

The Hugs System

- Hugs is an implementation of Haskell 98, and is the most widely used Haskell system;
- The interactive nature of Hugs makes it well suited for teaching and prototyping purposes;
- Hugs is available on the web from:

www.haskell.org/hugs

Starting Hugs

On a Unix system, Hugs can be started from the % prompt by simply typing hugs:

```
% hugs
```

```
  _  _  _  _  _  _  
||  ||  ||  ||  ||  || | | | |
||__||  ||__||  ||__||  ||__||  
||---||      ||__||  
||  ||  
||  ||
```

```
Hugs 98: Based on the Haskell 98 standard  
Copyright (c) 1994-2005  
World Wide Web: http://haskell.org/hugs  
Report bugs to: hugs-bugs@haskell.org
```

```
>
```

The Hugs > prompt means that the Hugs system is ready to evaluate an expression.

For example:

```
> 2+3*4  
14
```

```
> (2+3)*4  
20
```

```
> sqrt (3^2 + 4^2)  
5.0
```

The Standard Prelude

The library file Prelude.hs provides a large number of standard functions. In addition to the familiar numeric functions such as `+` and `*`, the library also provides many useful functions on lists.

- Select the first element of a list:

```
> head [1,2,3,4,5]  
1
```

- Remove the first element from a list:

```
> tail [1,2,3,4,5]  
[2,3,4,5]
```

- Select the nth element of a list:

```
> [1,2,3,4,5] !! 2  
3
```

- Select the first n elements of a list:

```
> take 3 [1,2,3,4,5]  
[1,2,3]
```

- Remove the first n elements from a list:

```
> drop 3 [1,2,3,4,5]  
[4,5]
```

- Calculate the length of a list:

```
> length [1,2,3,4,5]  
5
```

- Calculate the sum of a list of numbers:

```
> sum [1,2,3,4,5]  
15
```

- Calculate the product of a list of numbers:

```
> product [1,2,3,4,5]  
120
```

- Append two lists:

```
> [1,2,3] ++ [4,5]  
[1,2,3,4,5]
```

- Reverse a list:

```
> reverse [1,2,3,4,5]  
[5,4,3,2,1]
```

Function Application

In mathematics, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space.

$$f(a,b) + c d$$



Apply the function f to a and b , and add the result to the product of c and d .

In Haskell, function application is denoted using space, and multiplication is denoted using `*`.

```
f a b + c*d
```



As previously, but in Haskell syntax.

Moreover, function application is assumed to have higher priority than all other operators.

$f\ a + b$

Means $(f\ a) + b$, rather than $f\ (a + b)$.

Examples

Mathematics

$f(x)$

$f(x, y)$

$f(g(x))$

$f(x, g(y))$

$f(x)g(y)$

Haskell

$f\ x$

$f\ x\ y$

$f\ (g\ x)$

$f\ x\ (g\ y)$

$f\ x\ * \ g\ y$

Haskell Scripts

- As well as the functions in the standard prelude, you can also define your own functions;
- New functions are defined within a script, a text file comprising a sequence of definitions;
- By convention, Haskell scripts usually have a .hs suffix on their filename. This is not mandatory, but is useful for identification purposes.

My First Script

When developing a Haskell script, it is useful to keep two windows open, one running an editor for the script, and the other running Hugs.

Start an editor, type in the following two function definitions, and save the script as test.hs:

```
double x      = x + x
```

```
quadruple x = double (double x)
```

Leaving the editor open, in another window start up Hugs with the new script:

```
% hugs test.hs
```

Now both Prelude.hs and test.hs are loaded, and functions from both scripts can be used:

```
> quadruple 10  
40
```

```
> take (double 2) [1,2,3,4,5,6]  
[1,2,3,4]
```

Leaving Hugs open, return to the editor, add the following two definitions, and resave:

```
factorial n = product [1..n]  
average ns = sum ns `div` length ns
```

Note:

- `div` is enclosed in back quotes, not forward;
- `x `f` y` is just syntactic sugar for `f x y`.

Hugs does not automatically detect that the script has been changed, so a reload command must be executed before the new definitions can be used:

```
> :reload  
Reading file "test.hs"  
  
> factorial 10  
3628800  
  
> average [1,2,3,4,5]  
3
```


Naming Requirements

- Function and argument names must begin with a lower-case letter. For example:

myFun

fun1

arg_2

x'

- By convention, list arguments usually have an s suffix on their name. For example:

xs

ns

nss

The Layout Rule

In a sequence of definitions, each definition must begin in precisely the same column:

```
a = 10  
b = 20  
c = 30
```



```
a = 10  
  b = 20  
c = 30
```

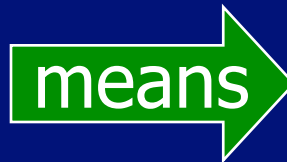


```
  a = 10  
b = 20  
  c = 30
```



The layout rule avoids the need for explicit syntax to indicate the grouping of definitions.

```
a = b + c
  where
    b = 1
    c = 2
d = a * 2
```



```
a = b + c
  where
    {b = 1;
     c = 2}
d = a * 2
```

implicit grouping

explicit grouping

Useful Hugs Commands

<u>Command</u>	<u>Meaning</u>
:load <i>name</i>	load script <i>name</i>
:reload	reload current script
:edit <i>name</i>	edit script <i>name</i>
:edit	edit current script
:type <i>expr</i>	show type of <i>expr</i>
:?	show all commands
:quit	quit Hugs

Exercises

- (1) Try out slides 2-8 and 14-17 using Hugs.
- (2) Fix the syntax errors in the program below, and test your solution using Hugs.

```
N = a ' div' length xs
  where
    a = 10
    xs = [1,2,3,4,5]
```

- (3) Show how the library function last that selects the last element of a list can be defined using the functions introduced in this lecture.
- (4) Can you think of another possible definition?
- (5) Similarly, show how the library function init that removes the last element from a list can be defined in two different ways.