ITU GAE HW

Generated by Doxygen 1.10.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 matrix Namespace Reference

myMatrix

Classes

- class Matrix

4.1.1 Detailed Description

myMatrix

This is main matrix This is matrix

Chapter 5

Class Documentation

5.1 Calculator Class Reference

Public Member Functions

- template<typename T1 , typename T2 >
  double **addition** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  double **subtraction** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  double **division** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  double **multiplication** (T1 a, T2 b)
- template<typename T1 >
  double **square** (T1 a)
- template<typename T1 , typename T2 >
  double **exponentiation** (T1 a, T2 b)
- template<typename T1 , typename T2 >
  double **modulus** (T1 a, T2 b)
- std::pair< double, double > **getvals** (int a)

The documentation for this class was generated from the following file:

- include/CalculatorPart2.h

5.2 matrix::Matrix< T1 > Class Template Reference

Public Member Functions

- **Matrix** (const std::vector< std::vector< T1 > > &a)
- Matrix (int axis, const std::vector< T1 > &a)

  Adds two integers.
- **Matrix** (int size, T1 val=0)
- **Matrix** (std::pair< int, int > dimensions, T1 val=0)
- std::pair< int, int > **getSize** ()
- void **print** ()
- T1 **determinant** ()
- void **setData** (int c, int r, T1 val)
- Matrix **transpose** ()
- Matrix **inverse** ()

5.2.1 Constructor & Destructor Documentation

5.2.1.1 Matrix()

```
template<typename T1 >
matrix::Matrix< T1 >::Matrix (
            int axis,
            const std::vector< T1 > & a )  [inline]
```

Adds two integers.

This function takes two integers and returns their sum.

Parameters

a	The first integer.
b	The second integer.

Returns

The sum of a and b.

The documentation for this class was generated from the following file:

- include/MatrixPart3.h

5.3 Point Struct Reference

Public Member Functions

- **Point** (float x, float y, float z)
- void **print** ()
- float **zero_distance** ()
- float **distance** (Point p1, Point p2)
- float **distance** (Point p)
- int **region** ()
- bool **is_in_same_region** (Point p1, Point p2)
- bool **is_in_same_region** (Point p)

The documentation for this struct was generated from the following file:

- include/PointStructurePart1.h

Chapter 6

File Documentation

6.1 CalculatorPart2.h

```
00001 #include <iostream>
00002 #include <cmath>
00003 #include <sstream>
00004 #include <vector>
00005
00006 class Calculator{
00007 public:
00008     template<typename T1,typename T2>
00009     double addition(T1 a, T2 b) {
00010         return a+b;
00011     }
00012
00013     template<typename T1,typename T2>
00014     double subtraction(T1 a,T2 b) {
00015         return addition(a,-b);
00016     }
00017
00018     template<typename T1,typename T2>
00019     double division(T1 a,T2 b) {
00020         try {
00021             if (b) return a/b;
00022             else throw std::runtime_error("Division by 0 error");
00023         } catch (const std::exception& e) {
00024             std::cout « "Exception caught: " « e.what() « std::endl;
00025             std::cout « "Reenter vals: ";
00026             std::pair<T1,T2> z;
00027             z = getvals(2);
00028             return division(z.first,z.second);
00029         }
00030     }
00031
00032     template<typename T1,typename T2>
00033     double multiplication(T1 a,T2 b){
00034         return a*b;
00035     }
00036
00037     template<typename T1>
00038     double square(T1 a){
00039         return multiplication(a,a);
00040     }
00041
00042     template<typename T1, typename T2>
00043     double exponentiation(T1 a, T2 b){
00044         return (T1)pow(a,b);
00045     }
00046
00047     template<typename T1, typename T2>
00048     double modulus(T1 a, T2 b) {
00049         if (b) return subtraction(a,multiplication(int(division(a,b)),b));
00050         else return a;
00051     }
00052
00053     std::pair<double,double> getvals(int a){
00054
00055         try {
00056             std::string i;
00057             std::getline(std::cin, i);
00058             std::stringstream ss(i);
```

```
00059             std::vector<std::string> subs;
00060             std::string sub;
00061             while (std::getline(ss, sub, ' ')) {
00062                 if (sub!="")
00063                     subs.push_back(sub);
00064             }
00065             if (subs.size()!=a) {throw std::runtime_error("Incorrect input count");}
00066             else if (a==1) return std::make_pair(std::stod(subs[0]),1.0f);
00067             else return std::make_pair(std::stod(subs[0]),std::stod(subs[1]));
00068         } catch (const std::exception& e) {
00069             std::cout << "Exception caught: " << e.what() << std::endl;
00070             std::cout << "Reenter val(s): ";
00071 
00072         }
00073         return getvals(a);
00074     }
00075 
00076 };
```

6.2 MatrixPart3.h

```
00001 #include <vector>
00002 
00004 
00007 namespace matrix {
00008 template<typename T1>
00009 class Matrix {
00010 private:
00011     std::vector<std::vector<T1>> mat;
00012     std::vector<std::vector<T1>> matHelper;
00013     int col=0,row=0;
00014     void resize(){
00015         col = mat.size();
00016         (col) && (row = mat[0].size());
00017 
00018     }
00019 public:
00020 
00021     Matrix(){
00022         mat = std::vector(1,std::vector(1,0));
00023         resize();
00024     }
00025 
00026     Matrix(const std::vector<std::vector<T1>> &a) : mat(a) {
00027         resize();
00028     }
00029 
00039     Matrix(int axis ,const std::vector<T1> &a ){
00040         if (axis){
00041             for (auto x:a){
00042                 std::vector<T1> temp(1,x);
00043                 mat.push_back(temp);
00044             }
00045         }
00046         else{
00047             mat.resize(1);
00048             for (auto x:a){
00049                 mat[0].push_back(x);
00050             }
00051         }
00052         resize();
00053     }
00054 
00055     Matrix(int size , T1 val = 0) : mat(std::vector(size, std::vector<T1>(size, val))) {
00056         resize();
00057     }
00058 
00059     Matrix(std::pair<int,int> dimensions, T1 val = 0) : mat(std::vector(dimensions.first,
      std::vector<T1>(dimensions.second, val))) {
00060         resize();
00061     }
00062 
00063     std::pair<int,int> getSize(){
00064         return std::make_pair(col,row);
00065     }
00066 
00067     void print(){
00068         for (int i=0;i<col;i++){
00069             for (int j=0;j<row;j++)
00070                 std::cout<<mat[i][j]<<" ";
00071             std::cout<<std::endl;
00072         }
00073     }
```

```
00074
00075     T1 determinant(){
00076         try {
00077             if (col!=row) throw std::runtime_error("not a square matrix");
00078             else if (col==0) throw std::runtime_error("not enough size");
00079         }catch (const std::exception& err){
00080             std::cout«"Error: "«err.what()«std::endl;
00081             return 0;
00082         }
00083         if (!col) return 1;
00084         // 1X1
00085         if (col==1) return mat[0][0];
00086         T1 det=0;
00087         for (int i=0;i<row;i++){
00088             Matrix<T1> ms({col - 1, row - 1},0);
00089             for (int j=1;j<col;j++){
00090                 int r=0;
00091                 for (int k=0;k<row;k++){
00092                     if (k!=i){
00093                     ms.setData(j-1,r,mat[j][k]);
00094                     r++;}
00095                 }
00096             }
00097             det+=(i%2==0?1:-1)*mat[0][i]*ms.determinant();
00098         }
00099         return det;
00100     }
00101
00102     void setData(int c,int r,T1 val){
00103         mat[c][r]=val;
00104     }
00105
00106     Matrix transpose() {
00107         std::vector<std::vector<T1» z = mat;
00108         int rows = col;
00109         int cols = row;
00110         std::vector<std::vector<T1» result(cols, std::vector<T1>(rows));
00111         for (int i = 0; i < rows; ++i) {
00112             for (int j = 0; j < cols; ++j) {
00113                 result[j][i] = z[i][j];
00114             }
00115         }
00116         return Matrix(result);
00117     }
00118
00119     Matrix inverse(){
00120         double det=this->determinant();
00121         if (det==0) throw std::runtime_error("Non-Invertable matrix determinant equals to 0");
00122         double var=1.0/det;
00123
00124     }
00125 };
00126
00127 }
```

6.3 PointStructurePart1.h

```
00001 #include <iostream>
00002 #include <cmath>
00003 #include <bitset>
00004
00005 enum quadrants{
00006     FIRST=1,
00007     SECOND=2,
00008     THIRD=3,
00009     FOURTH=4,
00010     FIFTH=5,
00011     SIXTH=6,
00012     SEVENTH=7,
00013     EIGHTH=8,
00014     ONLINE=9
00015 };
00016
00017
00018
00019 struct Point {
00020     private:
00021         float x;
00022         float y;
00023         float z;
00024
00025     public:
00026         Point(float x, float y, float z) : x(x), y(y), z(z) {}
```

```
00027         Point() : x(0), y(0), z(0) {}
00028         void print();
00029         float zero_distance();
00030         float distance(Point p1, Point p2);
00031         float distance(Point p);
00032         int region();
00033         bool is_in_same_region(Point p1, Point p2);
00034         bool is_in_same_region(Point p);
00035
00036 };
```

Index