

The TeachScheme! Project

Adelphi University

Brown University

Northeastern University

University of Chicago

University of Utah

Worcester Polytechnic Institute

The Revolution

Two principles:

Shift away from machine details

Emphasis on correctness over efficiency
(ie, focus on program design)

What's Wrong with Machine-Oriented Languages?

machine arithmetic, pointers and
memory addresses, even i/o

- Make students waste time on unimportant and uninteresting details
- Force students to confront issues they are not prepared for
- and ...

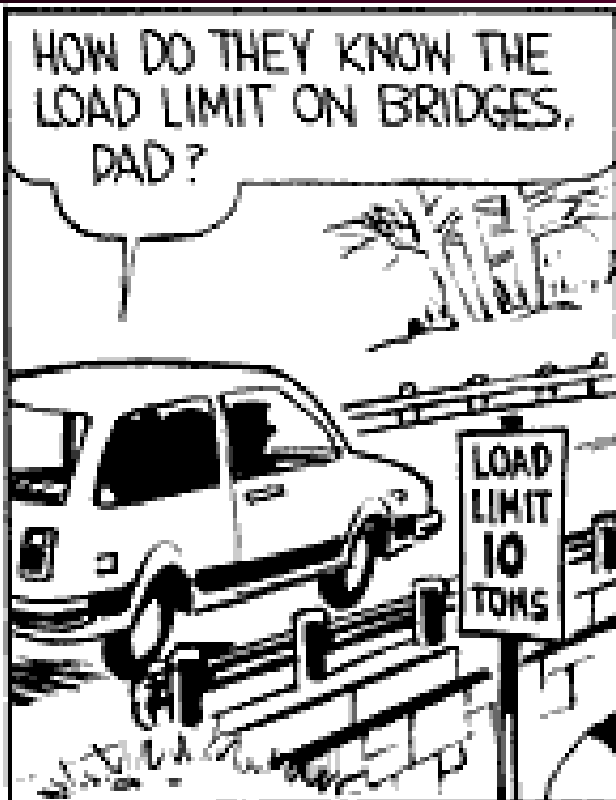
What Computer Science is *Not* About

The computer!



Just as biology isn't "microscope science"
and writing isn't "pen science" ...

What's This About Program Design?



Why Am I Here?

- The TeachScheme! Project: Outreach program hosted by six universities
- Specially designed for high schools
- Provides all material -- books, software, etc -- free of charge

What Teachers Experience

K.I.S.S. Keep It Simple Syntactically

C++/Pascal

10% Problem-solving vs 90% Syntax

Scheme

90% Problem-solving vs 10% Syntax



The Golden Rule of Scheme Syntax

(Operation List-of-Arguments)

or

(Operation Arg₁)

The Golden Rule of Scheme Syntax

(Operation List-of-Arguments)

or

(Operation Arg₁ Arg₂)

The Golden Rule of Scheme Syntax

(Operation List-of-Arguments)

or

(Operation Arg₁ Arg₂ . . . Arg_n)

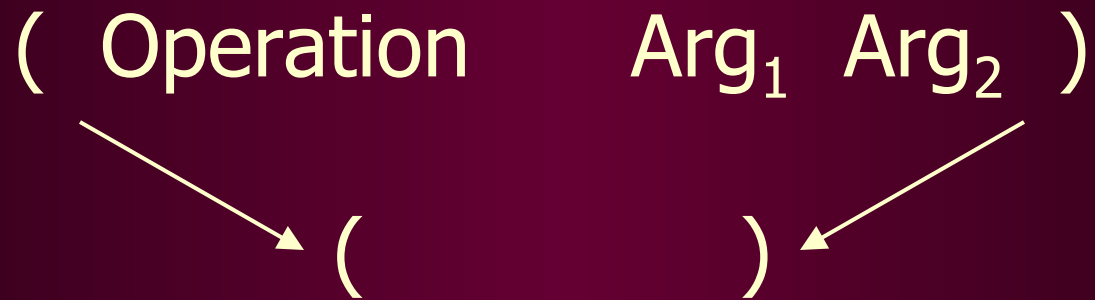
An Example From Arithmetic

$$4 + 5$$

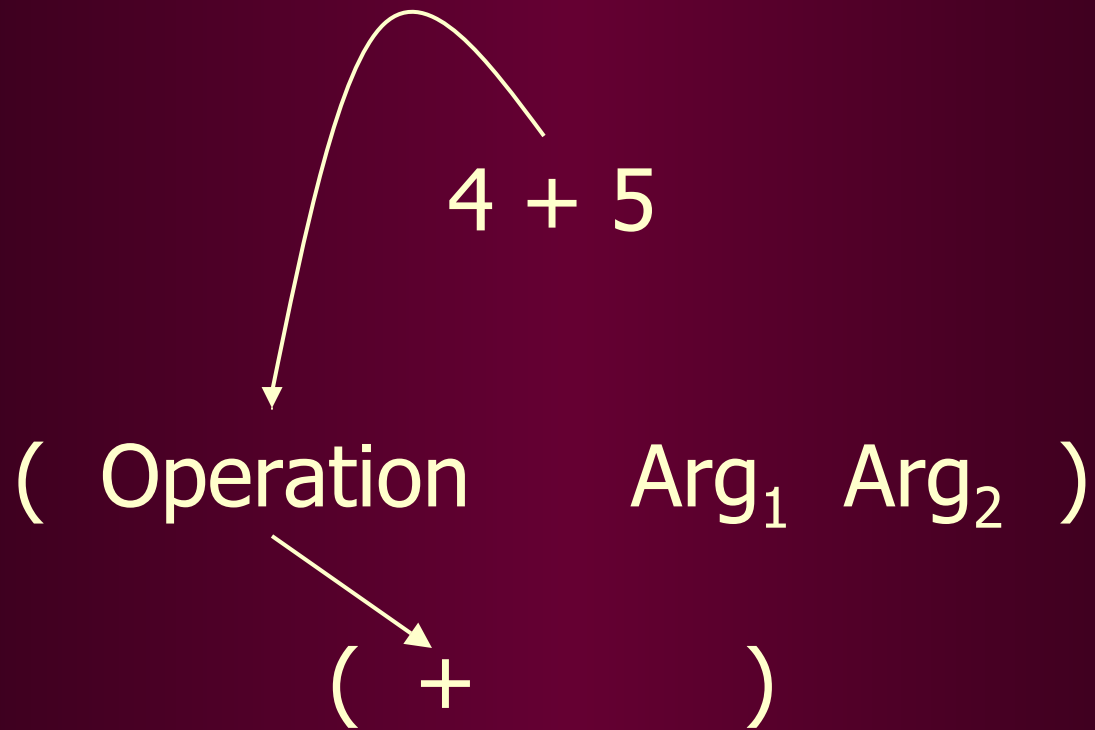
(Operation Arg₁ Arg₂)

Example #1 (cont'd)

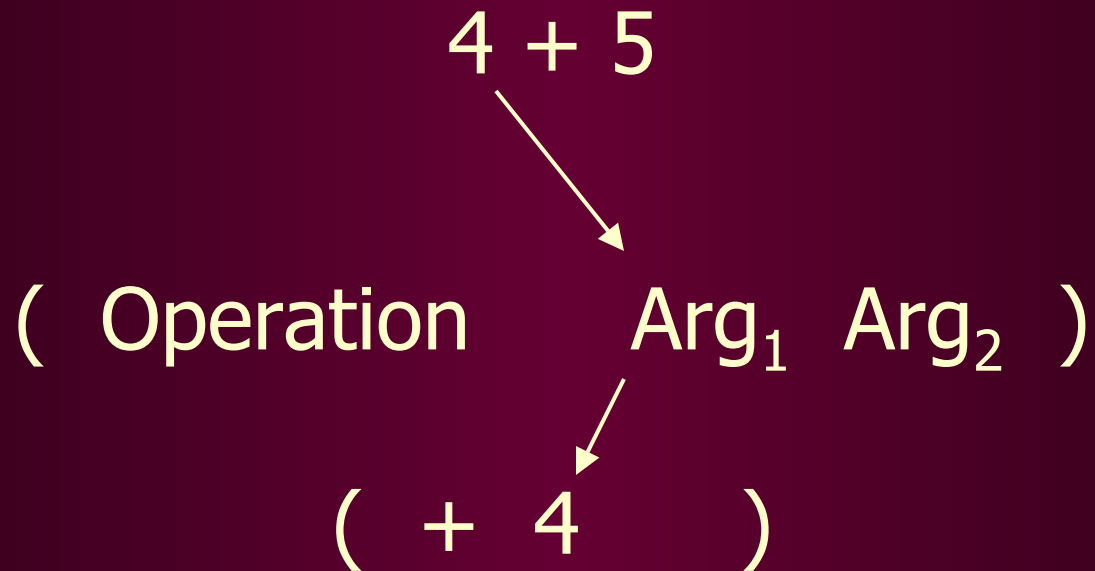
4 + 5



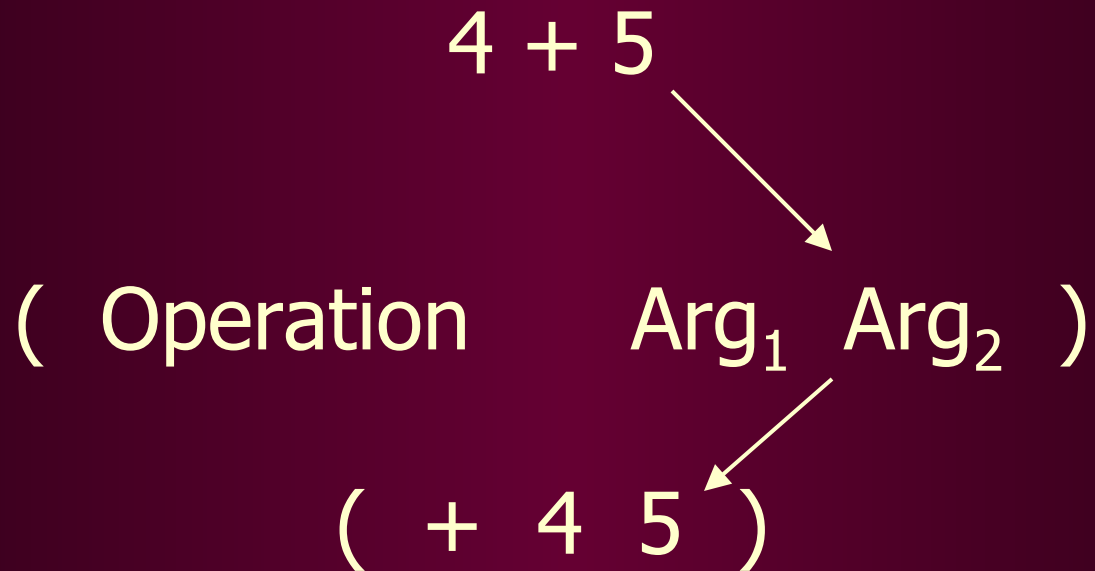
Example #1 (cont'd)



Example #1 (cont'd)



Example #1 (cont'd)



Example #1 (cont'd)

$$4 + 5$$

$$(+ 4 5)$$

Another Arithmetic Example

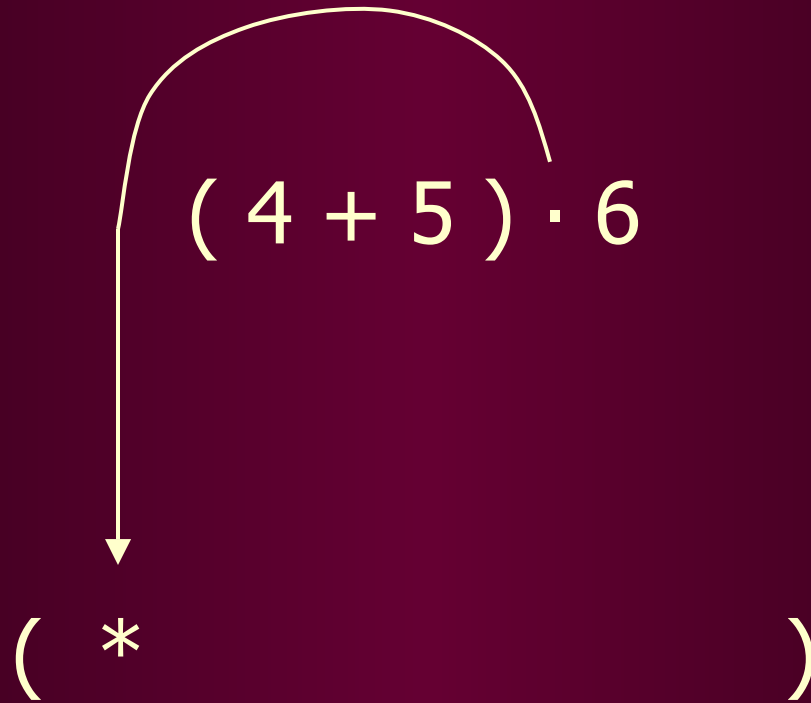
$$(4 + 5) \cdot 6$$

Example #2 (cont'd)

$$(4 + 5) \cdot 6$$

()

Example #2 (cont'd)



Example #2 (cont'd)

$$(4 + 5) \cdot 6$$



$$(* (+ 4 5) 6)$$

Example #2 (cont'd)

$$(4 + 5) \cdot 6$$



$$(* (+ 4 5) 6)$$

Example #2 (cont'd)

$$(4 + 5) \cdot 6$$

$$(\ * \ (\ + \ 4 \ 5 \) \ 6 \)$$

An Example From Algebra

$$4 + 5$$

$$f(x) = x + 5$$

Example #3 (cont'd)

$$f(x) = x + 5$$

(Operation Arg₁ Arg₂)

Example #3 (cont'd)

$$f(x) = x + 5$$



Example #3 (cont'd)

$$f(x) = x + 5$$

(Operation Arg₁ Arg₂)

(define

)

Example #3 (cont'd)

$$f(x) = x + 5$$

(Operation Arg₁ Arg₂)



(function-name input-name)



(f x)

Example #3 (cont'd)

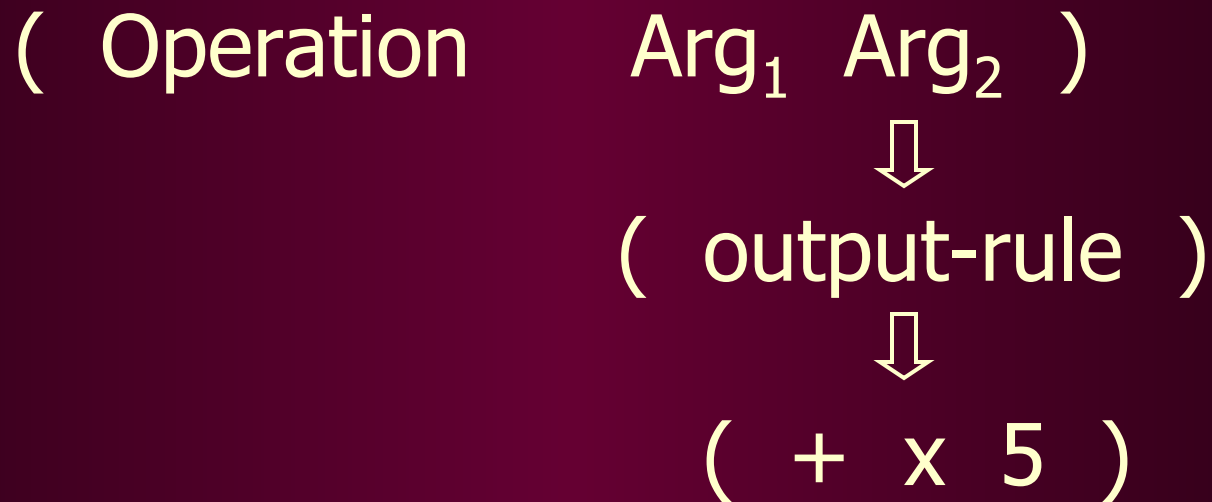
$$f(x) = x + 5$$

(Operation Arg₁ Arg₂)

(define (f x)
)

Example #3 (cont'd)

$$f(x) = x + 5$$



Example #3 (cont'd)

$$f(x) = \underbrace{x + 5}$$

(Operation Arg₁ Arg₂)

```
( define ( f x )  
  ( + x 5 ) )
```



Example #3 (cont'd)

$$f(x) = x + 5$$

```
( define ( f x )  
  ( + x 5 ) )
```


Algebra vs Scheme vs Pascal

Algebra

$$f(x) = x + 5$$

Scheme

```
( define ( f x )  
  ( + x 5 ) )
```

Pascal

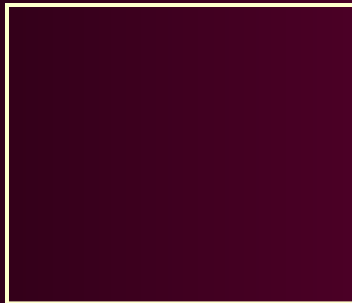
```
Program f (Input, Output) ;  
Var  
  x : Integer ;  
Begin  
  Readln ( x ) ;  
  Writeln ( x + 5 )  
End .
```

Design

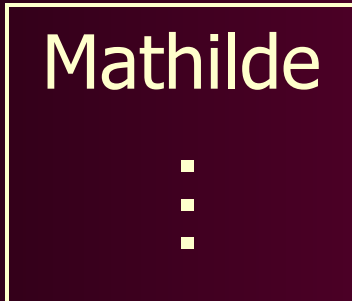
D³: Data Drive Design (A Non-Numeric Example)

Consider program *guest*, which determines whether a friend's *name* is in a party's invitation *list*.

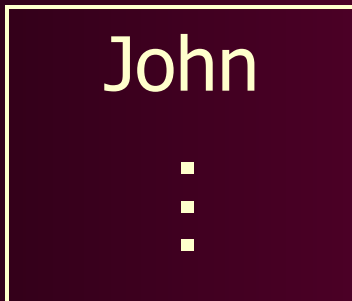
Is *Mathilde* In The List?



No

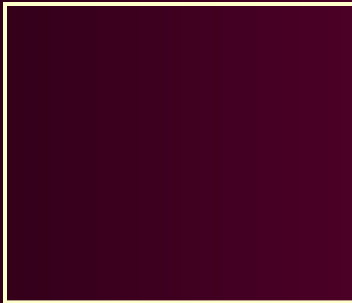


Yes

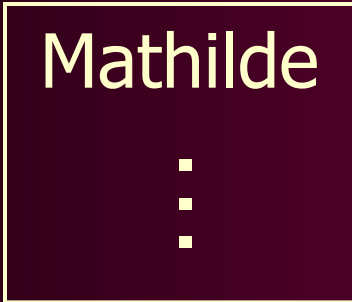


Look in the **Rest of the List**

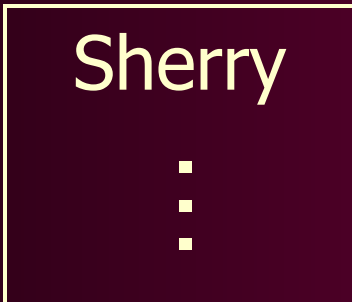
Is *Mathilde* In The Rest of the List?



No



Yes



Look in the **Rest of the List**

Pattern To Algebra

Algebra

quest(name, list) =

Algebra

quest(name, list) =

{

Algebra

quest(*name*, *list*) =

{

if *list* is empty

Algebra

$quest(name, list) =$

{

no

if $list$ is empty

Algebra

$quest(name, list) =$

{
no

if $list$ is empty

if $name = first(list)$

Algebra

$quest(name, list) =$

{

no

yes

if $list$ is empty

if $name = first(list)$

Algebra

$quest(name, list) =$

{

no

yes

if $list$ is empty

if $name = first(list)$

otherwise

Algebra

$quest(name, list) =$

{	no	if $list$ is empty
	yes	if $name = first(list)$
	$quest(name, rest(list))$	otherwise

Algebra

guest(*name*, *list*) =

Scheme

```
( define ( guest name list )
```

```
)
```

Algebra

guest(*name*, *list*) =

{

Scheme

(define (guest name list)

)

Algebra

guest(*name*, *list*) =

{

Scheme

```
( define ( guest name list )
```

```
  ( cond
```

```
    ) )
```

Algebra

$guest(name, list) =$

{

if $list$ is empty

if $name = first(list)$

otherwise

Scheme

```
( define ( guest name list )
```

```
  ( cond
```

```
    ))
```

Algebra

$guest(name, list) =$

{

if $list$ is empty

if $name = first(list)$

otherwise

Scheme

```
( define ( guest name list )
```

```
  ( cond
```

```
    ( ( = list ) #f )
```

```
    ( ( = name ( first list ) ) #t )
```

```
    ( #t #f ) ) )
```

Algebra

$guest(name, list) =$

{

if $list$ is empty

if $name = first(list)$

otherwise

Scheme

```
( define ( guest name list )
```

```
  ( cond
```

```
    ( ( empty? list )                                     )
```

```
    ( ( equal? name ( first list ) )                   )
```

```
    ( else                                             ) ) )
```

Algebra

$$\text{guest}(name, list) = \begin{cases} \text{no} & \text{if } list \text{ is empty} \\ \text{yes} & \text{if } name = \text{first}(list) \\ \text{guest}(name, \text{rest}(list)) & \text{otherwise} \end{cases}$$

Scheme

```
(define (guest name list)
  (cond
    ((empty? list) 'no)
    ((equal? name (first list)) 'yes)
    (else (guest name (rest list)))))
```

Did You Notice?

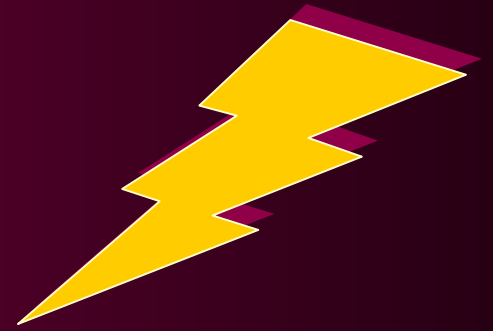
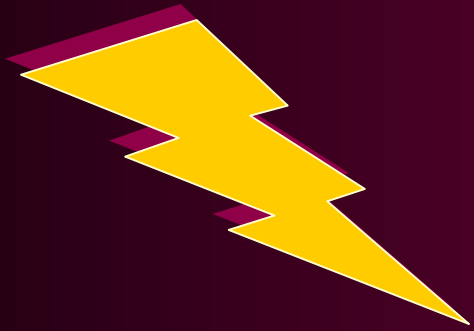
Algebra

$guest(name, list) =$

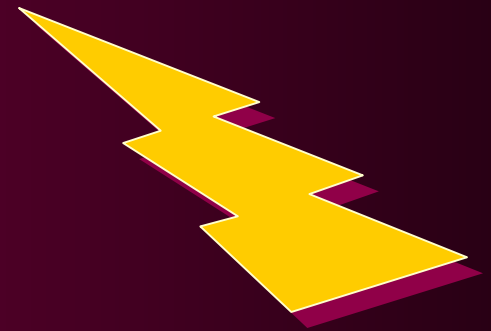
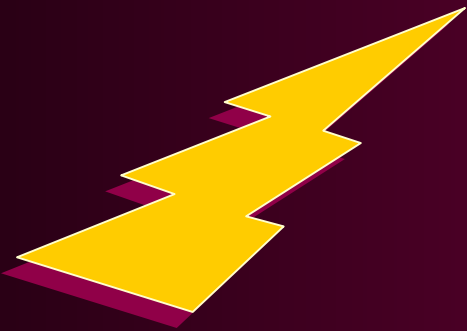
{	no	if <i>list</i> is empty
	yes	if <i>name</i> = first (<i>list</i>)
	$guest(name, rest (list))$	otherwise

Scheme

```
( define ( guest name list )  
  ( cond  
    ( ( empty? list ) 'no )  
    ( ( equal? name ( first list ) ) 'yes )  
    ( else ( guest name ( rest list ) ) ) ) )
```



Recursion Is Natural



Comparisons

Pascal

```
Program NameOnList (Input, Output) ;
Type
  ListType  = ^NodeType;
  NodeType  = Record
                First : String;
                Rest  : ListType
            End;
Var
  List      : ListType;
  Name     : String;
Procedure GetList (Var List: ListType); ...
Function Guest (Name : String; List : ListType) :
String;
Begin
  If List = nil
  Then Guest := 'no'
  Else If Name = List^.First
  Then Guest := 'yes'
  Else Guest := Guest ( Name, List^.Rest)
End;
Begin
  Readln ( Name );
  GetList ( List );
  Writeln (Guest ( Name, List ) )
End .
```

Pascal

Scheme

```
( define ( guest name list )  
  ( cond  
    ( ( empty? list ) 'no )  
    ( ( equal? name ( first list ) ) 'yes )  
    ( else ( guest name ( rest list ) ) ) ) )
```

```
Program NameOnList (Input, Output) ;
```

```
Type
```

```
ListType = ^NodeType;
```

```
NodeType = Record
```

```
First : String;
```

```
Rest : ListType
```

```
End;
```

```
Var
```

```
List : ListType;
```

```
Name : String;
```

```
Procedure GetList (Var List: ListType); ...
```

```
Function Member (Name : String; List : ListType) :  
String;
```

```
Begin
```

```
If List = nil
```

```
Then Member := 'no'
```

```
Else If Name = List^.First
```

```
Then Member := 'yes'
```

```
Else Member := Member ( Name, List^.Rest)
```

```
End;
```

```
Begin
```

```
Readln ( Name );
```

```
GetList ( List );
```

```
Writeln (Member ( Name, List ) )
```

```
End .
```

C or C++

Scheme

```
( define ( guest name list )  
  ( cond  
    ( ( empty? list ) 'no )  
    ( ( equal? name ( first list ) ) 'yes )  
    ( else ( guest name ( rest list ) ) ) ) )
```

```
#include <stdio.h>  
typedef struct listCell * list;  
struct listCell {  
    int first;  
    list rest;  
};  
bool guest (int x, list l) {  
    if (l == NULL)  
        return false;  
    else if (x == (l -> first))  
        return true;  
    else  
        return guest (x, l -> rest);  
}  
int main (int argc, char ** argv) {  
    list l1, l2, l3 = NULL; int x;  
    l1 = (list) malloc (sizeof (struct listCell));  
    l2 = (list) malloc (sizeof (struct listCell));  
    l2 -> first = 3; l2 -> rest = l3;  
    l1 -> first = 2; l1 -> rest = l2;  
    scanf ("%d", &x);  
    printf ("%d\n", member (x, l1));  
}
```

Principles of Program Design

- K.I.S.S.: Keep It Simple Syntactically
- D³: Data Drive Design
- Recursion Is Natural

The Ping-Pong Game

9th Graders With

- Algebra I
- 12 Weeks of Scheme

Curriculum Comparison

- introduction
- syntax
- Turbo Pascal, i/o
- numbers, strings
- simple arithmetic
- text files
- conditionals
- procedures, stubs
- algebra, functions
- conditionals
- design recipes
- symbols
- linked lists
- structures, records
- graphics
- lists containing lists

The Programming Environment

Salient DrScheme features:

- interactive evaluation
- immediate error-reporting with source highlighting
- language presented as a sequence of increasingly complex layers

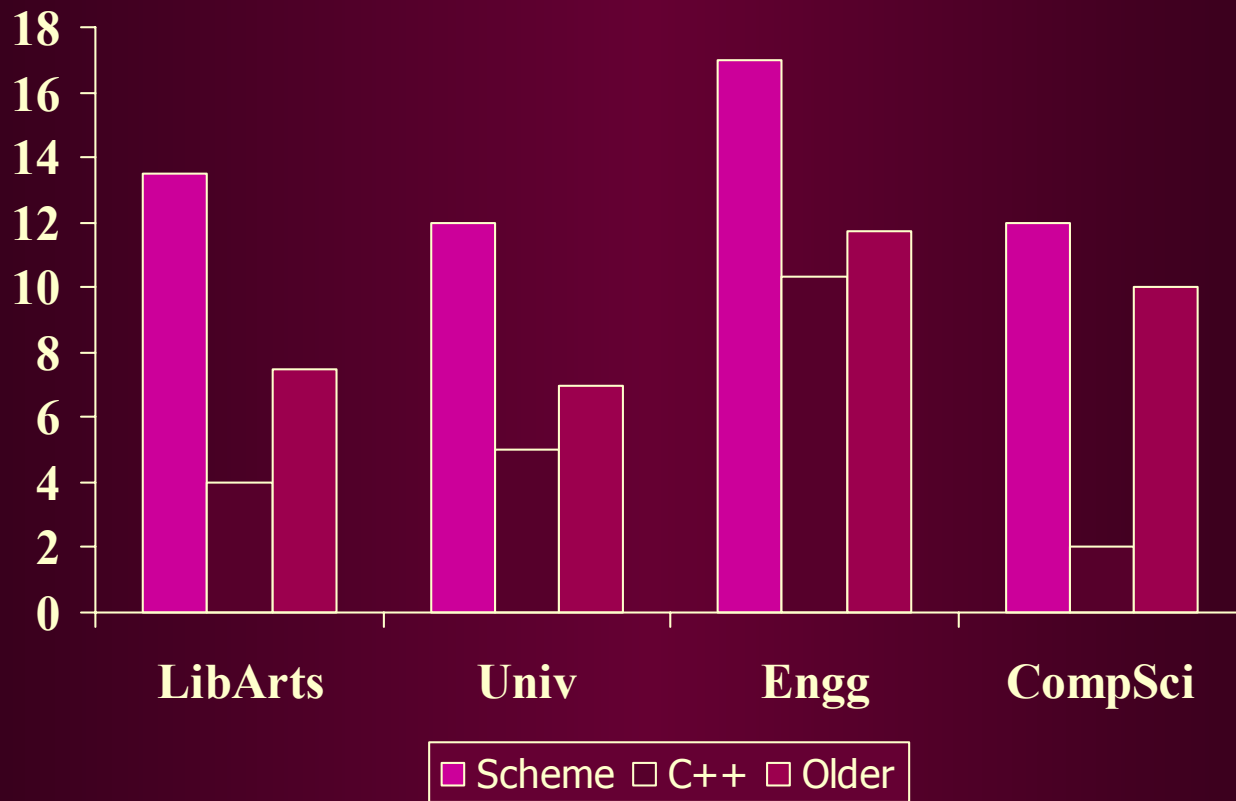
Putting it in Context

What a University Saw

Universities like Rice admit some of the best students in the nation; yet, the students cannot

- develop a program systematically
- separate problem solving from machine details
- explain why a program works (or doesn't)

What the ETS Wishes You Didn't Know (~1998)



Conclusion

- Computer science education is undergoing a revolution
- TeachScheme! is at the forefront
- Schools and universities must collaborate to reap the benefits

What We Offer

- Textbook (*How to Design Programs*)
- DrScheme programming environment
- Teacher's guide
- Programming environment guide
- Exercises and solution sets
- Miscellany: help, summer course, etc

All available for **free!**

Web Information

See

<http://www.teach-scheme.org/>

for information about the project,
especially the free summer courses