



Review of what has been covered so far

- ❑ An introduction to the notation of Miranda
- ❑ Some basic programming constructs:
 - Constant-valued programs
 - Conditional programs
 - Recursion
 - Composition
 - Relations and functions

Now ready to investigate some general CS concepts. This week: types and relational databases

Types

All things manipulated by a computer have a type. We use the notation $x ::$ to denote the type of x .

$3 :: \Rightarrow \text{num}$

$'a' :: \Rightarrow \text{char}$

$[1, 2, 3] :: \Rightarrow [\text{num}]$

$"abc" :: \Rightarrow [\text{char}]$

$(3, \text{true}) :: \Rightarrow (\text{num}, \text{bool})$

The Type of Operators

$+ :: \text{num} \rightarrow \text{num} \rightarrow \text{num}$

$* :: ?$

$(+ 1) :: ?$

Polymorphism

Some operators and programs can operate on different types of input. For example:

$++ :: \quad \Rightarrow \quad [*] \rightarrow [*] \rightarrow [*]$

$(:) :: \quad \Rightarrow \quad ?$

Polymorphism is a very important concept in CS

Program types: Constant-valued programs

Definitions:

`first_name = " Bill "`

`first_name :: ?`

`five = 2 + 3`

`five :: ?`

`pair_of_twos = (2, 2)`

`pair_of_twos :: ?`

`list_to_four = 1 : [2, 3, 4]`

`List_to_four :: ?`

Programs requiring input

Cube $x = x * x * x$

Cube :: ?

Add_squares $x\ y = (x * x) + (y * y)$

Add_squares :: ?

eg cube 5 \Rightarrow 125

add_squares 3 4 \Rightarrow 25

Conditional programs

Type of the output must always be the same

pick $x\ y\ z = x$, if $z \geq 0$
 $z = y$, if $z < 0$

pick :: ?

eg pick 3 5 2 => 3

Programs made with composition

$\text{sub_32 } n = n - 32$

$\text{sub_32} :: ?$

$\text{div_9 } n = n / 9$

$\text{div_9} :: ?$

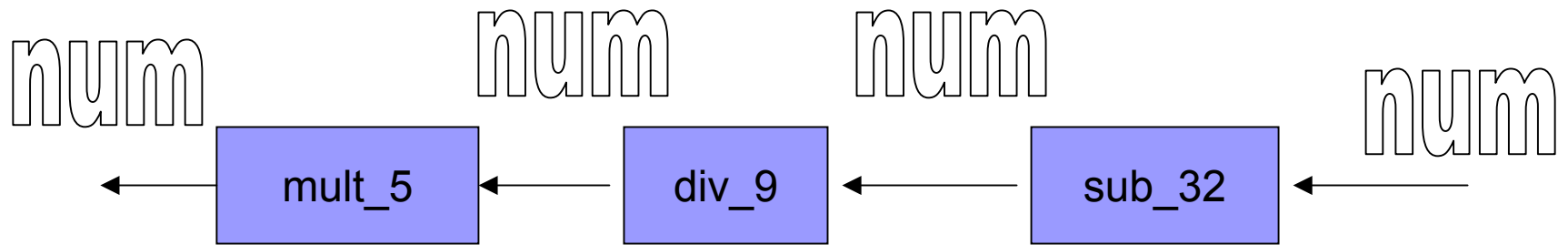
$\text{mult_5 } n = n * 5$

$\text{mult_5} :: ?$

$\text{f_to_cent} = \text{mult_5} . \text{div_9} . \text{Sub_32}$

$\text{f_to_cent} :: ?$

Use type info to check that parts fit together.



sumlist [] = 0

sumlist (e:es) = e + sumlist

prodlist [] = 1

prodlist (e:es) = e * prodlist

flatten [] = []

flatten (e:es) = e ++ flatten es

andlist [] = True

andlist (e:es) = e & andlist es

orlist [] = False

orlist (e:es) = e \/ orlist es

avglist [] = 0

avglist (e:es) = e \$AVG avglist es

where e1 \$AVG2 e2 = (e1+e2)/2

maxlist [e] = e

maxlist (e:es) = e \$MAX maxlist

where e1 \$MAX2 e2 = e1, if e1 >= e2
= e2, otherwise

foldr op id [] = id

foldr op id (e : es) = e \$op foldr op id es

sumlist = foldr (+) 0

prodlist = foldr (*) 1

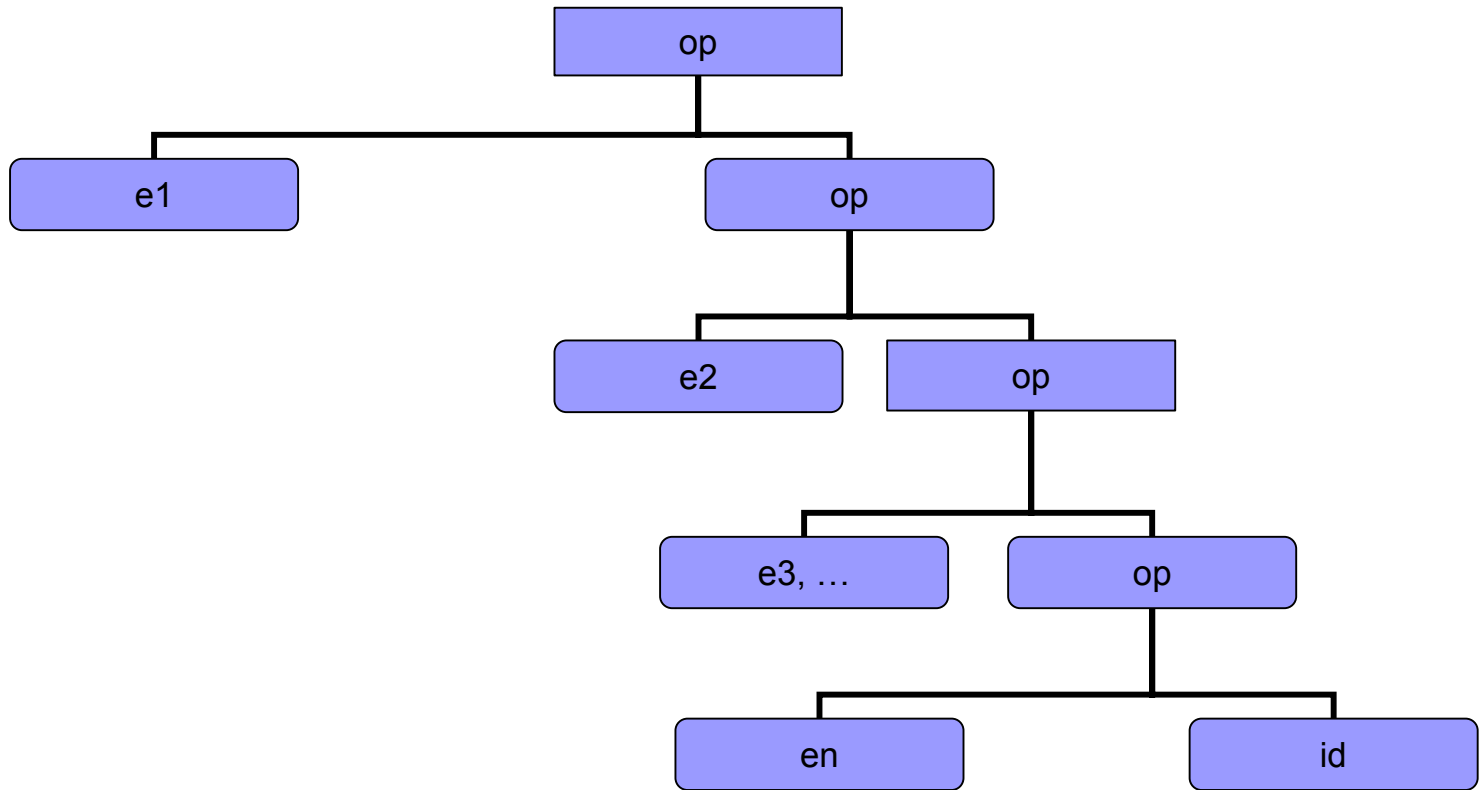
flatten = foldr (++) []

andlist = foldr (&) True

orlist = foldr (\ /) False

avglst = foldr (AVG2) 0

maxlist = foldr (MAX2) MINVALUE



Programs made with higher-order programs. Type depends on the operators etc.

sumlist = foldr (+) 0

sumlist :: ?

concat = foldr (++) []

concat :: ?

sum [1, 2, 3, 4] => 10

concat [[1, 2], [3, 4]] => [1, 2, 3, 4]

Another example

```
double_all = map ( * 2 )
```

```
double_all :: ?
```

```
double_all [ 1, 2, 3 ] => [ 2, 4, 6 ]
```

Working out the type of a program

What is the type of p:

$p \ x \ y \ z = 5, \text{ if } x = y$
 $\quad \quad \quad = z, \text{ otherwise}$

Look for clues

$p :: ? \rightarrow ? \rightarrow ? \rightarrow ?$

output is a num therefore, z must be a num
(output always same type)

type of x = type of y

$p :: * \rightarrow * \rightarrow \text{num} \rightarrow \text{num}$

A polymorphic example

What is the type of t ?

$$\begin{aligned} t \ x \ y \ z &= x ! 0, && \text{if } y = z \\ &= x ! 1, && \text{otherwise} \end{aligned}$$

$t :: ?$

Working out the type of a recursive program

What is the type of q:

$$q\ 0\ m = []$$
$$q\ n\ (x : xs) = n * x : q\ (n - 1)\ xs$$
$$q :: \Rightarrow ?$$

The type of map

Example use of map:

```
map (*2) [1, 2, 3] => [2, 4, 6]
```

What is the type of map?

```
map :: (num -> num) -> [ num ] -> [ num ]
```

Not quite. Consider the example:

```
map ( = [] ) [ [1, 2] , [3] , [ ] ]  
=> { Fale, False, True }
```

```
map :: ( [num] -> bool ) -> [num] -> [bool]
```

Not quite

The type of map-continued

$\text{map} :: (* \rightarrow **) \rightarrow [*] \rightarrow [**]$

eg

$\text{map } (+1) [2, 3, 4] \Rightarrow [3, 4, 5]$

Write polymorphic programs whenever possible.

RECURSIVE PROGRAMS ON NUMBERS

factorial 4 \Rightarrow 24 (ie $4 * 3 * 2 * 1$)

factorial 1 = 1

factorial n = n * factorial (n-1)

A Trace:

factorial 4 \Rightarrow 4 * factorial 3

\Rightarrow 4 * 3 * factorial 2

\Rightarrow 4 * 3 * 2 * factorial 1

\Rightarrow 4 * 3 * 2 * 1

\Rightarrow 24

EXPLANATION OF HOW WE OBTAINED THE PROGRAM

factorial 1 => 1

..

factorial 4 => 4 * 3 * 2 * 1

factorial 5 => 5 * 4 * 3 * 2 * 1

factorial 6 => 6 * 5 * 4 * 3 * 2 * 1

...

Recursive Pattern

factorial n = ?

Need base case:

TRACING RECURSIVE PROGRAMS

factorial 1 = 1

factorial n = n * factorial (n-1)

Trace by rewriting left to right using definition with substitution

factorial 4 => 4 * factorial 3

=> 4 * 3 * factorial 2

=> 4 * 3 * 2 * factorial 1

=> 4 * 3 * 2 * 1

=> 24

A conditional recursive program

Highest Common Factor

$$\text{hcf } 5 \ 5 \Rightarrow 5$$

$$\text{hcf } 12 \ 20 \Rightarrow 4$$

$$\text{hcf } 24 \ 36 \Rightarrow 12$$

Definition

$$\text{hcf } x \ y = ? \quad , \text{ if } ?$$

$$\text{hcf } x \ y = \dots x \dots y \dots \text{hcf} \dots , \text{ if } ?$$

$$\text{hcf } x \ y = \dots x \dots y \dots \text{hcf} \dots , \text{ if } ?$$

You need to think about the recurring patterns

$$\text{hcf } x \ x = x$$

$$\text{hcf } x \ y = \text{hcf } x \ (y-x), \text{ if } y > x$$

$$\text{hcf } x \ y = \text{hcf } (x-y) \ y, \text{ otherwise}$$

$$\begin{aligned} \text{hcf } 32 \ 6 &= \text{hcf } 26 \ 6 \\ &= \text{hcf } 20 \ 6 \\ &= \text{hcf } 14 \ 6 \\ &= \text{hcf } 8 \ 6 \\ &= \text{hcf } 2 \ 6 \\ &= \text{hcf } 2 \ 4 \\ &= \text{hcf } 2 \ 2 \\ &= 2 \end{aligned}$$

GENERAL FORMAT FOR SIMPLE RECURSIVE PROGRAMS ON NUMBERS

rprog base_case = ?

rprog n = ...n...rprog (n-1)

Example

sum_to_n 5 => 15

sum_to_n 6 => 21

Create definition

(by plugging in to the format)

sum_to_n ? = ?

sum_to_n n = ...n ...sum_to_n (n-1)

GENERAL FORMAT FOR SIMPLE RECURSIVE PROGRAMS ON NUMBERS

rprog base_case = ?

rprog (a:as) = ...a...rprog as

Example

last [4, 3, 5] => 5

last base_case = x

last (a:as) = ...a...last as

Trace

last [1, 2, 3] => last [2, 3]

=> last [3]

=> 3

Recursive reverse

rever [1, 2, 3] => [3, 2, 1]

rever [23, 67, 5] => [5, 67, 23]

rever ? = ?

rever (a:as) = ...a...rever as

Answer

This is very inefficient (see later in course)

front [1, 2, 3, 4] => [1, 2, 3]

front [1, 2] => [1]

front [1] => []

front [] = Undefined

front [e] = []

front (e:es) = es : front es

tail (e:es) = es

middle = front . Tail

middle [1, 2, 3] = [2]

middle [1, 2, 3, 4] = [2, 3]

middle [1, 2, 3, 4, 5] = [2, 3, 4]

THE PALINDROME PROBLEM

pal [1, 2, 3, 2, 1] => True

pal [34, 2, 34] => True

pal [8] => True

The palindrome program

pal [] = True

pal [x] = True

pal as = True, if ((head as) = (last as))

&

(pal (middle as) = True))

RECURSIVE INSERT_SORT

insert 2 [1, 3, 4] => [1, 2, 3, 4]

insert x [] = ?

insert x (a:as) = ...x...insert x as, if ?

= ...x...insert x as, otherwise

insert_sort [3, 2, 5, 8, 1] => [1, 2, 3, 5, 8]

insert_sort [] = ?

insert_sort (a:as) = ...a... insert_sort as

QUICKSORT

(The fastest sorting algorithm)

```
qsort [] = []
```

```
qsort (a:as) = qsort [e | e <- as; e < a]
```

```
  ++ [a] ++
```

```
  qsort [e | e <- as; e >= a]
```

`bin2decimal [] = 0`

`bin2decimal (b:bs) = b * 2^(#bs) + bin2decimal bs`

`bin2decimal seq = xbin2decimal seq (#seq)`

`xbin2decimal [] c = c`

`xbin2decimal (b:bs) = b * 2^c + xbin2decimal bs
(c-1)`

`reverse [] = []`

`reverse (e:es) = reverse es ++ [e]`

`reverse s = reverse2 s []`

`reverse2 [] c = c`

`reverse2 (e:es) c = reverse es (e:c)`

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	T	F

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

A	B	$A \rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

A	$\neg B$
T	F
F	T


$$(A \wedge B) \equiv (B \wedge A)$$

$$(A \sqcap B) \equiv (B \sqcap A)$$

$$(A \rightarrow B) \not\equiv (B \rightarrow A)$$

$$(A \rightarrow B) \equiv (\neg B \sqcup A)$$