Compiler Construction

The Tiger Language

The Tiger Language

- Toy Language ... but still effective
- Imperative Language: descendant of Algol and Pascal
- Functional flavor, nested functions
- Well defined: simple and sound grammar

Can be easily extended with Objects!
Or Overloading!
Or ...

Core of the language

- **C-style Comments**: $/*$ and $*/$
- **Identifier**: sequence of letters, digits and underscore starting with a letter
- **Builtins**: *int* and *string*
- **User defined types**: Records, arrays, recursive types
- **Nested scopes** with function declaration
- **Minimal standard library**

First Programs (1/3)

```
print("Hello World!\n")
```

```
let
  function cdown(i: int) : int =
      if i = 0 then 0
      else cdown(i-1)
in
  cdown(3*3+1)
end
```

```
let
  function hello(name: string) =
      print(concat("Hello", name))
in
  hello("you!")
end
```

First Programs (2/3)

```
let
   var myvar := 42
in
   print_int(myvar)
end
```

```
let function ten(): int =
  (print("Once.\n"); 10)
in
  for i := 0 to ten() do
    print_int(i)
end
```

```
let var useless := 0
in
  for i := 1 to 10 do break
end
```

First Programs (3/3)

```
let
  type intArray = array of int
  var row := intArray[8] of 0
in
  print_int(row[0])
end
```

```
let
  type rec = { a : int }
  var b :=
      if 0 then nil
      else rec { a = 1 }
in
  print_int(b.a)
end
```

Features many extensions

```
let
  import "myimport.tih"
in
  1
end
```

```
let
  class B
  {
    var a := 42
    method m() : int = self.a
  }
  var b := new B
in
  b.a := 51
end
```

Illegal

```
let
  var a := nil
in
  /* ... */
end
```

```
if nil = nil then /* ... */
```

nil must be used in a context where the type can be determined

Remarks (1/3)

Fails to typecheck

```
(a := b) + c
```

Comparison operator

*, /, +, - are left associative while = does not associate, so you should write:

```
a = (b = c)
```

Remarks (2/3)

True and False

Since there is no boolean type, 0 is used to denote false (c-style)

Array creation

The expression **typeid** $[e_1]$ **of** e_2 evaluates first e_1 then e_2

Remarks (3/3)

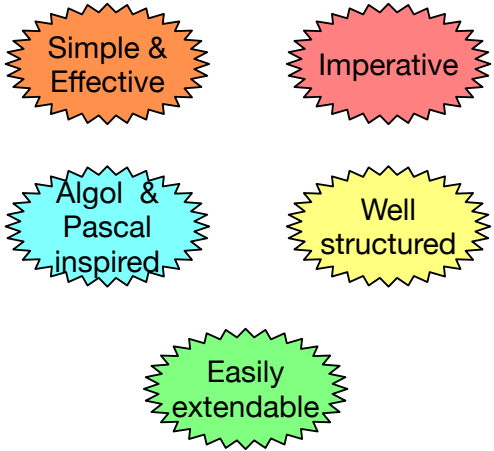
Namespaces

There are different namespaces: *a* can both refer a type and a variable or a function.

Breaks

A **break** in procedure *p* cannot terminate a loop in procedure *q* even if *p* is nested within *q*

Summary


Simple &
Effective
Imperative
Algol &
Pascal
inspired
Well
structured
Easily
extendable