Lecture 3:

- Wrap up overview of MyPL
- Compilation and Interpretation

Homework:

- HW-0 due
- HW-1 out

13. **Arrays**: fixed size, indexed collection of values of the same type

- dynamically allocated (on the heap)
- the expression "**new** $t[n]$" creates an array with n null values

```
array int xs;                          // xs is null, type is int array
array int ys = new int[10];            // ys is 10 element int array
ys[2] = 42;                            // assign third element
int y = ys[2];                         // get third element

array Node ys = new Node[y];           // array of linked-list nodes

// function that returns an array
array int init(int length, int value) {
  array int tmp = new int[length]
  for (int i = 0; i < length; i = i + 1) {
    tmp[i] = value;
  }
  return tmp;
}

// function that takes an array
int get(array int xs, int i, int or_value) {
  if (i >= 0 and i < length(xs)) {
    return xs[i];
  }
  else {
    return or_value;
  }
}
```

Note on deallocation: no garbage collection or delete (could be a final project)

14. Automatic type conversion not supported

- for example, the following result in **type errors**:

```
int x = "4";       // type error (expected int, found string)
bool y = 1;        // type error (expected bool, found int)
double z = 3;      // type error (expected double, found int)
```

15. Built-in functions for type conversion

```
int x1 = dtoi(3.14);         // double to int
int x2 = stoi("4");          // string to int

double x3 = itod(4);         // int to double
double x4 = stod("3.14");    // string to double

string x5 = itos(4);         // int to string
string x6 = dtos(3.14);      // double to string
```

- `stoi()`, `stod()` can result in runtime errors

16. Additional built-in functions

```
print("Hello World\n");       // print any base type value
string y1 = input();          // read string from standard in
int y2 = length("foo");       // string or array length
string y4 = get(0, "foo");    // get i-th string char
```

17. MyPL Programs

- only consider single-file programs (in a text file, but we use `.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)

Example 1: Simple Struct

```
struct Car {
  string make;
  string model;
  int year;
}

void print_car(Car c) {
  print(c.make + " " + c.model + " " + itos(c.year) + "\n");
}

void main() {
  Car c1 = new Car("Toyota", "Corolla", 2023);
  Car c2 = new Car("Honda", "Civic", 2022);
  print_car(c1);
  print_car(c2);
}
```

Example 2: Simple Linked List

```
struct Node {
  int val;
  Node next;
}

void main() {
  // creates a linked list: 10, 20, 30, 40, 50
  Node head = null;
  int len = 5;
  for (int i = 0; i <= (len - 1); i = i + 1) {
    head = new Node((len - i) * 10, head);
  }
  // print the list
  Node curr = head;
  int i = 0;
  while (i < len) {
    print(curr.val);
    if (i < (len - 1)) {
      print(", ");
    }
    curr = curr.next;
    i = i + 1;
  }
  print("\n");
}
```

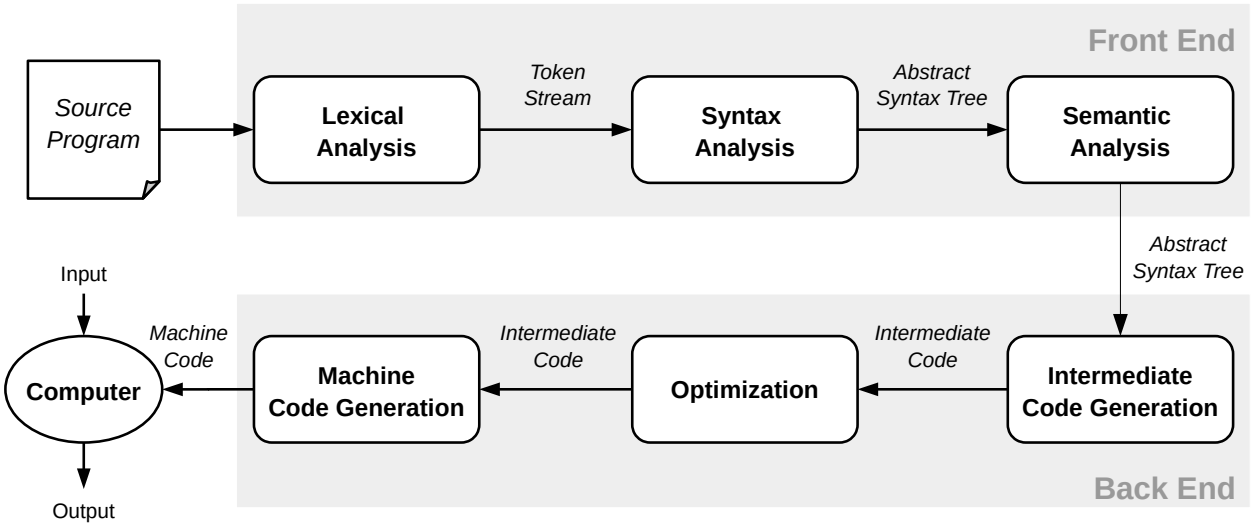
Example 3: Catalan Numbers

```
int fac(int n) {
  if (n <= 0) {
    return 1;
  }
  return r = n * fac(n - 1);
}

int catalan_number(int n) {
  if (n < 0) {
    // only defined for n >= 0
    return 0;
  }
  return fac(2 * n) / (fac(n + 1) * fac(n));
}

// prints: 1, 1, 2, 5, 14, 42, 132, ...
void main() {
  print("Enter the number of catalan numbers to print: ")
  int m = stoi(input());
  for (int n = 0; n < m; n = n + 1) {
    int c = catalan_number(n);
    print("Catalan number " + itos(n) + " = " + c + "\n");
  }
}
```

PL Implementation Basics: Compilation



Compilation (typically) involves the following steps (pipeline)

1. identify language "tokens" in source ... "*lexer/scanner*"
2. ensure syntax correct ... "*parser*"
3. ensure program "correct" (use-before-def, type errors) ... "*static analyzer*"
4. generate intermediate representation (e.g., for optimization)
5. improve performance of code ... "*optimizer*"
6. generate executable code (machine code) ... "*code generator*"

Example of "separation of concerns" (engineering design strategy)

- too complex to do "all at once" (single-pass vs multi-pass)
- also easier to manage, maintain, update/improve, reuse

Summary – Things to Know

Note: See Lecture 2 for MyPL specific items ...

1. Each of the compilation steps

2. Explain the high-level purpose of each step

3. Draw the sequence of steps including input/output of each step