ECS 140A Discussion 3

Ty Feng, Chris Fernandez

2:10-3:50pm Thursday, Aug 24 2023

UC DAVIS

Today’s agenda

First set

Construct a parser

Homework 2

Haskell

Q&A

First Set

- First set of a non terminal is the set of all terminals that can begin a string derived from X.
- If X derives ε, first set of X will include ε.
- First set is a concept used in syntax analysis, specifically in the context of parsing algorithms. It is a set of terminals that can appear immediately after a given non-terminal in a grammar.
- The FIRST set for each nonterminal symbol is calculated by examining the productions for that symbol and determining which terminal symbols can appear as the first symbol in a string derived from that production.

Rules to Calculate First Set

- If X-> c is a production rule, where c is a terminal, then FIRST(X) = { ‘c’ }
- If X-> c | d is a production rule, then FIRST(X) = { ‘c’, ‘d’ }
- If X-> Є is a production rule, then FIRST(X) = {Є}.
- If X -> Y1 is a production rule, FIRST(X) -> FIRST(Y1)
- If X->Y1 | Y2 is a production rule, FIRST(X) = {FIRST(Y1)} U {FIRST(Y2)}
- If X->Y1 Y2 Y3….Yn is a production,
  - FIRST(X) = FIRST(Y1)
  - If FIRST(Y1) contains Є, FIRST(X) = { FIRST(Y1) – Є } U { FIRST(Y2) }
  - If FIRST (Yi) contains Є for all i = 1 to n, then add Є to FIRST(X).

Example 1

Grammar : <Program> ::= <Block>
<Block> ::= { <Stmt> }
<Stmt> ::= <Assign> | <While>
<Assign> ::= id ':=' <Exp>
<Exp> ::= id | num
<While> ::= while <Exp> do <Block> end

- <While> ::= while <Exp> do <Block> end
  First(While) = {while}
- <Exp> ::= id | num
  First(Exp) = {id, num}
- <Assign> ::= id ':=' <Exp>
  First(Assign) = {id}

Example 1

- <Stmt> ::= <Assign> | <While>
  First(Stmt) = First(Assign) U First(While)
  First(Stmt) = {id} U {while}
  First(Stmt) = {id, while}

- <Block> ::= {<Stmt>}
  First(Block) = { ε } U First(Stmt)
  First(Block) = { ε, id, while }

- <Program> ::= <Block>
  First(Program) = First(Block)
  First(Program) = { ε, id, while }

Example 2

Grammar : <A> -> <B><C>
<C> -> +<B><C>|Є
<B> -> <D>< E>
<E> -> *<D><E> | Є
<D> -> -<A> | id

- FIRST(D) = { - , id }
- FIRST(E) = { *, Є }
- FIRST(C) = { +, Є }
- FIRST(B) = FIRST(D) = { - , id }
- FIRST(A) = FIRST(B) = { - , id }

Example 3

Grammar : <S> -> <A><C><B> | <C>bb | <B>a
<A> -> da | <B><C>
<B> -> g | Є
<C> -> h | Є

- FIRST(C) = { h , Є }
- FIRST(B) = { g, Є }
- FIRST(A) = {d} U FIRST(B) = {d, g,h, Є}
- FIRST(S) = FIRST(ACB) U FIRST(Cbb) U FIRST(Ba)

  = { d, g, h, b, a, Є}

Parser

- It's the heart of a typical compiler.
- It calls on the scanner to obtain the tokens of the input program.
- Assembles tokens into a parse tree.
- Passes the tree to later phases of the compiler (semantic analysis, code generation, optimization).
- A grammar is a generator for a language.
- A parser is a recognizer for a language.

Construct a Parser

- Tokens are matched directly with input tokens from the scanner
- Non-terminals are interpreted as calls to the procedures corresponding to the non-terminals.
- EBNF rules correspond to the code of a recursive-descent parser.
- Parser should not have to deal with backtrack, hence we use EBNF grammar to remove the recursion.
- A parser that commits to a specific action based only on the lookahead is called a predictive parser.
- A predictive parser requires that the different choices in a grammar rule start with different tokens.
- Hence, we use First Set to figure out which terminal symbols can be the first to appear from a Non terminal

Construct a Parser

We will need the following to construct our parser

- sym: a global variable to store a token
- next(): a routine that sets sym to the next token in the input
- f_X: denotes the set First(X) where X is a non-terminal
- error(): an error handling routine

Construct a Parser

- **For a a terminal x**, the parser recognizes it with this pseudocode:

  ```
  if (sym is an x)
        next();
  else
        error();
  ```

- **For non-terminal X:**

  ```
        X();
  ```

Construct a Parser

- **For a sequence X Y**:

```
X(); Y();
```

- **For Repetition {X}:**

```
while (sym in f_X)
    X();
```

- **For alternation X|Y:**

```
if (sym in f_X)
    X();
else if (sym in f_Y)
    Y();
else
    error();
```

Construct a Parser

<Program> ::= <Block>

<Block> ::= { <Stmt> }

<Stmt> ::= <Assign> | <While>

<Assign> ::= id ':=' <Exp>

<Exp> ::= id | num

<While> ::= while <Exp> do <Block> end

id is any identifier like x or y, and num is any numeric literal like 9, 33, 0.

Construct a Parser

First(While) = {while}

First(Exp) = {id, num}

First(Assign) = {id}

First(Stmt) = {id, while}

First(Block) = { ε, id, while }

First(Program) = { ε, id, while }

Construct a Parser

```
main() {
    /* read the first token */
    next();
    /* parse the input with the starting non-terminal */
    Program();
    /* do something to ensure all input was parsed */
}
```

```
<Program> ::= <Block>
    Program() {
        Block();
    }
```

```
<Block> ::= { <Stmt> }
    Block() {
        while (sym in f_Stmt) Stmt();
    }
```

Construct a Parser

```
<Stmt> ::= <Assign> | <While>
      Stmt() {
            if (sym in f_Assign){
                  Assign();
            }
            else if (sym in f_While){
                  While();
            }
            else{
                  error();
            }
            }
```

```
<Assign> ::= id ':=' <Exp>
      Assign() {
            if (sym is an id){
                  next();}
            else{
                  error();}
            if (sym is a ':='){
                  next();}
            else{
                  error();}
            Exp();
            }
```

Construct a Parser

<Exp> ::= id | num

```
Exp() {
    if (sym is an id){
        next();
    }
    else if (sym is a num){
        next();
    }
    else{
         error;
    }
    }
```

<While> ::= while <Exp> do <Block> end

```
While() {
    if (sym is a while){
        next();}
     else{error();}
    Exp();
    if (sym is a do){
        next();}
    else{ error();}
    Block();
    if (sym is an end){
        next();}
     else{error();}
}
```

Construct a Parser

Test the parser we created with the input text

- x := 5

- while x do
  ```
  while x do
      x := y
      y := 3
  end
  ```

Homework 2

A few things you might find useful for the homework:

1. Read line from file (use either BufferReader, or Scanner)
2. Split a string by a delimiter
3. Switch statements
4. `instanceof` to check if an object is of a specific class
5. `interface`, `implements`, `extends`, … → review lecture 5

Java code is on tyfeng.com/ecs140a/discussion3

Read file using BufferedReader `(ReadFileBufferedReader.java)`

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFileBufferedReader {
    public static void main(String[] args) {
        BufferedReader reader;
        try {
            reader = new BufferedReader(new FileReader("hw2.txt"));
            String line = reader.readLine();
            while (line != null) {
                System.out.println(line);
                line = reader.readLine(); // reads the next line
            }
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Read file using Scanner `(ReadFileScanner.java)`

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadFileScanner {
    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(new File("hw2.txt"));
            while (scanner.hasNextLine()) {
                System.out.println(scanner.nextLine());
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Where to put hw2.txt? (in Eclipse workspace)

/workspace path that you defined when first loading Eclipse

For example, I defined named my workspace as ecs140a, and hw2.txt goes right under there.

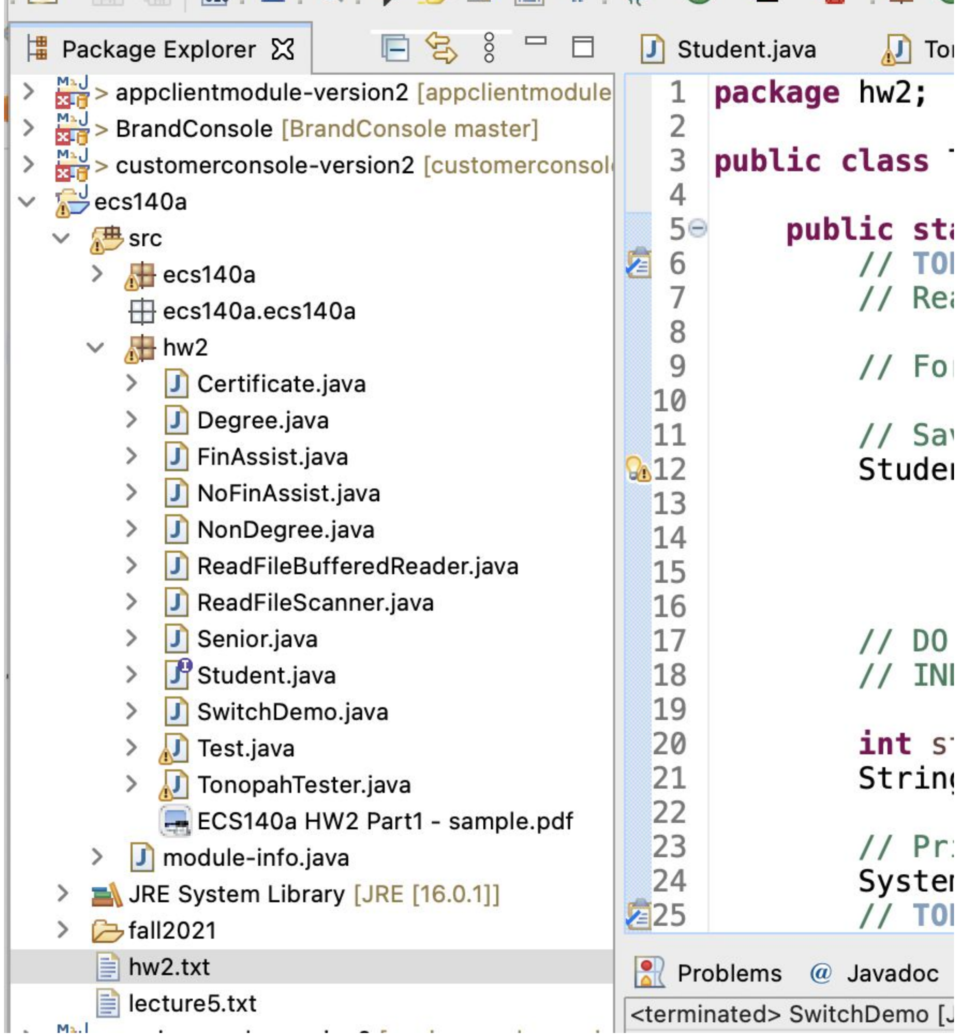
/ecs140a

— src/

— (packageName) **hw2**

— classes and interface (Student.java, DegreeStudent.java, etc.)

— hw2.txt



Package Explorer
Student.java
appclientmodule-version2 [appclientmodule
BrandConsole [BrandConsole master]
customerconsole-version2 [customerconsol
ecs140a
src
ecs140a
ecs140a.ecs140a
hw2
Certificate.java
Degree.java
FinAssist.java
NoFinAssist.java
NonDegree.java
ReadFileBufferedReader.java
ReadFileScanner.java
Senior.java
Student.java
SwitchDemo.java
Test.java
TonopahTester.java
ECS140a HW2 Part1 - sample.pdf
module-info.java
JRE System Library [JRE [16.0.1]]
fall2021
hw2.txt
lecture5.txt
package hw2;
public class
Problems
Javadoc
<terminated> SwitchDemo

Split a string by a delimiter

The Java String class has a `split(<delimiter>)` method. The following code splits the input String s into a String array named studentData.

```
String s = “046352;Moe;Howard;32;11;Y;E;G;Y;500.0”;

String[] studentData = s.split(“;”);

System.out.println("The student's name is: " +
studentData[1] + " " + studentData[2] + " \n");

------OUTPUT------

The student's name is: Moe Howard
```

Switch (SwitchDemo.java)

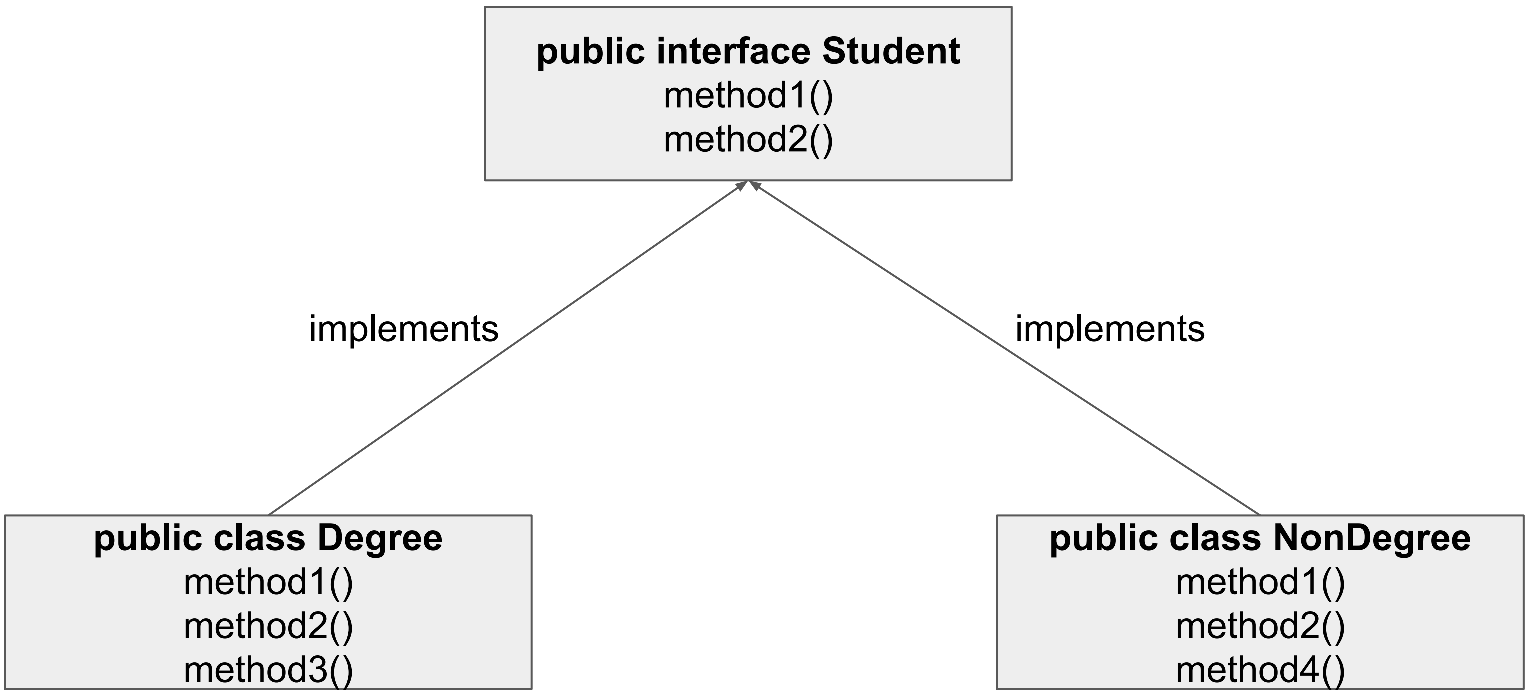
```
public class SwitchDemo {
    public static void main(String[] args) {

        int month = 8;
        String monthString;
        switch (month) {
            case 1:  monthString = "January";
                     break;
            case 2:  monthString = "February";
                     break;
            case 3:  monthString = "March";
                     break;
            case 4:  monthString = "April";
                     break;
            case 5:  monthString = "May";
                     break;
            case 6:  monthString = "June";
                     break;
            case 7:  monthString = "July";
                     break;
            case 8:  monthString = "August";
                     break;
            case 9:  monthString = "September";
                     break;
            case 10: monthString = "October";
                     break;
            case 11: monthString = "November";
                     break;
            case 12: monthString = "December";
                     break;
            default: monthString = "Invalid month";
                     break;
        }
        System.out.println(monthString);
    }
}
```

instanceof

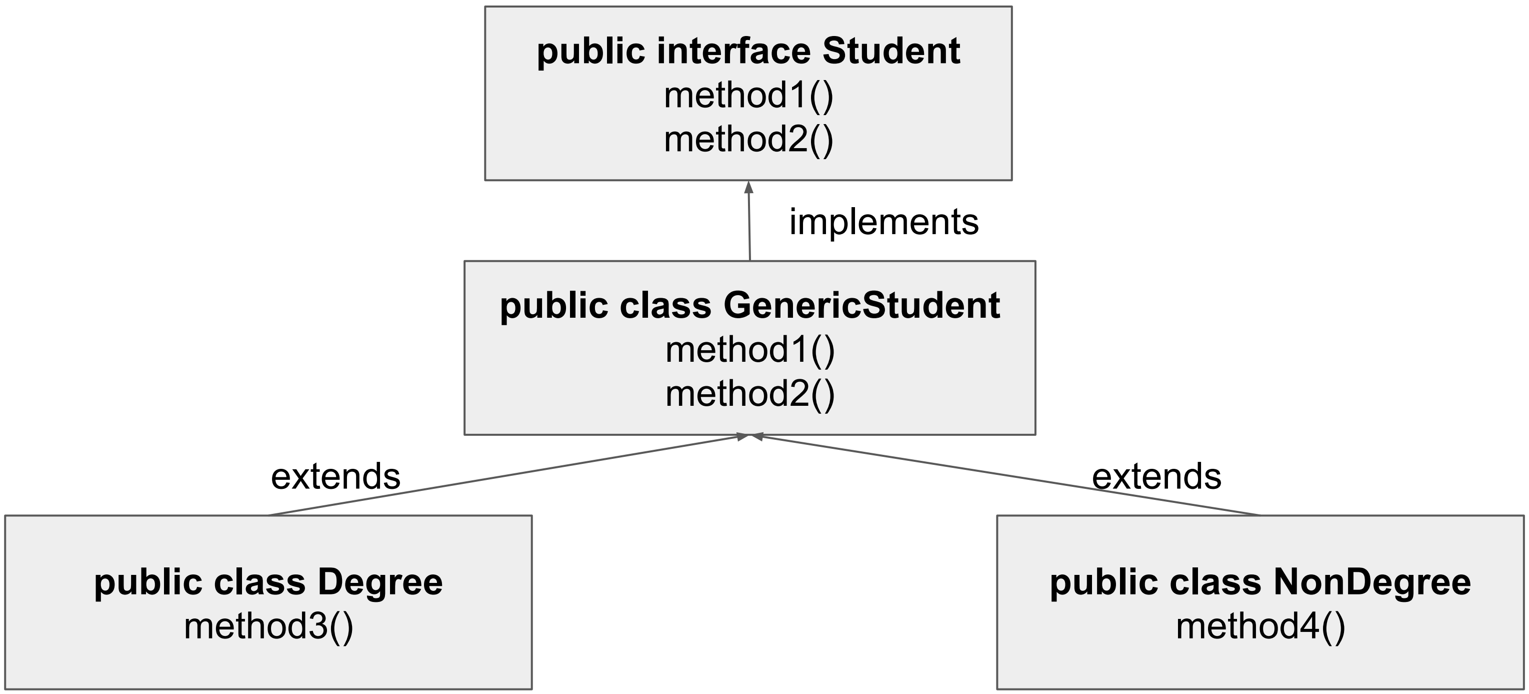
Suppose you want to check whether a student object belongs to DegreeStudent or NonDegreeStudent class, you can do this:

```
Student student1 = new DegreeStudent(...);

if (student1 instanceof DegreeStudent) {   // true

  …

}

if (student1 instanceof NonDegreeStudent) {   // false

  …

}
```

OOP in Java


public interface Student
method1()
method2()
implements
implements
public class Degree
method1()
method2()
method3()
public class NonDegree
method1()
method2()
method4()

OOP in Java


public interface Student
method1()
method2()
implements
public class GenericStudent
method1()
method2()
extends
extends
public class Degree
method3()
public class NonDegree
method4()

OOP in Java

How you design your class hierarchy is your choice. This OO design skill is highly relevant to programming in the real world. You may be given some business requirements, and you are asked to design various classes and interfaces to satisfy those requirements while following good programming practices. A carefully designed system saves a lot of headache down the line, when new features are added.

Autograder

We will use an autograder to grade hw 2. To ensure you get graded correctly, please follow these instructions:

In Eclipse, create a **package** named **hw2**. Put your code under the package.

We provided a file called **Test.java** in Canvas, which is where all the reports in part 2 are printed. We have provided some skeleton code and StringBuilder statements that generate the reports. Don’t add, modify, or delete the StringBuilder statements in this file. Add your code to where there is a TODO comment in Test.java. You may call classes defined in other files.

To make sure your code passes the autograder, check if your output is exactly the same as hw2_output.txt in Canvas. Hint: use String.equals() to compare

Expected output (hw2_output.txt)

Summary of each student's fees assessed:

Rick Reichardt has $3,450 fees assessed
Harriet Nelson has $2,350 fees assessed
Stanley Park has $3,450 fees assessed
Pamela Brown has $3,150 fees assessed
June Cleaver has $2,500 fees assessed
Martha Piper has $1,900 fees assessed
Shirley Muldowney has $2,900 fees assessed
Bobby Cheung has $0 fees assessed
William Clinton has $100 fees assessed
Paul Martin has $100 fees assessed
Groucho Marx has $4,600 fees assessed
Jessica Wong has $2,050 fees assessed
Ulysses Grant has $1,150 fees assessed
Vladimir Lenin has $3,400 fees assessed
Moe Howard has $2,675 fees assessed
Michael Jackson has $100 fees assessed
George Bush has $250 fees assessed
Keith Utley has $0 fees assessed
Larry Fine has $3,450 fees assessed

Summary of all student fees assessed:

Degree-seeking students without financial assistance: $15,600
Degree-seeking students with financial assistance: $9,025
Certificate students: $12,400
Senior citizens: $550

Total fees assessed: $37,575

Haskell

Let's run some code

tyfeng.com/ecs140a/haskell

To run any Haskell code:

```
$ ghc haskellCode.hs     // compiles to haskellCode
$ ./haskellCode
```

Pattern Matching

factorial.hs

```
fact 0 = 1
fact n = n * fact(n-1)
main :: IO ()
main = do
    print(fact(4))
```

Pattern Matching

```
fact 0 = 1

fact n = n * fact(n-1)

main :: IO ()

main = do

        print(fact(4))
```

The compiler will start searching for a function called "fact" with an argument. If the argument is not equal to 0, then the number will keep on calling the same function with 1 less than that of the actual argument.

When the pattern of the argument exactly matches with 0, it will call our pattern which is "fact 0 = 1".

Can also do the same thing with guards

factorialGuards.hs

```
fact n
    | n==0 = 1
    | n==1 = 1
    | n>1 = n*fact(n-1)
main :: IO ()
main = do
    print(fact(4))
```

Generate all even numbers after 2

evens.hs

```
evens = [2,4..]
main :: IO ()
main = do
    print (tail evens)
```

→ `tail` removes the first element in a list and returns the rest of the list

Haskell allows infinite lists.

[4, 6..]

How do you generate all positive numbers?

How do you generate all positive numbers?

```
positives = [1..]

main :: IO ()

main = do

    print (positives)
```

How do you generate first 10 positive numbers?

```
positives = [1..]
main :: IO ()
main = do
    print (take 10 positives)
```

Head

evens.hs

```
evens = [2,4..]
main :: IO ()
main = do
    print (head evens)
```

→ first element in a list

2

First element

```
evens = [2,4..]
main :: IO ()
main = do
    print(evens!!0)     -> first element
```

2

Second element

```
evens = [2,4..]
main :: IO ()
main = do
    print(evens!!1)    -> second element
```

4

Third element

```
evens = [2,4..]
main :: IO ()
main = do
    print(evens!!2)    -> third element
```

6

Squares

```
squares = [n*n | n <- [0..]]
main :: IO ()
main = do
    print(squares)
```

infinite squared numbers

squares5.hs

```haskell
squares = [n*n | n <- [0..5]]
main :: IO ()
main = do
    print(squares)
```

```
[0,1,4,9,16,25]
```

Is 2 a member in [1..4]?

```
member n (m:x)
      | m<n = member n x
      | m==n = True
      | otherwise = False
main :: IO ()
main = do
    print(member 2 [1..4])
```

True

member function takes in n, and m:x
m:x destructures a list x into head element m and the rest of the list x.

m<n: if the first element of the list x is less than n, it recursively calls member n on the remaining elements of the list

m==n: if the first element of the list x is equal to n, it returns True

Otherwise: if neither of the above conditions is true, it returns False

First step: member 2, [1..4] → m<n, recurse
Second step: member 2, [2..4] → True

Is 2 a member in [1,3..]?

```
member n (m:x)
      | m<n = member n x
      | m==n = True
      | otherwise = False
main :: IO ()
main = do
    print(member 2 [1,3..])
```

False