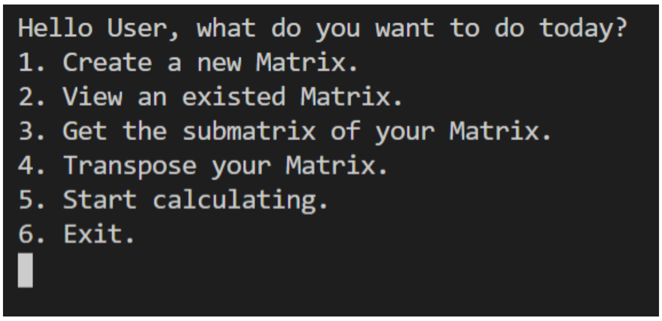
# Data Structure HW3

40847013S 王瑞渝

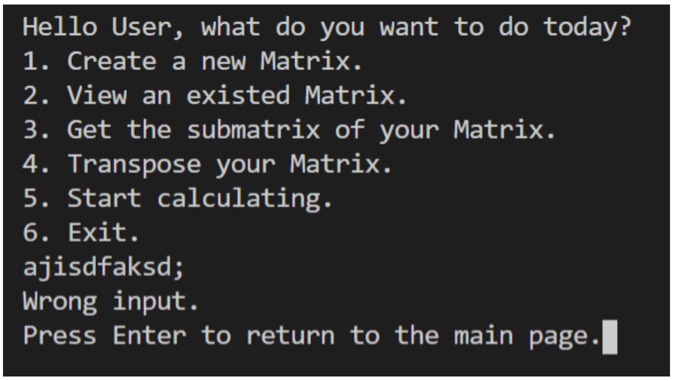
## 目錄

## 1.1 介面說明——MAIN PAGE



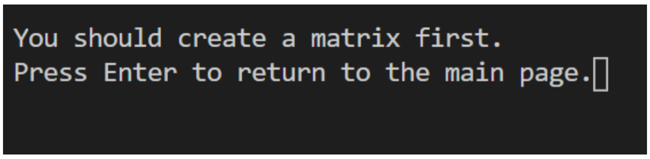
```
Hello User, what do you want to do today?
1. Create a new Matrix.
2. View an existed Matrix.
3. Get the submatrix of your Matrix.
4. Transpose your Matrix.
5. Start calculating.
6. Exit.
```

▲ 可輸入 1~6 執行功能



```
Hello User, what do you want to do today?
1. Create a new Matrix.
2. View an existed Matrix.
3. Get the submatrix of your Matrix.
4. Transpose your Matrix.
5. Start calculating.
6. Exit.
ajisdfaksd;
Wrong input.
Press Enter to return to the main page.
```

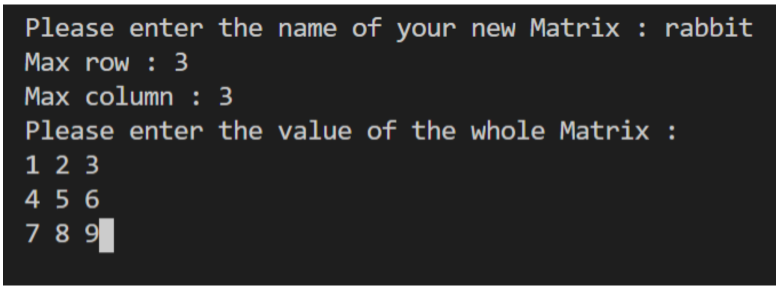
▲ 輸入其他字符視為錯誤，輸出警示



```
You should create a matrix first.
Press Enter to return to the main page.
```

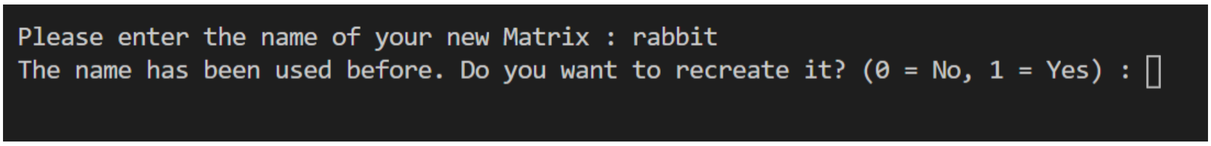
▲ 未建立矩陣時(選單 1)，輸入 2~5 判定錯誤

## 1.2 介面說明——CREATE MATRIX



```
Please enter the name of your new Matrix : rabbit
Max row : 3
Max column : 3
Please enter the value of the whole Matrix :
1 2 3
4 5 6
7 8 9
```

▲ 依照提示輸入矩陣資料

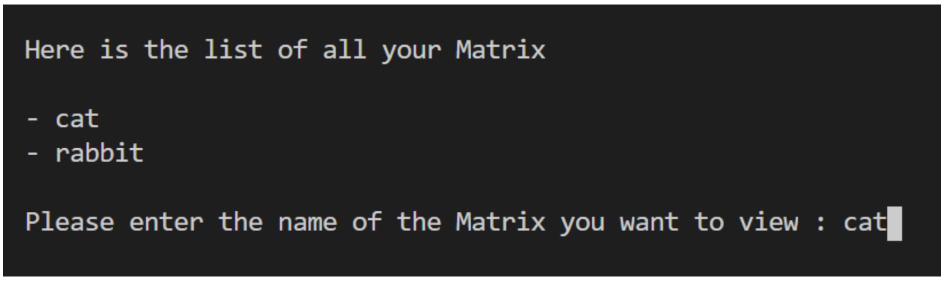


```
Please enter the name of your new Matrix : rabbit
The name has been used before. Do you want to recreate it? (0 = No, 1 = Yes) :
```

▲ 若名稱已被使用，詢問使用者是否重建舊矩陣

▲ (所有儲存新矩陣相關功能皆有此保護)

## 1.3 介面說明——VIEW MATRIX

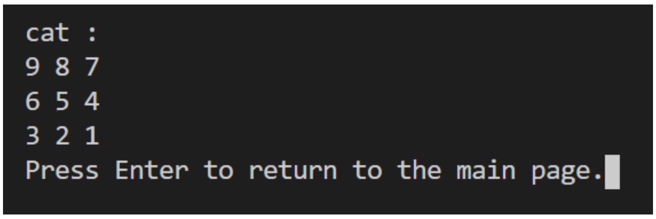


```
Here is the list of all your Matrix

- cat
- rabbit

Please enter the name of the Matrix you want to view : cat
```

▲ 列出現有矩陣列表供使用者選擇所需展示之矩陣(輸入矩陣名以選擇)

▲ 輸入不存在矩陣名會自動忽略，並重新刷新頁面

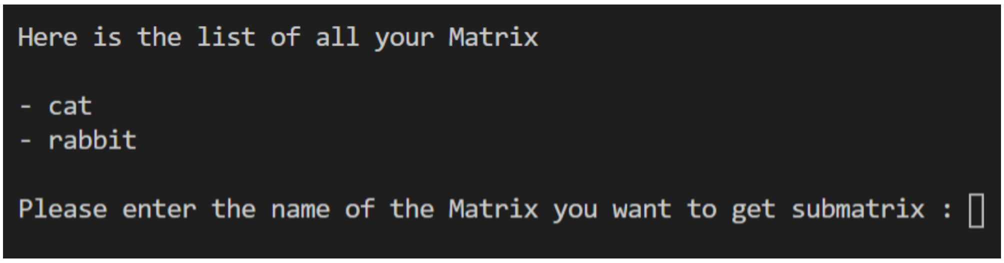
▲ (所有選擇矩陣相關功能皆有此保護)



```
cat :
9 8 7
6 5 4
3 2 1
Press Enter to return to the main page.
```

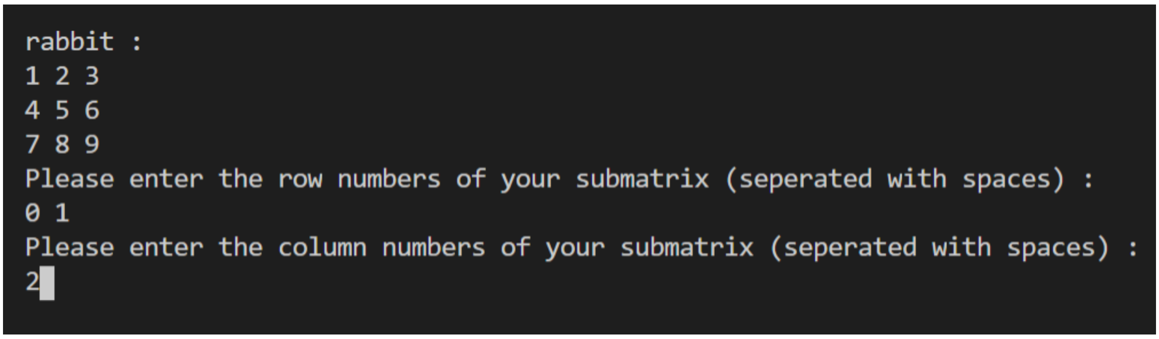
▲ 成功選擇後印出所選矩陣

## 1.4 介面說明——Get Submatrix



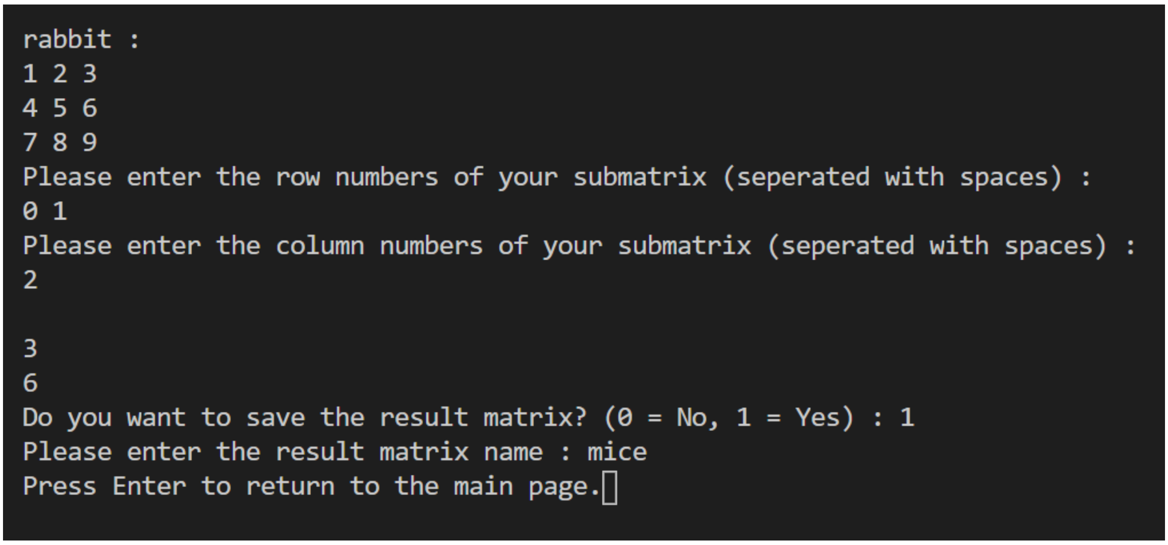
```
Here is the list of all your Matrix

- cat
- rabbit

Please enter the name of the Matrix you want to get submatrix :
```

▲ 列出現有矩陣列表供使用者選擇子矩陣之原矩陣



```
rabbit :
1 2 3
4 5 6
7 8 9
Please enter the row numbers of your submatrix (seperated with spaces) :
0 1
Please enter the column numbers of your submatrix (seperated with spaces) :
2
```

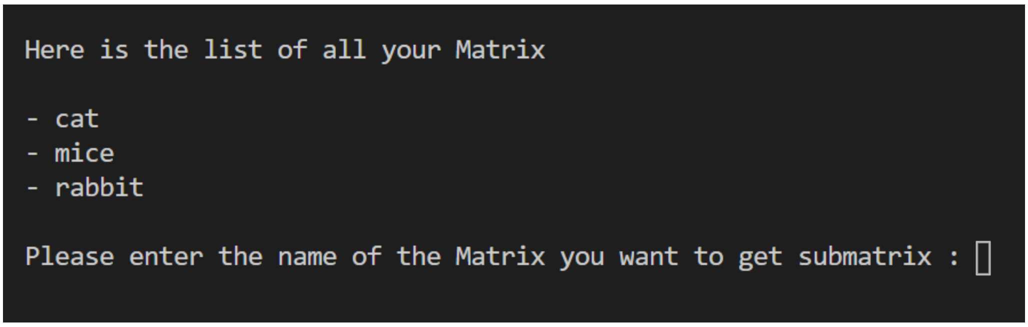
▲ 根據提示輸入所需子矩陣之行列編號



```
rabbit :
1 2 3
4 5 6
7 8 9
Please enter the row numbers of your submatrix (seperated with spaces) :
0 1
Please enter the column numbers of your submatrix (seperated with spaces) :
2

3
6
Do you want to save the result matrix? (0 = No, 1 = Yes) : 1
Please enter the result matrix name : mice
Press Enter to return to the main page.
```

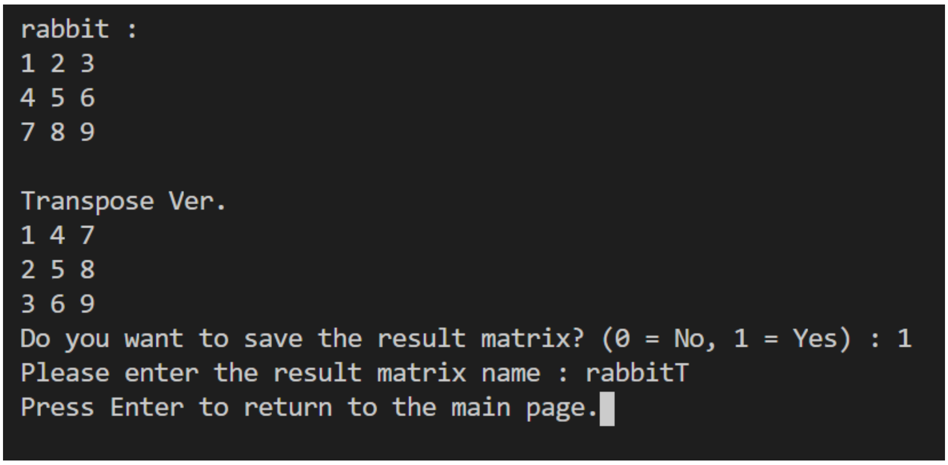
▲ 印出子矩陣並詢問使用者是否保存新矩陣

## 1.5 介面說明──TRANSPOSE MATRIX



```
Here is the list of all your Matrix

- cat
- mice
- rabbit

Please enter the name of the Matrix you want to get submatrix : 
```

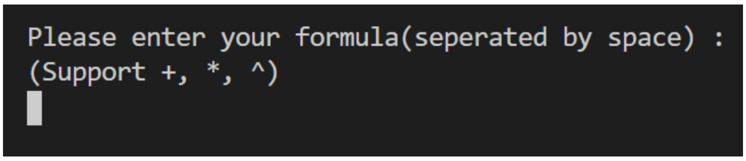
▲ 列出現有矩陣列表供使用者選擇所需轉置之矩陣



```
rabbit :
1 2 3
4 5 6
7 8 9

Transpose Ver.
1 4 7
2 5 8
3 6 9
Do you want to save the result matrix? (0 = No, 1 = Yes) : 1
Please enter the result matrix name : rabbitT
Press Enter to return to the main page.
```

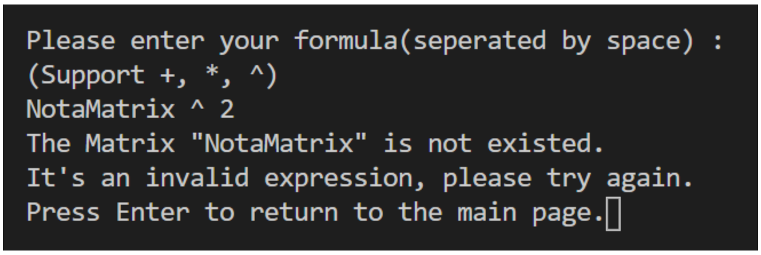
▲ 印出原矩陣和轉置後矩陣並詢問使用者是否保存轉置矩陣

## 1.6 介面說明——CALCULATE



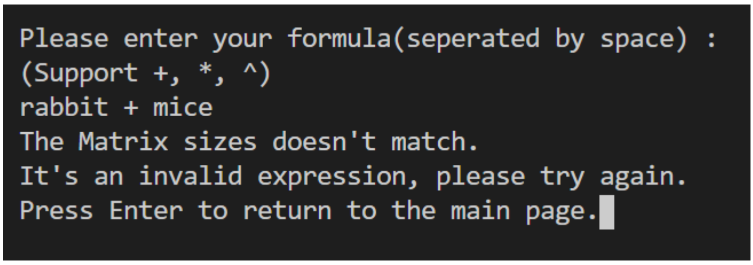
```
Please enter your formula(seperated by space) :
(Support +, *, ^)
```

▲ 使用空白分隔輸入算式(不支援先乘除後加減，一律從左到右)



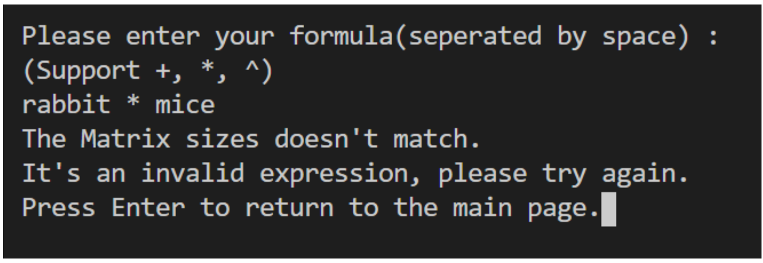
```
Please enter your formula(seperated by space) :
(Support +, *, ^)
NotaMatrix ^ 2
The Matrix "NotaMatrix" is not existed.
It's an invalid expression, please try again.
Press Enter to return to the main page.
```

(不存在矩陣)



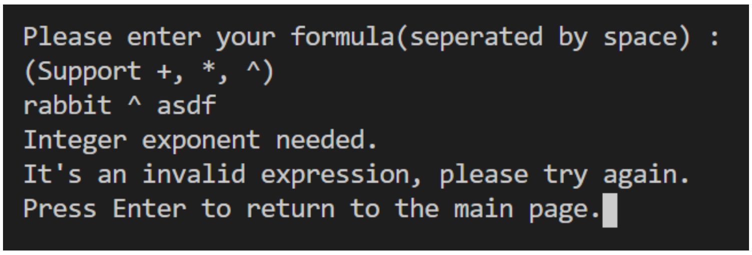
```
Please enter your formula(seperated by space) :
(Support +, *, ^)
rabbit + mice
The Matrix sizes doesn't match.
It's an invalid expression, please try again.
Press Enter to return to the main page.
```

(大小不同矩陣相加)



```
Please enter your formula(seperated by space) :
(Support +, *, ^)
rabbit * mice
The Matrix sizes doesn't match.
It's an invalid expression, please try again.
Press Enter to return to the main page.
```

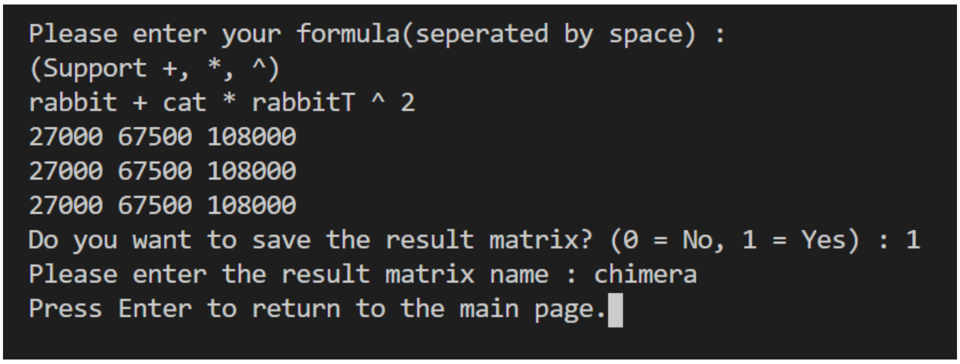
(大小不同矩陣相乘)



```
Please enter your formula(seperated by space) :
(Support +, *, ^)
rabbit ^ asdf
Integer exponent needed.
It's an invalid expression, please try again.
Press Enter to return to the main page.
```

(給定非 int 的指數)

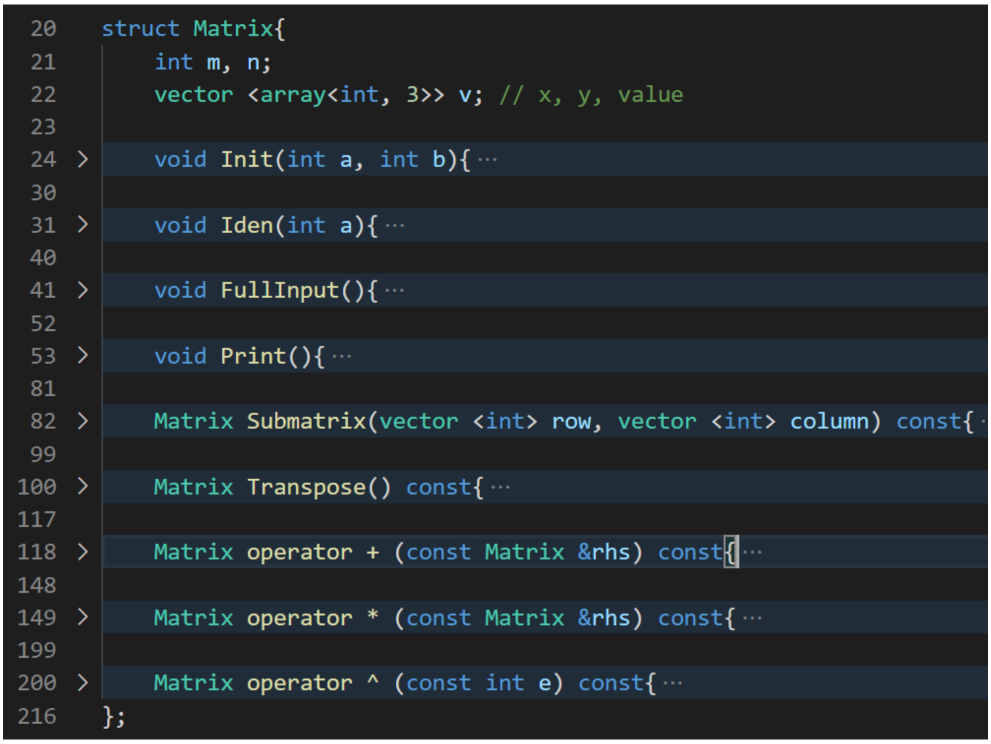
▲ 以上情況視為錯誤，輸出警示



```
Please enter your formula(seperated by space) :
(Support +, *, ^)
rabbit + cat * rabbitT ^ 2
27000 67500 108000
27000 67500 108000
27000 67500 108000
Do you want to save the result matrix? (0 = No, 1 = Yes) : 1
Please enter the result matrix name : chimera
Press Enter to return to the main page.
```

▲ 算式輸入成功後印出結果，並詢問是否保存結果矩陣

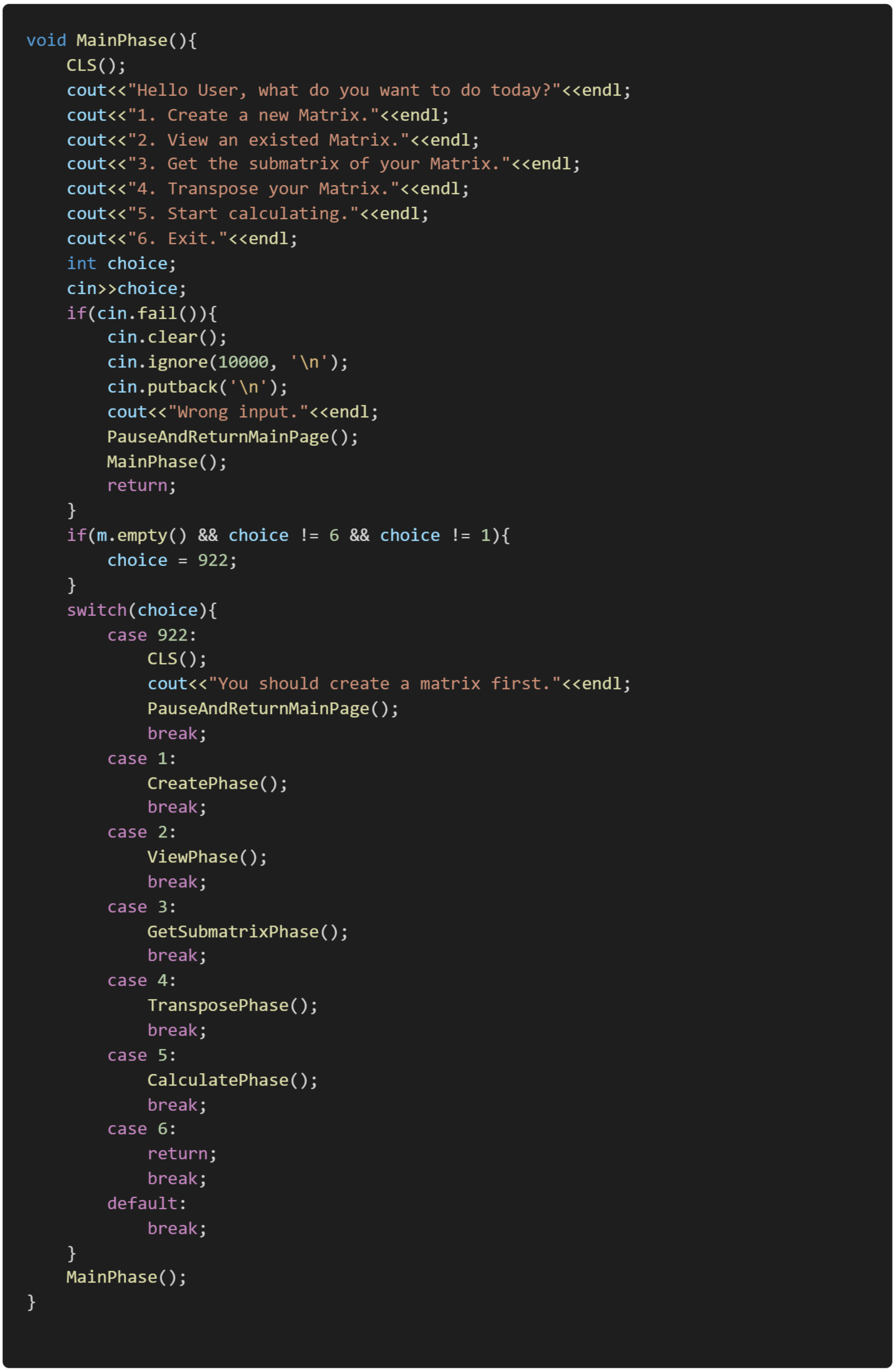
## 2.1 功能分析——Matrix 結構



```
struct Matrix{
    int m, n;
    vector <array<int, 3>> v; // x, y, value

    void Init(int a, int b){…

    void Iden(int a){…

    void FullInput(){…

    void Print(){…

    Matrix Submatrix(vector <int> row, vector <int> column) const{

    Matrix Transpose() const{…

    Matrix operator + (const Matrix &rhs) const{…

    Matrix operator * (const Matrix &rhs) const{…

    Matrix operator ^ (const int e) const{…
};
```

- m → max row
- n → max column
- vector <array<int, 3>> v → Sparse Matrix 儲存位置
- Init(a, b) → 初始化成一個有 a rows, b columns 的空矩陣
- Iden(a) → 初始化成一個 a 階的標準矩陣$I_a$
- FullInput() → 輸入整個矩陣資料 (詳情可見 2.3 輸入 Matrix)
- Print() → 印出整個矩陣 (詳情可見 2.4 顯示 Matrix)
- Submatrix(row, column) → 透過給定的行列編號 vector，回傳子矩陣 (詳情可見 2.5 Submatrix)
- Transpose() → 回傳反矩陣 (詳情可見 2.6 Transpose Matrix)
- operator + → 矩陣加法 (詳情可見 2.7 Transpose Matrix)
- operator * → 矩陣乘法 (詳情可見 2.8 Transpose Matrix)
- operator ^ → 矩陣次方/矩陣快速冪 (詳情可見 2.9 矩陣次方/矩陣快速冪)

## 2.2 功能分析——主選單

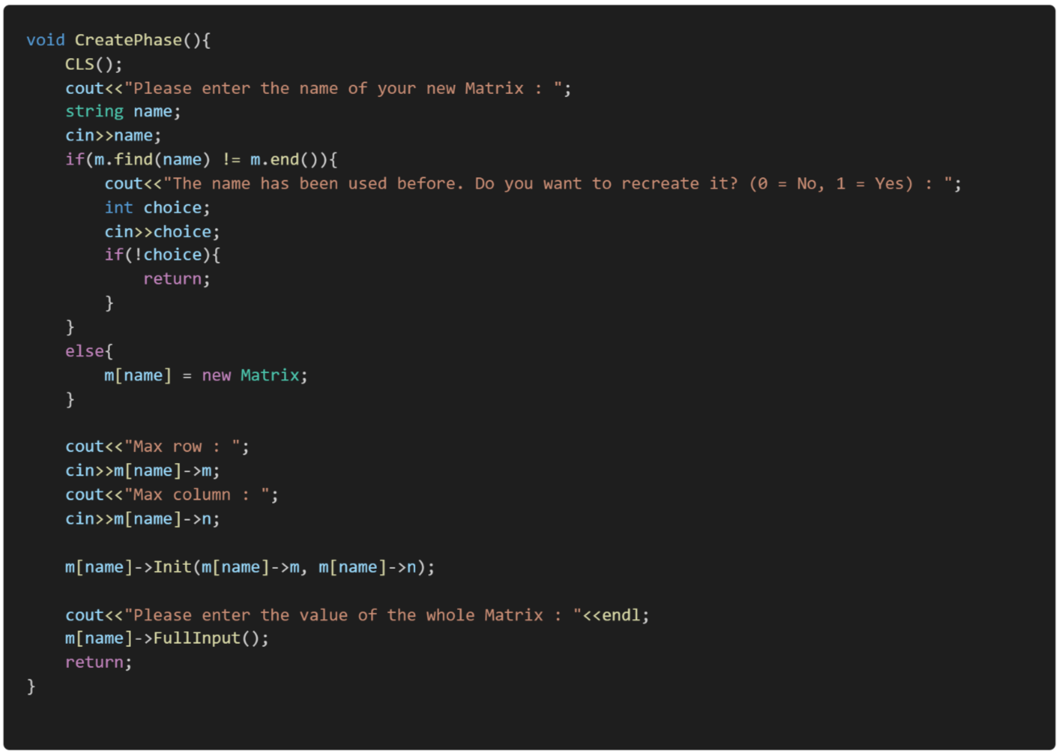


```
void MainPhase(){
    CLS();
    cout<<"Hello User, what do you want to do today?"<<endl;
    cout<<"1. Create a new Matrix."<<endl;
    cout<<"2. View an existed Matrix."<<endl;
    cout<<"3. Get the submatrix of your Matrix."<<endl;
    cout<<"4. Transpose your Matrix."<<endl;
    cout<<"5. Start calculating."<<endl;
    cout<<"6. Exit."<<endl;
    int choice;
    cin>>choice;
    if(cin.fail()){
        cin.clear();
        cin.ignore(10000, '\n');
        cin.putback('\n');
        cout<<"Wrong input."<<endl;
        PauseAndReturnMainPage();
        MainPhase();
        return;
    }
    if(m.empty() && choice != 6 && choice != 1){
        choice = 922;
    }
    switch(choice){
        case 922:
            CLS();
            cout<<"You should create a matrix first."<<endl;
            PauseAndReturnMainPage();
            break;
        case 1:
            CreatePhase();
            break;
        case 2:
            ViewPhase();
            break;
        case 3:
            GetSubmatrixPhase();
            break;
        case 4:
            TransposePhase();
            break;
        case 5:
            CalculatePhase();
            break;
        case 6:
            return;
            break;
        default:
            break;
    }
    MainPhase();
}
```

- CLS() → 會偵測作業系統使用清屏
- PauseAndReturnMainPage() → 按 Enter 以回到主選單
- m → 全域變數 `map <string, Matrix *> m;` 用來存不同名字的矩陣

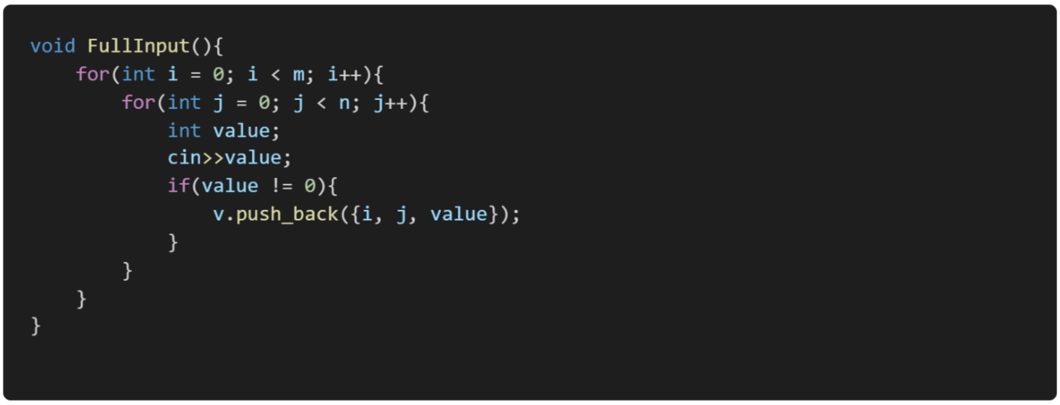
程式碼解釋：輸入 choice 後，如果沒有任何已建立矩陣，輸出錯誤，否則根據 choice 呼叫不同的 Phase 函式。

## 2.3 功能分析——輸入 MATRIX



```cpp
void CreatePhase(){
    CLS();
    cout<<"Please enter the name of your new Matrix : ";
    string name;
    cin>>name;
    if(m.find(name) != m.end()){
        cout<<"The name has been used before. Do you want to recreate it? (0 = No, 1 = Yes) : ";
        int choice;
        cin>>choice;
        if(!choice){
            return;
        }
    }
    else{
        m[name] = new Matrix;
    }

    cout<<"Max row : ";
    cin>>m[name]->m;
    cout<<"Max column : ";
    cin>>m[name]->n;

    m[name]->Init(m[name]->m, m[name]->n);

    cout<<"Please enter the value of the whole Matrix : "<<endl;
    m[name]->FullInput();
    return;
}
```

輸入欲創建的矩陣名，如果該名字未使用就給這個名字分配一個 Matrix 的空間，已使用就向使用者確認是否覆蓋，最後再將名字對應的空間初始化後呼叫 FullInput()輸入整個矩陣。

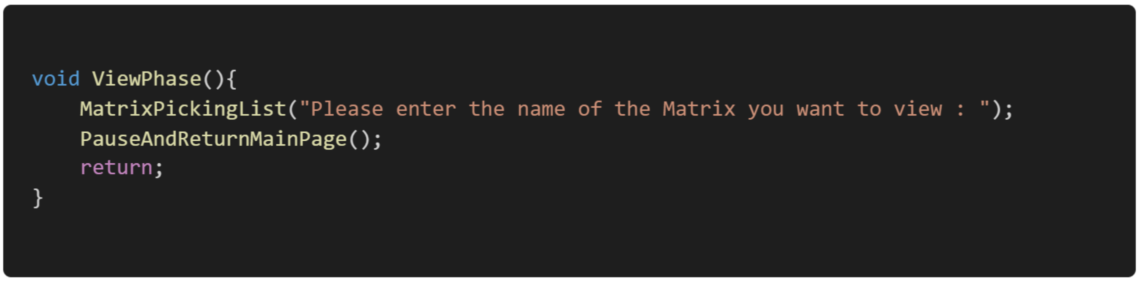


```cpp
void FullInput(){
    for(int i = 0; i < m; i++){
        for(int j = 0; j < n; j++){
            int value;
            cin>>value;
            if(value != 0){
                v.push_back({i, j, value});
            }
        }
    }
}
```

雙重迴圈輸入，如果輸入不為零則把{i, j, value}塞進 v 裡面。

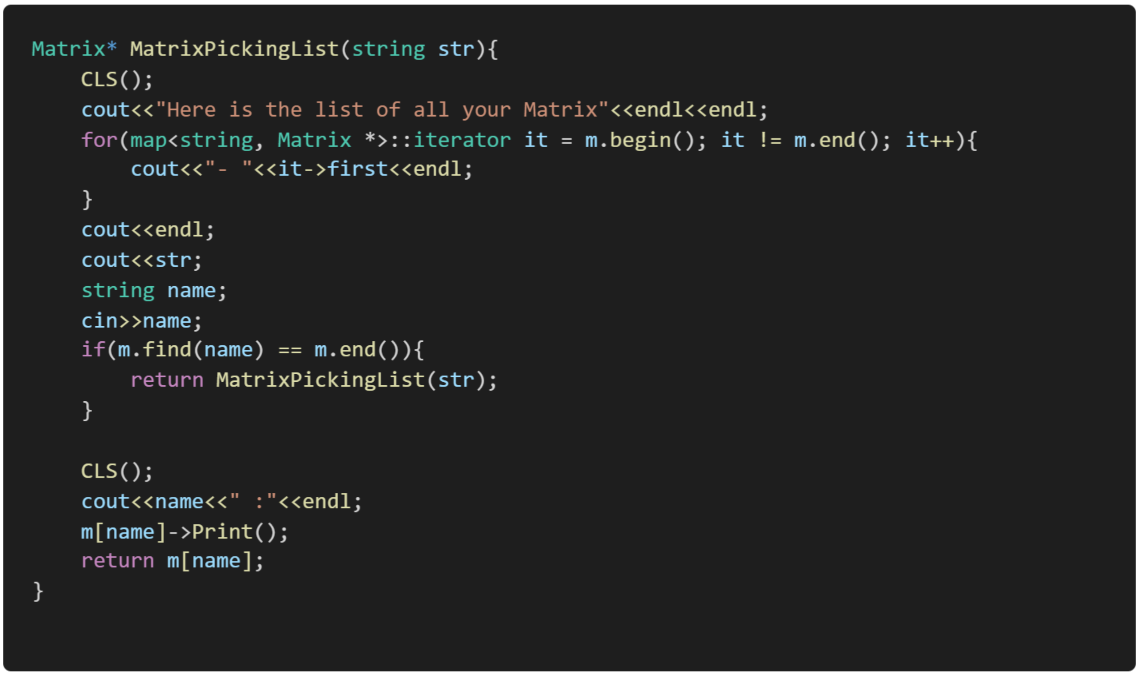
時間複雜度：O(nm)。

## 2.4 功能分析——顯示 MATRIX



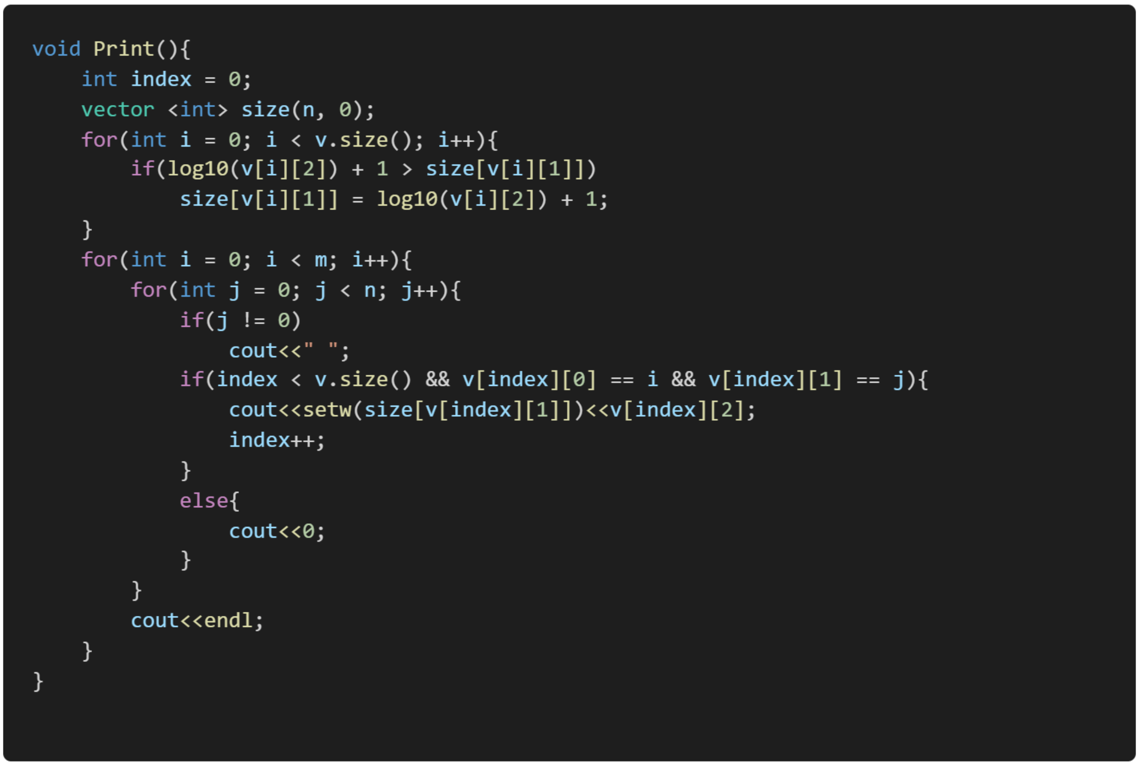
```
void ViewPhase(){
    MatrixPickingList("Please enter the name of the Matrix you want to view : ");
    PauseAndReturnMainPage();
    return;
}
```

呼叫 MatrixPickingList()。



```
Matrix* MatrixPickingList(string str){
    CLS();
    cout<<"Here is the list of all your Matrix"<<endl<<endl;
    for(map<string, Matrix *>::iterator it = m.begin(); it != m.end(); it++){
        cout<<"- "<<it->first<<endl;
    }
    cout<<endl;
    cout<<str;
    string name;
    cin>>name;
    if(m.find(name) == m.end()){
        return MatrixPickingList(str);
    }

    CLS();
    cout<<name<<" :"<<endl;
    m[name]->Print();
    return m[name];
}
```

把整個 map 裡面的東西印成列表並讓使用者輸入名字，若沒有此名字的矩陣，呼叫自己重新來一次，取得矩陣後用 Print()印出來。

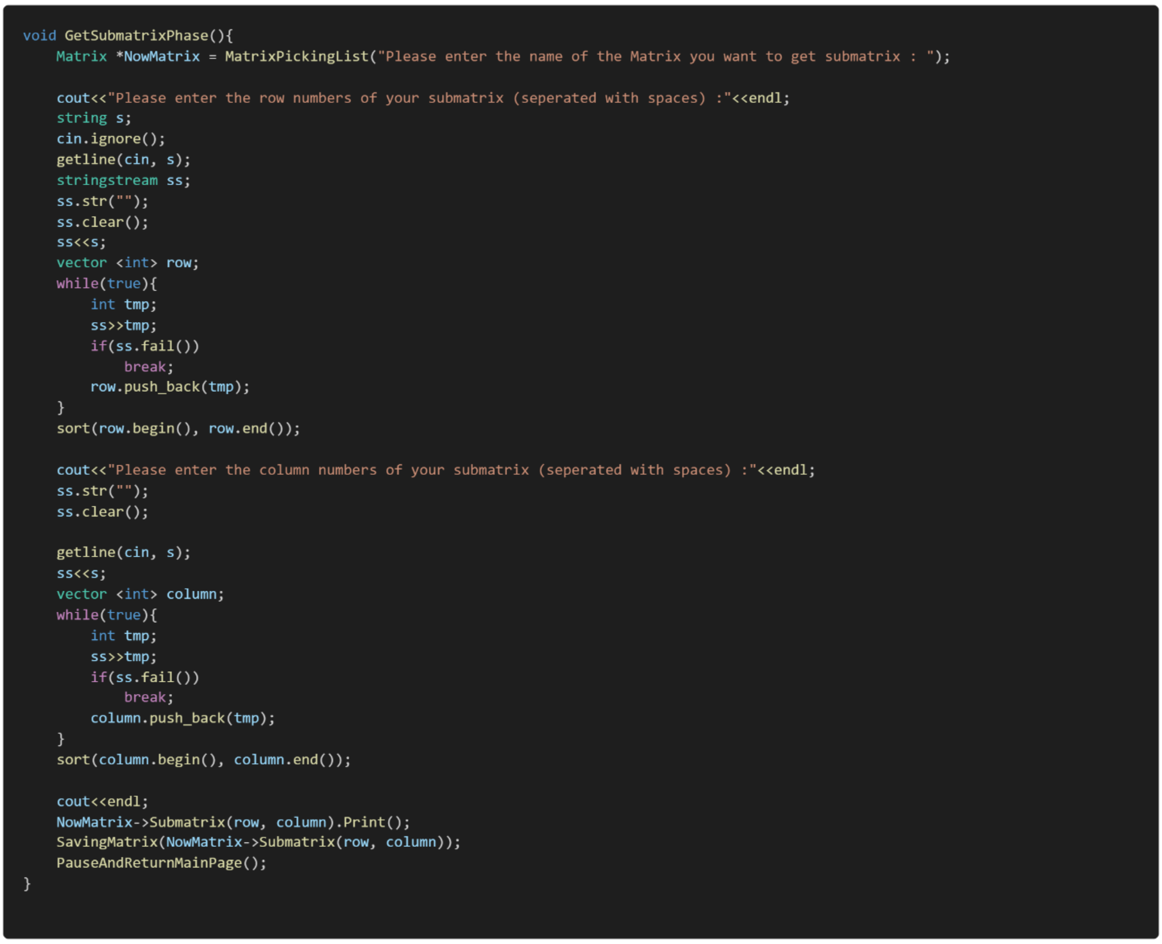


```
void Print(){
    int index = 0;
    vector <int> size(n, 0);
    for(int i = 0; i < v.size(); i++){
        if(log10(v[i][2]) + 1 > size[v[i][1]])
            size[v[i][1]] = log10(v[i][2]) + 1;
    }
    for(int i = 0; i < m; i++){
        for(int j = 0; j < n; j++){
            if(j != 0)
                cout<<" ";
            if(index < v.size() && v[index][0] == i && v[index][1] == j){
                cout<<setw(size[v[index][1]])<<v[index][2];
                index++;
            }
            else{
                cout<<0;
            }
        }
        cout<<endl;
    }
}
```

跑一個迴圈取得對每個 column 的最大位數，跑雙重迴圈——如果當下的(i, j)和index 指向項的(x, y)相同便印出 value 否則印出 0。

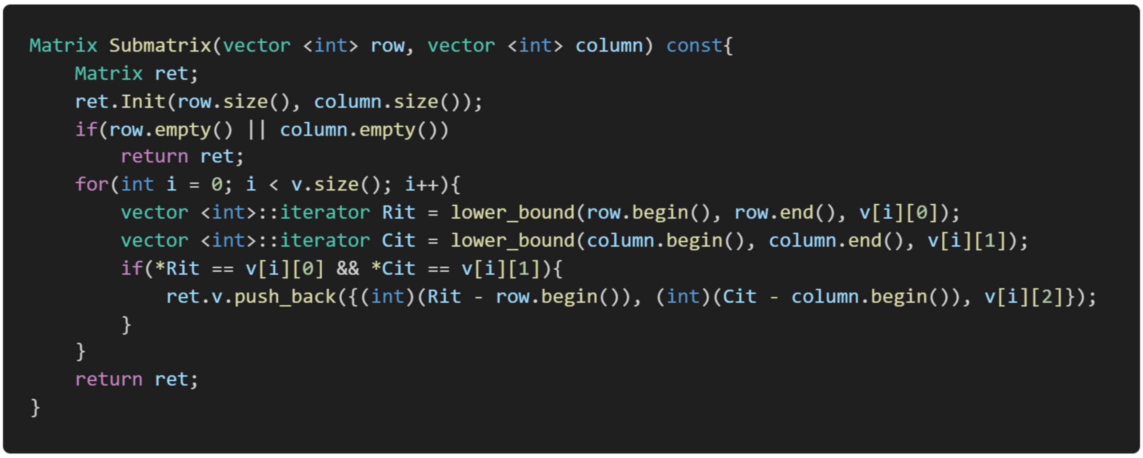
時間複雜度：$\mathrm{O}(\mathrm{Elements}+\mathrm{nm})=\mathrm{O}(\mathrm{nm})$。

## 2.5 功能分析——SUBMATRIX



```
void GetSubmatrixPhase(){
    Matrix *NowMatrix = MatrixPickingList("Please enter the name of the Matrix you want to get submatrix : ");

    cout<<"Please enter the row numbers of your submatrix (seperated with spaces) :"<<endl;
    string s;
    cin.ignore();
    getline(cin, s);
    stringstream ss;
    ss.str("");
    ss.clear();
    ss<<s;
    vector <int> row;
    while(true){
        int tmp;
        ss>>tmp;
        if(ss.fail())
            break;
        row.push_back(tmp);
    }
    sort(row.begin(), row.end());

    cout<<"Please enter the column numbers of your submatrix (seperated with spaces) :"<<endl;
    ss.str("");
    ss.clear();

    getline(cin, s);
    ss<<s;
    vector <int> column;
    while(true){
        int tmp;
        ss>>tmp;
        if(ss.fail())
            break;
        column.push_back(tmp);
    }
    sort(column.begin(), column.end());

    cout<<endl;
    NowMatrix->Submatrix(row, column).Print();
    SavingMatrix(NowMatrix->Submatrix(row, column));
    PauseAndReturnMainPage();
}
```

讓使用者選取原矩陣後，用兩個 vector<int>分別存行列編號後傳進 Submatrix()

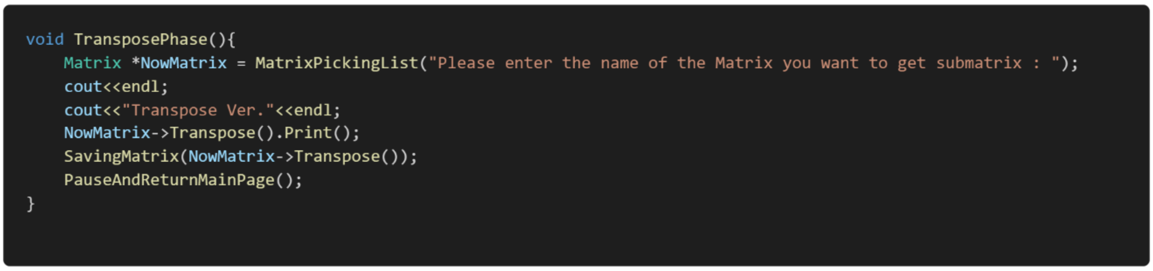


```
Matrix Submatrix(vector <int> row, vector <int> column) const{
    Matrix ret;
    ret.Init(row.size(), column.size());
    if(row.empty() || column.empty())
        return ret;
    for(int i = 0; i < v.size(); i++){
        vector <int>::iterator Rit = lower_bound(row.begin(), row.end(), v[i][0]);
        vector <int>::iterator Cit = lower_bound(column.begin(), column.end(), v[i][1]);
        if(*Rit == v[i][0] && *Cit == v[i][1]){
            ret.v.push_back({(int)(Rit - row.begin()), (int)(Cit - column.begin()), v[i][2]});
        }
    }
    return ret;
}
```

先 Init 個空矩陣 ret，然後跑個 Elements 迴圈用 lower_bound 檢查這一項 element 的行列編號有沒有對應 row, column 裡的值，最後用 Rit, Cit 算新的編號後塞入 ret。

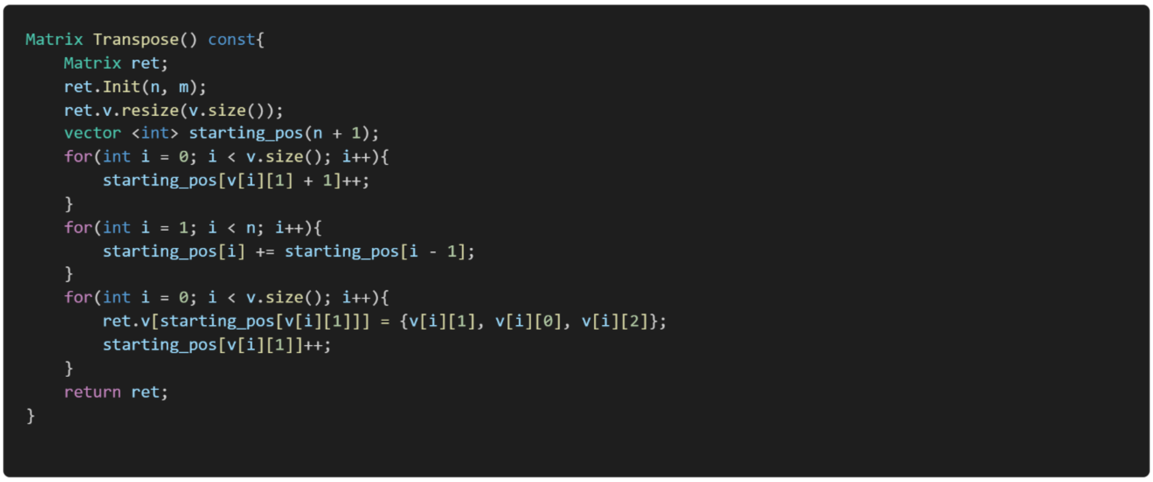
時間複雜度：$O\big(\mathrm{NewRow} + \mathrm{NewColumn} + \mathrm{NewElements} \times (\lg(\mathrm{NewRow}) + \lg(\mathrm{NewColumn}))\big) = O(\mathrm{NewElements} \times \lg(\mathrm{NewElements}))$。

## 2.6 功能分析——轉置矩陣



```
void TransposePhase(){
    Matrix *NowMatrix = MatrixPickingList("Please enter the name of the Matrix you want to get submatrix : ");
    cout<<endl;
    cout<<"Transpose Ver."<<endl;
    NowMatrix->Transpose().Print();
    SavingMatrix(NowMatrix->Transpose());
    PauseAndReturnMainPage();
}
```

讓使用者選取原矩陣後再呼叫 Transpose()

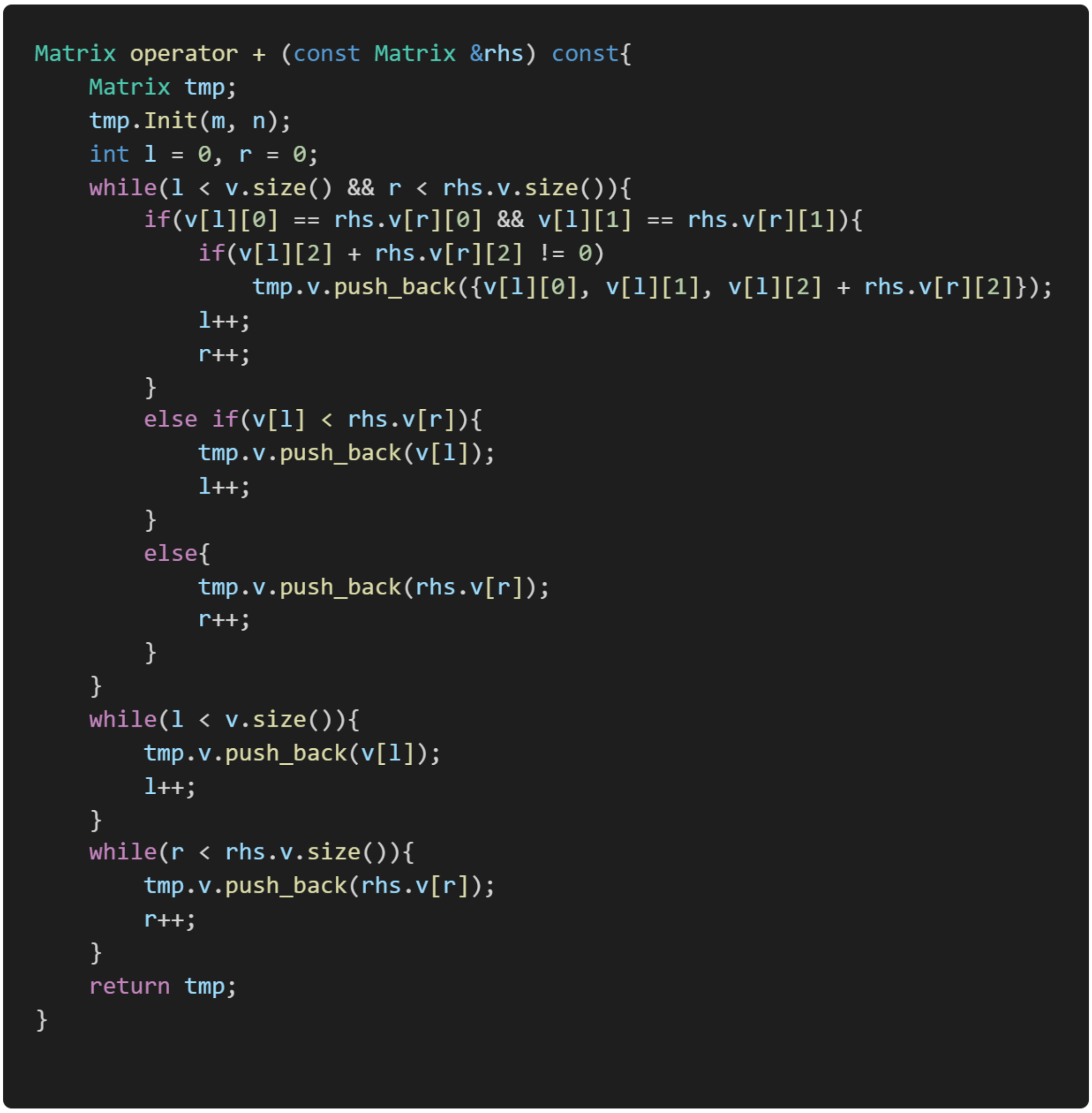


```
Matrix Transpose() const{
    Matrix ret;
    ret.Init(n, m);
    ret.v.resize(v.size());
    vector <int> starting_pos(n + 1);
    for(int i = 0; i < v.size(); i++){
        starting_pos[v[i][1] + 1]++;
    }
    for(int i = 1; i < n; i++){
        starting_pos[i] += starting_pos[i - 1];
    }
    for(int i = 0; i < v.size(); i++){
        ret.v[starting_pos[v[i][1]]] = {v[i][1], v[i][0], v[i][2]};
        starting_pos[v[i][1]]++;
    }
    return ret;
}
```

先 Init 一個空矩陣，將 ret.v resize 成原矩陣 v 之大小，跑一個 Elements 迴圈紀錄每一個 column 編號佔幾格，再跑一個 n 迴圈把 starting_pos 更新成編號為 i 的第一個位置，最後跑個 Elements 迴圈把行列交換後的 array<int, 3>塞至對應的位置。

時間複雜度：$O(\mathrm{Elements} + n + \mathrm{Elements}) = O(\mathrm{Elements})$。

## 2.7 功能分析——矩陣加法

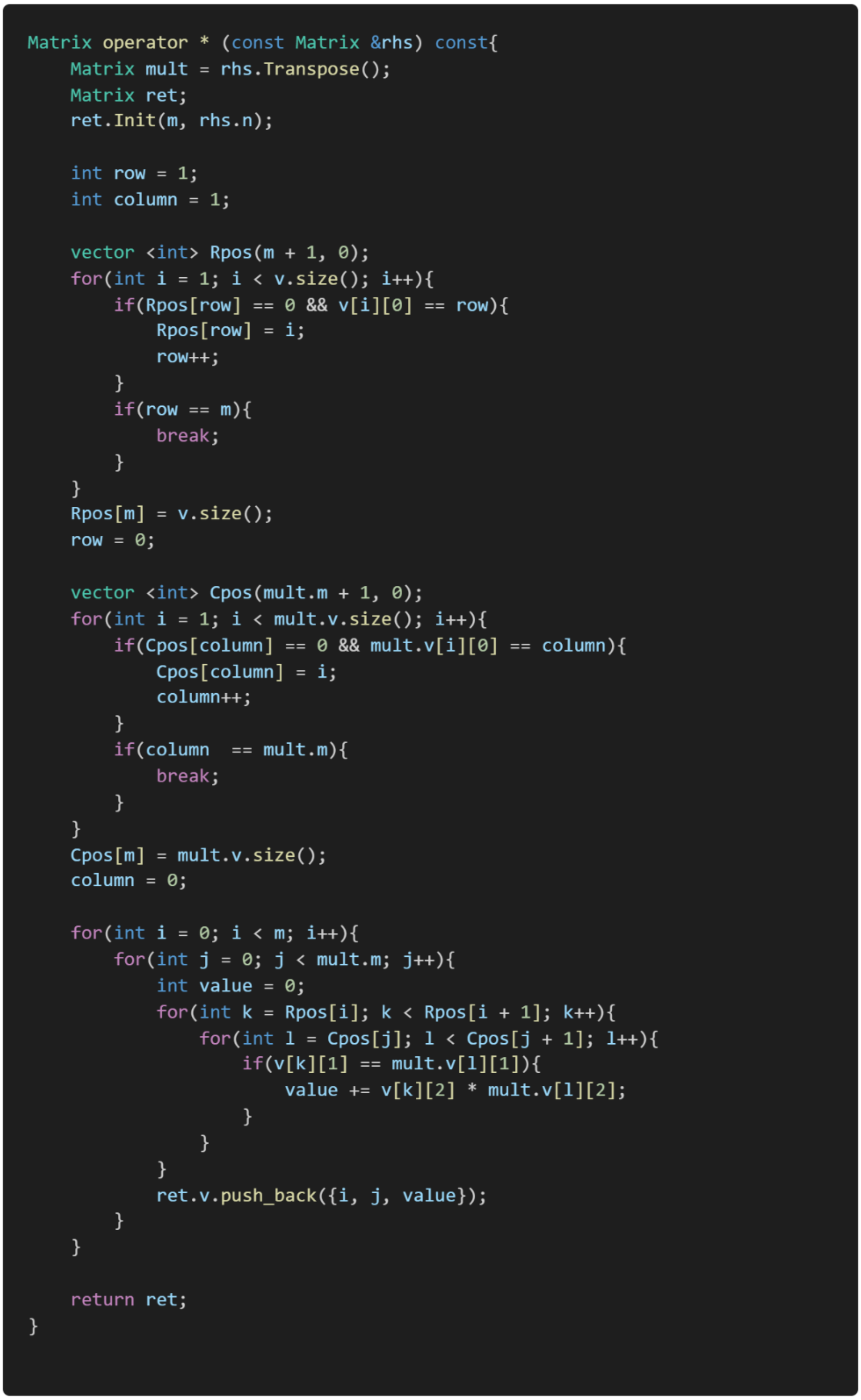


```
Matrix operator + (const Matrix &rhs) const{
    Matrix tmp;
    tmp.Init(m, n);
    int l = 0, r = 0;
    while(l < v.size() && r < rhs.v.size()){
        if(v[l][0] == rhs.v[r][0] && v[l][1] == rhs.v[r][1]){
            if(v[l][2] + rhs.v[r][2] != 0)
                tmp.v.push_back({v[l][0], v[l][1], v[l][2] + rhs.v[r][2]});
            l++;
            r++;
        }
        else if(v[l] < rhs.v[r]){
            tmp.v.push_back(v[l]);
            l++;
        }
        else{
            tmp.v.push_back(rhs.v[r]);
            r++;
        }
    }
    while(l < v.size()){
        tmp.v.push_back(v[l]);
        l++;
    }
    while(r < rhs.v.size()){
        tmp.v.push_back(rhs.v[r]);
        r++;
    }
    return tmp;
}
```

用兩個指針指著兩個 Sparse Matrix List 從小到大遍歷，將順位較小的那項放進去後指針往後一格，若兩格順位一致則 value 相加後再放進去、兩個指針同時往後。

時間複雜度：O(NewElements)

## 2.8 功能分析——矩陣乘法

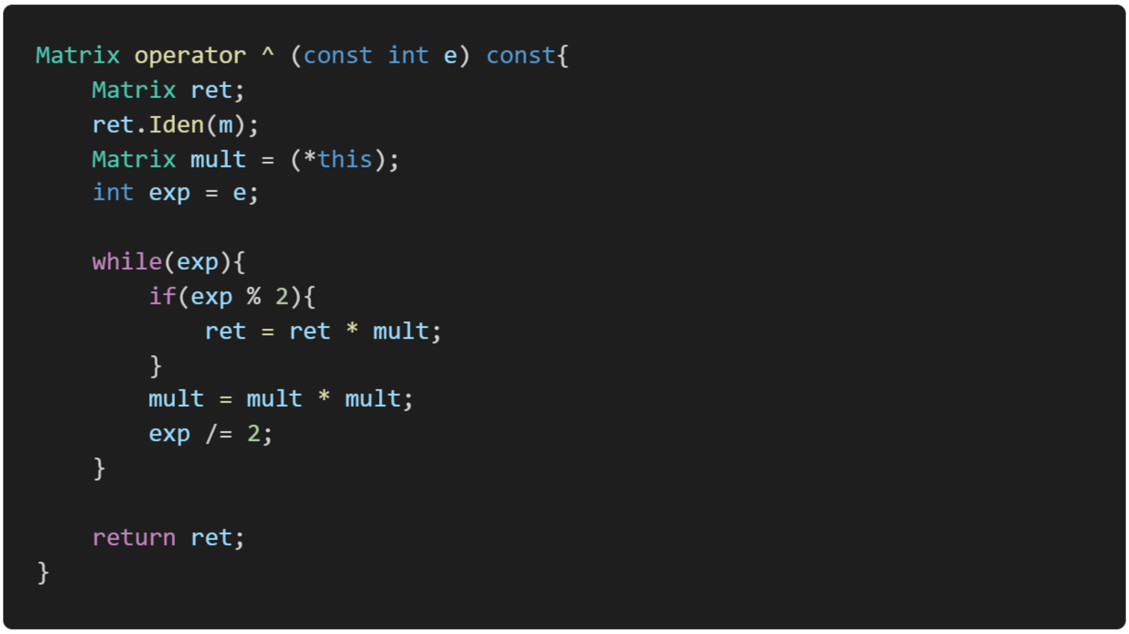


```
Matrix operator * (const Matrix &rhs) const{
    Matrix mult = rhs.Transpose();
    Matrix ret;
    ret.Init(m, rhs.n);

    int row = 1;
    int column = 1;

    vector <int> Rpos(m + 1, 0);
    for(int i = 1; i < v.size(); i++){
        if(Rpos[row] == 0 && v[i][0] == row){
            Rpos[row] = i;
            row++;
        }
        if(row == m){
            break;
        }
    }
    Rpos[m] = v.size();
    row = 0;

    vector <int> Cpos(mult.m + 1, 0);
    for(int i = 1; i < mult.v.size(); i++){
        if(Cpos[column] == 0 && mult.v[i][0] == column){
            Cpos[column] = i;
            column++;
        }
        if(column  == mult.m){
            break;
        }
    }
    Cpos[m] = mult.v.size();
    column = 0;

    for(int i = 0; i < m; i++){
        for(int j = 0; j < mult.m; j++){
            int value = 0;
            for(int k = Rpos[i]; k < Rpos[i + 1]; k++){
                for(int l = Cpos[j]; l < Cpos[j + 1]; l++){
                    if(v[k][1] == mult.v[l][1]){
                        value += v[k][2] * mult.v[l][2];
                    }
                }
            }
            ret.v.push_back({i, j, value});
        }
    }

    return ret;
}
```

先跑兩個迴圈確認每個編號的起始點，然後$\forall(i, j)$去遍歷那個區間，將所有符合的匹配組相乘後加進$m_{ij}$。

時間複雜度：$O(\text{lhsElements} + \text{rhsElements} + \text{lhsElements} \times \text{rhsElements}) = O(\text{lhsElements} \times \text{rhsElements})$

## 2.9 功能分析——矩陣次方/矩陣快速冪



```cpp
Matrix operator ^ (const int e) const{
    Matrix ret;
    ret.Iden(m);
    Matrix mult = (*this);
    int exp = e;

    while(exp){
        if(exp % 2){
            ret = ret * mult;
        }
        mult = mult * mult;
        exp /= 2;
    }

    return ret;
}
```

欲求最少次數乘法得出指定次方，可以透過將矩陣不斷平方來將次方數/2，剩餘次方除 2 有餘數則將當前矩陣乘進結果矩陣(初始為 m 階標準矩陣)中。

時間複雜度：$O(\lg(\text{Exponet}) \times \text{Elements}^2)$