

l2 mov #1 #6 -> L3       //guarda el valor x

L3 mov #3 #7 -> L4       //guarda el valor 0

L4 rest #6 $1 -> L5

L5 mov #2 #10 -> L6      //guarda el valor de y

L6 call #4 <- mul(x-1,#10) -> L7

L7 call # 5 <- rest #7 #4 -> L8

L8 rest #2 #5 -> Lf

L9 mov #3 #11 -> Lf
```

Tenemos las instrucciones Lt (etiqueta traducida), Lf (etiqueta final) y Rop (resultado de la operación) y su traducción a RTL queda de la siguiente forma:

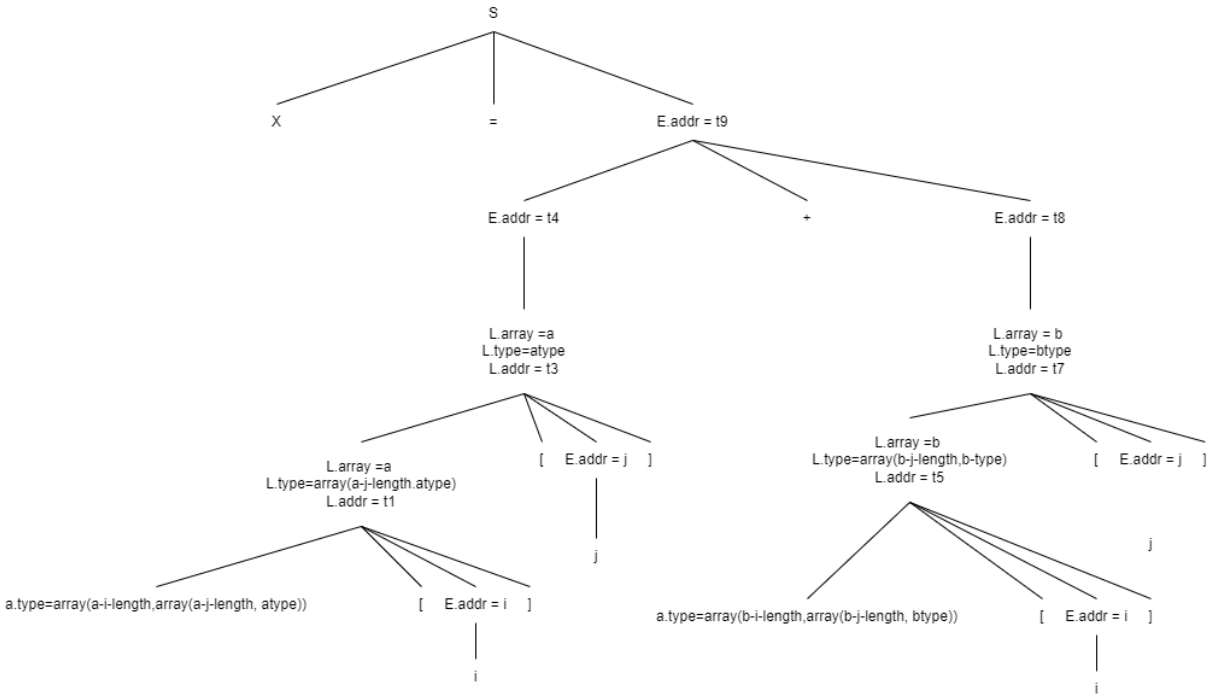
```
RTL(return y - (0-mul(x-1, y)) Rop, Lf)

    RTL(return y, Lf)

    RTL((0-mul(x-1, y)) Lf, Lt)

        RTL(return 0 Lf)

        RTL(mul(x-1,y)Lf, Lt)

RTL(return 0)
```

4. Utiliza la siguiente gramática para obtener el código de tres direcciones de la expresión:

```
x=a[i][j] + b[i][j]
```

$S \rightarrow$	$\mathbf{id} = E\ ;$	$\{\ gen(\ top.get(\mathbf{id}.lexeme)\ '='\ E.addr);\ \}$
$\mid$	$L = E\ ;$	$\{\ gen(L.array.base\ '['\ L.addr\ ']'\ '='\ E.addr);\ \}$
$E \rightarrow$	$E_1 + E_2$	$\{\ E.addr = \mathbf{new}\ Temp\,();$ $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr);\ \}$
$\mid$	$\mathbf{id}$	$\{\ E.addr = top.get(\mathbf{id}.lexeme);\ \}$
$\mid$	L	$\{\ E.addr = \mathbf{new}\ Temp\,();$ $gen(E.addr\ '='\ L.array.base\ '['\ L.addr\ ']');\ \}$
$L \rightarrow$	$\mathbf{id}\ [\ E\ ]$	$\{\ L.array = top.get(\mathbf{id}.lexeme);$ $L.type = L.array.type.elem;$ $L.addr = \mathbf{new}\ Temp\,();$ $gen(L.addr\ '='\ E.addr\ '*'\ L.type.width);\ \}$
$\mid$	$L_1\ [\ E\ ]$	$\{\ L.array = L_1.array;$ $L.type = L_1.type.elem;$ $t = \mathbf{new}\ Temp\,();$ $L.addr = \mathbf{new}\ Temp\,();$ $gen(t\ '='\ E.addr\ '*'\ L.type.width);$ $gen(L.addr\ '='\ L_1.addr\ '+'\ t);\ \}$

Como no tenemos información sobre las especificaciones de los arreglos para determinar el tipo del arreglo a y b, utilizamos "type" y como no sabemos la longitud de los componentes de cada arreglo utilizaremos length. Así podremos generalizar la operación.




```
t1 = i * a-i_width
t2 = j * a-j_width
t3 = t1 + t2
t4 = a[t3]
t5 = i * b-i_width
t6 = j * b-j_width
t7 = t5 + t6
t8 = b[t7]
t9 = t4 + t8
x = t9
```