CSE443
Compilers

Dr. Carl Alphonse
alphonse@buffalo.edu
343 Davis Hall

http://www.cse.buffalo.edu/faculty/alphonse/SP17/CSE443/index.php
https://piazza.com/class/iybn4ndqa1s3ei
handling of type checking in:

- 'assignable ablock'
- 'assignable rec0p identifier'

handling of records/arrays

recap project 3

intermediate code expectations for pr04
Assignment status

- PR04 - due M 4/17 @5
- HW4 - out W 4/5 @5, due T 4/11 @5
Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
Three address code instructions
(see 6.2.1, pages 364-5)

1. \( x = y \text{ op } z \)
2. \( x = \text{ op } y \)  \( \) (treat \( i2r \) and \( r2i \) as unary ops)
3. \( x = y \)
4. goto L
5. if x goto L / ifFalse x goto L
6. if x relop y goto L
7. function calls:
   - param x
   - call p, n
   - y = call p
   - return y
8. \( x = y[i] \) and \( x[i] = y \)
9. \( x = &y, x = *y, *x = y \)
Our Language

- primitive types: integer, real, Boolean, character
- built-in type: string (array of character)
- user-defined types:
  - record types have names
    - type rec : (real x := -4.79,  y := 1.3)
  - array types have names
    - type arr : (5,10) -> string : ( "" )
  - function types have names
    - type bar : ( real : x ) -> rec
    - function f : bar { ... }
Our language

primitive types: integer, real, Boolean, character

built-in type: string (array of character)

user-defined types:
- record types have names
  - type rec : (real x := -4.79, y := 1.3)
- array types have names
  - type arr : (5,10) -> string : (""")
- function types have names
  - type bar : (real : x) -> rec
  - function f : bar { ... }
Our Language

- primitive types: integer, real, Boolean, character
- built-in type: string (array of character)
- user-defined types:
  - record types have names
    - `type rec : (real x := -4.79, y := 1.3)`
  - array types have names
    - `type arr : (5,10) -> string : ( "" )`
  - function types have names
    - `type bar : ( real : x ) -> rec`
    - `function f : bar { ... }`

We now have allow multi-dimensional arrays
Our Language

- primitive types: integer, real, Boolean, character
- built-in type: string (array of character)
- user-defined types:
  - record types have names:
    - type rec : (real x := -4.79, y := 1.3)
  - array types have names:
    - type arr : (5,10) -> string : ( "" )
  - function types have names:
    - type bar : ( real : x ) -> rec
    - function f : bar { ... }
Recursive records
Recursive functions

A record type must allow a component to be of the same type as the type itself:

type Node: ( integer datum:=0 ; Node rest:=null )
type information

- type indicates size
- type indicates storage location
  - primitives: in current environment
  - records: on heap
  - arrays: on heap
  - functions: code in static, locals on stack
- need to determine how to lay out records, arrays, invocation records in memory
Sizes of types

- int: 32 bits (2's complement)
- real: 64 bits (IEEE 754)
- Boolean: 8 bits (TBD)
- character: 8 bit (ASCII)
# Sizes of types

- **type string:** `?` → **character**
  - 5 bytes + length of string * size of character (= 1 byte)

<table>
<thead>
<tr>
<th># dims</th>
<th>size of dimension 1 (integer)</th>
<th>(0)</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>V</td>
<td>A</td>
</tr>
</tbody>
</table>

Sizes of types

What is the size of a multi-dimensional array of type T?
# of dimensions (X): 1 byte
sizes of dimensions (S_i): X*4 bytes
data: (\prod_{i \in X} S_i) * \text{sizeOf}(T)
Figure 6.16, p 375

\[ w = 2 \times 3 \]

\[ w = 2 + 6 \times 1 = 15 \]

\[ 9 + \text{array}(2, \text{arr}(3, \text{character})) \]

\[ \epsilon \]
**dblocks (6.3.5 and 6.3.6)**

records (in separate symbol table), sequence of declarations at start of sblock

definition → type identifier ':' dblock

\[
\begin{array}{l}
st\.put(\text{identifier.lxeme}, \text{TYPE}, \text{dblock.type}, \text{dblock.width})
\end{array}
\]

dblock → '('

\[
\begin{array}{l}
\quad \text{Env.push}(\text{st}); \text{st} = \text{new Env}(); \text{Stack.push}(\text{offset}); \text{offset} = 0;
\end{array}
\]

destination-list ')

\[
\begin{array}{l}
\quad \text{dblock.type} = \text{record}(\text{st}); \text{dblock.width} = \text{offset}; \text{st} = \text{Env.pop}(); \text{offset} = \text{Stack.pop}();
\end{array}
\]

destination-list → declaration ';' destination-list

destination-list → declaration

declaration → identifier ':'

\[
\begin{array}{l}
\quad \text{id-list.type} = \text{identifier};
\end{array}
\]

← however you store types

(identifier-list)

identifier-list → identifier ( sBinOp constant ) ', '

\[
\begin{array}{l}
\quad \text{st.put(\text{identifier.lxeme}, \text{VAR}, \text{identifier-list.type}, \text{offset}); offset} = \text{offset} + \text{identifier-list.type.width};
\end{array}
\]

identifier-list

identifier-list → identifier ( sBinOp constant )

\[
\begin{array}{l}
\quad \text{st.put(\text{identifier.lxeme}, \text{VAR}, \text{identifier-list.type}, \text{offset}); offset} = \text{offset} + \text{identifier-list.type.width};
\end{array}
\]