Lecture 2:

- Quick demo of MyPL
- MyPL part 2 (constructs)

Announcements:

- HW-0 out (see course webpage and piazza for instructions)
- No Class Monday: MLK Jr Holiday (also see course webpage)

Intro to MyPL

MyPL Basics: *... a simple PL for learning about PL implementation*

(1) programs as a single-file set of functions (required `main`)

(2) variables and simple arithmetic ops

(3) if, while, and (simple) for statements

(4) structs and 1-D arrays ... as "objects" allocated on the heap

(5) functions use pass-by-value, objects passed as "references" (via object ids)

(6) small set of built-in functions ... I/O, type cohersion, arrays

(7) explict type declarations ... but some type inference for variables

(8) static typing ... type errors detected "statically" (before runtime)

Note: we are using Java 21+ (with Maven) to implement MyPL this semester

MyPL Overview

1. Comments:

```
# start with a '#' sign
# comments are single-line
```

2. Primitive Data Types:

```
int       # 32 bit integer
double    # 64 bit (double precision) floating point
bool      # either true or false (not 0, 1)
string    # characters wrapped in "..." (double quotes)
void      # can only be used as a function return type
```

3. Values:

```
0, 1, 7, 10, 20, 876132      # int values
1.0, 1.01, 10.3, 0.505       # double values
true, false                  # bool values
"foo", "bar", ""             # string values
null                         # legal value (any type)
```


MyPL Overview

4. Variable declarations:

```
var x1: int         # declare x1 as int, initialized to null
var x2: int = 5     # declare x2 as int, initialized to 5
var x3 = 42         # declare x3, initialized to 42, inferred as int
var x4 = (5*4)/2    # declare x4, initialized to 10, inferred as int

var x5: double = 3.14  # double (explicit type)
var x6: bool = true    # bool (explicit type)
var x7 = false         # bool (implicit type)

var x8 = x3 + x5       # type error
var x9 = x1 + x2       # runtime error: null + 5
var x10 = null         # type error (can't infer type)

var x11: string        # string set to null (explicit type)
var x12 = ""           # empty string (implicit type)
```

MyPL Overview

5. Assignment:

```
w = 10              # valid if w: int
x = 3.1 * 4.2       # valid if x: double
y = false           # valid if y: bool
z = null            # works for any type
```

Notes on `null`:

- for initialization, assignment, comparison (`==`, `!=`)
- other uses result in runtime errors (`x < null`, `5 + null`, etc.)

6. Relational Comparators, Boolean Operators

- normal six: >, <, >=, <=, ==, !=
- normal three: `and`, `or`, `not`
- parentheses also allowed, e.g., `((x <= y) and (y > z)) or (y < 0)`

MyPL Overview

7. For Loops: *simple range-style iteration*

```
for i from 0 to 9 {           # i = 0, i = 1, ..., i = 9
  println(i)                  # println is a built in function
}

for j from 10 to 1 {          # nothing is printed!
  print(str_val(j) + ", ")    # casts j, + also for strings
}
```

8. While Loops: *just the normal thing*

```
while i > 0 and i < 10 {      # note: parens not necessary!
  println(i)
}
```

MyPL Overview

9. If Statements: *also just the normal thing*

```
if i == 0 or i == 10 {     # parens not necessary!
  ...
}
else if i < 5 {
  ...
}
else if i < 10 {
  ...
}
else {
  ...
}
```


MyPL Overview

10. Arithmetic Expressions:

```
x + y      # int/double addition, string concatenation
x - y      # int/double subtraction
x * y      # int/double multiplication
x / y      # int/double division
```

- can be parenthesized, chained together, etc.
- no automatic type coercion (no type mixing!)

11. Functions:

Take the "normal" form: *return-type* function_name(*params*) {...}

```
int add(x: int, y: int) {
  if x < 0 or y < 0 {
    return 0
  }
  return x + y
}
```

MyPL Overview

MyPL supports recursion:

```
int fib(n: int) {
  if n < 0 {
    return null
  }
  if n == 0 or n == 1 {
    return n
  }
  return fib(n-2) + fib(n-1)
}
```

Each MyPL program (file) must have a `main` function:

```
void main() {
  ...
}
```


MyPL Overview

12. Structs: a collection of variables

- assigned a (hidden) object id on instantiation
- when created, must provide initial values ("constructor")

```
struct node {
  value: int,
  next: node
}

void main() {
  var n1: node = new node(10, null)  # explicit type
  var n2 = new node(20, n1)          # implicit type
  var n3 = new node(30, null)        # same
  n3.next = n2                       # set field
  println(n3 == n2)                  # false (diff oids)
  var n4 = n3.next.next              # field access
  print(n4.value)                    # prints 10
}
```

MyPL Overview

13. Arrays: fixed size list of same-typed values

- also assigned a (hidden) object id on instantiation
- when created, must provide size

```
void main() {
  var xs: [int] = new int[5]     # int array of 5 null values
  var ys = new string[3]         # string array of 3 nulls
  var zs: [bool]                 # zs is null

  # initialize xs
  for i from 1 to size(xs) {
    xs[i-1] = i                  # 1, 2, 3, 4, 5
  }

  # print xs
  for i from 0 to size(xs) - 1 {
    var x = xs[i]
    println(x)
  }
}
```


MyPL Overview

Function to create and initialize an int array

```
[int] init(length: int, value: int) {
  var xs = new int[length]
  for i from 0 to length - 1 {
    xs[i] = value
  }
  return xs
}
```

Function that "safely" returns an array value

```
int get(xs: [int], index: int, or_val: int) {
  if index >= 0 and index < size(xs) {
    return xs[index]
  }
  return or_val
}
```

MyPL Overview

14. Built-in functions for type coercion:

```
int_val(3.14)        # returns integer 3
int_val("42")        # returns integer 42

dbl_val(13)          # returns double 13.0
dbl_val("2.7")       # returns double 2.7

str_val(24)          # returns string "24"
str_val(0.58)        # returns string "0.58"
```

15. Additional built-in functions:

```
print("Hello World")     # std output, no newline
println("Hi!")           # std output, adds newline
readln()                 # returns line from std input
size("foo")              # number of characters
size(xs)                 # number of array elems
get(i, "foo")            # i-th character in string
```

Note: `print` and `println` can display any base type value along with null

MyPL Overview

16. MyPL Programs

- only single-file programs (e.g., in a text file with `.mypl` extension)
- a MyPL file consists of struct and function definitions
- normal scoping rules (more later)
- must have a `void main()` function (called when program is run)

More details as we go ...

- so far, only an informal definition of the language
- we'll formalize it further as we implement it

See companion "study guide" for examples, etc.