Homework 3 — Bools, Scope Elimination, and Thunks

Code submitted without complete test coverage will result in an automatic 20% penalty.

This assignment is an individual assignment.

Part 1 — Booleans

Start with the interpreter with function values, and extend the implementation to support boolean literals, an equality test, and a conditional form:

```
<Exp> = ....
      | true
      | false
      | {= <Exp> <Exp>}
      | {if <Exp> <Exp> <Exp>}
```

The = operator should only work on number values, and if should only work when the value of the first subexpression is a boolean. Report a “not a number” error if a subexpression = produces a non-number value and report a “not a boolean” error when the first subexpression of if does not produce a boolean. The if form should evaluate its second subexpression only when the first subexpression’s value is true, and it should evaluate its third subexpression only when the first subexpression’s value is false. True and false are always spelled true and false, not #t or #f.

Note that you not only need to extend Exp with new kinds of expressions, but you will also need to add booleans to Value.

For example,

```
true
```

should produce a true value, while

```
{if true {+ 1 2} 5}
```

should produce 3, and

```
{if 1 2 3}
```

should report a “not a boolean” error.

As usual, update parse to support the extended language.

More examples:

```
(test (interp (parse `{if {= 2 {+ 1 1}} 7 8})
```

```
               mt-env)
       (interp (parse `7)
               mt-env))
  (test (interp (parse `{if false {+ 1 {lambda {x} x}} 9})
               mt-env)
       (interp (parse `9)
               mt-env))
  (test (interp (parse `{if true 10 {+ 1 {lambda {x} x}}})
               mt-env)
       (interp (parse `10)
               mt-env))
  (test/exn (interp (parse `{if 1 2 3})
                    mt-env)
            "not a boolean")
```

Part 2 — Hiding Variables

Add an unlet form that hides the nearest visible binding (if any) of a specified variable but let’s other bindings through. For example,

```
  {let {[x 1]}
    {unlet x
      x}}
```

should raise a “free variable” exception, but

```
  {let {[x 1]}
    {let {[x 2]}
      {unlet x
        x}}}
```

should produce 1.

As before, you must update the parse function.

Some examples:

```
  (test/exn (interp (parse `{let {[x 1]}
                              {unlet x
                               x}})
                    mt-env)
            "free variable")
  (test (interp (parse `{let {[x 1]}
                          {+ x {unlet x 1}}})
               mt-env)
       (interp (parse `2) mt-env))
  (test (interp (parse `{let {[x 1]}
                          {let {[x 2]}
```

```
                          {+ x {unlet x x}}}})
               mt-env)
        (interp (parse `3) mt-env))
  (test (interp (parse `{let {[x 1]}
                          {let {[x 2]}
                            {let {[z 3]}
                              {+ x {unlet x {+ x z}}}}}})
               mt-env)
        (interp (parse `6) mt-env))
  (test (interp (parse `{let {[f {lambda {z}
                                   {let {[z 8]}
                                     {unlet z
                                       z}}}]}
                          {f 2}})
               mt-env)
        (interp (parse `2) mt-env))
```

Part 3 — Thunks

A *thunk* is like a function of zero arguments, whose purpose is to delay a computation. Extend your interpreter with a delay form that creates a thunk, and a force form that causes a thunk's expression to be evaluated:

```
  <Exp> = ....
        | {delay <Exp>}
        | {force <Exp>}
```

A thunk is a new kind of value, like a number, function, or boolean.

For example,

```
  {delay {+ 1 {lambda {x} x}}}
```

produces a thunk value without complaining that a function is not a number, while

```
  {force {delay {+ 1 {lambda {x} x}}}}
```

triggers a "not a number" error. As another example,

```
   {let {[ok {delay {+ 1 2}}]}
     {let {[bad {delay {+ 1 false}}]}
       {force ok}}}
```

produces 3, while

```
   {let {[ok {delay {+ 1 2}}]}
     {let {[bad {delay {+ 1 false}}]}
       {force bad}}}
```

triggers a “not a number” error.

More examples:

```
  (test/exn (interp (parse `{force 1})
                    mt-env)
            "not a thunk")
  (test (interp (parse `{force {if {= 8 8} {delay 7} {delay 9}}})
                mt-env)
        (interp (parse `7)
                mt-env))
  (test (interp (parse `{let {[d {let {[y 8]}
                                   {delay {+ y 7}}}]}
                          {let {[y 9]}
                            {force d}}})
                mt-env)
        (interp (parse `15)
                mt-env))
```