Lecture 12 - High Level Design and C++

Prof. Brendan Kochunas

October 10 2022

Contents

1 Review of Lab06 Software Design

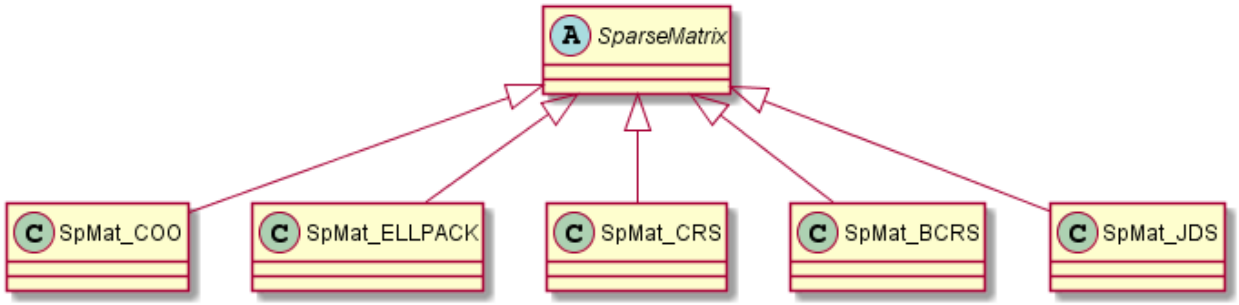
To begin, we review the software design developed in Lab 06.

1.1 Class Hierarchy Abstraction

Steps to developing a class hierarchy

1. Start with the sparse matrix class
2. Abstract or concrete? Why?
3. What are the specific types of sparse matrix?
4. One level hierarchy?

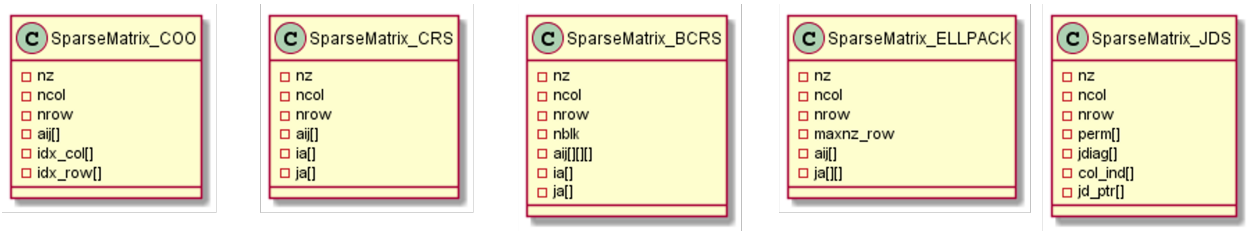
```
@startuml

abstract class "SparseMatrix"
SparseMatrix <|-- SpMat_COO
SparseMatrix <|-- SpMat_ELLPACK
SparseMatrix <|-- SpMat_CRS
SparseMatrix <|-- SpMat_BCRS
SparseMatrix <|-- SpMat_JDS

@enduml
```



Class Hierarchy with Attributes

Here we want to determine which attributes are required for us to be able to implement the methods. This means understanding what *defines* each class. Like least common denominators or greatest common factors, we strive to identify the minimal sets of attributes that define a class and are shared amongst the classes.

Our approach here is to attempt to define the attributes for each concrete class, then identify shared attributes amongst the the different concrete classes to define on the base class. To make good use of encapsulation, we desire that all our attributes are private.



1.1.1 COO Format

```
@startuml

class SparseMatrix_COO
SparseMatrix_COO : - nz
SparseMatrix_COO : - ncol
SparseMatrix_COO : - nrow
SparseMatrix_COO : - aij[]
SparseMatrix_COO : - idx_col[]
SparseMatrix_COO : - idx_row[]

@enduml
```

1.1.2 CRS Format

```
@startuml

class SparseMatrix_CRS
SparseMatrix_CRS : - nz
SparseMatrix_CRS : - ncol
SparseMatrix_CRS : - nrow
SparseMatrix_CRS : - aij[]
SparseMatrix_CRS : - ia[]
SparseMatrix_CRS : - ja[]

@enduml
```

1.1.3 BCRS Format

```
@startuml

class SparseMatrix_BCRS
SparseMatrix_BCRS : - nz
SparseMatrix_BCRS : - ncol
SparseMatrix_BCRS : - nrow
SparseMatrix_BCRS : - nblk
SparseMatrix_BCRS : - aij[][][]
SparseMatrix_BCRS : - ia[]
SparseMatrix_BCRS : - ja[]

@enduml
```

1.1.4 ELLPACK Format

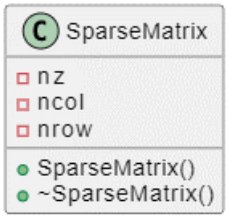
```
@startuml

class SparseMatrix_ELLPACK
SparseMatrix_ELLPACK : - nz
SparseMatrix_ELLPACK : - ncol
SparseMatrix_ELLPACK : - nrow
SparseMatrix_ELLPACK : - maxnz_row
SparseMatrix_ELLPACK : - aij[]
SparseMatrix_ELLPACK : - ja[][]

@enduml
```

1.1.5 JDS Format

```
@startuml

class SparseMatrix_JDS
SparseMatrix_JDS : - nz
SparseMatrix_JDS : - ncol
SparseMatrix_JDS : - nrow
SparseMatrix_JDS : - perm[]
SparseMatrix_JDS : - jdiag[]
SparseMatrix_JDS : - col_ind[]
SparseMatrix_JDS : - jd_ptr[]

@enduml
```

From here, we can identify the following attributes. The generic class can have `nz`, `nrow`, and `ncol`.

```
@startuml

class SparseMatrix
SparseMatrix : - nz
SparseMatrix : - ncol
SparseMatrix : - nrow
' Requried constructor/destructor
SparseMatrix : + SparseMatrix()
SparseMatrix : + ~SparseMatrix()

@enduml
```



2 Defining C++ Classes

Before continuing in the complexity of the design, it'll be good to first establish some basic knowledge on how to implement a class in C++.

2.1 Step 1 – Setup

To begin we will define a main program that is a simple hello world. This file will also include a header file for our class. For now we'll leave it blank.

```
1 //An empty header file
```

Listing 1: Empty C++ Header file (SpMV.hpp)

```
1 #include <iostream>
2 #include "SpMV.hpp"
3
4 int main(int argc, char* argv[])
5 {
6     std::cout << "Hello World!" << std::endl;
7     return 0;
8 }
```

Listing 2: Simple C++ Program (SpMV.cpp)

2.2 Step 2 – Best Practices: Include Guards and Namespacing

Before we put a basic class definition in our header file, we'll demonstrate some C++ best practices. Include guards prevent multiple inclusion of the same header information when included across multiple files. Not doing this can lead to compiler errors. The preprocessor symbol should be something globally unique.

By using a namespace we present our class names from "colliding" with other C++ classes/libraries/etc. This is a best practice as you might imagine someone may want to use your classes with classes from another package that may use the same name for a class!

```
1 #ifndef _SPMV570_
2 #define _SPMV570_
3
4 // These first two lines prevent the following code from being included more than once!
5
6 namespace SpMV
7 {
8     //Code goes here
9 }
10
11 #endif
```

Listing 3: C++ Header file with include guards and namespacing

2.3 Step 3 – A simple class definition

Here we show how to declare a basic class in C++. Then we show how to declare variables of this type in the main program and show how the constructor and destructor are implicitly called.

```
#ifndef _SPMV570_
#define _SPMV570_

#include <iostream>

namespace SpMV
{
    class SparseMatrix
    {
    private:
        /* data */
    public:
        SparseMatrix(/* args */);
        ~SparseMatrix();
    };

    SparseMatrix::SparseMatrix(/* args */)
    {
        std::cout << "Hello from SparseMatrix Constructor!" << std::endl;
    }

    SparseMatrix::~SparseMatrix()
    {
        std::cout << "Hello from SparseMatrix Destructor!" << std::endl;
    }
}

#endif
```

Listing 4: C++ Header with basic class definition

```
#include <iostream>
#include "SpMV.hpp"

int main(int argc, char* argv[])
{
    std::cout << "Hello World!" << std::endl;

    SpMV::SparseMatrix* ptr_A = new SpMV::SparseMatrix();

    // New scoping unit. This means variables defined in here, stay here.
    {
        SpMV::SparseMatrix A = SpMV::SparseMatrix();
        std::cout << "Lets do stuff to A!" << std::endl;

    }

    delete(ptr_A);

    return 0;
}
```

Listing 5: C++ Program using our simple class

2.4 Step 4 – Adding attributes to the class

```cpp
#ifndef _SPMV570_
#define _SPMV570_

#include <iostream>
#include <cstddef> //for size_t

namespace SpMV
{
    class SparseMatrix
    {
    private:
        size_t _nrows = 0;
        size_t _ncols = 0;
        size_t _nnz   = 0;

    public:
        SparseMatrix(const int nrows, const int ncols, const int nnz);
        ~SparseMatrix();
    };

    SparseMatrix::SparseMatrix(const int nrows, const int ncols, const int nnz)
    {
        std::cout << "Hello from SparseMatrix Constructor!" << std::endl;
        this->_nrows = nrows;
        this->_ncols = ncols;
        this->_nnz   = nnz;
    }

    SparseMatrix::~SparseMatrix()
    {
        std::cout << "Hello from SparseMatrix Destructor!" << std::endl;
        std::cout << "this->_ncols=" << this->_ncols << std::endl;
        std::cout << "this->_nrows=" << this->_nrows << std::endl;
        std::cout << "this->_nnz  =" << this->_nnz   << std::endl;
    }

}

#endif
```

Listing 6: SparseMatrix class with attributes

Some improvements to this. We may simplify our coding with the "using" keyword. Also, this type of constructor is very common, so there is a simplified syntax that can be used.

```
#ifndef _SPMV570_
#define _SPMV570_

#include <iostream>
#include <cstddef>

using namespace std;

namespace SpMV
{
    class SparseMatrix
    {
    private:
        size_t _nrows = 0;
        size_t _ncols = 0;
        size_t _nnz   = 0;

    public:
        SparseMatrix(const int nrows, const int ncols);
        ~SparseMatrix();
    };

    SparseMatrix::SparseMatrix(const int nrows, const int ncols) :
        _nrows(nrows), _ncols(ncols)
    {
        cout << "Hello from SparseMatrix Constructor!" << endl;
    }

    SparseMatrix::~SparseMatrix()
    {
        cout << "Hello from SparseMatrix Destructor!" << endl;
        cout << "this->_ncols=" << this->_ncols << endl;
        cout << "this->_nrows=" << this->_nrows << endl;
        cout << "this->_nnz  =" << this->_nnz   << endl;
    }

}

#endif
```

Listing 7: A better SparseMatrix class with attributes

```
#include <iostream>
#include "SpMV.hpp"

int main(int argc, char* argv[])
{
    std::cout << "Hello World!" << std::endl;

    SpMV::SparseMatrix* ptr_A = new SpMV::SparseMatrix(1,1);

    // New scoping unit. This means variables defined in here, stay here.
    {
        SpMV::SparseMatrix A = SpMV::SparseMatrix(2,2);
        std::cout << "Lets do stuff to A!" << std::endl;
    }

    delete(ptr_A);
    ptr_A = NULL;

    return 0;
}
```

Listing 8: Program using our SparseMatrix Class

3 Continued review of Lab06

3.1 Abstract Class Details (Methods)

Objective: Identify methods

Process of abstraction

- Generally want answer: How do we want to use our class?
- Approach: look for verbs in requirements
- What's a meaningful name for each?
  - `getMatrixAs()`
  - `matVec()`
  - By default, constructor (`SparseMatrix`) and destructor (`~SparseMatrix`)

Lets consider the usage pattern offered by PETSc for defining a matrix

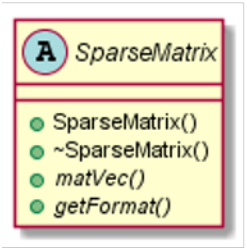
```
MatCreate()
MatSetSizes()
MatSetFromOptions()
MatSetup()
MatGetOwnershipRange()
MatSetValues()
MatAssemblyBegin()
MatAssemblyEnd()
MatDestroy()
```

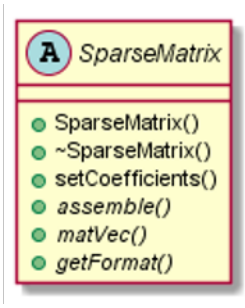
1. `MatCreate()` is like constructor
2. `MatDestroy()` is the destructor
3. We do not have command line option processing so ignore `MatSetFromOptions()`
4. We do not have distributed memory so we ignore `MatGetOwnershipRange()`
5. Simplify creation and add include sizes in constructor. So, ignore `MatSetSizes()`
6. Simplify construction/assemblage into single step by combining `MatAssemblyBegin()` and `MatAssemblyEnd()`
7. From the above, we can also defer the storage format from an initial temporary format to some final format that gets set at assembly time. So, we can ignore `MatSetup()`

SparseMatrix Class with Methods

Therefore, one possible solution is an abstract class like the following.

```
@startuml

abstract class "SparseMatrix"
'Requried constructor/destructor
SparseMatrix : + SparseMatrix()
SparseMatrix : + ~SparseMatrix()

'Additional interfaces to work like PETSc
SparseMatrix : + setCoefficients()
SparseMatrix : + {abstract} assemble()

'Required Interfaces
SparseMatrix : + {abstract} matVec()
SparseMatrix : + {abstract} getMatrixAs()

@enduml
```

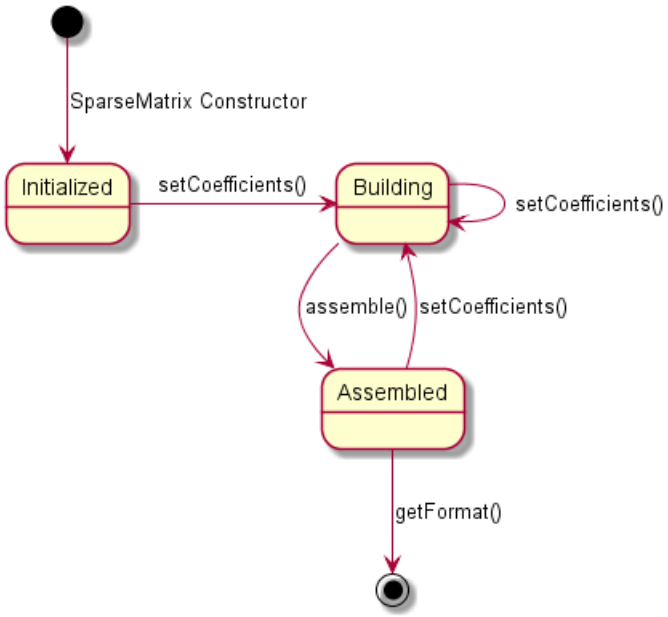




3.2 UML State Diagram (Builder Pattern)

In Lab06 Prof. Kochunas showed we might understand the behavior of our class with a state diagram, and drew something like the followong:

```
@startuml

[*] --> Initialized : SparseMatrix Constructor
Initialized -> Building : setCoefficients()
Building -> Building : setCoefficients()
Building --> Assembled : assemble()
Assembled --> Building : setCoefficients()
Assembled --> [*] : getFormat()

@enduml
```



As we will learn in the next lecture, this is related to the builder pattern.

4 Builder Pattern and DBC

We now have an expected behavior of our software. We may enforce this behavior, and proper use of our class, by adopting design by contract.

```
#ifndef _SPMV570_
#define _SPMV570_

#include <iostream>
#include <cstddef>

using namespace std;

namespace SpMV
{
    class SparseMatrix
    {
    private:
        size_t _nrows = 0;
        size_t _ncols = 0;
        size_t _nnz   = 0;

    public:
        SparseMatrix(const int nrows, const int ncols);
        ~SparseMatrix();

        void setCoefficient(const size_t row, const size_t col, const double aij);
        void assembleStorage();
        SparseMatrix getFormat();
    };

    SparseMatrix::SparseMatrix(const int nrows, const int ncols) :
        _nrows(nrows), _ncols(ncols)
    {
        cout << "Hello from SparseMatrix Constructor!" << endl;
    }

    SparseMatrix::~SparseMatrix()
    {
        cout << "Hello from SparseMatrix Destructor!" << endl;
        cout << "this->_ncols=" << this->_ncols << endl;
        cout << "this->_nrows=" << this->_nrows << endl;
        cout << "this->_nnz  =" << this->_nnz   << endl;
    }

    void SparseMatrix::setCoefficient(const size_t row, const size_t col, const double aij)
    {
        cout << "Hello from SparseMatrix::setCoefficient!" << endl;
    }

    void SparseMatrix::assembleStorage()
    {
        cout << "Hello from SparseMatrix::assembleStorage!" << endl;
    }
    SparseMatrix SparseMatrix::getFormat()
    {
        cout << "Hello from SparseMatrix::getFormat!" << endl;

        return SparseMatrix(this->_ncols,this->_nrows);
    }

}

#endif
```

Listing 9: SparseMatrix class with all methods

```cpp
#include <iostream>
#include "SpMV.hpp"

int main(int argc, char* argv[])
{
    std::cout << "Hello World!" << std::endl;

    SpMV::SparseMatrix* ptr_A = new SpMV::SparseMatrix(1,1);

    // New scoping unit. This means variables defined in here, stay here.
    {
        SpMV::SparseMatrix A = SpMV::SparseMatrix(2,2);

        A.assembleStorage();
        A.setCoefficient(5,6, 1.0);

        SpMV::SparseMatrix B = A.getFormat();
    }

    delete(ptr_A);
    ptr_A = NULL;

    return 0;
}
```

Listing 10: Program using our SparseMatrix Class