Universidad Nacional Autónoma de México
Facultad de Ingeniería
Compiladores
Grupo 4, Semestre 2016-2
Programa Final

1 Objetivo

Elaborar un front-end utilizando lex y yacc para una definición dirigida por sintaxis que implementa un pequeño conjunto de instrucciones del lenguaje C.

2 Entregables

Se deberan entregar junto con el código fuente los siguientes documentos:

- El Esquema de traducción que se obtuvo de la Definición Dirigida por Sintaxis
- El manual de usuario del programa
- Un programa ejemplo para prueba del front-end

3 Consideraciones generales

Funciones

1. Función getTypeElement(type) retorna el tipo de base de type en la tabla de tipos
2. Función getWidthElement(type) retorna el tamaño en bytes de type en la tabla de tipos
3. Función get(id) retorna si el id esta dentro de la tabla de símbolos
4. Función getType(id) retorna el tipo del id en la tabla de símbolos
5. Función put(id, ...) inserta en la tabla de símbolos el id con sus propiedades, siempre y cuando no este dentro de ella, en caso contrario retorna -1
6. Función getTop() retorna la tabla de símbolos que está en el tope de la pila
7. Función getTypeTop() retorna la tabla de tipos que se encuentra en la cima de la pila de tipos.
8. Función getGlobal() retorna la tabla de símbolos que corresponde al ámbito global de programa y se encuentra en la base de la pila, debido a que fue el primero que entro en la pila.
9. Función setNumParam(id, num) inserta en la tabla de símbolos el número de parámetros de una función.
10. Función setParams(id, params) inserta en la tabla de símbolos la lista de parámetros para una función.
11. Función getParams(id) retorna la lista de parámetros de una función.

Tipos

1. Al tipo char le corresponde el valor 0
2. Al tipo int le corresponde el valor 1
3. Al tipo float le corresponde el valor 2
4. Al tipo double le corresponde el valor 3
5. Al tipo string le corresponde el valor 4

Tablas

1. Tabla de Símbolos guarda los siguientes datos
    - El id
    - El tipo del id (apuntador a la tabla de tipos)
    - El tipo de variable (variable=0, función=1, parámetro=2)
    - La dirección en caso de ser variable o parámetro
    - El apuntador a la tabla de símbolos en caso de ser una función para poder recuperar sus parámetros
    - El número de parámetros en caso de ser función.
2. Tabla de Tipos guarda la siguiente información:
    - La posición, es decir el lugar que ocupa en la tabla de tipos.
    - El tipo este puede ser representado numéricamente (char= 0, int= 1, float=1, double=2, array =3)
    - Su dimensión, es el número de localidades en caso de ser un arreglo
    - Su tamaño en bytes
    - El tipo base en caso de ser un tipo derivado (como los arreglos).

Estructuras adicionales

1. Pila para las tablas de símbolos
2. Pila para las tablas de tipos
3. Pila para el offset que permite manejar la direcciones locales (ámbito de las variables)
4. Una lista para guardar el tipo de los parámetros en una función.

Función ampliar

$Dir\ ampliar(Dir\ a, Type\ t, Type\ w)$
if w=t **then**
    return a
else if $t == char\ and\ w == int$ **then**
    $t1 = newTemp()$
    $gen(t1' =' charToint()a)$
    $return\ t1$
else if $t == int\ and\ w == float$ **then**
    $t1 = newTemp()$
    $gen(t1' =' intTofloat()a)$
    $return\ t1$
else if $t == float\ and\ w == double$ **then**
    $t1 = newTemp()$
    $gen(t1' =' floatTodouble()a)$
    $return\ t1$
else if $t == int\ and\ w == double$ **then**
    $t1 = newTemp()$
    $gen(t1' =' intTodouble()a)$
    $return\ t1$
end if

Función compatibles

$int\ compatibles(Type\ t, Type\ w)$
if $w = t$ **then**
    $return\ 1$
else if $t == char$ ***and*** $w == int$ ***or*** $t == int$ ***and*** $w == char$ **then**
    $return\ 1$
else if $t == int\ and\ w == float$ ***or*** $t == float$ ***and*** $w == int$ **then**
    $return\ 1$

else if $t == float\ and\ w == double$ ***or*** $t == double$ ***and*** $w == float$ **then**
    $return\ 1$
else if $t == int\ and\ w == double$ ***or*** $t == int$ ***and*** $w == double$ **then**
    $return\ 1$
else
    $return\ -1$
end if

Función max

$Type\ compatibles(Type\ t, Type\ w)$
if $w = t$ **then**
    $return\ t$
else if $t == char$ ***and*** $w == int$ ***or*** $t == int$ ***and*** $w == char$ **then**
    $return\ int$
else if $t == int\ and\ w == float$ ***or*** $t == float$ ***and*** $w == int$ **then**
    $return\ float$
else if $t == float\ and\ w == double$ ***or*** $t == double$ ***and*** $w == float$ **then**
    $return\ double$
else if $t == int\ and\ w == double$ ***or*** $t == int$ ***and*** $w == double$ **then**
    $return\ double$
else
    $return\ -1$
end if

4 Definición Dirigida por Sintaxis

PRODUCCIÓN	REGLAS SEMÁNTICAS
$P \rightarrow D\ D_f$	$offset = 0$
$D \rightarrow D\ T\ L_d$;	$c\_w = T.width$ $c\_t = T.type$
$D \rightarrow \varepsilon$	

$D_f \rightarrow D_f$ ***define*** T ***id*** (F) { L_s }	**if** $symbols.top.put(id.lexval, T.type, "func") != -1$ **then**
	$top = newSymbols();\ symbols.push(top)$
	$topType = newTypes();\ types.push(topType);$
	$offsets.push(offset)$
	$offset = 0$
	if $f\_type != T.type$ **then**
	$error("El\ tipo\ de\ retorno\ no\ coincide")$
	else
	$D_f.code = label(id) \parallel L.code \parallel label(L.next) \parallel gen('halt')$
	end if
	$symbols.pop();\ types.pop();\ offset = offsets.pop();$
	else
	$error("Id\ duplicado")$
	end if

$D_f \rightarrow \varepsilon$	

$F \rightarrow F_1$, T ***id***	**if** $(symbols.top.put($***id***.lexval, T.type$, offset, "param") == -1)$ **then**
	$error("Id\ duplicado")$
	else
	$offset = offset + T.width$
	$F_1.list.add(E.type);\ F.list = F_1.list$
	end if

$F \rightarrow T$ ***id***	**if** $(symbols.top.put($***id***.lexval, T.type$, offset, "param") == -1)$ **then**
	$error("Id\ duplicado")$
	else
	$offset = offset + T.width;$
	$F.list = newList();\ F.list.add(E.type)$
	end if

$T \rightarrow B\ C$	w = $B.width$ t = $B.type$ $T.type = C.type$ $T.width = C.width$
$B \rightarrow$ **int**	$B.type$= int $B.width = 4$
$B \rightarrow$ **float**	$B.type$= float $B.width = 8$
$B \rightarrow$ **double**	$B.type$= double $B.width = 16$
$B \rightarrow$ **char**	$B.type$= char $B.width = 1$
$C \rightarrow$ [**num**] C_1	$C.type = insert\_type("array",$***num***$.valor, C_1.type)$ $C.width =$ ***num***$.valor \times C_1.width$

PRODUCCIÓN	REGLAS SEMÁNTICAS
$C \rightarrow \varepsilon$	$C.type = t$ $C.width = w$

$L_d \rightarrow L_{d1}$, ***id***	**if** $(symbols.top.put(\textbf{\textit{id}}.lexval, c\_t, offset, "var") == -1)$ **then**
	$error("Id\ duplicado")$
	else
	$offset = offset + c\_w;$
	end if

$L_d \rightarrow$ ***id***	**if** $(symbols.top.put(\textbf{\textit{id}}.lexval, c\_t, offset, "var") == -1)$ **then**
	$error("Id\ duplicado")$
	else
	$offset = offset + c\_w$
	end if

$L_s \rightarrow L_{s1}\ S$	$L.next = S.next$ $L.code = L_1.code \parallel label(L_1.next) \parallel S.code$
$L_s \rightarrow S$	$L.next = S.next$ $L.code = S.code$
$S \rightarrow$ ***while*** (B_c) S_1	$S.next = B_c.false$ $S.code = lable(\ S_1.next) \parallel B_c.code \parallel label(B_C.true)$ $\parallel S_1.code \parallel gen('goto'\ S_1.next)$
$S \rightarrow$ ***if*** (B_c) $S_1\ S'$	$S'.false = B_c.false$ $S.next = S_1.next \parallel S'.next$ $S.code = B_c.code \parallel label(B_c.true)$ $\parallel S_1.code \parallel S'.next$
$S \rightarrow$ { L }	$S.next = L.next$ $S.code = L.code$

$S \rightarrow I\ O_a E$;	$S.next = newLabel()$
	if $I.type == E.type$ **then**
	$S.code = I.code \parallel E.code \parallel gen(I.dir\ O_a.val\ E.dir)$
	else
	$error("Tipos\ no\ compatibles")$
	end if

$S \rightarrow$ ***return*** E ;	$S.next = newLabel$ $S.code = E.code \parallel gen('return'\ E.dir)$ $f\_type = E.type$
$S' \rightarrow$ ***else*** S	$S'.next = S.next$ $S.code = gen(\ 'goto'\ S.next\ ) \parallel label(S'.false) \parallel S.code$
$S' \rightarrow \varepsilon$	$S'.next = ''$ $S.code = label(S'.false)$
$I \rightarrow A$	$I.dir = A.base' =' ['A.dir']')$ $I.code = ""$

$I \rightarrow$ ***id***	**if** $(symbols.top.get(id.lexval)\ != -1)$ **then**
	$I.dir = id.lexval$
	$I.code = ''$
	else
	$error("El\ no\ existe\ el\ id)$
	end if

PRODUCCIÓN	REGLAS SEMÁNTICAS
$O_a \rightarrow$ **=**	$O_a.val = '='$
$O_a \rightarrow$ **+ =**	$O_a.val = '+='$
$O_a \rightarrow - =$	$O_a.val = '-='$
$O_a \rightarrow$ **/ =**	$O_a.val = '/='$
$O_a \rightarrow$ *** =**	$O_a.val = '*='$
$O_a \rightarrow$ **% =**	$O_a.val = '\%='$
$B_c \rightarrow B_{c1}$ **||** B_{c2}	$B_c.true = B_{c1}.true \parallel B_{c2}.true$ $B_c.false = B_{c2}.false$ $B_c.code = B_{c1}.code \parallel label(B_{c1}.false) \parallel B_{c2}.code$
$B \rightarrow B_{c1}$ **&&** B_{c2}	$B_c.true = B_{c2}.true$ $B_c.false = B_{c1}.false \parallel B_{c2}.false$ $B_c.code = B_{c1}.code \parallel label(B_{c1}.true) \parallel B_{c2}.code$
$B_c \rightarrow$ **(** B_{c1} **)**	$B_c.true = B_{c1}.true$ $B_c.false = B_{c1}.false$ $B_c.code = B_{c1}.code$
$B_c \rightarrow$ **!** B_{c1}	$B_c.true = B_{c1}.false$ $B_c.false = B_{c1}.true$ $B_c.code = B_{c1}.code$
$B_c \rightarrow E_1\ O_r\ E_2$	$B_c.true = newLabel()$ $B_c.false = newLabel()$ $B_c.code = E_1.code \parallel E_2.code$ $\parallel gen('if'\ E_1.dir\ O_r.val\ E_2.dir\ 'goto'\ B.true)$ $\parallel gen('goto'\ B.false)$
$O_r \rightarrow$ **>**	$O_r.val = '>'$
$O_r \rightarrow$ **>=**	$O_r.val = '>='$
$O_r \rightarrow$ **<**	$O_r.val = '<'$
$O_r \rightarrow$ **<=**	$O_r.val = '<='$
$O_r \rightarrow$ **==**	$O_r.val = '=='$
$O_r \rightarrow$ **! =**	$O_r.val = '!='$
$O_s \rightarrow$ **+**	$O_s.val = '+'$
$O_s \rightarrow -$	$O_s.val = '-'$
$O_f \rightarrow$ *****	$O_f.val = '*'$
$O_f \rightarrow$ **/**	$O_f.val = '/'$
$O_f \rightarrow$ **%**	$O_f.val = '\%'$

PRODUCCIÓN	REGLAS SEMÁNTICAS

$E \rightarrow E_1\ O_s\ E_t$	**if** $E_1.type == E_t.type$ **then**
	$E.type = E_1.type$
	$E.dir = newTemp()$
	$E.code = E_1.code \parallel E_t.code$
	$\parallel gen(E.dir = E_1.dir\ O_s.val\ E_t.dir)$
	else if $compatibles(E_1.type, E_t.type)$ **then**
	$E.type = max(E_1.type, E_t.type)$
	$\alpha_1 = amp(E_1.dir,E_1.type,E.type)$
	$\alpha_2 = amp(E_t.dir,Et1.type,E.type)$
	$E.code = E_1.code \parallel E_t.code$
	$\parallel gen(E.dir = \alpha_1\ O_s.val\ \alpha_2)$
	else
	$error(\text{"}id\ no\ declarado\text{"})$
	end if

$E \rightarrow E_t$	$E.type = E_t.type$ $E.dir = E_t.dir$ $E.code = E_t.code$

$E_t \rightarrow E_{t1}\ O_f\ E_f$	**if** $E_{t1}.type == E_f.type$ **then**
	$E_t.type = E_{t1}.type$
	$E_t.dir = newTemp()$
	$E_t.code = E_{t1}.code \parallel E_f.code$
	$\parallel gen(E_t.dir = E_{t1}.dir\ O_f.val\ E_f.dir)$
	else if $compatibles(E_{t1}.type, E_f.type)$ **then**
	$E_t.type = max(E_{t1}.type, E_f.type)$
	$\alpha_1 = amp(E_{t1}.dir,E_{t1}.type,E_t.type)$
	$\alpha_2 = amp(E_f.dir,E_f.type,E_t.type)$
	$E_t.code = E_{t1}.code \parallel E_f.code$
	$\parallel gen(E_t.dir = \alpha_1\ O_f.val\ \alpha_2)$
	else
	$error(\text{"}id\ no\ declarado\text{"})$
	end if

$E_t \rightarrow E_f$	$E_t.type = E_f.type$ $E_t.dir = E_f.dir$ $E_t.code = E_f.code$
$E_f \rightarrow$ **(** E **)**	$E_f.type = E.type$ $E_f.dir = E.dir$ $E_f.code = E.code$

$E_f \rightarrow$ ***id***	**if** symbols.top.get(***id.****lexval*)!=-1 **then**
	$E_f.dir =$ ***id.****lexval*
	$E_f.type =$ symbols.top.getType(***id.****lexval*)
	$E_f.code =$ ""
	else
	error("el id no existe")
	end if

$E_f \rightarrow A$	$E_f.dir = newTemp()$ $E_f.code = gen(E.dir' =' A.base'['A.dir']')$
$E_f \rightarrow$ ***num***	$E_f.dir =$ ***num.****lexval* $E_f.type =$ ***num.****type* $E_f.code = ''$

PRODUCCIÓN	REGLAS SEMÁNTICAS
$E_f \rightarrow$ ***chr***	$E_f.dir = \textbf{chr}.lexval$ $E_f.type = \textbf{chr}.type$ $E_f.code = ''$
$E_f \rightarrow$ ***str***	$E_f.dir = \textbf{str}.lexval$ $E_f.type = \textbf{str}.type$ $E_f.code = ''$

$E_f \rightarrow$ ***id*** **(** A_f **)**	**if** $symbols.global.get(id.lexval) != -1$ **then**
	  if $id.type ==' funcion'$ **then**
	  $list = symbols.top.getSymbols(\textbf{id}.lexval)$
	  if $list.tam == A_f.list.tam$ **then**
	  for $i = 0 : list.tam$ **do**
	  if $list[i].type != A_f.list[i]$ **then**
	  $error(''El\ tipo\ de\ dato\ no\ coincide'')$
	  end if
	  end for
	  else
	  $error(''No\ coincide\ el\ numero\ de\ argumentos'')$
	  end if
	  else
	  $error(''El\ id\ no\ es\ una\ funcion'')$
	  end if
	  $E_f.dir = newTemp()$
	  $E_f.type = symbols.top.getType(\textbf{id}.lexval)$
	  $E_f.code = A_f.code \parallel A_f.param \parallel gen(E.dir' =' call\ id,\ A_f.tam)$
	else
	  $error(''id\ no\ declarado'')$
	end if

$A_f \rightarrow A_{f1}$ **,** E	$A_f.code = A_{f1}.code \parallel E.code$ $A_f.params = A_{f1}.params \parallel gen('param'\ E.dir)$ $A_{f1}.list.insert(E.type)$ $A_f.list = A_{f1}.list$
$A_f \rightarrow E$	$A.code = E.code$ $A.params = gen('param'\ E.dir)$ $A_f.list = newList()$ $A_f.list.insert(E.type)$

$A \rightarrow$ ***id*** **[** E **]**	**if** $symbols.top.get(id.lexval) != -1$ **then**
	  $A.base = id.lexval$
	  $typeb = symbols.top.getType(id.lexval)$
	  $A.type = types.top.getTypeElement(typeb)$
	  $A.width = types.top.getWidthElement(A.type)$
	  $A.dir = newTemp()$
	  $A.code = gen(A.dir\ ' =' \ E.dir\ ' *' \ A.width)$
	else
	  $error(''el\ id\ no\ existe'')$
	end if

$A \rightarrow A_1$ **[** E **]**	$A.base = A_1.base$ $A.type = types.top.getTypeElement(A_1.type)$ $A.width = types.top.getWidthElement(A.type)$ $t = newTemp()$ $A.dir = newTemp()$ $A.code = A_1.code$ $\parallel = gen(t\ ' =' \ E.dir\ ' *' \ A.width)$ $\parallel gen(A.dir\ ' =' \ L_1.dir\ ' +' \ t)$