LẬP TRÌNH ÂM THANH Spring 2022 - D19PT			
Nhóm Bài tập lớn: SNDPROJSP2205	**Bài:** Project #01		
Họ và tên: Phạm Thị Hương	Mã sinh viên: B19DCPT117	Nhóm lớp: 01	**Điểm đánh giá 10**
Họ và tên : Nguyễn Thị Linh	Mã sinh viên: B19DCPT140	Nhóm lớp: 01	**Điểm đánh giá 10**
Họ và tên : Nguyễn Văn Sang (NT)	Mã sinh viên: B19DCPT190	Nhóm lớp: 01	**Điểm đánh giá 10**
Mức độ hoàn thành: 100%			
Đánh giá cụ thể			

Hàm	Cài đặt	Script sử dụng
Giảm tần số lấy mẫu	Hương, Sang	C++
Tăng tần số lấy mẫu	Hương, Sang	C++
Đảo thời gian	Hương, Sang	C++
Dịch thời gian	Hương, Sang	C++
Nhân hằng số	Linh, Sang	C++
Cộng tín hiệu	Linh, Sang	C++
Trừ tín hiệu	Linh, Sang	C++
Nhân tín hiệu	Linh, Sang	C++
Tích chập	Sang	C++
Tương quan chéo	Sang	C++

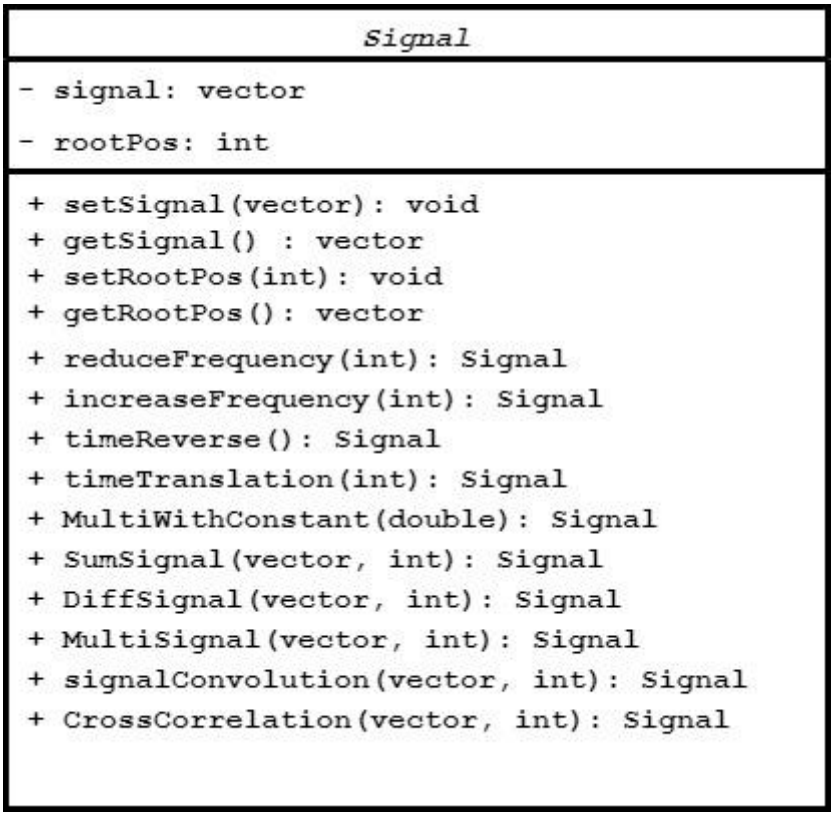
1. Cách thiết kế xây dựng đối tượng:

Với yêu cầu đề bài là "Xây dựng các lớp đối tượng mô tả tín hiệu rời rạc và các phép toán với tín hiệu rời rạc", chúng em hướng tới xây dựng đối tượng ***Signal*** với 2 thuộc tính: *vector<double> signal* – dùng để lưu trữ các giá trị tín hiệu và *int rootPos* – lưu trữ vị trí gốc tọa độ trong dãy tín hiệu

Các phép toán với tín hiệu bao gồm: cộng, trừ, nhân với hằng số, nhân tín hiệu với tín hiệu, tích chập, tương quan chéo, phép dịch thời gian, đảo thời gian, tăng giảm tần số.

Toàn bộ các phép toán đều được trả về kiểu ***Signal***

2. UML cho lớp Signal:



3. Các phần code chính trong cài đặt & thực hiện minh họa sử dụng kèm theo các giải thích

a. Phương thức giảm tần số lấy mẫu (Down sampling - reduceFrequency)

```
Signal reduceFrequency(int m)
{
    /*

       Tang hay giam 1 thi van la chinh no

     */
    vector<double> temp = getSignal();
    int rootPos = getRootPos();
    int rootRes_pos = rootPos;
    vector<double> filterBe;
    vector<double> filterAf;
    int count_del = 0;
    for (int i = rootPos; i >= 0; i -= m){
        if (i != rootPos){
            filterBe.push_back(temp[i]);
            count_del++;
        }
    }
    rootRes_pos = count_del;
    reverse(filterBe.begin(), filterBe.end());
    for (int i = rootPos; i < temp.size(); i += m)
        filterAf.push_back(temp[i]);

    vector<double> res(filterBe);
    res.insert(res.end(), filterAf.begin(), filterAf.end());

    return {res, rootRes_pos};
}
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Giá trị đầu vào là ***m*** – là giá trị nguyên thể hiện cho giá trị giảm bao nhiêu lần tín hiệu của tần số
- Cách hoạt động của đoạn code này như sau:
    - Có hai vector filterBe và filter Af dùng để chứa các giá trị tín hiệu trong tín hiệu gốc
    - rootRes_pos là biến dùng để lưu vị trí gốc mới khi thực hiện giàm tần số lấy mẫu. Biến này tương ứng với số các giá trị được lưu vào vector filterBe
    - Sau khi lưu được các giá trị vào 2 vector filterBe và filterAf thì vector res có nhiệm vụ gom chúng lại. Lưu ý, trước khi được nối lại với nhau thì filterBe phải được reverse (đảo ngược các giá trị trong vector)

b. Phương thức tăng tần số lấy mẫu (Up sampling - increaseFrequency)

```
Signal increaseFrequency(int m)
{
    vector<double> temp = getSignal();
    vector<double> res;
    int rootPos = getRootPos();
    int rootRes_pos = rootPos;
    for (int i = 0; i < temp.size() - 1; i++)
    {
        res.push_back(temp[i]);
        for (int j = 1; j < m; j++){
            if(i < rootPos) rootRes_pos++;
            res.push_back(0);
        }

    }
    res.push_back(temp[temp.size() - 1]);
    return {res, rootRes_pos};
}
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Giá trị đầu vào là ***m*** – là giá trị nguyên thể hiện cho giá trị tăng bao nhiêu lần tín hiệu của tần số
- Cách hoạt động của đoạn code này như sau:
    - rootRes_pos là biến dùng để lưu vị trí gốc mới khi thực hiện tăng tần số lấy mẫu. Biến này tương ứng với số các giá trị đứng trước rootPos khi được chèn thêm 0 vào tín hiệu
    - res là vector lưu các giá trị tín hiệu cũ và đồng thời chèn thêm các giá trị 0 mới vào.

c. Phương thức đảo thời gian (timeReverse)

```
Signal timeReverse()
{
    /*
        x[n] --> x[-n]
        lat x[n] qua truc tung
     */
    vector<double> temp = getSignal();
    // vector<double> temp_verCopy{temp.begin(), temp.end()};

    int rootPos = getRootPos();
    reverse(temp.begin(), temp.end());
    vector<double> res{temp.begin(), temp.end()};
    int new_rootPos = temp.size()-rootPos-1;
    if(new_rootPos < temp.size()-new_rootPos-1){
        for(int i = 0 ; i < (temp.size()-new_rootPos-1) - new_rootPos; i++)
res.insert(res.begin(), 0);
    }else if(new_rootPos > temp.size()-new_rootPos-1){
        for(int i = 0 ; i < new_rootPos - (temp.size()-new_rootPos-1); i++)
res.push_back(0);
    }
    return {res, new_rootPos};
}
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Không có tham số đầu vào
- Cách hoạt động của đoạn code này như sau:
    - Temp là biến chứa tín hiệu gốc, rootPos là biến chứa vị trí gốc tọa độ trong tín hiệu gốc
    - Đảo ngược temp
    - Vector res copy temp và sau đó chèn thêm các giá trị 0 vào đầu hoặc cuối (cho trường hợp vị trí gốc tọa độ không ở chính giữa đoạn tín hiệu đang xét). Và do đó new_rootPos mới cũng được cập nhật

d. Phương thức dịch thời gian (timeTranslation)

```
Signal timeTranslation(int m)
{

    /*
        x[n-1] => m = 1 => tre mot mau thi dich sang phai mot mau
        va nguoc lai
        suy ra m duong thi insert, am thi push_back
     */

    vector<double> temp = getSignal();
    vector<double> res(temp.size(), 0);
    if (m > 0)
        for (int i = 0; i < m; i++)
            temp.insert(temp.begin(), 0);
    else
        for (int i = 0; i < abs(m); i++)
            temp.push_back(0);

    if (m > 0)
    {
```

```
        for (int i = 0; i < temp.size(); i++)
        {
            if (i >= res.size() && temp[i] != 0)
                res.push_back(temp[i]);
            else if (i < res.size())
                res[i] = temp[i];
        }
    }
    else
    {
        int run = 0;
        for (int i = abs(m); i < temp.size(); i++)
            res[run++] = temp[i];
    }
    return {res, getRootPos()};
}
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Tham số đầu vào là ***m*** – là giá trị nguyên thể hiện cho giá trị dịch tín hiệu của tần số (m dương thì dịch phải và ngược lại)
- Cách hoạt động của đoạn code này như sau:
    - Với lý thuyết về dịch thời gian như trên, chúng em vận dụng hai hàm cơ bản đó là insert và push_back đối với vector. Nếu m dương – trễ m mẫu => dịch sang phải m mẫu và ngược lại.
    - Dịch phải thì insert m số 0, dịch trái thì push_back m số 0 và sau đó xem xét xem liệu dãy gốc có các giá trị khác 0 bị vượt ra ngoài vector lưu quá trình insert và push_back vừa rồi thì phải giữ lại (vì mục đích demo kết quả)

e. Phương thức nhân tín hiệu với một hằng số (MultiWithConstant)

```
Signal MultiWithConstant(double c)
{
    vector<double> temp = getSignal();
```

```
    for (auto &x : temp){
        if(x!=0)
            x *= c;
    }

    return {temp, getRootPos()};
}
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Tham số đầu vào là ***c*** – là giá trị số thực thể hiện cho giá trị hằng số nhân với tín hiệu
- Cách hoạt động của đoạn code này như sau:
    - Nhân lần lượt giá trị c vào các giá trị vector chứa các giá trị tín hiệu gốc
    - Nghiễm nhiên vị trí gốc tọa không thay đổi

f. Phương thức tính tổng tín hiệu (SumSignal)

```
Signal SumSignal(vector<double> signal_ex, int rootPosition)
{
    vector<double> itsSignal = getSignal();
    int itsRoot = getRootPos();
    int rootRes = itsRoot;
    vector<double> res;

    vector<double> signalFirstBe{itsSignal.begin(), itsSignal.begin() + itsRoot};
    int d1 = signalFirstBe.size();

    vector<double> signalFirstAf{itsSignal.begin() + itsRoot, itsSignal.end()};
    int d2 = signalFirstAf.size();
```

```
    vector<double> signalSecondBe{signal_ex.begin(), signal_ex.begin() +
rootPosition};
    int d3 = signalSecondBe.size();


    vector<double> signalSecondAf{signal_ex.begin() + rootPosition,
signal_ex.end()};
    int d4 = signalSecondAf.size();


    int dis1 = abs(d1 - d3);


    if (signalFirstBe.size() < signalSecondBe.size())
    {
       for (int i = 0; i < dis1; i++){
          signalFirstBe.insert(signalFirstBe.begin(), 0);
          rootRes++;
       }


    }
    else if (signalFirstBe.size() > signalSecondBe.size())
    {
       for (int i = 0; i < dis1; i++)
          signalSecondBe.insert(signalSecondBe.begin(), 0);
    }


    int dis2 = abs(d2 - d4);


    if (signalFirstAf.size() < signalSecondAf.size())
    {
       for (int i = 0; i < dis2; i++)
```

```
            signalFirstAf.push_back(0);
        }
        else if (signalFirstAf.size() > signalSecondAf.size())
        {
            for (int i = 0; i < dis2; i++)
                signalSecondAf.push_back(0);
        }

        vector<double> total1(signalFirstBe);
        total1.insert(total1.end(), signalFirstAf.begin(), signalFirstAf.end());
        vector<double> total2(signalSecondBe);
        total2.insert(total2.end(), signalSecondAf.begin(), signalSecondAf.end());

        for (int i = 0; i < total1.size(); i++) res.push_back(total1[i] + total2[i]);

        return {res, rootRes};
    }
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Tham số đầu vào vector – chứa giá trị tín hiệu và int rootPosition chứa vị trí gốc tọa độ của tín hiệu truyền vào
- Cách hoạt động của đoạn code này như sau:
    - Mấu chốt để thực hiện phép toán này là cần code sao cho vị trí 2 gốc tọa độ trùng nhau. Do đó chúng em xây dụng các vector lần lượt là signalFirstBe, signalFirstAf, signalSecondBe, signalSeconAdAdff chứa các giá trí tín hiệu của 2 tín hiệu ở 2 nửa (trước và sau vị trí gốc tọa độ trong tín hiệu). Sau đó so sánh độ dài của chúng để insert hoặc push_back thêm các giá trị 0 nhằm sau này khi nối hai vector tương ứng (signalFirstBe – signalFirstAf; signalSecondBe – signalSecondAf) để chúng có độ dài giống nhau
    - Việc còn lại sau khi có được hai vector với độ dài giống nhau và vị trí gốc tọa độ trùng nhau là cộng các giá trị tương ứng

g. Phương thức tính hiệu tín hiệu (DiffSignal)

```
Signal DiffSignal(vector<double> signal_ex, int rootPosition)
{
    vector<double> itsSignal = getSignal();
    int itsRoot = getRootPos();
    int rootRes = itsRoot;
    vector<double> res;
    vector<double> signalFirstBe{itsSignal.begin(), itsSignal.begin() + itsRoot};
    int d1 = signalFirstBe.size();

    vector<double> signalFirstAf{itsSignal.begin() + itsRoot, itsSignal.end()};
    int d2 = signalFirstAf.size();

    vector<double> signalSecondBe{signal_ex.begin(), signal_ex.begin() +
rootPosition};
    int d3 = signalSecondBe.size();

    vector<double> signalSecondAf{signal_ex.begin() + rootPosition,
signal_ex.end()};
    int d4 = signalSecondAf.size();

    int dis1 = abs(d1 - d3);

    if (signalFirstBe.size() < signalSecondBe.size())
    {
        for (int i = 0; i < dis1; i++){
            signalFirstBe.insert(signalFirstBe.begin(), 0);
            rootRes++;
        }

    }
```

```
    else if (signalFirstBe.size() > signalSecondBe.size())
    {
        for (int i = 0; i < dis1; i++)
            signalSecondBe.insert(signalSecondBe.begin(), 0);
    }

    int dis2 = abs(d2 - d4);

    if (signalFirstAf.size() < signalSecondAf.size())
    {
        for (int i = 0; i < dis2; i++)
            signalFirstAf.push_back(0);
    }
    else if (signalFirstAf.size() > signalSecondAf.size())
    {
        for (int i = 0; i < dis2; i++)
            signalSecondAf.push_back(0);
    }

    vector<double> total1(signalFirstBe);
    total1.insert(total1.end(), signalFirstAf.begin(), signalFirstAf.end());
    vector<double> total2(signalSecondBe);
    total2.insert(total2.end(), signalSecondAf.begin(), signalSecondAf.end());

    for (int i = 0; i < total1.size(); i++)
        res.push_back(total1[i] - total2[i]);
    return {res, rootRes};
}
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Tham số đầu vào vector – chứa giá trị tín hiệu và int rootPosition chứa vị trí gốc tọa độ của tín hiệu truyền vào
- Cách hoạt động của đoạn code này như sau:
    - Mấu chốt để thực hiện phép toán này là cần code sao cho vị trí 2 gốc tọa độ trùng nhau. Do đó chúng em xây dựng các vector lần lượt là signalFirstBe, signalFirstAf, signalSecondBe, signalSeconAdAdff chứa các giá trí tín hiệu của 2 tín hiệu ở 2 nửa (trước và sau vị trí gốc tọa độ trong tín hiệu). Sau đó so sánh độ dài của chúng để insert hoặc push_back thêm các giá trị 0 nhằm sau này khi nối hai vector tương ứng (signalFirstBe – signalFirstAf; signalSecondBe – signalSecondAf) để chúng có độ dài giống nhau
    - Việc còn lại sau khi có được hai vector với độ dài giống nhau và vị trí gốc tọa độ trùng nhau là trừ các giá trị tương ứng

h. Phương thức tính tích tín hiệu (MultiSignal)

```
Signal MultiSignal(vector<double> signal_ex, int rootPosition)
{
    vector<double> itsSignal = getSignal();
    int itsRoot = getRootPos();
    int rootRes = itsRoot;
    vector<double> res;

    vector<double> signalFirstBe{itsSignal.begin(), itsSignal.begin() + itsRoot};
    int d1 = signalFirstBe.size();

    vector<double> signalFirstAf{itsSignal.begin() + itsRoot, itsSignal.end()};
    int d2 = signalFirstAf.size();

    vector<double> signalSecondBe{signal_ex.begin(), signal_ex.begin() +
rootPosition};
    int d3 = signalSecondBe.size();
```

```
    vector<double> signalSecondAf{signal_ex.begin() + rootPosition,
signal_ex.end()};
    int d4 = signalSecondAf.size();


    int dis1 = abs(d1 - d3);


    if (signalFirstBe.size() < signalSecondBe.size())
    {
       for (int i = 0; i < dis1; i++){
          signalFirstBe.insert(signalFirstBe.begin(), 0);
          rootRes++;
       }


    }
    else if (signalFirstBe.size() > signalSecondBe.size())
    {
       for (int i = 0; i < dis1; i++)
          signalSecondBe.insert(signalSecondBe.begin(), 0);
    }


    int dis2 = abs(d2 - d4);


    if (signalFirstAf.size() < signalSecondAf.size())
    {
       for (int i = 0; i < dis2; i++)
          signalFirstAf.push_back(0);
    }
    else if (signalFirstAf.size() > signalSecondAf.size())
    {
```

```
        for (int i = 0; i < dis2; i++)
            signalSecondAf.push_back(0);
    }

    vector<double> total1(signalFirstBe);
    total1.insert(total1.end(), signalFirstAf.begin(), signalFirstAf.end());
    vector<double> total2(signalSecondBe);
    total2.insert(total2.end(), signalSecondAf.begin(), signalSecondAf.end());

    for (int i = 0; i < total1.size(); i++)
        res.push_back(total1[i] * total2[i]);
    return {res, rootRes};
}
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Tham số đầu vào vector – chứa giá trị tín hiệu và int rootPosition chứa vị trí gốc tọa độ của tín hiệu truyền vào
- Cách hoạt động của đoạn code này như sau:
    - Mấu chốt để thực hiện phép toán này là cần code sao cho vị trí 2 gốc tọa độ trùng nhau. Do đó chúng em xây dựng các vector lần lượt là signalFirstBe, signalFirstAf, signalSecondBe, signalSeconAdAdff chứa các giá trí tín hiệu của 2 tín hiệu ở 2 nửa (trước và sau vị trí gốc tọa độ trong tín hiệu). Sau đó so sánh độ dài của chúng để insert hoặc push_back thêm các giá trị 0 nhằm sau này khi nối hai vector tương ứng (signalFirstBe – signalFirstAf; signalSecondBe – signalSecondAf) để chúng có độ dài giống nhau
    - Việc còn lại sau khi có được hai vector với độ dài giống nhau và vị trí gốc tọa độ trùng nhau là nhân các giá trị tương ứng

i. Phương thức tính tích chập tín hiệu (signalConvolution)

```
Signal signalConvolution(vector<double> signal_sample, int rootPos_sample)
{
    vector<double> itsSignal = getSignal();
```

```
int itsRoot = getRootPos();
int row = itsSignal.size();
int col = signal_sample.size();


// xuat cac phan tu nam phia tren duong cheo chinh

vector<vector<double>> matrix;
for (int i = 0; i < row; i++)
{
    vector<double> temp;
    for (int j = 0; j < col; j++)
    {
        temp.push_back(itsSignal[i] * signal_sample[j]);
    }
    matrix.push_back(temp);
}

// cout << "Cac phan tu nam phia tren duong cheo chinh la: " << endl;

vector<double> res;
vector<vector<vector<int>>> checkRoot;
int rootRes = 0;

for (int i = 0; i < row; i++)
{
    int temp = i;
    double sum = 0;
    vector<vector<int>> temp_checkR;
```

```
        for (int j = 0; j <= i; j++)
        {
            temp_checkR.push_back({temp, j});
            sum += matrix[temp][j];
            temp--;
        }
        res.push_back(sum);
        checkRoot.push_back(temp_checkR);
    }

    for (int i = 1; i < col; i++)
    {
        int temp = row-1;
        int sum = 0;
        vector<vector<int>> temp_checkR;
        for (int j = i; j < col; j++)
        {
            temp_checkR.push_back({temp, j});
            sum += matrix[temp][j];
            temp--;
        }
        checkRoot.push_back(temp_checkR);
        res.push_back(sum);
    }
    // for(auto x : res)   cout << x << ' ';
    for (int i = 0; i < checkRoot.size(); i++)
    {
        bool flag = true;
        for (int j = 0; j < checkRoot[i].size(); j++)
```

```
        {
            if(itsRoot == checkRoot[i][j][0] && rootPos_sample == checkRoot[i][j][1]){
                rootRes = i;
                flag = false;
                break;
            }
        }
        if(!flag)   break;
    }
    return {res, rootRes};
}
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Tham số đầu vào vector – chứa giá trị tín hiệu và int rootPosition chứa vị trí gốc tọa độ của tín hiệu truyền vào
- Cách hoạt động của đoạn code này như sau:
    - Tư tưởng xây dựng đoạn code xử lý này theo cách giải bài toán khi xây dựng bảng
    - Xây dựng vector<vector<double>> matrix chứa các giá trị khi nhân 2 tín hiệu tương ứng
    - Vector<vector<vector<int>>> checkRoot chứa các chỉ số theo đường chéo phụ trong ma trận (việc lưu trữ các chỉ số trong ma trận này giúp tìm ra đâu là giá trị nằm tại vị trí gốc tọa độ của dãy tín hiệu kết quả)

j. Phương thức tính tương quan chéo tín hiệu (CrossCorreclation)

```
Signal CrossCorrelation(vector<double> signal_sample, int rootPos_sample)
{
    vector<double> itsSignal = getSignal();

    Signal s{{0, 0, 1, 1, 0, 0}, 2};
    Signal secondSign = s.timeReverse();
```

```
vector<double> signal_sample_ver2 = secondSign.getSignal();

int rootPos_sample_ver2 = secondSign.getRootPos();

int itsRoot = getRootPos();
int row = itsSignal.size();
int col = signal_sample_ver2.size();



// xuat cac phan tu nam phia tren duong cheo chinh

vector<vector<double>> matrix;
for (int i = 0; i < row; i++)
{
    vector<double> temp;
    for (int j = 0; j < col; j++)
    {
        temp.push_back(itsSignal[i] * signal_sample_ver2[j]);
    }
    matrix.push_back(temp);
}

// cout << "Cac phan tu nam phia tren duong cheo chinh la: " << endl;

vector<double> res;
vector<vector<vector<int>>> checkRoot;
int rootRes = 0;

for (int i = 0; i < row; i++)
```

```
    {
        int temp = i;
        double sum = 0;
        vector<vector<int>> temp_checkR;
        for (int j = 0; j <= i; j++)
        {
            temp_checkR.push_back({temp, j});
            sum += matrix[temp][j];
            temp--;
        }
        res.push_back(sum);
        checkRoot.push_back(temp_checkR);
    }

    for (int i = 1; i < col; i++)
    {
        int temp = row-1;
        int sum = 0;
        vector<vector<int>> temp_checkR;
        for (int j = i; j < col; j++)
        {
            temp_checkR.push_back({temp, j});
            sum += matrix[temp][j];
            temp--;
        }
        checkRoot.push_back(temp_checkR);
        res.push_back(sum);
    }
    // for(auto x : res)   cout << x << ' ';
```

```
    for (int i = 0; i < checkRoot.size(); i++)
    {
        bool flag = true;
        for (int j = 0; j < checkRoot[i].size(); j++)
        {
            if(itsRoot == checkRoot[i][j][0] && rootPos_sample_ver2 ==
checkRoot[i][j][1]){
                rootRes = i;
                flag = false;
                break;
            }
        }
        if(!flag)   break;
    }
    return {res, rootRes};
  }
```

Giải thích:

- Phương thức này trả về dữ liệu kiểu ***Signal***
- Tham số đầu vào vector – chứa giá trị tín hiệu và int rootPosition chứa vị trí gốc tọa độ của tín hiệu truyền vào
- Cách hoạt động của đoạn code này như sau:
    - Tư tưởng xây dựng đoạn code xử lý này theo cách giải bài toán khi xây dựng bảng. Điểm khác biệt đối với code tích chập bên trên là dãy tín hiệu thứ 2 sẽ được timeReverse() trước rồi mới cho vào nhân các giá trị tương ứng.
    - Xây dựng vector<vector<double>> matrix chứa các giá trị khi nhân 2 tín hiệu tương ứng
    - Vector<vector<vector<int>>> checkRoot chứa các chỉ số theo đường chéo phụ trong ma trận (việc lưu trữ các chỉ số trong ma trận này giúp tìm ra đâu là giá trị nằm tại vị trí gốc tọa độ của dãy tín hiệu kết quả)

4. Các kết quả:

Trong lý thuyết, cần chia ra thành 2 loại dãy (dãy có chiều dài hữu hạn, dãy có chiều dài vô hạn). Tuy nhiên, khi minh họa khái niệm, chúng em lấy tín hiệu có chiều dài hữu hạn.

a. Phép toán tổng hai tín hiệu:

Ví dụ 1: Giả sử ta thực hiện phép toán cộng hai dãy tín hiệu sau:

X[n] = {0, 0, 1, 1, 0.5, 1, 0, 0} – vị trí gốc tọa độ là 1 (bôi đỏ)

Y[n] = {0, 0, 0, 1, 0.5, 0.5, 0} – vị trí gốc tọa độ là 1 (bôi đỏ)

Sau khi chạy, chương trinh cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\So
al_Processing\Project1\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
0 0 0 2 1.5 1 1 0 0
3
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 0 0 0 2 1.5 1 1 0 0 là dãy kết quả và vị trí gốc tọa độ nằm tại giá trị 2 (bôi đỏ)

Ví dụ 2: Giả sử ta thực hiện phép toán cộng hai dãy tín hiệu sau:

X[n] = {0, 0, 1, 1, 0.5, 1, 0, 0} – vị trí gốc tọa độ tại giá trị 1 (bôi đỏ)

Y[n] = {1, 0.5, 0.5, 0, 0, 1.5, 0.5} – vị trí gốc tọa độ tại giá trị 1.5 (bôi đỏ)

Sau khi chạy, chương trình cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\Sou
al_Processing\Project1\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
1 0.5 0.5 0 0 2.5 1.5 0.5 1 0 0
5
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 1 0.5 0.5 0 0 2.5 1.5 0.5 1 0 0 là dãy kết quả và vị trí gốc tọa độ nằm tại giá trị 2.5 (bôi đỏ)

b. Phép hiệu hai tín hiệu:

Ví dụ 1: Giả sử ta thực hiện phép toán hiệu hai dãy tín hiệu sau:

X[n] = {0, 0, 1, 1, 0.5, 1, 0, 0} – vị trí gốc tọa độ tại giá trị 1 (bôi đỏ)

Y[n] = {1, 0.5, 0.5, 0, 0, 1.5, 0.5} – vị trí gốc tọa độ tại giá trị 0.5 (bôi đỏ)

Sau khi chạy, chương trình cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_su
bjects\Sound_Signal_Processing\Project1\" ; if ($?) { g++ test.cpp -o te
st } ; if ($?) { .\test }
0 -1 0.5 0.5 0.5 1 -1.5 -0.5
2
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 0 -1 0.5 0.5 0.5 1 -1.5 -0.5 là dãy kết quả và vị trí gốc tọa độ nằm tại giá trị 0.5 (bôi đỏ)

Ví dụ 2: Giả sử ta thực hiện phép toán hiệu hai dãy tín hiệu sau:

X[n] = { 0, 1, 2, 1, 0} – vị trí gốc tọa độ tại giá trị 0 (bôi đỏ)
Y[n] = {1, 0.5, 0.5, 0, 0, 1.5, 0.5} – vị trí gốc tọa độ tại giá trị 0.5 (bôi đỏ)

Sau khi chạy, chương trình cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\So
al_Processing\Project1\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
0 1 2 0 -0.5 -0.5 0 0 -1.5 -0.5
4
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 0 1 2 0 -0.5 -0.5 0 0 -1.5 -0.5 là dãy kết quả và vị trí gốc tọa độ nằm tại giá trị -0.5 (bôi đỏ)

c. Phép toán tích hai tín hiệu:

Ví dụ 1: Giả sử ta thực hiện phép toán nhân hai dãy tín hiệu sau:

X[n] = { 0, 0, 1, 0, 0.5, 0.25, 0} – vị trí gốc tọa độ tại giá trị 1 (bôi đỏ)

Y[n] = { 0, 1, 0.5, 0.25, 0, 1, 0} – vị trí gốc tọa độ tại giá trị 1 (bôi đỏ)

Sau khi chạy, chương trinh cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\Sou
{ g++ test.cpp -o test } ; if ($?) { .\test }
0 0 1 0 0.125 0 0 0
2
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

0 0 1 0 0.123 0 0 0 là dãy kết quả và vị trí gốc tọa độ là 1 (bôi đỏ)

Ví dụ 2: Giả sử ta thực hiện phép toán nhân hai dãy tín hiệu sau:

X[n] = {0, 0, 1, 1, 0.5, 1, 0, 0} – vị trí gốc tọa độ tại giá trị 1 (bôi đỏ)

Y[n] = {1, 1, 0.5, 0, 0.2, 1.5} – vị trí gốc tọa độ tại giá trị 0.2 (bôi đỏ)

Sau khi chạy, chương trình cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\S
{ g++ test.cpp -o test } ; if ($?) { .\test }
0 0 0 0 0.2 0 0 0 0
4
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 0 0 0 0 0.2 0 0 0 0 là dãy kết quả và vị trí gốc tọa độ nằm tại vị trí 0.2 (bôi đỏ)

d. Phép nhân tín hiệu với một hằng số

Ví dụ 1: Giả sử ta thực hiện phép toán nhân một dãy tín hiệu với một hằng số như sau:

X[n] = {0, 0, 0.5, 1.0, 1.2, 0.2, 0.1, 0} – vị trí gốc tọa độ tại giá trị 0.5 (bôi đỏ)

Thực hiện nhân X[n] với 2

Sau khi chạy, chương trình cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\So
{ g++ test.cpp -o test } ; if ($?) { .\test }
0 0 1 2 2.4 0.4 0.2 0
2
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 0 0 1 2 2.4 0.4 0.2 0 là dãy kết quả với vị trí gốc tọa độ nằm tại giá trị 1 (bôi đỏ)

Ví dụ 2: Giả sử ta thực hiện phép toán nhân một dãy tín hiệu với một hằng số như sau:

X[n] = {0, 1, 2.2, 0.2, 1, 0, 0, 0.1} – vị trí gốc tọa độ tại giá trị 0.1 (bôi đỏ)

Thực hiện nhân dãy X[n] với -5

Sau khi chạy, chương trình cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\So
al_Processing\Project1\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
0 -5 -11 -1 -5 0 0 -0.5
7
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Ln 84, Col 13 Spaces: 4 UTF-8 CRLF C++ Go Live Win32 Prettie

Quan sát hình trên, ta thấy 0 -5 -11 -1 -5 0 0 -0.5 là dãy kết quả và vị trí của gốc tọa độ nằm tại vị trí -0.5 (bôi đỏ)

e. Phép tích chập

Ví dụ 1: Giả sử ta thực hiện phép toán tích chập hai dãy tín hiệu sau:

X[n] = {1, 3, 4} – vị trí gốc tọa độ tại giá trị 3 (bôi đỏ)
Y[n] = {2, -1, 5} – vị trí gốc tọa độ tại giá trị 2 (bôi đỏ)

Sau khi chạy, chương trinh cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\So
{ g++ test.cpp -o test } ; if ($?) { .\test }
2 5 10 11 20
1
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta có 2 5 10 11 20 là dãy kết quả với vị trí gốc tọa độ tại giá trị 5(bôi đỏ)

f. Phép tương quan chéo

Ví dụ 2: Giả sử ta thực hiện phép toán tương quan chéo hai dãy tín hiệu sau:

X[n] = { 0, 0, 1, 1, 1, 0, 0} – vị trí gốc tọa độ tại giá trị 1 (bôi đỏ)
Y[n] = { 0, 0, 1, 1, 0, 0} – vị trí gốc tọa độ tại giá trị 1 (bôi đỏ)

Sau khi chạy, chương trình cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\Sou
ect1\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
0 0 0 0 1 2 2 1 0 0 0 0 0
5
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát kết quả ta thấy 0 0 0 0 1 2 2 1 0 0 0 0 0 là dãy kết quả và vị trí gốc tọa độ nằm tại giá trị 2 (bôi đỏ)

Như vậy, với phương pháp tính tay thông thường (sử dụng đồ thị) cũng cho kết quả tương tự. Kết quả của bài toán trên theo phương pháp tính tay dử dụng đồ thị thông thường như sau:

R[0] = 2

R[-1] = 1
R[n] = 0 (với n <= -2)

R[1] = 2

R[n] = 0 (với n>=3)

⇨ Quan sát dãy kết quả trên sử dụng chương trình cũng có thể kết luận kết quả tương tự như vậy

g. Phép dịch thời gian của tín hiệu

Ví dụ 1: Giả sử ta thực hiện phép dịch thời gian dãy tín hiệu sau:

X[n] = {0, 0, 1, 1, 0.5, 1, 0, 1, 0} – vị trí gốc tọa độ tại giá trị 1 (bôi đỏ)

X[n-1] = ?

Sau khi chạy, chương trình cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\S
ect1\" ; if ($?) { g++ test.cpp -o test } ; if ($?) { .\test }
0 0 0 1 1 0.5 1 0 1
3
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 0 0 0 1 1 0.5 1 0 1 là dãy kết quả và vị trí gốc tọa độ nằm tại giá trị 1 (bôi đỏ)

h. Tăng tần số lấy mẫu (Up sampling)

Giả sử ta thực hiện phép tang tần số lấy mẫu với dãy tín hiệu sau:
X[n] = { 0, 1, 0.2, 0.3, 0.4, 0.5, 0.6} và vị trí gốc tọa độ tại giá trị 0.2 (bôi đỏ)

Y[n] = $X_2[n]$ = ?

Kết quả sau khi chạy chương trình là:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\S
{ g++ test.cpp -o test } ; if ($?) { .\test }
0 0 1 0 0.2 0 0.3 0 0.4 0 0.5 0 0.6
4
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 0 0 1 0 0.2 0 0.3 0 0.4 0 0.5 0 0.6 là dãy kết quả và vị trí của gốc tọa độ nằm tại giá trị 0.2 (bôi đỏ)

i. Giảm tần số lấy mẫu (Down sampling)

Ví dụ 1: Giả sử ta thực hiện phép giảm tần số lấy mẫu của dãy tín hiệu sau:

X[n] = {0, 0, 1, 0, 1.2, 1.1, 1.3, 1.0, 0.5, 0.2} – vị trí của gốc tọa độ tại giá trị 1 (bôi đỏ)

Y[n] = X[2n] = ?
Sau khi chạy chương trình cho kết quả như sau:

```
0 1 1.2 1.3 0.5
1
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\S
{ g++ test.cpp -o test } ; if ($?) { .\test }
0 1 1.2 1.3 0.5
1
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Ln 92, Col 43 Spaces: 4 UTF-8

Quan hình trên, ta thấy 0 1 1.2 1.3 0.5 là dãy kết quả và vị trí gốc tọa độ nằm tại giá trị 1(bôi đỏ)

Ví dụ 2: Giả sử ta thực hiện phép giảm tần số lấy mẫu của dãy tín hiệu sau:

X[n] = {0, 1, 2, 2, 3, 5, 2, 3, 1, 2, 1} – vị trí của gốc tọa độ tại giá trị 5 (bôi đỏ)

Y[n] = X[3n] = ?

Sau khi chạy chương trình cho kết quả như sau:

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\So
{ g++ test.cpp -o test } ; if ($?) { .\test }
2 5 1
1
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 2 5 1 là dãy kết quả và vị trí gốc tọa độ nằm tại giá trị 5 (bôi đỏ)

j. Đảo thời gian:

Cho dãy tín hiệu x[n] = {0, 0, 0, 1, 0.5, 0.25, 0, 0} – vị trí gốc tọa độ tại giá trị 1 (bôi đỏ)

```
PS D:\Record_subjects\Sound_Signal_Processing\Project1> cd "d:\Record_subjects\S
{ g++ test.cpp -o test } ; if ($?) { .\test }
0 0 0.25 0.5 1 0 0 0 0
4
PS D:\Record_subjects\Sound_Signal_Processing\Project1>
```

Quan sát hình trên, ta thấy 0 0 0.25 0.5 1 0 0 0 0 là dãy kết quả và vị trí gốc tọa độ nằm trị giá trị 1 (bôi đỏ)

Kết luận:

Trong khuôn khổ xây dựng lớp thực hiện mô phỏng các phép toán với tín hiệu, chúng em đã lập trình gần như đầy đủ các phép toán (còn tự tương quan)

Tất cả các hàm đều được xây dựng bên trong một class duy nhất là Signal. Do đó không có bất cứ mối quan hệ với các lớp nào khác.

Tài liệu tham khảo:

1. Các phép toán với tín hiệu:

   https://www.youtube.com/watch?v=ecUUbQGUArw&list=PLZ0Z4sYiJXgZpMnj_syYYSR2Fiuno88Ks

2. Tính tích chập, tương quan chéo, tự tương quan bằng bảng

   https://www.youtube.com/watch?v=tMEwZlNRkWU&t=274s