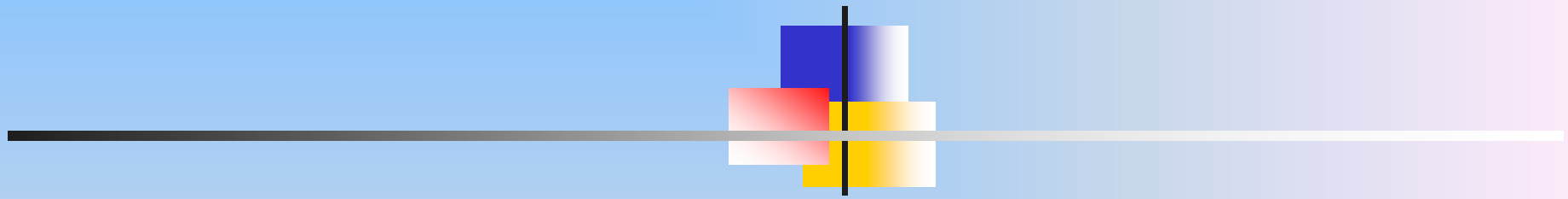
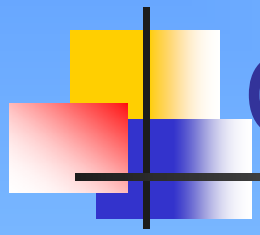
CSE302: Compiler Design



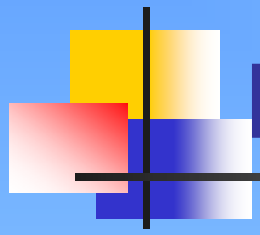
Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

April 12, 2007



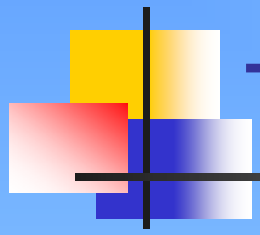
Outline

- Recap
- Intermediate code generation
  - Type expression and storage
- Summary



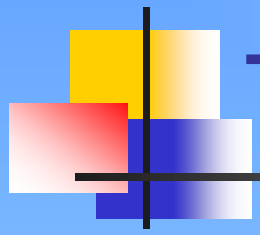
How to Address Array Elements?

- Row-major order with index starting at 0
  - A[i] locates at base+i×w
  - A[i1][i2] locates at base+i1×w1+i2×w2
    - int[2][3]: w1=12, w2=4
- L → **id** [E] | L [E]
- SDT for array references



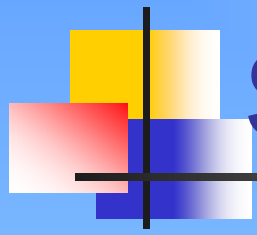
Translation of Array References

- L→ **id** [E]
  - L.array = top.get(id.lexeme)
  - L.type = L.array.type.elem
  - L.addr = **new** Temp()
  - gen(L.addr '=' E.addr '*' L.type.width)
- L→ L1 [E]
  - L.array = L1.array
  - L.type = L1.type.elem
  - t = **new** Temp()
  - L.addr = **new** Temp()
  - gen(t '=' E.addr '*' L.type.width)
  - gen(L.addr '=' L1.addr '+' t)
- S → L = E ;
  - gen(L.array.base '[' L.addr ']' '=' E.addr
- E → L
  - E.addr = new Temp()
  - gen(E.addr '=' L.array.base '[' L.addr ']')
- S → id = E ;
  - S.code = E.code || gen(top.get(id.lexeme) '=' E.addr)



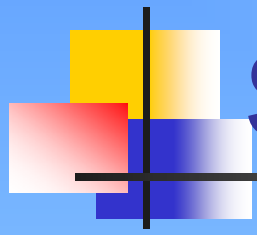
Type Expressions and Storage

- Declarations
  - D → T **id**; D | ε
  - T → B C | **record** '{' D '}'
  - B → **int** | **float**
  - C → [**num**] C | ε
- A type expression
  - A basic type
  - A type constructor
    - array
    - record
    - →
- Names: Types and their widths in storage



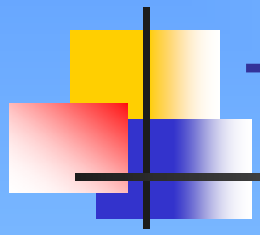
SDT of Array Type Declaration

- T → B {t = B.type; w = B.width;} C {
  T.type = C.type; T.width = C.width;
  }
- B → **int** {B.type = integer; B.width = 4;}
- B → **float** {B.type = float; B.width = 8;}
- C → [**num**] C1 {
  C.type = array(**num**.value, C1.type);
  C.width = **num**.value × C1.width;
  }
- C → ε {C.type = t; C.width = w;}



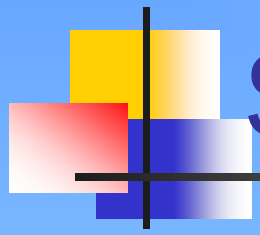
SDT of Array Type Declaration

- T → B {t = B.type; w = B.width;} C {
  T.type = C.type; T.width = C.width;
  }
- B → **int** {B.type = integer; B.width = 4;}
- B → **float** {B.type = float; B.width = 8;}
- C → [**num**] C1 {
  C.type = array(**num**.value, C1.type);
  C.width = **num**.value × C1.width;
  }
- C → ε {C.type = t; C.width = w;}
- int[2][3] a;



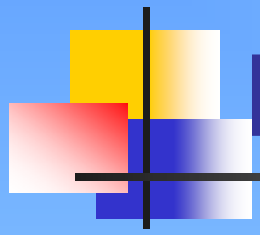
Translation of Array References

- L→ **id** [E]
  - L.array = top.get(id.lexeme)
  - L.type = L.array.type.elem
  - L.addr = **new** Temp()
  - gen(L.addr '=' E.addr '*' L.type.width)
- L→ L1 [E]
  - L.array = L1.array
  - L.type = L1.type.elem
  - t = **new** Temp()
  - L.addr = **new** Temp()
  - gen(t '=' E.addr '*' L.type.width)
  - gen(L.addr '=' L1.addr '+' t)
- S → L = E ;
  - gen(L.array.base '[' L.addr ']' '=' E.addr
- E → L
  - E.addr = new Temp()
  - gen(E.addr '=' L.array.base '[' L.addr ']')



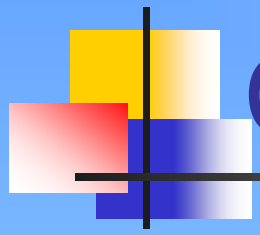
Sequences of Declarations

- P → {offset = 0;} D
- D → T id; {
  top.put(id.lexeme, T.type, offset);
  offset = offset + T.width;
  } D1
- D → ε



Fields in Records and Classes

- T → **record** ‘{’
  {
  Env.push(top); top = new Env();
  Stack.push(offset); offset = 0;
  }
  D ‘}’
  {
  T.type = record(top);
  T.width = offset;
  top = Env.pop();
  offset = Stack.pop();
  }



Outline

- Recap
- Intermediate code generation
- Summary