In [1]:
```
# Define the directory paths for control and parkinson folders
control_folder = "/Users/JOSEPHSPC04292016/Desktop/BE 175/Final Project/p
parkinson_folder = "/Users/JOSEPHSPC04292016/Desktop/BE 175/Final Projec
```

# So to take it step by step, this first cell takes in the folder paths and outputs two master arrays that hold information on the std. deviation of pressure and the condition( control or parkinson) of each patient( or file )

In [2]:
```
import numpy as np

def average_std_deviation(arr):
    # Calculate the standard deviation of the array
    std_deviation = np.std(arr)
    return std_deviation

def calculate_distance_from_center(x_values, y_values, result_array):
    # Calculate the midpoint of x and y ranges
    x_midpoint = (np.max(x_values) + np.min(x_values)) / 2
    y_midpoint = (np.max(y_values) + np.min(y_values)) / 2

    # Calculate distances from center for each point
    for x, y in zip(x_values, y_values):
        distance = np.sqrt((x - x_midpoint)**2 + (y - y_midpoint)**2)
        result_array.append(distance)



def read_data(file_path, x_array, y_array, x1, y1, x2, y2, time_0, time_
    with open(file_path, 'r') as file:
        for line in file:
            data = line.strip().split(';')
            if data[-1] == '0':  # Check if the last element is '0'
                x_array.append(int(data[0]))
                y_array.append(int(data[1]))
                time_0.append(int(data[-2]))
                grip_0.append(int(data[3]))
            if data[-1] == '1':  # Check if the last element is '1'
                x1.append(int(data[0]))
                y1.append(int(data[1]))
                time_1.append(int(data[-2]))
                grip_1.append(int(data[3]))

            if data[-1] == '2':  # Check if the last element is '2'
                x2.append(int(data[0]))
                y2.append(int(data[1]))
                time_2.append(int(data[-2]))
                grip_2.append(int(data[3]))



def masterfunction( file_path , fileID ):
    x_val_0 = []
    y_val_0 = []

    x_val_1 = []
    y_val_1 = []

    x_val_2 = []
    y_val_2 = []

    time_0 = []
    time_1 = []
    time_2 = []

    grip_0 = []
```

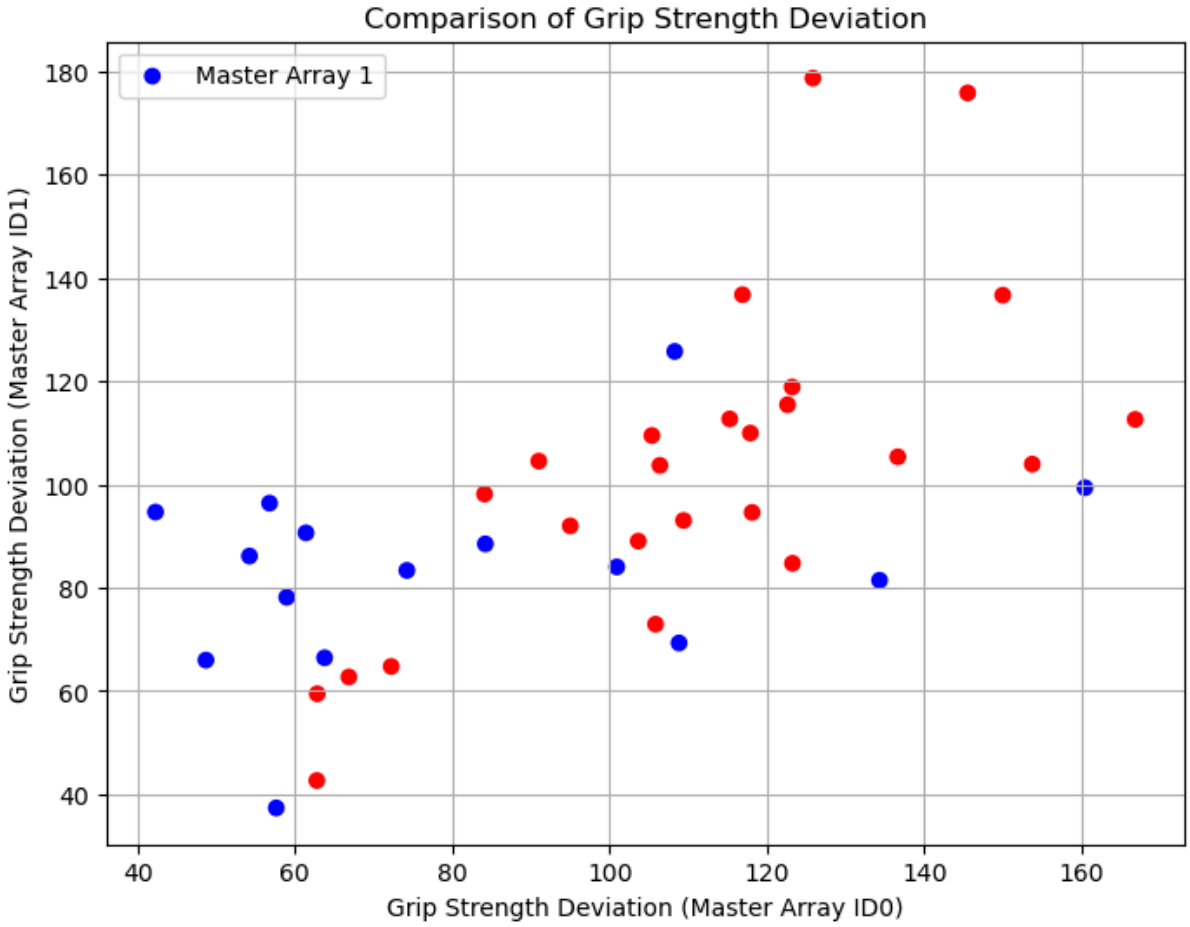
```
    grip_1 = []
    grip_2 = []

    read_data(file_path, x_val_0, y_val_0, x_val_1, y_val_1, x_val_2, y_
    # now we have filled in all of our data into the respective arrays

    if fileID == "0":
        radius_0 = []
        calculate_distance_from_center(x_val_0, y_val_0, radius_0)

        result_neg_slop_prop = 0 #calculate_slope_proportions(time_0, ra
        result_avg_grip_deviation = average_std_deviation(grip_0)
    if fileID == "1":
        radius_1 = []
        calculate_distance_from_center(x_val_1, y_val_1, radius_1)

        result_neg_slop_prop = 0 #calculate_slope_proportions(time_1, ra
        result_avg_grip_deviation = average_std_deviation(grip_1)

    return result_neg_slop_prop, result_avg_grip_deviation




import os

# Define an empty master 2D array to hold patient information BUILDING IL
master_arrayID0 = []

# Function to determine the value of the third column based on folder nam
def determine_label(folder_name):
    if folder_name == "control":
        return 0
    elif folder_name == "parkinson":
        return 1
    else:
        return None

# Iterate through the control folder
for filename in os.listdir(control_folder):
    file_path = os.path.join(control_folder, filename)
    if os.path.isfile(file_path):
        result_neg_slop_prop, result_avg_grip_deviation = masterfunction
        label = determine_label("control")
        master_arrayID0.append([result_neg_slop_prop, result_avg_grip_dev

# Iterate through the parkinson folder
for filename in os.listdir(parkinson_folder):
    file_path = os.path.join(parkinson_folder, filename)
    if os.path.isfile(file_path):
        result_neg_slop_prop, result_avg_grip_deviation = masterfunction
        label = determine_label("parkinson")
        master_arrayID0.append([result_neg_slop_prop, result_avg_grip_dev

# Print the master 2D array
```

```
master_arrayID1 = []

# Function to determine the value of the third column based on folder nar
def determine_label(folder_name):
    if folder_name == "control":
        return 0
    elif folder_name == "parkinson":
        return 1
    else:
        return None

# Iterate through the control folder
for filename in os.listdir(control_folder):
    file_path = os.path.join(control_folder, filename)
    if os.path.isfile(file_path):
        result_neg_slop_prop, result_avg_grip_deviation = masterfunction
        label = determine_label("control")
        master_arrayID1.append([result_neg_slop_prop, result_avg_grip_dev

# Iterate through the parkinson folder
for filename in os.listdir(parkinson_folder):
    file_path = os.path.join(parkinson_folder, filename)
    if os.path.isfile(file_path):
        result_neg_slop_prop, result_avg_grip_deviation = masterfunction
        label = determine_label("parkinson")
        master_arrayID1.append([result_neg_slop_prop, result_avg_grip_dev

# Print the master 2D array
print("Master 2D Array:")
# for row in master_arrayID1:
#     print(row)


# print("Master 2D Array:")
# for row in master_arrayID0:
#     print(row)
```

```
Master 2D Array:
```

# then with the two "master arrays" corresponding to each test ID 0 ( dynamic or spiral test) it will then look to plot the std. dev of pressure of each test ID against each other, while labeling which patients are control or parkinson's by the color

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

# Assuming master_arrayID0 and master_arrayID1 are defined

master_array_1 = master_arrayID0
master_array_2 = master_arrayID1

# Extract grip strength deviation from master arrays
grip_deviation_1 = np.array([row[1] for row in master_array_1]).reshape(
grip_deviation_2 = np.array([row[1] for row in master_array_2]).reshape(

# Flatten grip_deviation_2 to avoid DataConversionWarning
grip_deviation_2 = grip_deviation_2.ravel()

# Extract labels from master arrays
labels_1 = [row[2] for row in master_array_1]
labels_2 = [row[2] for row in master_array_2]

# Define colors based on labels
colors_1 = ['blue' if label == 0 else 'red' for label in labels_1]
colors_2 = ['green' if label == 0 else 'blue' for label in labels_2]

# Plot data
plt.figure(figsize=(8, 6))
plt.scatter(grip_deviation_1, grip_deviation_2, c=colors_1, marker='o',
plt.title('Comparison of Grip Strength Deviation')
plt.xlabel('Grip Strength Deviation (Master Array ID0)')
plt.ylabel('Grip Strength Deviation (Master Array ID1)')
plt.legend()
plt.grid(True)
plt.show()


print(colors_1)
```




```
['blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blu
e', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'red', 'red', 're
d', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 're
d', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 're
d', 'red', 'red']
```

## these next two cells look to parse the arrays regarding color and the std. deviations, into a more usable form that the SVM function can take in, essentially just reformatting!

In [4]:
```
# color_ar = ['blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'blue', 'b

# Convert colors to numerical values
numerical_array = [0 if color == 'blue' else 1 for color in colors_1]

# Append 0 to the numerical array
# numerical_array.append(0)

# Print the length of the numerical array
print("Length of numerical array:", len(numerical_array))


print(numerical_array)
```

```
Length of numerical array: 40
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

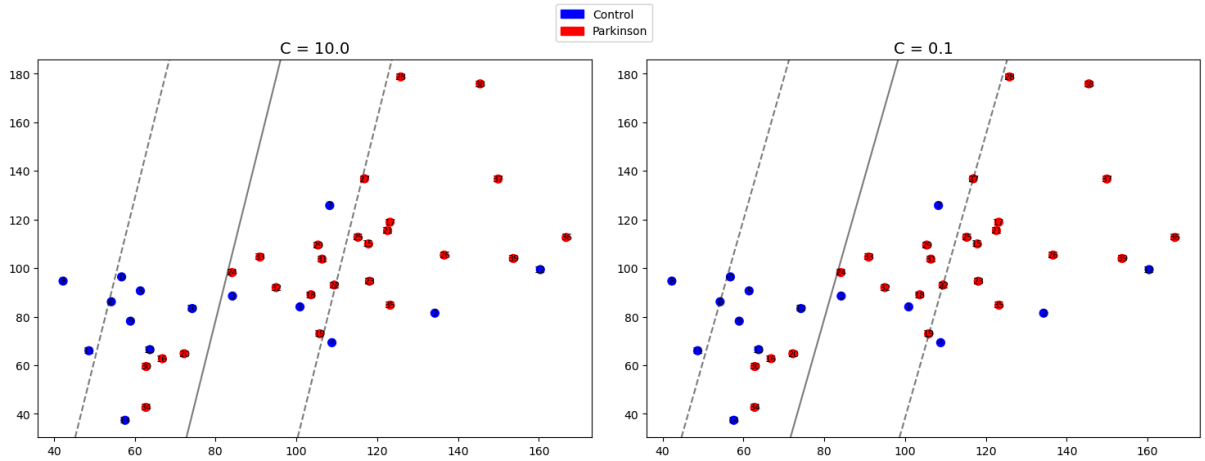
In [5]:
```
import numpy as np
import matplotlib.pyplot as plt

# Assuming grip_deviation_1 and grip_deviation_2 are defined

# Convert grip deviation arrays to one-dimensional numpy arrays
grip_deviation_1 = np.ravel(grip_deviation_1)
grip_deviation_2 = np.ravel(grip_deviation_2)

# Combine grip deviations into one big 2D array
combined_array = np.column_stack((grip_deviation_1, grip_deviation_2))

# Plot the graph
# plt.scatter(combined_array[:, 0], combined_array[:, 1])
# plt.xlabel('Grip Strength Deviation (ID 0)')
# plt.ylabel('Grip Strength Deviation (ID 1)')
# plt.title('Comparison of Grip Strength Deviation')
# plt.grid(True)
# plt.show()

print(combined_array)
```

```
[[ 56.77315768  96.37497372]
 [100.91022406  84.0334577 ]
 [134.303481    81.44286085]
 [ 84.20098324  88.48953288]
 [108.82030568  69.30592668]
 [ 61.403805    90.62500651]
 [ 54.22182406  86.13772141]
 [108.2692326  125.74136469]
 [ 42.27846714  94.64909324]
 [ 58.96029062  78.16058026]
 [ 74.22848566  83.35004943]
 [ 48.68896221  65.98686002]
 [160.40326366  99.36571537]
 [ 63.77263137  66.44717921]
 [ 57.62499538  37.38036058]
 [117.8877146  109.93085098]
 [ 66.85199151  62.73316822]
 [123.19680632 118.86393949]
 [103.63480362  89.02435569]
 [105.82783018  72.9508759 ]
 [ 72.23328176  64.77232856]
 [122.60438284 115.41738226]
 [109.39284744  93.02392928]
 [118.10510402  94.57737302]
 [ 84.09743316  98.141353  ]
 [115.26093716 112.67288449]
 [136.62203951 105.34091362]
 [116.88445617 136.72627941]
 [125.83459999 178.64248242]
 [105.36797685 109.47161584]
 [ 62.83494336  59.48005648]
 [106.38200878 103.68640994]
 [ 94.99210611  91.97590349]
 [ 90.99283576 104.51501638]
 [ 62.78287367  42.68302725]
 [123.24003119  84.72805978]
 [166.79567206 112.58531665]
 [149.96443175 136.62902596]
 [145.50412279 175.77991856]
 [153.71023929 103.93404364]]
```

# Lastly we are throwing our data into the SVM model with soft margins and then creating a simple function that uses the parameters of our decision boundary to predict if someone has parkinson, with a confidence interval that is based on how far from the decision boundary it is

In [6]:
```
from sklearn.datasets import make_blobs
from sklearn.svm import SVC
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

# Assuming combined_array and numerical_array are defined somewhere
X, y = combined_array, numerical_array

def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none')
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)

for axi, C in zip(ax, [10.0, 0.1]):
    model = SVC(kernel='linear', C=C).fit(X, y)
    # Change yellow points to red and red points to blue
    colors = ['blue' if label == 0 else 'red' for label in y]
    axi.scatter(X[:, 0], X[:, 1], c=colors, s=50)
    plot_svc_decision_function(model, axi)
    axi.scatter(model.support_vectors_[:, 0],
                model.support_vectors_[:, 1],
                s=300, lw=1, facecolors='none')

    # Label each point with its index
    for i, (x, y_val) in enumerate(zip(X[:, 0], X[:, 1])):
        axi.text(x, y_val, i, color='black', fontsize=8, ha='center', va

    axi.set_title('C = {0:.1f}'.format(C), size=14)

# Create legend
legend_elements = [
```

```
    mpatches.Patch(color='blue', label='Control'),
    mpatches.Patch(color='red', label='Parkinson')
]

fig.legend(handles=legend_elements, loc='upper center')

plt.show()
```




In [9]:
```
def predict_parkinson_with_confidence(model, x_coord, y_coord):
    """Predict whether a new point belongs to Parkinson's class or not."""
    # Reshape the coordinates into a 2D array
    new_point = np.array([[x_coord, y_coord]])
    # Use the decision function to get the signed distance to the hyperp
    distance_to_hyperplane = model.decision_function(new_point)
    # Compute confidence based on the signed distance
    confidence = np.abs(distance_to_hyperplane)
    # Use the trained model to predict the class of the new point
    prediction = model.predict(new_point)
    return prediction[0], confidence[0]

# Example usage:
new_x = 80.5  # Example x coordinate of the new point
new_y = 100.0  # Example y coordinate of the new point

# Assuming 'model' is your trained SVM model
prediction, confidence = predict_parkinson_with_confidence(model, new_x,
print("Prediction for point ({}, {}): {}, Confidence: {:.2f}".format(new_
```

```
Prediction for point (80.5, 100.0): Control, Confidence: 0.11
```

In [ ]:

In [ ]: