CSCI 334:
Principles of Programming Languages

Lecture 22: Recap

Instructor: Dan Barowy

Williams

Announcements

Project due: Sunday, Dec 9 by 10pm

Presentation: Tuesday, Dec 11 at 11:20am
in TCL 202

SWELL user testing

This weekend, 30-40 minutes.

`https://bit.ly/2EgWKgi`

It would be a real help if you have the time!

Topics

Recap

Project Q&A

Recap

First, a SPJ video

What did we learn?

Programming languages are for people

VB.NET Quick Sort [on hold]

-5

I am trying to write a quick sort program in vb.net. The program is compilable but when i run it, it throws a run time exception. The code in ASCII is:

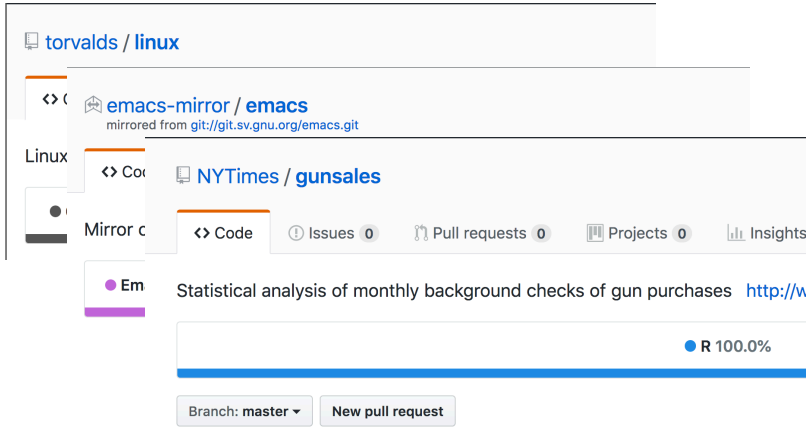
```
077 111 100 117 108 101 032 077 111 100 117 108 101 049 013 010 032 032 032 032 070
117 110 099 116 105 111 110 032 112 097 114 116 105 116 105 111 110 040 066 121 082
101 102 032 097 032 065 115 032 065 114 114 097 121 044 032 066 121 086 097 108 032
108 032 065 115 032 073 110 116 101 103 101 114 044 032 066 121 086 097 108 032 114
032 065 115 032 073 110 116 101 103 101 114 041 013 010 032 032 032 032 032 032 032
032 068 105 109 032 106 032 065 115 032 073 110 116 101 103 101 114 032 061 032 108
013 010 032 032 032 032 032 032 032 032 068 105 109 032 112 032 065 115 032 073 110
116 101 103 101 114 032 061 032 097 040 114 041 013 010 013 010 032 032 032 032 032
032 032 032 070 111 114 032 105 032 061 032 108 032 084 111 032 114 032 045 032 049
013 010 032 032 032 032 032 032 032 032 032 032 032 032 073 102 032 097 040 105 041
032 060 032 112 032 084 104 101 110 013 010 032 032 032 032 032 032 032 032 032 032
032 032 032 032 032 032 068 105 109 032 116 109 112 032 065 115 032 073 110 116 101
103 101 114 032 061 032 097 040 105 041 013 010 032 032 032 032 032 032 032 032 032
032 032 032 032 032 032 032 097 040 105 041 032 061 032 097 040 106 041 013 010 032
032 032 032 032 032 032 032 032 032 032 032 032 032 032 097 040 106 041 032 061
032 116 109 112 013 010 032 032 032 032 032 032 032 032 032 032 032 032 032 032 032
032 106 032 061 032 106 032 043 032 049 013 010 032 032 032 032 032 032 032 032 032
032 032 032 069 110 100 032 073 102 013 010 032 032 032 032 032 032 032 032 078 101
120 116 013 010 032 032 032 032 032 032 032 032 097 040 114 041 032 061 032 097 040
106 041 013 010 032 032 032 032 032 032 032 032 097 040 106 041 032 061 032 112 013
010 013 010 032 032 032 032 032 032 032 032 082 101 116 117 114 110 032 106 013 010
032 032 032 032 069 110 100 032 070 117 110 099 116 105 111 110 013 010 032 032 032
```

A good language makes elegant algorithms *look* elegant!

```
let rec quicksort list =
   match list with
   | [] -> []
   | x::xs ->
       let xs_small =
           xs
           |> List.filter (fun e -> e < x)
           |> quicksort
       let xs_large =
           xs
           |> List.filter (fun e -> e >= x)
           |> quicksort
       xs_small @ [x] @ xs_large
```

But a good language is more about looks

It's about being the right tool for the job

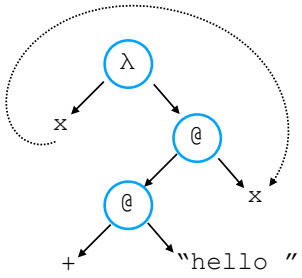


Nonetheless, all languages have common parts

```
e ::= n | e+e | e-e
n ::= d | nd
d ::= 0 | 1 | … | 9
```

Syntax

Nonetheless, all languages have common parts



Abstract syntax

Nonetheless, all languages have common parts

```
let rec eval (e: Expr) : int =
    match e with
    | Number n        -> n
    | Plus (e1, e2) -> (eval e1) + (eval e2)
    | Minus(e1, e2) -> (eval e1) - (eval e2)
    | Mult (e1, e2) -> (eval e1) * (eval e2)
```

Evaluation rules

With knowledge of a few principles, we can understand how all languages work

```
<expr>  ::= <var>
         |  <abs>
         |  <app>
<var>   ::= x
<abs>   ::= λ<var>.<expr>
<app>   ::= <expr><expr>
```

Call stack

There are limits to what we can do on a computer

`Halt(P,x)` = { returns `true` if `P(x)` halts; returns `false` otherwise }

But there are also clever workarounds that approximate the impossible

Nothing happens until the program runs out of free storage. When a free register is wanted, and there is none left on the free-storage list, a reclamation cycle starts.

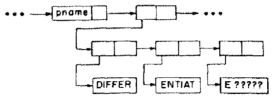


Fig. 4

First, the program finds all registers accessible from the base registers and makes their signs negative. This is accomplished by starting from each of the base registers

And using those approximations, we can abstract away incidental complexity

```
(define (fizzbuzz x y)
  (println
    (cond (( = (modulo x 15) 0 ) "FizzBuzz")
          (( = (modulo x 3) 0 ) "Fizz")
          (( = (modulo x 5) 0 ) "Buzz")
          (else x)))

    (if (< x y) (fizzbuzz (+ x 1) y)))
```

Good abstractions let us build complex things simply

```
 Welcome to
                  EEEEEE  LL      IIII  ZZZZZZZ   AAAAA
                  EE      LL       II       ZZ   AA   AA
                  EEEEE   LL       II     ZZZ    AAAAAAA
                  EE      LL       II    ZZ      AA   AA
                  EEEEEE  LLLLLL  IIII  ZZZZZZZ  AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

Good abstractions let us build complex things simply

Deep Dream

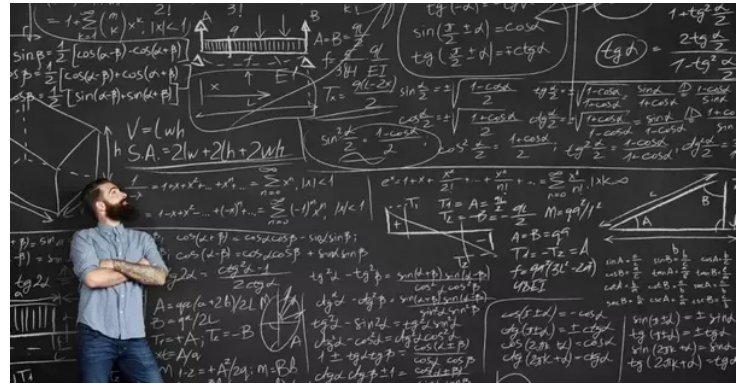
Good abstractions let us build complex things simply

```
alpha = { 'a'..'z' | 'A'..'Z' }
digit = { '0'..'9' }

ident = { (alpha | digit)+ }

ident_list = _{ !digit ~ ident ~ (" " ~ ident)+ }
```

Parsing Expression Grammars (PEGs)

One important problem in the real world is scale

Sometimes we need to scale operations



Other times it's about the data

When scaling operations, we use functional approach

```
type Color =
    | Red = 0
    | Green = 1
    | Blue = 2

let printColorName (color:Color) =
    match color with
    | Color.Red -> printfn "Red"
    | Color.Green -> printfn "Green"
    | Color.Blue -> printfn "Blue"
    | _ -> ()
```

When scaling data, we use object-orientation

```
class Person:
    def say_hi(self):
        print('Hello, how are you?')

p = Person()
p.say_hi()
```

In either model, many tools can help us scale

```
[<TestClass>]
type TestClass () =

    [<TestMethod>]
    member this.TestMethodPassing() =
        Assert.IsTrue(true)
```

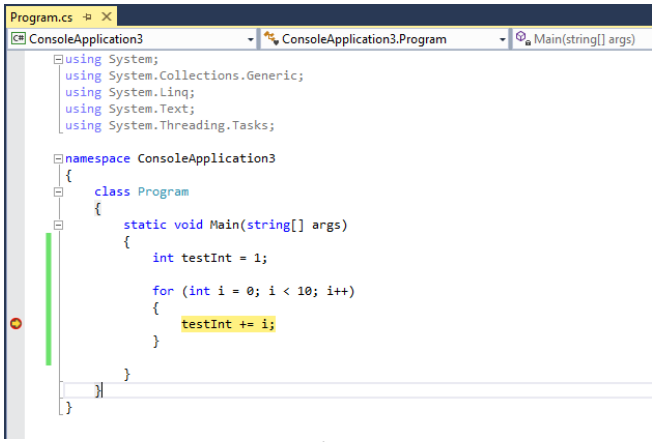
Tests

In either model, many tools can help us scale

```
let insertKeepsOrder (x:int) xs =
  ordered xs ==> ordered (insert x xs)

Check.Quick insertKeepsOrder
```

Randomized testing

In either model, many tools can help us scale



Debuggers

In either model, many tools can help us scale

```
scala> type AA[-T] = A[A[T]]
<console>:15: error: contravariant type T
occurs in covariant position in type [-
T]A[A[T]] of type AA
       type AA[-T] = A[A[T]]
            ^
```

Types!

If you want to be a great programmer, take the time to understand your tools

One way to do that is to build lots of stuff!

If it is not impossible, you can build it!

Use your imagination!

With enough practice, you will transcend “coding”

You will become a craftsperson

That is the quality possessed by all of our big thinkers

And it is the quality that leads you toward a fulfilling career in computer science

Good luck on your final project!

I’m sure that you’re going to do great!

Project Q&A