CANONIZADO, Michael Xavier E.
BSCS 2A
Programming Languages
Homework 3

1. The following C function computes for the power ab where a is a floating point and b is a (nonnegative) integer.

```
double power(double a, int b) {
Int i; double temp = 1.0;
for (i = 0; i <= b; i++)
  temp *= a: return temp;
}
```

a. Rewrite the procedure in a functional form.

Answer:

```
double power(double a, int b) {
    // Base case
    if (b == 0) return 1.0;
    return a * power(a, b - 1);
}
```

b. Rewrite your answer to (a) using an accumulating parameter to make it tail recursive.

Answer:

```
double powerHelper(double a, int b, double acc) {
    if (b == 0) return acc;
    return power_helper(a, b-1, acc * a);
}

double power(double a, int b) {
    return power_helper(a, b, 1.0);
}
```

2. The binomial coefficients are a frequent computational task in computer science. They are defined as follows for n>=0, 0<=k<=n:

B(n,k) = (n!)/((n-k)!k!)

a) Write a procedure using a loop to computer B(n,k). Test your program on B(10,5).

Answer:

```c
#include <stdio.h>

double binomialCoefficients(int n, int k) {
    double top = 1.0, bottomsub = 1.0, bottomk = 1.0;

    for (int i = 1; i <= n; i++) {
        top *= i;
    }

    for (int i = 1; i <= (n - k); i++) {
        bottomsub *= i;
    }

    for (int i = 1; i <= k; i++) {
        bottomk *= i;
    }

    return top / (bottomsub * bottomk);
}

int main() {
    int n = 10, k = 5;
    printf("BC(%d, %d) = %lf\n", n, k, binomialCoefficients(n, k));
    return 0;
}
```

Output

```
C(10, 5) = 252.000000


=== Code Execution Successful ===
```

b) Use the following recurrence and the fact that B(n,0) = 1 and B(n,n) = 1 to write a functional procedure to compute B(n,k):

B(n, k) = B(n – 1, k – 1) + B(n – 1, k)

Test your program on B(10, 5).

Answer:

```c
#include <stdio.h>

int binomialCoefficients(int n, int k) {
    if (k == 0 || n == k)
        return 1;
    return binomialCoefficients(n - 1, k - 1) + binomialCoefficients(n - 1, k);
}

int main() {
    int n = 10, k = 5;
    printf("BC(%d, %d) = %d\n", n, k, binomialCoefficients(n, k));
    return 0;
}

```

Output

```
BC(10, 5) = 252


=== Code Execution Successful ===
```

3. A refutation system is a logical system that proves a statement by assuming it is false and deriving a contradiction. Show that Horn clause logic with resolution is a refutation system. (Hint: The empty clause is assumed to be false, so a goal ← a is equivalent to a →false. Show that this is equivalent to not(a).)

Answer:

a	False	a → False	!a
True	False	False	False
False	False	True	True

4. Write the following statements in the first-order predicate calculus:

If it is raining or snowing, then there is precipitation.
If it is freezing and there is precipitation, then it is snowing.
If it is not freezing and there is precipitation, then it is raining.
It is snowing.

R(x) → "It is raining at time x"
S(x) → "It is snowing at time x"
P(x) → "There is precipitation at time x"
F(x) → "It is freezing at time x"

"If it is raining or snowing, then there is precipitation."

$\forall x\,(R(x) \lor S(x) \rightarrow P(x))$

"If it is freezing and there is precipitation, then it is snowing."

$\forall x\,(F(x) \land P(x) \rightarrow S(x))$

"If it is not freezing and there is precipitation, then it is raining."

$\forall x\,(\neg F(x) \land P(x) \rightarrow R(x))$

"It is snowing."

$\exists x\, S(x)$

5. Show that the following grammar does not satisfy the second rule of predictive parsing:

stmt → if-stmt | other
if-stmt → if-stmt [else stmt]

Example:

1) stmt
2) if-stmt
3) if-if-stmt
4) if-if-if-stmt
5) if-if-if-if-stmt
6) …

Answer:
With the example above. The given grammar is left recursive, thus violating the second rule of predictive parsing

6. Given the following grammar in EBNF:

expr → (list) | a

list → expr [list]

a) Show that the two conditions for predictive parsing are satisfied.

Answer:

Condition one:

"list → expr [list]"

The grammar is not left recursive. So condition one is met

Condition two:

"expr → (list) | a"

"list → expr [list]"

The grammar has no common prefix and gives the ability to choose among several alternatives

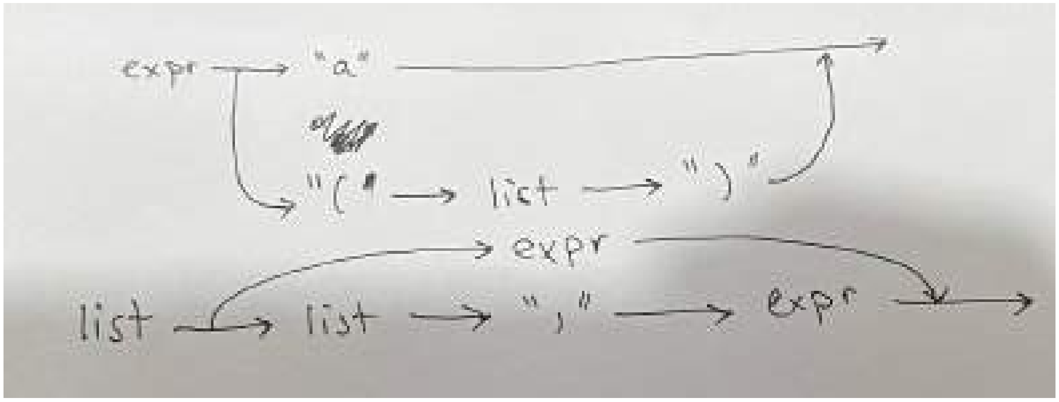
b) Write a recursive-descent recognizer for the language

```
class RecursiveDescentParser:
    def __init__(self, tokens):
        self.tokens = tokens
        self.pos = 0

    def match(self, expected):
        if self.pos < len(self.tokens) and self.tokens[self.pos] == expected:
            self.pos += 1
        else:
            raise SyntaxError(f"Expected '{expected}', but found '{self.tokens[sel
                if self.pos < len(self.tokens) else 'Unexpected end of input'")

    def expr(self):
        if self.pos < len(self.tokens) and self.tokens[self.pos] == '(':
            self.match('(')
            self.list()
            self.match(')')
        elif self.pos < len(self.tokens) and self.tokens[self.pos] == 'a':
            self.match('a')
        else:
            raise SyntaxError(f"Invalid token '{self.tokens[self.pos]}' at positio
                .pos}")

    def list(self):
        self.expr()
        if self.pos < len(self.tokens) and self.tokens[self.pos] in ('(', 'a'):
            self.list()

    def parse(self):
        self.expr()
        if self.pos < len(self.tokens):
            raise SyntaxError(f"Unexpected token '{self.tokens[self.pos]}' at pos
                {self.pos}")

test_cases = [
    ['a'],
    ['(', 'a', ')'],
    ['(', '(', 'a', ')', ')'],
    ['(', 'a', 'a', ')'],
    ['(', '(', 'a', ')', 'a', ')'],
    ['(', 'a'],
    ['a', ')'],
    ['(', ')'],
]

for tokens in test_cases:
    try:
        parser = RecursiveDescentParser(tokens)
        parser.parse()
        print(f"Accepted: {tokens}")
    except SyntaxError as e:
        print(f"Rejected: {tokens} - {e}")
```

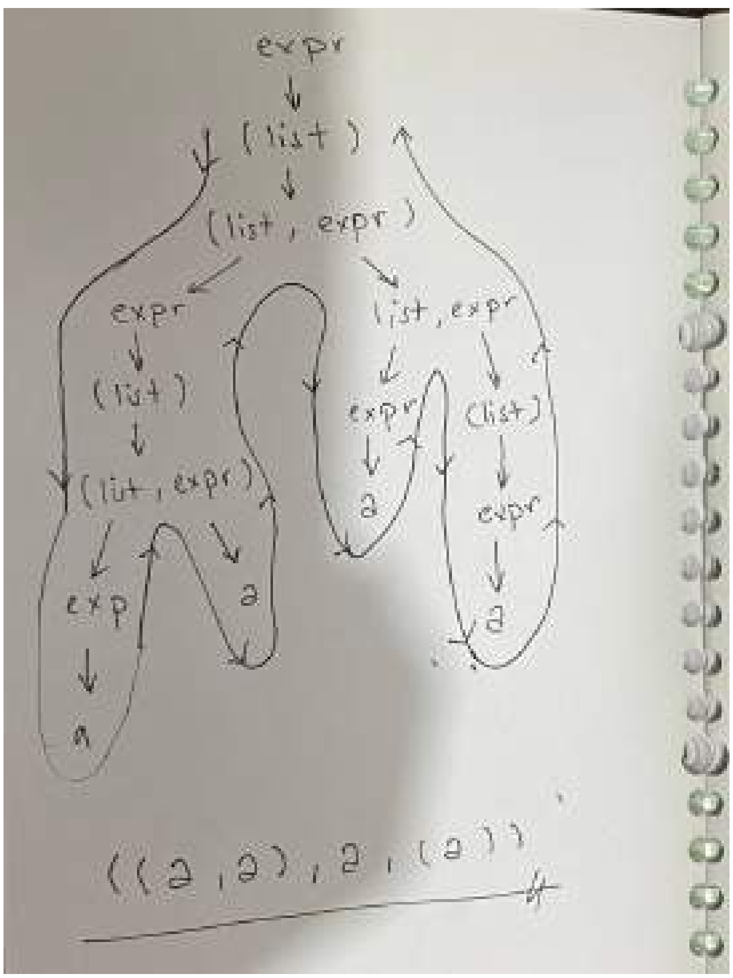
7. Given the following BNF:

expr → (list) | a
list → list , expr | expr

a) Write EBNF and/or syntax diagrams for the languages



b) Draw the parse tree for ((a , a), a, (a))



c) Write a recursive-descent for the language

expr → (list) | a
list → expr [list]