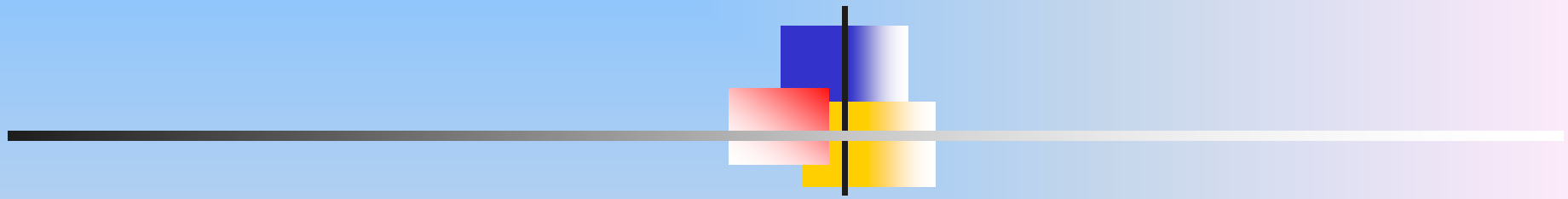
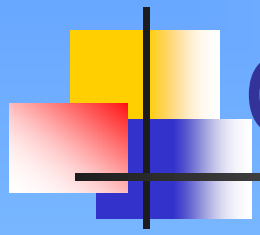
CSE302: Compiler Design



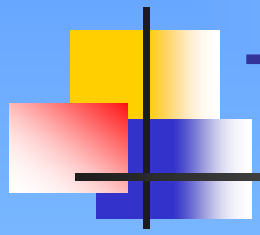
Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

April 24, 2007



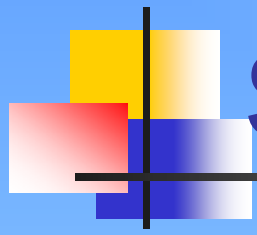
Outline

- Recap
  - Three address code generation
- Type checking
- Brief introduction to runtime environments
- Summary



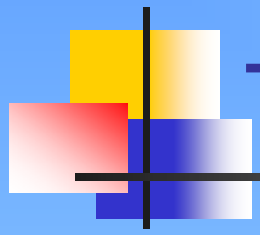
Translation of Expressions

- S → id = E ;
  - S.code = E.code || gen(top.get(id.lexeme) ‘=’ E.addr)
- E → E1 + E2
  - E.addr = new Temp()
  - E.code = E1.code || E2.code
    || gen(E.addr ‘=’ E1.addr ‘+’ E2.addr)
- E → - E1
  - E.addr = new Temp()
  - E.code = E1.code || gen(E.addr ‘=’ ‘minus’ E1.addr)
- E → (E1)
  - E.addr = E1.addr
  - E.code = E1.code
- E → **id**
  - E.addr = top.get(id.lexeme)
  - E.code = ‘’



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;}



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 ']')

Translation of Control Flows and Boolean Expressions

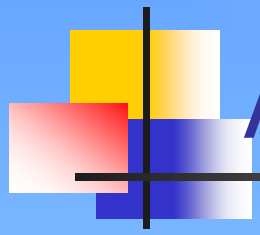
- S→if(B) S1 {B.true=newlabel(); B.false=S.next;} {S.code=B.code || label(B.true) || S1.code
  - if B goto L1
  - goto L2
  - L1: S1
  - L2:
- B → B1 || B2
- S → if(B1||B2) S1 {B.true=newlabel(); B.false=S.next; B1.true=B.true; B1.false=newlabel(); B2.true=B.true; B2.false=B.false;} {B.code=B1.code || label(B1.false) || B2.code} {S.code=B.code || label(B.true) || S1.code
  - if B1 goto L1
  - goto L2
  - L2: if B2 goto L1
  - goto L3
  - L1: S1
  - L3:

Translation of Boolean Expressions

- B → B1 && B2
- S → if({ B.true=newlabel(); B.false=S.next;
  B1.true=newlabel(); B1.false=B.false;
  B2.true=B.true; B2.false=B.false;}
  B1&&B2) {B.code=B1.code || label(B1.true) || B2.code}
  S1 {S.code=B.code || label(B.true) || S1.code

- B → !B1
- S → if({B.true=newlabel(); B.false=S.next;
  B1.true=B.false; B1.false=B.true;}
  !B1) {B.code=B1.code}
  S1 {S.code=B.code || label(B.true) || S1.code}

- B → E1 rel E2
- S → if({B.true=newlabel(); B.false=S.next;}
  E1 rel E2) {B.code=E1.code || E2.code ||
  gen('if E1.addr rel.op E2.addr 'goto' B.true ||
  gen('goto' B.false);
  }
  S1 {S.code=B.code || label(B.true) || S1.code}



An Example

- while(a<b && a<c) a=(a+d)

An Example

- B → B1 && B2
- S → if({B.true=newlabel(); B.false=S.next; B1.true=newlabel();
  B1.false=B.false; B2.true=B.true; B2.false=B.false;}
  B1&&B2) {B.code=B1.code || label(B1.true) || B2.code}
  S1 {S.code=B.code || label(B.true) || S1.code

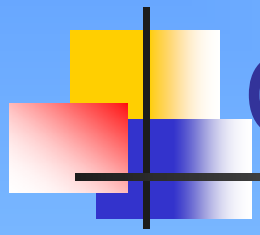
- B → E1 rel E2
- S → if({B.true=newlabel(); B.false=S.next;}
  E1 rel E2) {B.code=E1.code || E2.code ||
  gen('if E1.addr rel.op E2.addr 'goto' B.true || gen('goto' B.false); }
  S1 {S.code=B.code || label(B.true) || S1.code}

- S → while({begin = newlabel();
  B.true=newlabel(); B.false=S.next;}
  B) {S1.next=begin;}
  S1 {S.code=label(begin) || B.code || label(B.true) ||
  S1.code || gen('goto' begin);}

- E → **id** {E.addr = top.get(id.lexeme); E.code = '';}

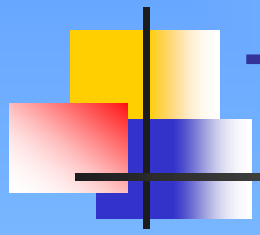
- S → id = E ; {S.code = E.code || gen(top.get(id.lexeme) '=' E.addr);}

- E → E1 + E2 {E.addr = new Temp();
  E.code = E1.code || E2.code || gen(E.addr '=' E1.addr '+' E2.addr);}



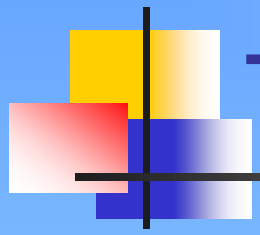
Outline

- Recap
- Type checking
- Brief introduction to runtime environments
- Summary



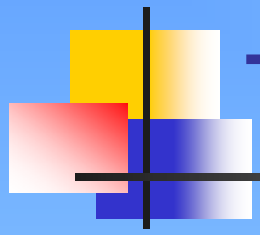
Type Expressions

- 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
    - →



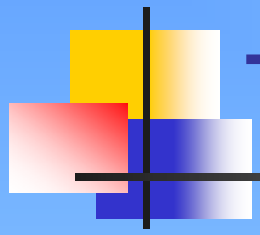
Type Equivalence

- **if** two type expressions are equal
  then return a certain type
  else error
  - Same basic type
  - Same constructor applied to structurally equivalent types
  - A type name denoting the other
    - typedef int aaa, bbb, ccc;
      /* all ints */
      aaa int1;
      bbb int2;
      ccc int3;
    - struct _node
      {
      char *name;
      char *value;
      struct _node *next;
      };
      typedef struct _node Node;



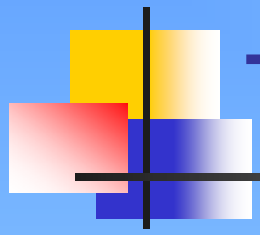
Type Checking

- Each component has a type expression assigned
  - Conform to the type system
  - Strongly typed languages



Type Synthesis

- if f has type $s \rightarrow t$ and x has type s then expression $f(x)$ has type t
- Type declarations are needed
- A type conversion SDD
  - E $\rightarrow$ E1 + E2
    - E.addr = new Temp()
    - E.code = E1.code || E2.code
      || gen(E.addr '=' E1.addr '+' E2.addr)

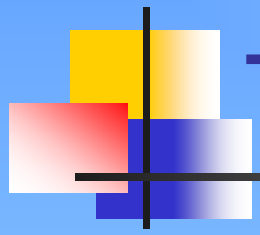


Type Conversion Hierarchy

- Double, float, long, int, short, char, byte
  - Widening conversion hierarchy
  - Narrowing conversion hierarchy
- Addr widen(Addr a, Type t, Type w) {

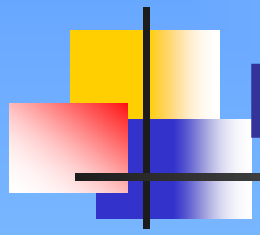
```
if (t=w) return a;
else if (t=int and w=float) {
        temp = new Temp();
        gen(temp '=' '(float)' a);
        return temp;
}
else error;
```

  }
- max(t1,t2)



Type Synthesis

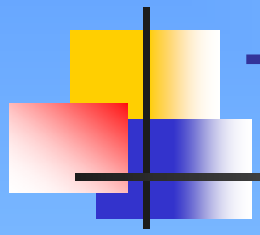
- if f has type $s \rightarrow t$ and x has type s then expression $f(x)$ has type t
- A type conversion SDD
  - E $\rightarrow$ E1 + E2
    - E.type = max(E1.type, E2.type)
    - a1 = widen(E1.addr, E1.type, E.type)
    - a2 = widen(E2.addr, E2.type, E.type)
    - E.addr = new Temp()
    - E.code = E1.code || E2.code
      || gen(E.addr '=' a1 '+' a2)



Function/Operator Overloading

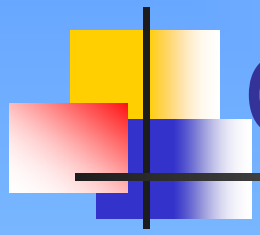
- void err() {...}; void err(String s) {...}
- 1+2; A+B;

- if f have type $s_i \rightarrow t_i$ and x has type s_k then expression $f(x_k)$ has type t_k



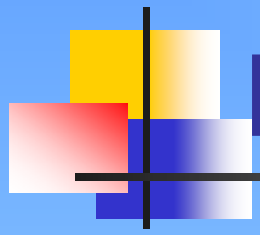
Type Inference

- No need to declare type before usage
- A Scheme example
  - *length*.scm
  - (+ (length '("String A" "String B")) (length '(1 2 3)))
  - The type expression of *length*
    - $\forall\alpha$. list(α) $\rightarrow$ integer



Outline

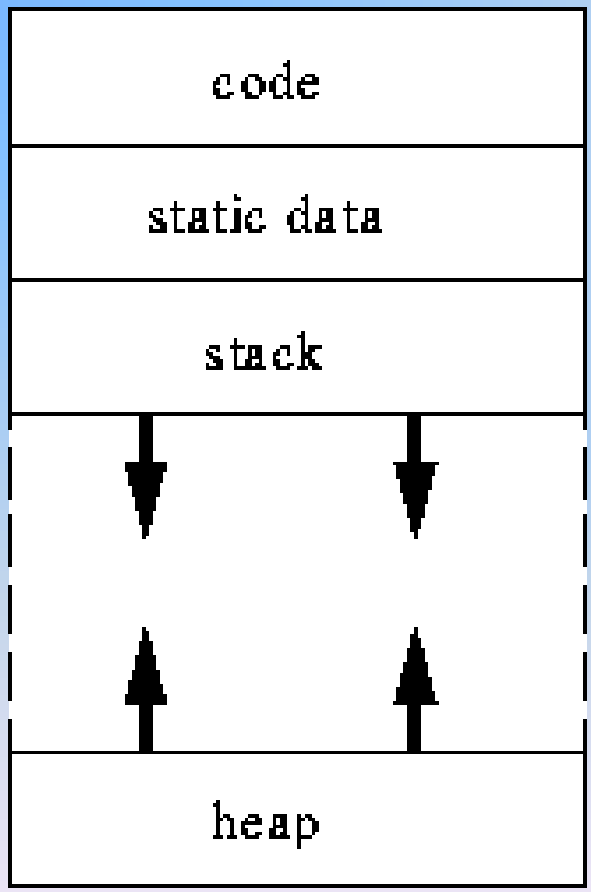
- Recap
- Type checking
- Brief introduction to runtime environments
- Summary

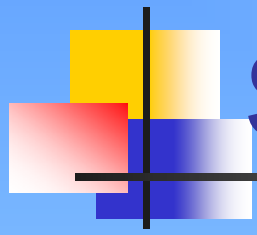


Run-Time Environments

- Storage layout and allocation
- Subprogram linkages
- Variable reference accessing mechanisms
- Parameter passing mechanisms
- Interface to OS and I/O

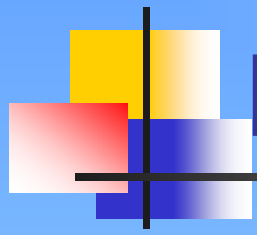
Storage Organization





Subprogram Linkages

- Activation record instance
  - Stack frame
- Subprogram call
  - Push stack frames
- Subprogram exit
  - Pop stack frames
- Stack-dynamic variable

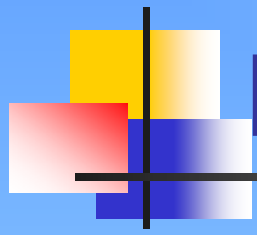


Heap-dynamic Variables

- C
  - **malloc** and **free**
- C++
  - **new** and **delete**

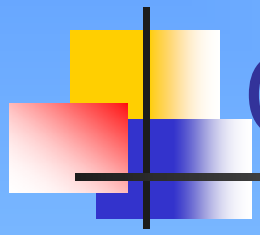
```
int *intnode;
...
intnode = new int;
...
delete intnode;
```

- Java
  - All objects



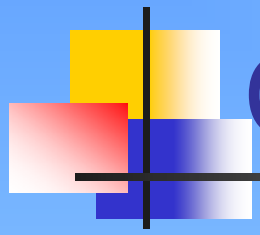
Heap Management

- Reference counters (eager approach)
  - Maintain a counter in every cell that store the number of pointers currently pointing at the cell
- Garbage collection (lazy approach)
  - Allocate and disconnect until all available cells allocated; then begin gathering all garbage



Garbage Collection

- Every heap cell has an extra bit used by collection algorithm
- All cells initially set to garbage
- All pointers traced into heap, and reachable cells marked as not garbage
- All garbage cells returned to a list of available cells



Outline

- Recap
- Type checking
- Brief introduction to runtime environments
- Summary