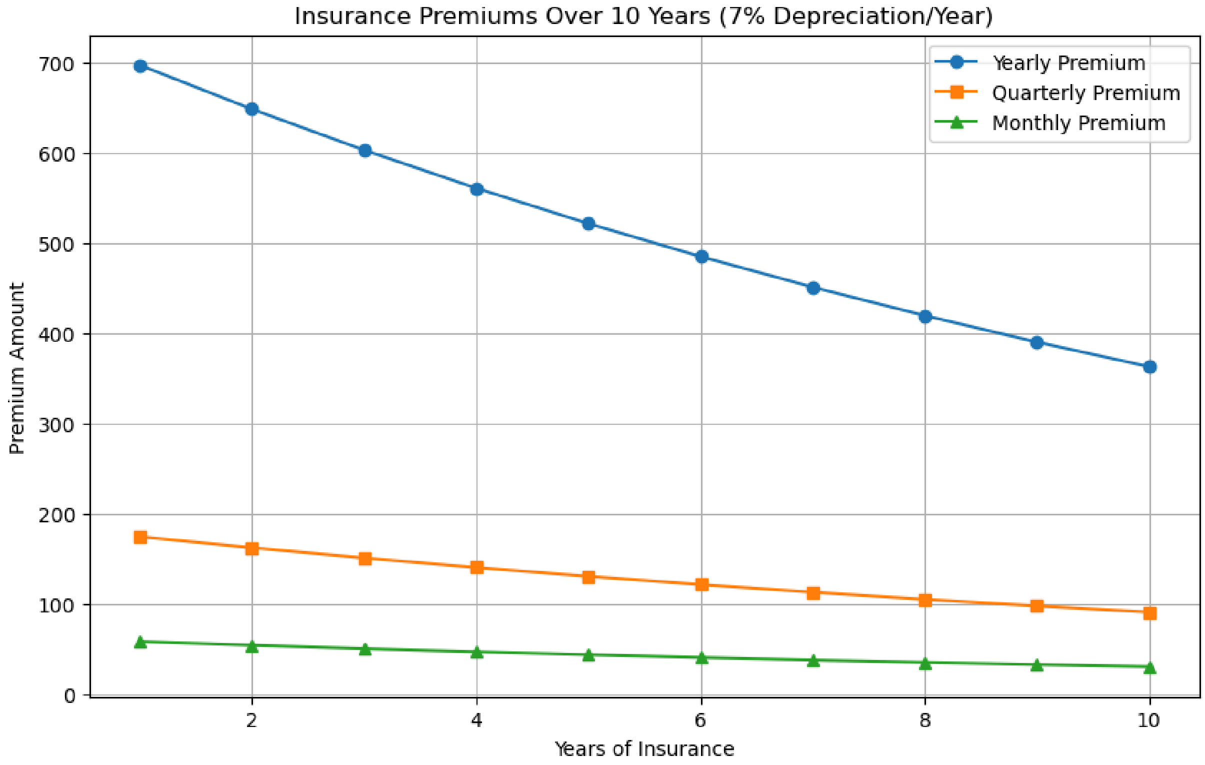
In [1]:

```
import random

# Define a class for a Bank Account
class SavingsAccount:
    def __init__(self, account_number, initial_balance):
        self.account_number = account_number
        self.balance = initial_balance
        self.transactions = []

    def deposit(self, amount):
        self.balance += amount
        self.transactions.append(f"Deposit: {amount}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            self.transactions.append(f"Withdraw: {amount}")
        else:
            self.transactions.append(f"Failed Withdraw: {amount} (Insufficient Balance)")

    def __repr__(self):
        return f"Account {self.account_number}: Balance = {self.balance}"

# Function to generate random bank accounts
def generate_accounts(num_accounts=100, months=12, max_transactions=10, seed_amount=1000):
    random.seed(seed_amount)  # Set the seed for reproducibility
    accounts = []

    for i in range(1, num_accounts + 1):
        # Generate a random initial balance
        initial_balance = random.randint(500, 5000)
        account = SavingsAccount(account_number=i, initial_balance=initial_balance)

        # Simulate random transactions for each account over a number of months
        for _ in range(months):
            for _ in range(random.randint(1, max_transactions)):
                if random.choice([True, False]):
                    account.deposit(random.randint(100, 1000))
                else:
                    account.withdraw(random.randint(100, 1000))

        accounts.append(account)

    # Sort accounts by balance (from lowest to highest)
    accounts.sort(key=lambda x: x.balance)

    return accounts

# Generate 100 accounts
accounts = generate_accounts()

# Print all accounts with their final balance
accounts
```

```
Out[1]: [Account 83: Balance = 14,
         Account 23: Balance = 105,
         Account 60: Balance = 214,
         Account 69: Balance = 431,
         Account 59: Balance = 449,
         Account 52: Balance = 633,
         Account 54: Balance = 807,
         Account 95: Balance = 825,
         Account 57: Balance = 832,
         Account 41: Balance = 880,
         Account 34: Balance = 913,
         Account 55: Balance = 1126,
         Account 85: Balance = 1210,
         Account 74: Balance = 1268,
         Account 77: Balance = 1331,
         Account 2: Balance = 1443,
         Account 64: Balance = 1603,
         Account 5: Balance = 1618,
         Account 43: Balance = 1632,
         Account 44: Balance = 1721,
         Account 25: Balance = 1911,
         Account 45: Balance = 1960,
         Account 39: Balance = 1989,
         Account 67: Balance = 2046,
         Account 42: Balance = 2176,
         Account 27: Balance = 2259,
         Account 1: Balance = 2281,
         Account 84: Balance = 2389,
         Account 40: Balance = 2474,
         Account 81: Balance = 2541,
         Account 31: Balance = 2601,
         Account 92: Balance = 2683,
         Account 35: Balance = 2776,
         Account 68: Balance = 2802,
         Account 22: Balance = 2973,
         Account 66: Balance = 3067,
         Account 24: Balance = 3105,
         Account 12: Balance = 3177,
         Account 99: Balance = 3279,
         Account 97: Balance = 3314,
         Account 75: Balance = 3391,
         Account 94: Balance = 3397,
         Account 50: Balance = 3467,
         Account 13: Balance = 3471,
         Account 9: Balance = 3496,
         Account 20: Balance = 3546,
         Account 76: Balance = 3775,
         Account 53: Balance = 3849,
         Account 78: Balance = 4021,
         Account 21: Balance = 4064,
         Account 30: Balance = 4067,
         Account 58: Balance = 4150,
         Account 6: Balance = 4169,
         Account 36: Balance = 4746,
         Account 48: Balance = 4838,
         Account 87: Balance = 4841,
         Account 38: Balance = 4854,
         Account 82: Balance = 4879,
         Account 46: Balance = 5098,
         Account 51: Balance = 5392,
         Account 10: Balance = 5417,
         Account 16: Balance = 5461,
         Account 32: Balance = 5472,
         Account 86: Balance = 5667,
         Account 62: Balance = 5734,
         Account 91: Balance = 5969,
         Account 37: Balance = 6111,
         Account 17: Balance = 6291,
         Account 98: Balance = 6402,
         Account 49: Balance = 6522,
         Account 33: Balance = 6549,
         Account 11: Balance = 6603,
         Account 47: Balance = 6646,
         Account 65: Balance = 6665,
         Account 61: Balance = 6696,
         Account 7: Balance = 6730,
         Account 96: Balance = 6788,
         Account 29: Balance = 6848,
         Account 28: Balance = 6885,
         Account 93: Balance = 7163,
         Account 8: Balance = 7379,
         Account 4: Balance = 7437,
         Account 71: Balance = 7473,
         Account 15: Balance = 7766,
         Account 79: Balance = 7771,
         Account 100: Balance = 7810,
         Account 3: Balance = 7974,
         Account 14: Balance = 8282,
         Account 89: Balance = 8619,
         Account 90: Balance = 8670,
         Account 80: Balance = 9289,
         Account 63: Balance = 9849,
         Account 70: Balance = 9974,
         Account 88: Balance = 10600,
         Account 72: Balance = 10919,
         Account 26: Balance = 11275,
         Account 18: Balance = 11638,
         Account 19: Balance = 12360,
         Account 56: Balance = 12699,
         Account 73: Balance = 13100]
```

In [3]:

```
import matplotlib.pyplot as plt

# Define a class for Insured Vehicle
class InsuredVehicle:
    def __init__(self, initial_value, yearly_premium_rate):
        self.initial_value = initial_value  # Initial value of the vehicle
        self.yearly_premium_rate = yearly_premium_rate  # Premium rate (percentage of the vehi

    # Method to calculate the vehicle's value after depreciation for a given year
    def value_after_years(self, years):
        return self.initial_value * ((1 - 0.07) ** years)

    # Method to calculate the yearly premium based on the vehicle's value for a given year
    def yearly_premium(self, years):
        value = self.value_after_years(years)
        return value * self.yearly_premium_rate

    # Method to calculate the quarterly premium
    def quarterly_premium(self, years):
        return self.yearly_premium(years) / 4

    # Method to calculate the monthly premium
    def monthly_premium(self, years):
        return self.yearly_premium(years) / 12

# Function to generate and plot premiums for a given number of years
def plot_premiums(vehicle, years_of_insurance):
    years = list(range(1, years_of_insurance + 1))
    yearly_premiums = [vehicle.yearly_premium(year) for year in years]
    quarterly_premiums = [vehicle.quarterly_premium(year) for year in years]
    monthly_premiums = [vehicle.monthly_premium(year) for year in years]

    # Plotting the data
    plt.figure(figsize=(10, 6))
    plt.plot(years, yearly_premiums, label='Yearly Premium', marker='o')
    plt.plot(years, quarterly_premiums, label='Quarterly Premium', marker='s')
    plt.plot(years, monthly_premiums, label='Monthly Premium', marker='^')

    # Adding labels and title
    plt.title(f"Insurance Premiums Over {years_of_insurance} Years (7% Depreciation/Year)")
    plt.xlabel("Years of Insurance")
    plt.ylabel("Premium Amount")
    plt.legend()
    plt.grid(True)
    plt.show()

# Example Usage
# Create an InsuredVehicle object with initial vehicle value of $30,000 and premium rate of 2.
vehicle = InsuredVehicle(initial_value=30000, yearly_premium_rate=0.025)

# Plot premiums for 10 years
plot_premiums(vehicle, years_of_insurance=10)
```


Insurance Premiums Over 10 Years (7% Depreciation/Year)
Premium Amount
Years of Insurance
Yearly Premium
Quarterly Premium
Monthly Premium