Midterm3-draft - Name ______________________________

For all problems, choose *best* possible answer and make answer clearly known by circling or writing the letter of your answer.

1. [5]What is the closure generated from the function definition on line 1?

```
let f = function(x)
         x + y
in let g = 5
in let y = 2
in f(g)
```

A. Closure(”x”, $x + y, [g \mapsto 5, y \mapsto 2, x \mapsto 5]$)
B. Closure(”x”, $x + y, [g \mapsto 5, y \mapsto 2]$)
C. Closure(”x”, $x + y, [g \mapsto 5]$)
[✓ D. Closure(”x”, $x + y, []$)]

2. [5]What is the value computed by this program?

```
let f = function(x)
         x + y
in let g = 5
in let y = 2
in f(g)
```

A. 10
B. Closure(”x”, $x + y$, $[x \mapsto 5]$)
C. $x + y$
D. 7
[✓ E. Error]

3. [5]What are the three parts of a Closure?
A. Argument, Type, Dictionary
B. Argument, Closure, Context
C. Parameter, Environment, Extensions
D. Argument, Function Body, Environment

4. [5]What does the following program evaluate to?

```
let fac = function(x)
            if x >= 0
              then x * fac(x - 1)
              else 1
in fac(4)
```

A. 6
B. 24
C. Zero
[✓ D. Error]

5. [5]Why do we need closures to implement the semantics of functions?
A. We don’t need them, they are merely for convenience
B. They let us look ahead and figure out where the function will be called, then we replace all those with the definition
C. They allow the program to make recursive calls
D. Without them we would have no way of capturing the environment of the program when the function is defined

6. [5]What is the closure generated by line 2 below?

```
1| let x = 5
2|  in let f = function(x)
3|                x + x
4|  in f(10)
```

A. Closure(”x”, $10 + 10$, [])
B. Closure(”x”, $5 + 5$, $[x \mapsto 10]$)
C. Closure(”x”, $x + x$, $[x \mapsto 10]$)
D. Closure(”x”, $x + x$, $[x \mapsto 5]$)

7. [5]What is the abstract syntax for the following expression?`(function(x) x + x) (2 + 3)`
A. `Error, Invalid Syntax`
B. `Let "f" (FunDef "x" (Plus (Ident "x") (Plus "x"))) (FunCall (Ident "f") (Num 5))`
C. `FunDef "x" (Plus (Ident "x") (Ident "x"))`
D. `FunCall (FunDef "x" (Plus (Ident "x") (Ident "x"))) (Plus (Num 2) (Num 3))`

8. [5]What is the abstract syntax for the following expression?

```
let f = function(x)
          f(x)
in f(5)
```

A. `LetRec "f" (FunDef "x" (FunCall (Ident "x") (Ident "x"))) (FunCall (Ident "f") (Num 5))`
B. `Let "f" (FunDef "x" (FunCall (Ident "x") (Ident "x"))) (FunCall (Ident "f") (Num 5))`

9. [5]What is the difference between `FunDef` and `LetRec`?
A. Only `FunDef` uses closures
B. `FunDef` defines functions while `LetRec` may define non-functions
C. `FunDef` supports recursion, while `LetRec` does not
D. `LetRec` adds the name of the function to its own closure

10. [5]What should the `???` be replaced with in the inference rule below?

$$\frac{}{\sigma \vdash ???\ x\ e \Downarrow \text{Closure}\ x\ e\ \sigma}$$

A. `LetRec`
B. `Closure`
C. `FunCall`
D. `FunDef`

11. [5]What should the `???` be replaced with in the inference rule below?

$$\frac{\sigma \vdash e_f \Downarrow \text{Closure } p\ e_b\ \pi \quad \sigma \vdash e_a \Downarrow v_a \quad ???[p \mapsto v_a] \vdash e_b \Downarrow v}{\sigma \vdash \text{FunCall } e_f\ e_a \Downarrow v}$$

A. $\emptyset$
B. Closure
C. σ
[✓ D. π]

12. [5]What should the `???` be replaced with in the inference rule below?

$$\frac{\sigma' = \sigma[f \mapsto ???] \quad \sigma' \vdash e_b \Downarrow v}{\sigma \vdash \texttt{LetRec} f\ x\ e_d\ e_b \Downarrow v}$$

A. $\text{Closure}(f, e_d, \sigma')$
B. $\text{Closure}(f, e_d, \sigma)$
C. $\text{Closure}(x, e_d, \sigma)$
[✓ D. $\text{Closure}(x, e_d, \sigma')$]

13. [5]In order to implement the interpreter that supported recursion(using `LetRec`) we needed to alter:
A. Function Definitions
B. Identifiers
C. Let Bindings
[✓ D. The Environment]

14. [5]What is a type synonym?
A. A dictionary of types
B. A function whose type changes based on the inputs
C. A type wrapped up inside of a new type definition in order to prevent misuse
[✓ D. An alternative naming for a type]

15. [5]Why do we use type synonyms?
A. To support functions that nay change their type depending on the input
B. To make our writing more eloquent
C. To represent new kinds of data we had no type for
[✓ D. To make complicated types easier to read by giving them a useful name]

16. [5]The type synonym `Parser S D` is equivalent to:
A. $[S] \rightarrow [([D], [S])]$
B. $[S] \rightarrow (D, [S])$
C. $S \rightarrow [(D, [S])]$
[✓ D. $[S] \rightarrow [(D, [S])]$]

17. [5]In the `Parser S D` what is the type `D`?
A. Expressions
B. The domain of errors for the parser
C. The set of symbols being parsed
[✓ D. The data structure that results from the parser]

18. [5]In the `Parser S D` what is the type `S`?
    A. Expressions
    B. The domain of errors for the parser
    C. The set of symbols being parsed
    D. The data structure that results from the parser

19. [5]What is each part of the tuple in the parser type?
    A. (Stream, Stack)
    B. (Error, String)
    C. (Parsed Expression, String)
    D. (Parsed Structure, Remaining Input)

20. [5]Why do we return a list of results from the parser type?
    A. To parse multiple types at a time
    B. To represent errors
    C. To parse more than one input stream at a time
    D. To support multiple partial-parses as we go

21. [5]Along with `success` and `failure`, what is the other primitive parser?
    A. `choose`
    B. `bind`
    C. `string`
    D. `char`

22. [5]What is a lexer and why are they used?
    A. A function which creates super villains for the Batman
    B. The step taken after parsing which produces the final AST, ready to be evaluated by an interpreter
    C. A function which removes whitespace from a string
    D. A type of intermediate parser that makes the job of parsing easier through tokenizing an input into its ”part of speech”

23. [5]What is a *Lexical Grammar*?
    A. A string without whitespace
    B. The different kinds of expression
    C. The constructors for an AST
    D. The ”Parts of Speech” of a programming language

24. [5]what is the result of the parser `(char ’a’)` on the input `"a"`?
    A. `Error`
    B. `’a’`
    C. `[("a", "")]`
    D. `[(’a’, "")]`

25. [5]what is the result of the parser `(char ’a’)` on the input `"b"`?
    A. `[("a", "")]`
    B. `[("b", "")]`
    C. `[(’b’, "a")]`
    D. `Error`

[✓ E. `[]`]

26. [5]what is the result of the parser `(char ’x’)` on the input `"xy"`?
    A. `Error`
    B. `[("x", "")]`
    C. `[("xy", "")]`
    [✓ D. `[(’x’, "y")]`]

27. [5]what is the result of the parser `choose (char ’a’) (char ’b’)` on the input `"ab"`?
    A. `[]`
    B. `[("ab", "")]`
    C. `[(’a’, "b"), (’b’, "a")]`
    D. `[(’b’, "a")]`
    [✓ E. `[(’a’, "b")]`]

28. [5]Descriptively, the parser below successfully parses:

```
bind (char ’a’)
  (lambda x -> bind (char ’b’)
  (lambda y -> success "ab"))
```

A. It is always successful and parses gives `"ab"`
B. The string `"ba"`
C. The characters `’a’` or `’b’`
[✓ D. The string `"ab"`]

29. [5]Which of the following strings is *not* parsed by `char ’a’`
    A. ”aaa”
    [✓ B. ” ”]
    C. ”a”
    D. All of them parse successfully

30. [5]What is the type synonym for `[Char]`
    A. `Symbol`
    B. `Stream`
    C. `Dict`
    [✓ D. `String`]

31. [5]What is the difference between `choose` and `option`?
    A. They are the same
    B. `option` parses both inputs while `choose` only uses the second parser if the first is unsuccessful
    [✓ C. `choose` parses both inputs while `option` only uses the second parser if the first is unsuccessful]