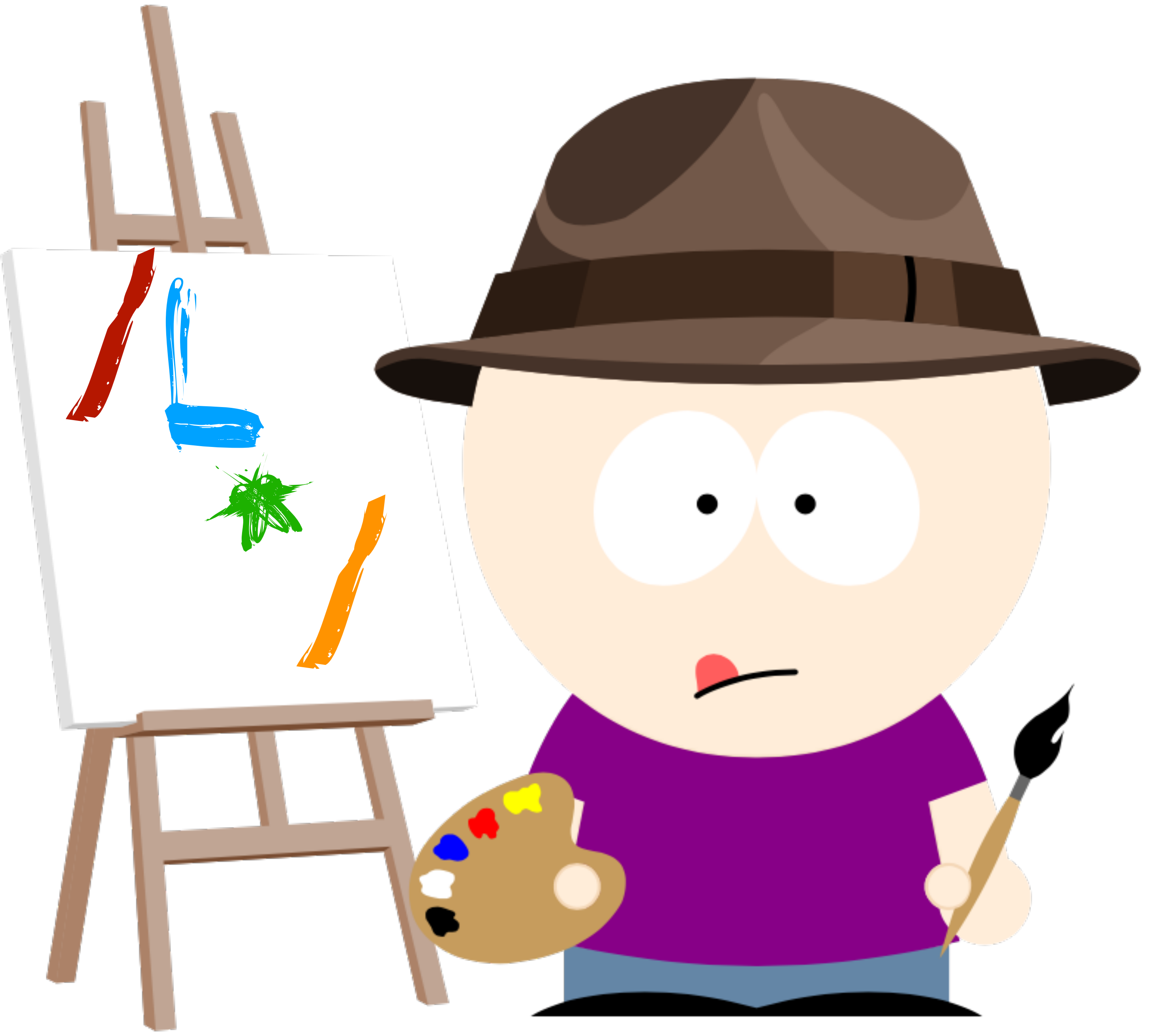
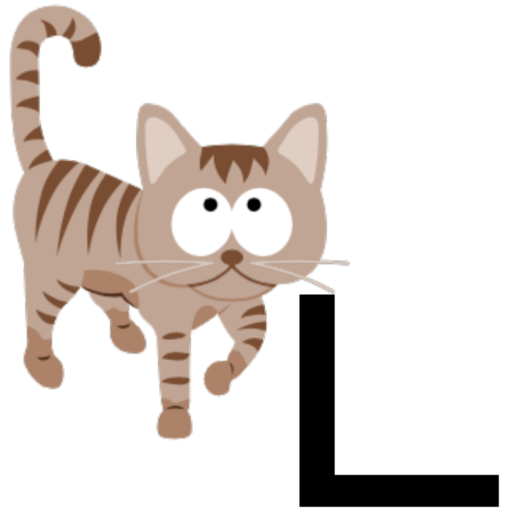


In-Class Exercises



Lexical Analysis 1

Version: 2025-12-27

Download exercises

1. Go to

   https://ligerlabs.org/compilers.html

2. Download the file

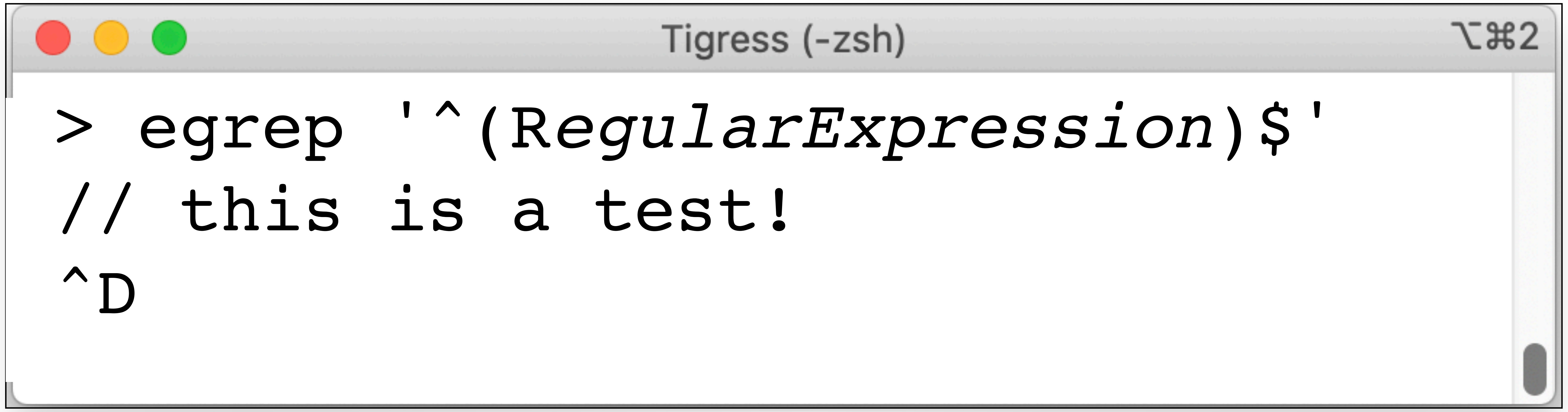
   ```
   lex-1-exercises.zip
   ```

3. Open up a terminal and Unzip the file

   ```
   > unzip lex-1-exercises.zip
   > cd lex-1-exercises
   > ls
   ```

Testing Regular Expressions

- In the tasks below you will be asked to construct regular expressions.
- You should test that your regular expressions are correct using `egrep`:



```
> egrep '^(RegularExpression)$'
// this is a test!
^D
```

- "^" matches the beginning of the line, "$" the end of the line.
- "`^D`" is `<control-D>` which terminates input.
- It's best to put parentheses around the regular expression.
- Instead of "`egrep`" you may have to say "`grep -E`"

Task 1

- C line comments look like this:

C

```c
// any text here
```

- In other words, "//" followed by any characters until the end of line ("`\n`").

1. Construct a regular grammar for C line comments.
2. Construct a regular expression for C line comments.

Task 2

- C decimal, octal, and hex constants look like this:

C

```c
int dec = 12345657890;
int hex = 0x12345ABCDEF;
int oct = 001234567;
```

- In other words, an octal constant starts with a 0 (zero) and is followed by a sequence of the digits `0` through `7`.
- A decimal constant cannot start with a 0 (zero).

1. Construct a regular grammar that accepts decimal, octal, and hex constants.
2. Do the same using a regular expression.

Task 3

- A C string constant looks like this:

C

```c
char* s = "abc\"def";
```

- In other words, a string can contain any character except " (double quote).
- Strings cannot span multiple lines. Let's assume that an end of line terminates a string.
- A character can be escaped with a \ (backslash).

1. Construct a regular grammar and a regular expression that accept C string constants.

Task 4

- Let's assume email address look like this:

```
alice@example.is
big_42.bob@foo.b-a-r.example.com
```

- Top-level domains (like "`.com`" and "`.is`") are a dot followed by 2 or 3 letters.
- Domain names can contain letters, dots, and hyphens.
- User names can also contain "_", "%", "+", and "-".

1. Construct a regular grammar and a regular expression that accept email addresses.

Task 4

- Construct regular expressions that match

1. An even number of `a`:s:
   `'', 'aa', 'aaaa', 'aaaaaa', ...`
2. An odd number of b's:
   `'b', 'bbb', 'bbbbb', 'bbbbbbb', ...`
3. An even number of a:s followed by an odd number of b's:
   `'b', 'aab', 'aaaabbb', 'aaaaaabbbbb', ...`
4. An even number of a:s **or** an an odd number of b's:
   `'', 'aa', 'bbb', 'aaaa', 'bbbbb'...`

Task 5

- Construct regular expressions that match

    1. Three dots, followed by '[', followed a non-empty sequence of '(' or ')', followed by ']', followed an 'E' or an 'e':

        `'...[(]E', '...[()(]e'`

Task 6

- Given this grammar for identifiers:

```
id     → letter | letter S
S      → letter | letter S
S      → digit  | digit S

digit  → 0 | 1 | ... | 9
letter → A | ... | Z | a | ... | z
```

- Show a derivation for the identifier "b77k"

Task 8

- Given this grammar for floating point numbers:

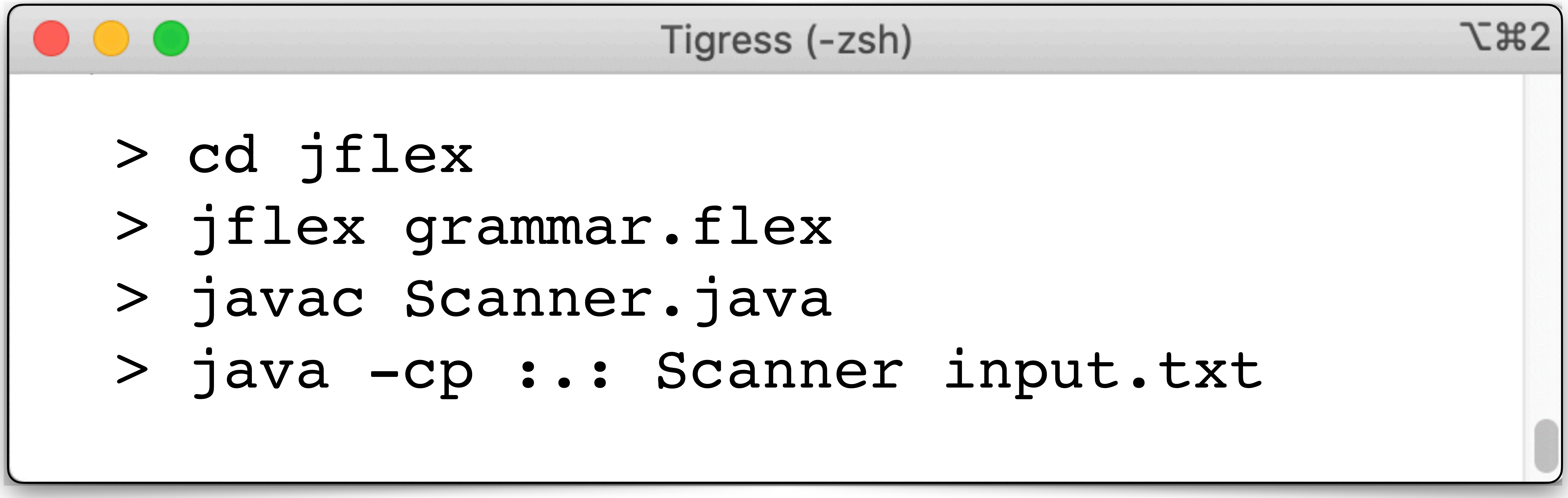
```
float  → + float1 | – float1 | float1
float1 → digit float1 | float2
float2 → . float3
float3 → digit float4 | digit
float4 → digit float4 | float5
float5 → E float6
float6 → + float7 | – float7 | float7
float7 → digit float7 | digit
```

- Show a derivation for the number "-44.5E-6",

Task 9

- Extend the `jflex` scanner to handle

    1. Floating point literals
    2. String literals



```
> cd jflex
> jflex grammar.flex
> javac Scanner.java
> java -cp :.: Scanner input.txt
```

```
%%
%public
%class Subst
%standalone
%unicode
%%

<YYINITIAL> {
  "BEGIN" {System.out.println("BEGIN"); }
  "END"   {System.out.println("END"); }
  ","     {System.out.println("COMMA"); }
  ":"     {System.out.println("COLON"); }
```

```
   [\ \t\b\012]+ { }


 [A-Za-z]                  -9])* {
       Syst               ("IDENT <"+yy              ;
   }


 [0-9][0-9]* {
       System.out.println("INTLIT <"+yytext()+">");
   }
}
```

Regular expression for integer literals

Returns the matched text

```
  \r|\n|\r\n { }




. {
  System.out.println("Illegal char:<"+yytext()+">");
}
```

Definitions and Algorithms

Regular Grammars

- A grammar is **regular** if all rules are of the form

$$A \rightarrow \underline{a}B$$
$$A \rightarrow \underline{a}$$

The "variables" of the grammar. Can't appear in the language.

- A, B, C, . . . are **non-terminals**
- $\underline{a}$, $\underline{b}$, $\underline{c}$, . . . are **terminals**
- α, β, γ, . . . are strings of symbols.

The tokens that appear in the language.

RE	Matches
a character	The character
. (dot)	Any single character
e1 | e2	S, if S is matched by e1 or e2
e1 e2	S1 S2, if e1 matches S1 and e2 matches S2
e+	One or more S if S is matched by e
e*	Zero or more S if S is matched by e
(e)	S, if S is matched by e.
\e	S, if S is matched by e.
e?	Zero or one S, if S is matched by e
[a-c],[a,b,c]	Any of the letters in the set

RE	Matches
[^az]	All characters except a and z
\w	All word-like characters: [a-z, A-Z, 0-9]
\d	All digit characters: [0-9]
\W	The complement of \w
\D	The complement of \d
^	Matches beginning of the string
$	Matches the end of the string.
a{2, 6}	Checks if a occurs between 2 and 6 times

collberg.cs.arizona.edu

ligerlabs.org

Supported by

NSF SATC/TTP-1525820

SATC/EDU-2029632

Christian Collberg

University of Arizona