CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
Intermediate Representations
Sizes of types

What is the size of a multi-dimensional array of type T?

sizes of dimensions \((S_i)\): X*4 bytes

data: \((T \cdot \prod S_i) \cdot \text{sizeof}(T)\)
For the purposes of type checking the number of dimensions is relevant, but the size of each dimension is not.

Q: if a and b are compatible array types, what are the semantics of a := b?
What if type info comes after
dimensions?

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

w = 9 + 6 \times 1 = 15

w = 2 \times 3

w = 6

B

\text{character}

w = 1

\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

```
{ st.put(identifier.lexeme, TYPE, dblock.type, dblock.width }
```

dblock → ']

```
{ Env.push(st); st = new Env(); Stack.push(offset); offset = 0; }
```

declaration-list ']

```
{ dblock.type=record(st); dblock.width=offset; st=Env.pop(); offset=Stack.pop(); }
```

declaration-list → declaration ';'; declaration-list

declaration-list → declaration

declaration → identifier ':

```
{ id-list.type = identifier; } <- however you store types
```

identifier-list

```
offset = offset + identifier-list.type.width; }
```

identifier-list → identifier ( sBinOp constant ) ';

```
{ st.put(identifier.lexeme, VAR, identifier-list.type, offset);
   offset = offset + identifier-list.type.width; }
```

identifier-list

```
{ st.put(identifier.lexeme, VAR, identifier-list.type, offset);
   offset = offset + identifier-list.type.width; }
```

**Just suggestions, not to be taken literally**
**dblocks (6.3.5 and 6.3.6)**

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

definition -> type identifier ' :' dblock
  { st.put(identifier.lexeme, TYPE, dblock.type, dblock.width }  

dblock -> '['
  { Env.push(st); st = new Env(); Stack.push(offset); offset = 0; }
  declaration-list ' ]'
  { dblock.type=record(st); dblock.width=offset; st=Env.pop(); offset=Stack.pop(); }

declaration-list -> declaration ' ; ' declaration-list
declaration-list -> declaration

declaration -> identifier ' :'  
  { id-list.type = identifier; }  <- however you store types
  identifier-list

identifier-list -> identifier ( sBinOp constant ) ' ; '  
  { st.put(identifier.lexeme, VAR, identifier-list.type, offset); 
    offset = offset + identifier-list.type.width; }
identifier-list

identifier-list -> identifier ( sBinOp constant )  
  { st.put(identifier.lexeme, VAR, identifier-list.type, offset); 
    offset = offset + identifier-list.type.width; }

*Just suggestions, not to be taken literally*

We can specialize due to the structure of our grammar: see next slide!
dblocks (6.3.5 and 6.3.6)
records (in separate symbol table), sequence of declarations at start of sblock

Since declarations must be gathered together at the start of an sblock, and cannot themselves be directly nested, we can do better:

dblock \rightarrow '[
{ Env.push(st); st = new Env(); Stack.push(offset); offset = 0; } declaration-list ']
{ dblock.type=record(st); dblock.width=offset; st=Env.pop(); offset=Stack.pop(); }

integer: x
integer: y
real: x
real: y
real: z
integer: x
boolean: y
character: z
dblocks (6.3.5 and 6.3.6)

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

Since declarations must be gathered together at the start of an sblock, and cannot themselves be directly nested, we can do better:

dblock → ']['

{ Env.push(st); st = new Env(); Stack.push(offset); offset = 0; }

declaration-list ']

{ dblock.type=record(st); dblock.width=offset; st=Env.pop(); offset=Stack.pop(); }
dblocks (6.3.5 and 6.3.6)
records (in separate symbol table), sequence of declarations at start of sblock

Since declarations must be gathered together at the start of an sblock, and cannot themselves be directly nested, we can do better:

```
dblock -> ']
  { Env.push(st); st = new Env(); Stack.push(offset); offset = 0; }
  declaration-list ']
  { dblock.type=record(st); dblock.width=offset; st=Env.pop(); offset=Stack.pop(); }
```

```plaintext
{ (integer : x , y )
  push offset = 8 onto stack
  offset = 4
  offset = 8
  (real : x , z ) .......
  offset = 16
  offset = 24
  pop offset = 8 from stack
  push offset = 8 onto stack
  offset = 8
  offset = 9
  offset = 10
  (Boolean : y ; character : z )
  ... 
  offset
  offset

  pop offset = 8 from stack
}
```

AT RUNTIME
dblocks (6.3.5 and 6.3.6)

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

Since declarations must be gathered together at the start of an sblock, and cannot themselves be directly nested, we can do better:

dblock $\rightarrow$ '[$'

Env.push(st); st = new Env(); Stack.push(offset); offset = 0; }
declaration-list ']'
dblocks (6.3.5 and 6.3.6)

Since declarations must be gathered together at the start of an sblock, and cannot themselves be directly nested, we can do better:

$\langle$offset $=$ 8$\rangle$

integer: $x$

$\langle$offset $=$ 8$\rangle$

integer: $y$

$\langle$offset $=$ 16$\rangle$

real: $x$

$\langle$offset $=$ 24$\rangle$

real: $z$

$\langle$offset $=$ 8$\rangle$

$\langle$offset $=$ 9$\rangle$

Boolean: $y$

$\langle$offset $=$ 10$\rangle$

character: $z$

$\langle$offset $=$ 8$\rangle$

$\langle$offset $=$ 8$\rangle$

AT RUNTIME

push offset = 8 onto stack

pop offset = 8 from stack

push offset = 8 onto stack

pop offset = 8 from stack

push offset = 8 onto stack

pop offset = 8 from stack
dblocks (6.3.5 and 6.3.6)

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

Since declarations must be gathered together at the start of an sblock, and cannot themselves be directly nested, we can do better:

dblock → '[
{ Env.push(st); st = new Env(); Stack.push(offset); offset = 0; }
declaration-list ']
{ dblock.type=record(st); dblock.width=offset; st=Env.pop(); offset=Stack.pop(); }

AT RUNTIME

offset = 0
push offset = 8 onto stack

{ ( integer : x , y )
offset = 4
push offset = 8 onto stack

{ ( real : x , z ) … … }
offset = 8

pop offset = 8 from stack
push offset = 8 onto stack

offset = 16
offset = 24

{ ( Boolean : y ; character : z ) … … }
offset = 9
offset = 10

pop offset = 8 from stack

offset = 8
dblocks (6.3.5 and 6.3.6)
records (in separate symbol table), sequence of declarations at start of sblock

Since declarations must be gathered together at the start of an sblock, and cannot themselves be directly nested, we can do better:

```
dblock → ']['
  { Env.push(st); st = new Env(); Stack.push(offset); offset = 0; }
  declaration-list ']
  { dblock.type=record(st); dblock.width=offset; st=Env.pop(); offset=Stack.pop(); }
```

```
offset = 0
{ ( integer : x , y )
push offset = 8 onto stack
offset = 4
offset = 8
{( real : x , z ) ... ...
offset = 16
offset = 20
offset = 24
pop offset = 8 from stack
push offset = 8 onto stack
offset = 8
{( Boolean : y ; character : z )
offset = 24
offset = 26
offset = 28
offset = 30
offset = 32
offset = 34
... }
pop offset = 8 from stack
}
```

AT RUNTIME

```
offset = 8
```