

## Compilation Example

Translating a C-like language to assembly language

```
int i;

void main () {
    for (i = 1; i <= 100; i++)
        fred(i);
}
```

{

```
    i:      ...
           data word 0
    main:   move 1 to i
    L1:     compare i with 100
           jump to L2 if greater
           push i
           call fred
           add 1 to i
           goto L1
    L2:     return
    fred:   ...
}
```

## Compilation Example

- Given the previous program, how does the compiler know how to interpret the various names in the program, such as `int`, `i`, `void`, `main`, *etc.*
- Names are bound to properties (characteristics the compiler can use to interpret the name) at various times during the life-time of a program.

## Compilation Example

- Language-definition Time
  - void, for
  - or any other keyword
- Language-implementation Time
  - int → 16, 32, 64 bit entity
  - anything else that is hardware specific
- Compile Time
  - the type of i is bound at compile time
- Link Time
  - the name fred is bound to a function at link time
- Load Time
  - the names main, fred, and i are determined at load time
- Runtime
  - the values stored in i are determined at runtime


## Where are we?

- Read chaps 4 & 5
- Homework #3
  - 4.4
  - due Friday 2/10
- Quiz
  - Wednesday 2/15
  - Chapters 1-4
  - Closed book/notes
- No Class Monday 2/13!

# ML

- ML is a functional programming language
- the ML environment runs in an interactive mode

```
C:\> sml
Standard ML of New Jersey v110.49 [FLINT
v1.5], September 13, 2004
```

 ← ML System Prompt

- At the prompt the system expects a sentence in ML

## ML – Constant Expressions

The simplest sentence in the ML language is a constant expression

```
Standard ML... Sentence Delimiter
- 1234;
val it = 1234 : int
- ↑           |           |
  Internal   Value      Type of
 Variable   the Value  the Value
```

### Other Constants:

real	123.4
bool	true/false
string	"Susan"
char	#"Q"

# ML – Operator and Simple Expressions

Example:

```
- ~ 1 + 2 - 3;  
val it = ~2 : int
```

~ is the unary -,  
here -2.

```
- (4 > 0) andalso (4 mod 2 = 0);  
val it = true : bool
```

Precedence	Operators
High	not ~
	* / div mod
	+ - <b>^</b>
	< > <= >= = <>
	andalso (logical and)
Low	orelse (logical or)

string concatenation;  
"abc" ^ "def"

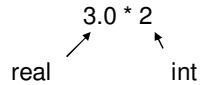
# ML – Conditional Expressions

if – then – else  
or  
if – then

```
-if 1 < 2 then #"x" else #"y";  
val it = #"x" : char
```

# ML – Type Conversions

Most programming languages we are used to allow for mixed-type expressions such as



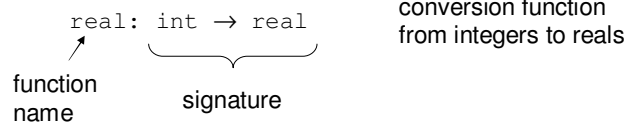
ML does not allow mixed-type expressions.

```
- 3.0 * 2;  
Error: operator and operand don't agree
```

# ML – Type Conversions

However, we can use type conversions to manipulate the types of an expression.

Example:



Other conversion functions:

```
floor: real → int (round down)  
ceil: real → int (round up)  
round: real → int (round to nearest int)
```

## ML – Type Conversions

We can now rewrite our illegal expression from before:

```
- 3.0 * real(2);
```

Convert 2 into 2.0

```
val it = 6.0 : real
```

or

```
- floor(3.0) * 2;  
val it = 6 : int
```

## ML – Variable Definitions

Very simple syntax:

```
- val x = 1 + 2 * floor(3.0);  
val x = 7 : int
```

No longer the  
default variable

In ML you do not need  
to declare the type of a variable;  
ML will determine its type through  
type inference.

Of course we can use the values of variables:

```
- x + 1;  
val it = 8 : int
```