Appendix A

The Initial Miranda Environment

This appendix provides information on three topics:

1. What names are allowable as identifiers.
2. Those names that are reserved and predefined.
3. The use of functions and type constructors in infix form.

A.1 Identifiers

An *Identifier* is a sequence of alphanumeric characters: including letters (A–Z, a–z), digits (0–9), underscores (_) or single quotes (') but *starting with a letter.* If the starting letter is *lower case* then the Identifier is used to name constants, functions and types (and is known as an *identifier*). If the starting letter is *Upper case* then the Identifier can only name a constructor (and is known as an *IDENTIFIER).*

A.2 Reserved and predefined names

Reserved names

The following names are reserved for use by the Miranda system and cannot be used as identifiers. They cannot be the names of formal parameters and they cannot be redefined within **where** blocks.

abstype div if mod otherwise
readvals show type where with

Predefined names

The following identifiers are predefined, and thus always in scope. They are available at the start of all Miranda sessions and constitute the *standard environment* of Miranda (release 2). For details of their functionality and possible implementation the reader is referred to the Miranda On-line Manual (Research Software, 1990).

In contrast to reserved names, these identifiers may be the names of formal parameters and may be redefined within **where** blocks. However, this practice is *not* recommended.

Predefined typenames

```
bool char num sys_message
```

Predefined constructors

```
False, True ::  bool

Appendfile, Closefile, Exit,
Stderr, Stdout, System, Tofile ::  sys_message
```

The undefined value

`undef` names the completely undefined value. Any attempt to access it results in an error message. Note that `undef` belongs to every type. It may be defined as:

```
undef ::  *
undef = error "undefined"
```

Predefined functions

```
abs and arctan cjustify code concat const converse cos decode
digit drop dropwhile e entier error exp filemode filter foldl
foldl1 foldr foldr1 force fst getenv hd hugenum id index init
integer iterate last lay layn letter limit lines ljustify log
log10 map map2 max max2 member merge min min2 mkset neg numval
or pi postfix product read rep repeat reverse rjustify scan
seq showfloat shownum showscaled sin snd sort spaces sqrt
subtract sum system take takewhile tinynum tl transpose until
zip2 zip3 zip4 zip5 zip6 zip
```

A.3 Functions as operators

The Miranda **$** token is the complement of the Miranda *section* facility, in that it is possible to use functions or algebraic type constructors in an *infix* manner. For example, given the prefix function `implies`, which corresponds to *logical implication*, then it may be used as an *infix* operator by preceding it with a **$**:

```
implies ::  bool -> bool -> bool

implies True False = False
implies any1 any2 = True
```

```
 Miranda False $implies False = False
True
```

Notice that it is *not* possible to use the **$** token to create an infix function or constructor:

```
not_infix_implies = $implies
```

```
Miranda True not_infix_implies False
type error in expression
cannot apply bool to bool->bool->bool
```