Solving Linear Systems

- Want to solve system of the form:

$$Ax = b$$

- ***A*** **is symmetric:**

$$A^T = A$$

- ***A*** **is positive-definite:**

$$x^T A x > 0 \quad \forall x \neq 0$$

Interface

```
// Matrix class the solver will accept
class implicitMatrix
{
 public:
  virtual void matVecMult(double x[], double b[]) = 0;
};


// Solve Ax = b for a symmetric, positive definite matrix A
double ConjGrad(int n, implicitMatrix *A, double x[], double b[],
      double epsilon,   // how low should we go?
      int    *steps);
```

Implicit Matrix

```
// Matrix class the solver will accept
class implicitMatrix
{
 public:
  virtual void matVecMult(double x[], double b[]) = 0;
};
```

- `matVecMult`: **a method that performs matrix multiplication**
- `x`: **the input vector**
- `b`: **the output vector**

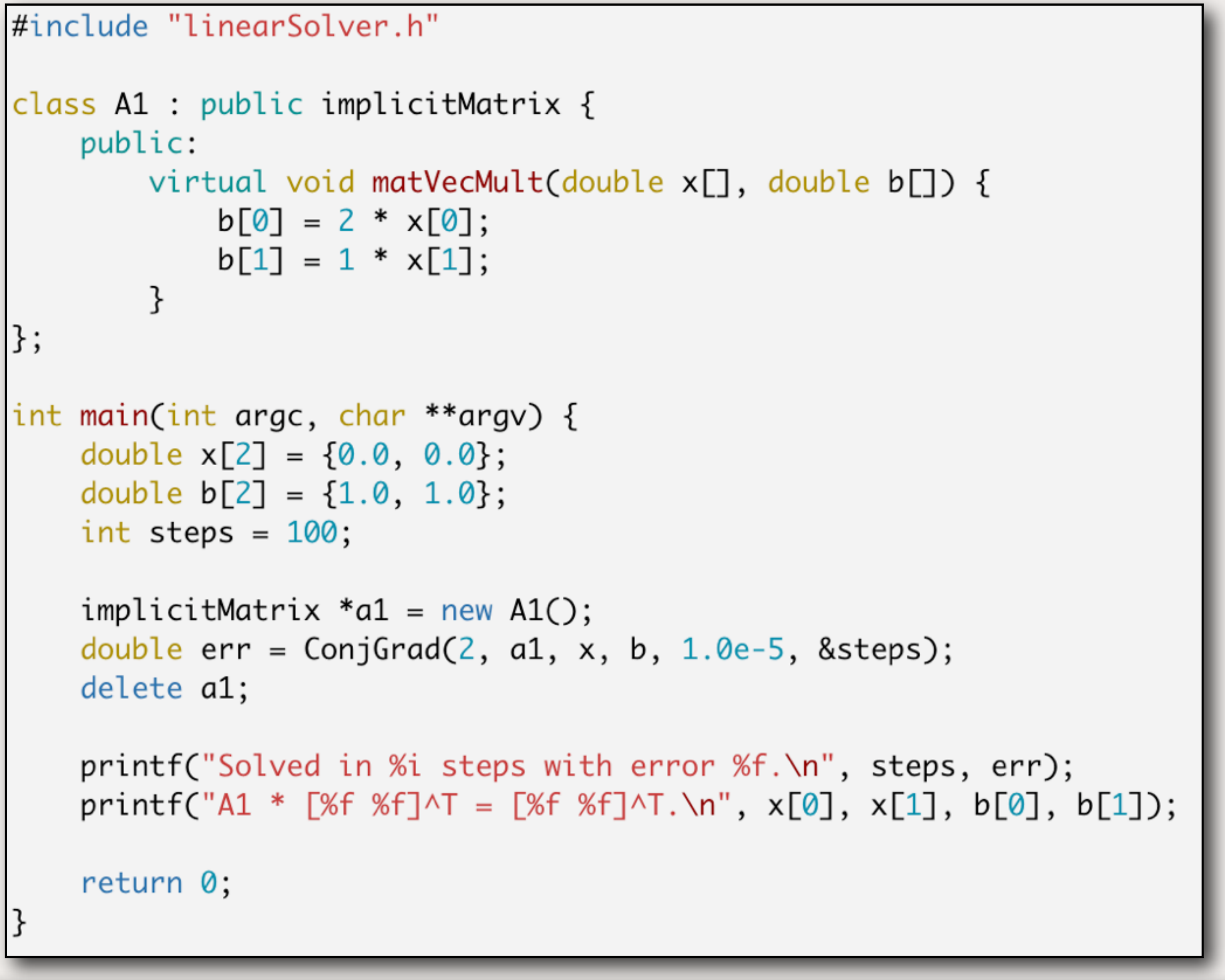
Implicit Matrix

```
// Solve Ax = b for a symmetric
// positive definite matrix A
double ConjGrad(int n, implicitMatrix *A,
  double x[], double b[],
  double epsilon,
  int    *steps);
```

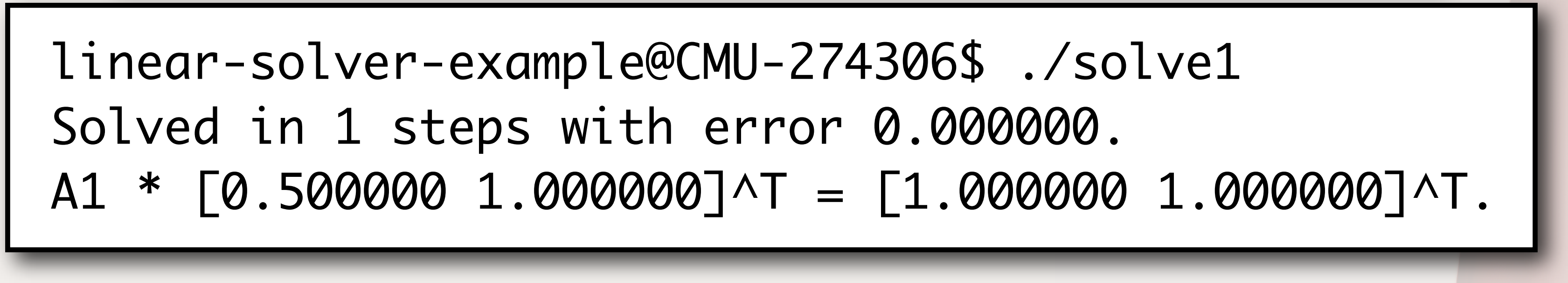
- `n`: **number of dimensions**
- `implicitMatrix`: **matrix instance**
- `x`: **the *output* vector**
- `b`: **the *input* vector**
- `epsilon`: **how low should we go? (1.0^{-5})**
- `steps`: ***inputs* the max steps and *outputs the actual steps***

Example 1

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



```
#include "linearSolver.h"

class A1 : public implicitMatrix {
    public:
        virtual void matVecMult(double x[], double b[]) {
            b[0] = 2 * x[0];
            b[1] = 1 * x[1];
        }
};

int main(int argc, char **argv) {
    double x[2] = {0.0, 0.0};
    double b[2] = {1.0, 1.0};
    int steps = 100;

    implicitMatrix *a1 = new A1();
    double err = ConjGrad(2, a1, x, b, 1.0e-5, &steps);
    delete a1;

    printf("Solved in %i steps with error %f.\n", steps, err);
    printf("A1 * [%f %f]^T = [%f %f]^T.\n", x[0], x[1], b[0], b[1]);

    return 0;
}
```

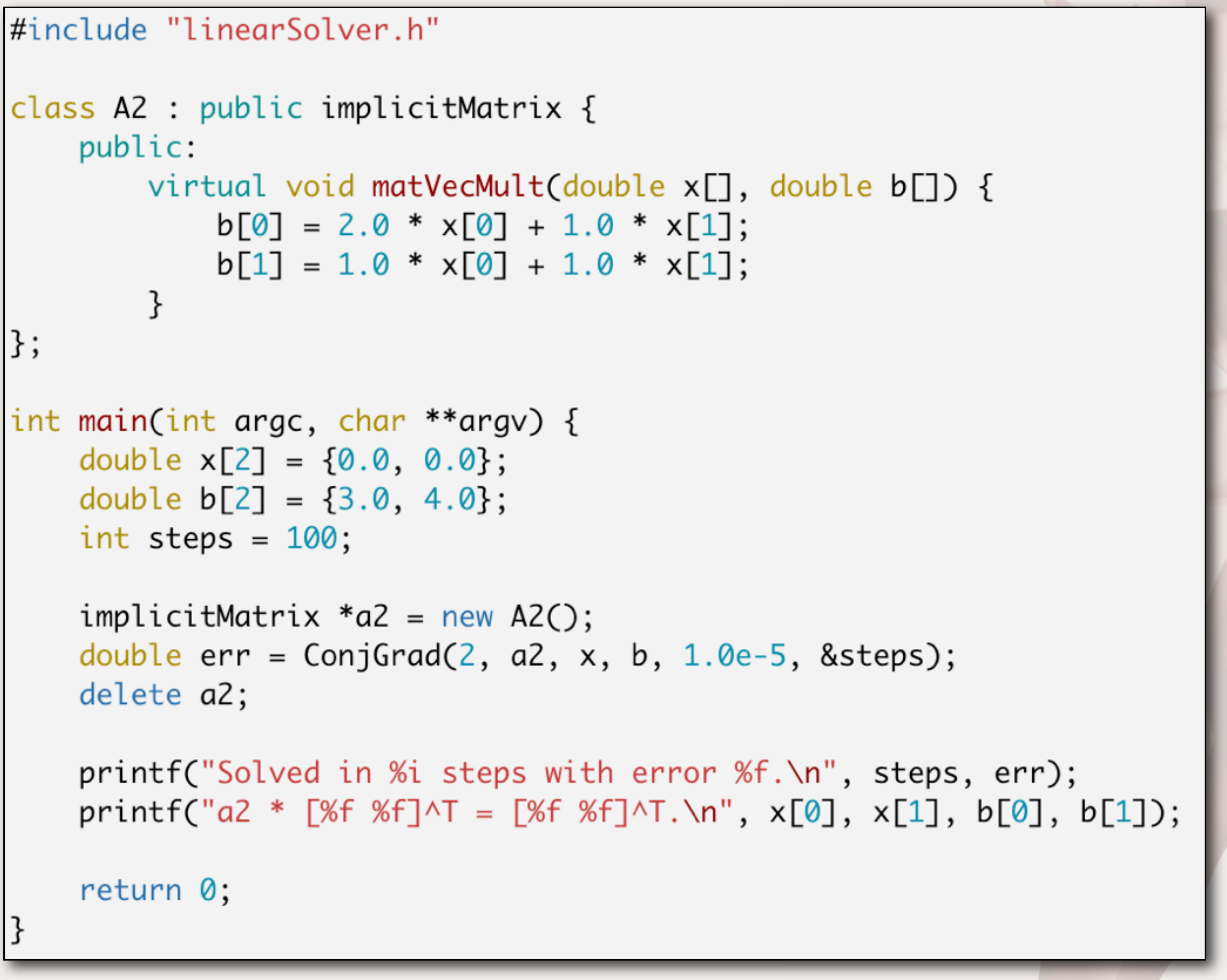


```
linear-solver-example@CMU-274306$ ./solve1
Solved in 1 steps with error 0.000000.
A1 * [0.500000 1.000000]^T = [1.000000 1.000000]^T.
```

Example 2

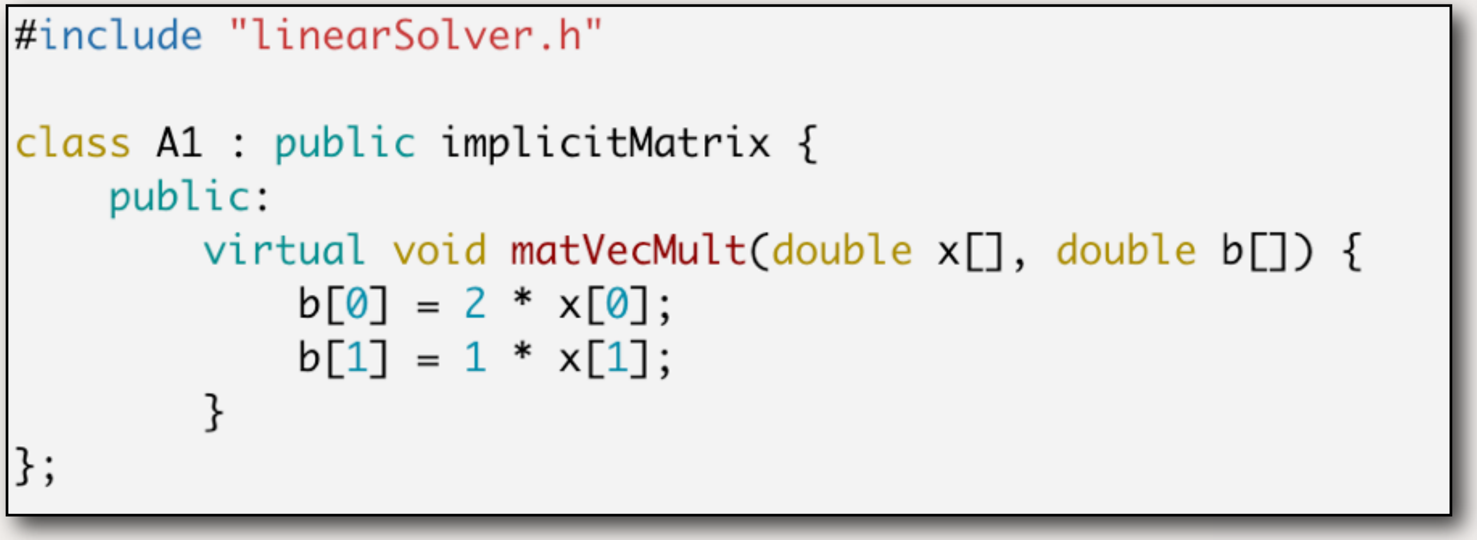
$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} x = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Example 2



```
#include "linearSolver.h"

class A2 : public implicitMatrix {
    public:
        virtual void matVecMult(double x[], double b[]) {
            b[0] = 2.0 * x[0] + 1.0 * x[1];
            b[1] = 1.0 * x[0] + 1.0 * x[1];
        }
};

int main(int argc, char **argv) {
    double x[2] = {0.0, 0.0};
    double b[2] = {3.0, 4.0};
    int steps = 100;

    implicitMatrix *a2 = new A2();
    double err = ConjGrad(2, a2, x, b, 1.0e-5, &steps);
    delete a2;

    printf("Solved in %i steps with error %f.\n", steps, err);
    printf("a2 * [%f %f]^T = [%f %f]^T.\n", x[0], x[1], b[0], b[1]);

    return 0;
}
```

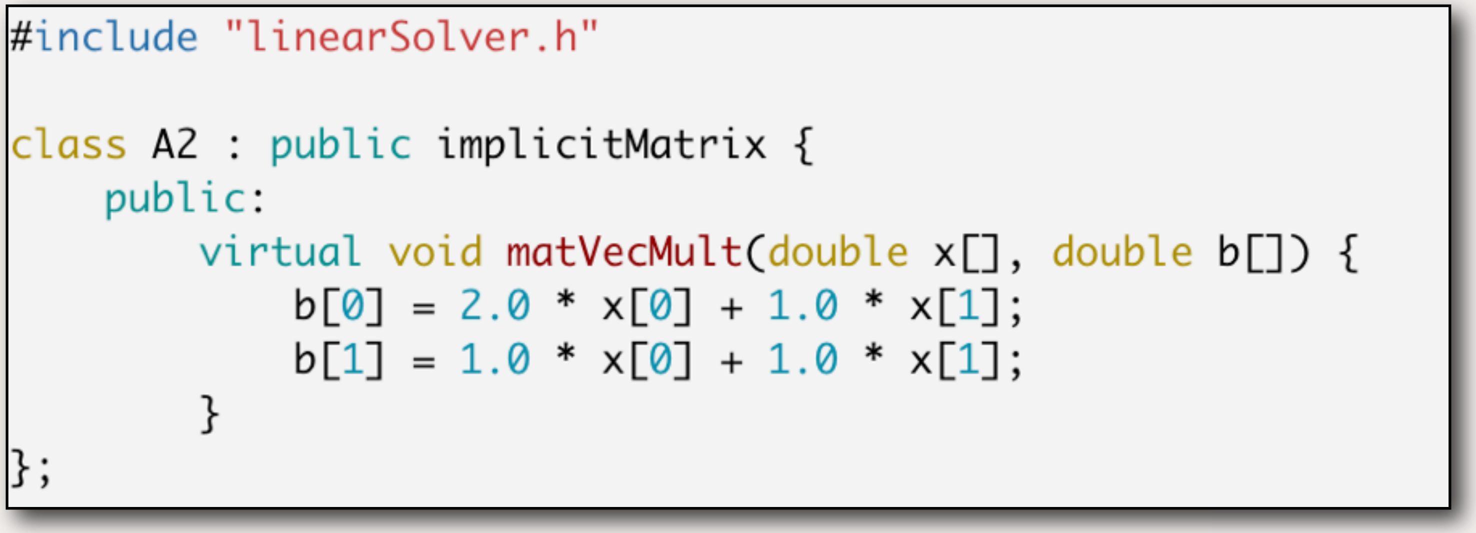
Why implicitMatrix?



```
#include "linearSolver.h"

class A1 : public implicitMatrix {
    public:
        virtual void matVecMult(double x[], double b[]) {
            b[0] = 2 * x[0];
            b[1] = 1 * x[1];
        }
};
```

***O*(*n*)**

vs



```
#include "linearSolver.h"

class A2 : public implicitMatrix {
    public:
        virtual void matVecMult(double x[], double b[]) {
            b[0] = 2.0 * x[0] + 1.0 * x[1];
            b[1] = 1.0 * x[0] + 1.0 * x[1];
        }
};
```

$O(n^2)$

Example 3

$$\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} x = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Not positive definite!

$$[1 \quad 0] \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -1$$

Example 3

- **What if *A* is not symmetric or not positive-definite?**

$$Ax = b$$

- **Then solve the *normal* equations:**

$$A^T A x = A^T b$$