CS471 Project 5

Generated by Doxygen 1.8.13

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 cs471 Namespace Reference

Classes

- class Experiment

  The experiment class runs takes a given ini file path, opens it, parses the parameters, then runs the NEH algorithm with the given parameters.
- struct TestParams

  Simple data structure that stores the test parameters for the experiment.

5.2 fshop Namespace Reference

Classes

- class FlowshopBasic

  The FlowshopBasic class runs the standard flowshop scheduling problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. This class also serves as a base class for the Flowshop with Blocking and Flowshop with No Wait problem variants.
- class FlowshopBlocking

  The FlowshopBlocking class runs the flowshop with blocking problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. Inherits from Flowshop↩ Basic.
- class FlowshopNoWait

  The FlowshopNoWait class runs the flowshop with no wait problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. Inherits from FlowshopBasic.
- struct FlowshopSolution

  The FlowshopSolution struct houses all solution data returned from calculating the objective flowshop function. This includes the cmax value, total flow time, job sequence used, start time matrix, and departure time matrix.
- struct JobTimePair

  Simple struct that pairs a job with it's total processing time. Used for sorting purposes.
- class NEH

  The NEH class runs the NEH algorithm on the given flowshop objective function and attempts to optimize the job sequence that produces the smallest cmax value.

5.3 mdata Namespace Reference

Classes

- class DataTable

  The DataTable class is a simple table of values with labeled columns.

5.4 util Namespace Reference

Classes

- class IniReader

  The IniReader class is a simple ∗.ini file reader and parser.

Functions

- template<class T = double>
  void initArray (T ∗a, size_t size, T val)

  Initializes an array with some set value.
- template<class T = double>
  void initMatrix (T ∗∗m, size_t rows, size_t cols, T val)

  Initializes a matrix with a set value for each entry.
- template<class T = double>
  bool releaseArray (T ∗&a)

  Releases an allocated array's memory and sets the pointer to nullptr.
- template<class T = double>
  void releaseMatrix (T ∗∗&m, size_t rows)

  Releases an allocated matrix's memory and sets the pointer to nullptr.
- template<class T = double>
  T ∗ allocArray (size_t size)

  Allocates a new array of the given data type.
- template<class T = double>
  T ∗∗ allocMatrix (size_t rows, size_t cols)

  Allocates a new matrix of the given data type.
- template<class T = double>
  T ∗∗ loadMatrixFromFile (const char ∗filePath, size_t &outNumRows, size_t &outNumCols)
- template<class T = double>
  void outputMatrix (std::ostream &os, T ∗∗matrix, size_t rows, size_t cols, int colWidth=3)
- template<class T = double>
  void copyArray (T ∗src, T ∗dest, size_t size)

  Copies the elements from one equal-sized array to another.

5.4.1 Function Documentation

5.4.1.1 allocArray()

```
template<class T = double>
T* util::allocArray (
              size_t size )   [inline]
```

Allocates a new array of the given data type.

Template Parameters

Data	type of the array

Parameters

size	Number of elements in the array

Returns

Returns a pointer to the new array, or nullptr allocation fails

Definition at line 121 of file mem.h.

```
{
    return new(std::nothrow) T[size];
}
```

5.4.1.2 allocMatrix()

```
template<class T = double>
T** util::allocMatrix (
            size_t rows,
            size_t cols ) [inline]
```

Allocates a new matrix of the given data type.

Template Parameters

Data	type of the matrix entries

Parameters

rows	The number of rows
cols	The number of columns

Returns

Returns a pointer to the new matrix, or nullptr if allocation fails

Definition at line 135 of file mem.h.

```
{
    T** m = (T**)allocArray<T*>(rows);
    if (m == nullptr) return nullptr;

    for (size_t i = 0; i < rows; i++)
    {
        m[i] = allocArray<T>(cols);
        if (m[i] == nullptr)
        {
            releaseMatrix<T>(m, rows);
            return nullptr;
        }
    }

    return m;
}
```

5.4.1.3 copyArray()

```
template<class T = double>
void util::copyArray (
              T * src,
              T * dest,
              size_t size )  [inline]
```

Copies the elements from one equal-sized array to another.

Template Parameters

Data	type of the array

Parameters

src	Source array from where the elements will be copied from
dest	Destination array from where the elements will be copied to
size	Number of elements in the array

Definition at line 255 of file mem.h.

```
    {
        for (size_t i = 0; i < size; i++)
            dest[i] = src[i];
    }
```

5.4.1.4 initArray()

```
template<class T = double>
void util::initArray (
              T * a,
              size_t size,
              T val )  [inline]
```

Initializes an array with some set value.

Template Parameters

Data	type of array

Parameters

a	Pointer to array
size	Size of the array
val	Value to initialize the array to

Definition at line 34 of file mem.h.

Referenced by initMatrix().

```
    {
        if (a == nullptr) return;

        for (size_t i = 0; i < size; i++)
        {
            a[i] = val;
        }
    }
```

5.4.1.5 initMatrix()

```
template<class T = double>
void util::initMatrix (
              T ** m,
              size_t rows,
              size_t cols,
              T val )  [inline]
```

Initializes a matrix with a set value for each entry.

Template Parameters

Data	type of matrix entries

Parameters

m	Pointer to a matrix
rows	Number of rows in matrix
cols	Number of columns in matrix
val	Value to initialize the matrix to

Definition at line 54 of file mem.h.

References initArray().

```
    {
        if (m == nullptr) return;

        for (size_t i = 0; i < rows; i++)
        {
            initArray(m[i], cols, val);
        }
    }
```

5.4.1.6 loadMatrixFromFile()

```
template<class T = double>
T** util::loadMatrixFromFile (
            const char * filePath,
            size_t & outNumRows,
            size_t & outNumCols )  [inline]
```

Definition at line 154 of file mem.h.

```
    {
        outNumRows = 0;
        outNumCols = 0;

        std::ifstream is(filePath);
        if (!is.good())
        {
            std::cerr << "Error loading matrix from file: Unable to open file." << std::endl;
            return nullptr;
        }

        std::string line;
        if (!std::getline(is, line))
        {
            std::cerr << "Error loading matrix from file: File is empty or invalid." << std::endl;
            is.close();
            return nullptr;
        }

        size_t rows = 0;
        size_t cols = 0;

        std::stringstream ss(line);
        if (!(ss >> rows >> cols) || rows == 0 || cols == 0)
        {
            std::cerr << "Error loading matrix from file: Row or column size is zero." << std::endl;
            is.close();
            return nullptr;
        }


        T** retMatrix = allocMatrix<T>(rows, cols);
        if (retMatrix == nullptr)
        {
            std::cerr << "Error loading matrix from file: Matrix memory allocation failed." << std::endl;
            is.close();
            return nullptr;
        }

        for (size_t r = 0; r < rows; r++)
        {
            if (!std::getline(is, line))
            {
                std::cerr << "Error loading matrix from file: EOF reached before reading all rows." <<
      std::endl;
                releaseMatrix<T>(retMatrix, rows);
                is.close();
                return nullptr;
            }

            std::stringstream ss(line);

            for (size_t c = 0; c < cols; c++)
            {
                T entry = 0;
                if (!(ss >> entry))
                {
                    std::cerr << "Error loading matrix from file: EOL reached before reading all cols." << std::endl;
                    releaseMatrix<T>(retMatrix, rows);
                    is.close();
                    return nullptr;
                }

                retMatrix[r][c] = entry;
            }
        }

        is.close();
        outNumRows = rows;
        outNumCols = cols;
        return retMatrix;
    }
```

5.4.1.7 outputMatrix()

```
template<class T = double>
void util::outputMatrix (
              std::ostream & os,
              T ** matrix,
              size_t rows,
              size_t cols,
              int colWidth = 3 )  [inline]
```

Definition at line 228 of file mem.h.

Referenced by fshop::FlowshopSolution::outputAll().

```
    {
        if (matrix == nullptr)
            return;

        for (size_t r = 0; r < rows; r++)
        {
            for (size_t c = 0; c < cols; c++)
            {
                os << std::setw(3) << matrix[r][c];
                if (c < cols - 1)
                    os << " ";
                else
                    os << std::endl;
            }
        }
    }
```

5.4.1.8 releaseArray()

```
template<class T = double>
bool util::releaseArray (
              T *& a )
```

Releases an allocated array's memory and sets the pointer to nullptr.

Template Parameters

Data	type of array

Parameters

a	Pointer to array

Definition at line 71 of file mem.h.

```
    {
        if (a == nullptr) return true;

        try
        {
            delete[] a;
            a = nullptr;
            return true;
        }
        catch(...)
        {
            return false;
        }
    }
```

5.4.1.9 releaseMatrix()

```
template<class T = double>
void util::releaseMatrix (
              T **& m,
              size_t rows )
```

Releases an allocated matrix's memory and sets the pointer to nullptr.

Template Parameters

Data	type of the matrix

Parameters

m	Pointer th the matrix
rows	The number of rows in the matrix

Definition at line 95 of file mem.h.

Referenced by mdata::DataTable< T >::~DataTable().

```
    {
        if (m == nullptr) return;

        for (size_t i = 0; i < rows; i++)
        {
            if (m[i] != nullptr)
            {
                // Release each row
                releaseArray<T>(m[i]);
            }
        }

        // Release columns
        delete[] m;
        m = nullptr;
    }
```

Chapter 6

Class Documentation

6.1 mdata::DataTable< T > Class Template Reference

The DataTable class is a simple table of values with labeled columns.

```
#include <datatable.h>
```

Public Member Functions

- DataTable (size_t _rows, size_t _cols)

  Construct a new Data Table object Throws std::length_error and std::bad_alloc.
- ∼DataTable ()

  Destroy the Data Table object.
- void clearData ()
- std::string getColLabel (size_t colIndex)

  Gets the string label for the column with the given index.
- void setColLabel (size_t colIndex, std::string newLabel)

  Sets the string label for the column with the given index.
- T getEntry (size_t row, size_t col)

  Returns the value in the table at the given row and column.
- void setEntry (size_t row, size_t col, T val)

  Set the value for the table entry at the given row and column.
- bool exportCSV (const char ∗filePath)

  Exports the contents of this DataTable to a .csv file.

6.1.1 Detailed Description

template<class T>
class mdata::DataTable< T >

The DataTable class is a simple table of values with labeled columns.

– Initialize a DataTable object with a specified number of rows and columns: DataTable table(rows, columns);

Set a column's label:

table.setColLabel(0, "Column 1");

Set an entry in the table:

table.setEntry(n, m, value);

Where 'n' is the row, 'm' is the column, and 'value' is the value of the entry

Export the table to a ∗.csv file:

bool success = table.exportCSV("my_file.csv");

Definition at line 50 of file datatable.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 DataTable()

```
template<class T>
mdata::DataTable< T >::DataTable (
              size_t _rows,
              size_t _cols )  [inline]
```

Construct a new Data Table object Throws std::length_error and std::bad_alloc.

Parameters

_rows	Number of rows in table
_cols	Number of columns in table

Definition at line 60 of file datatable.h.

```
                                              : rows(_rows), cols(_cols), dataMatrix(nullptr)
        {
            if (rows == 0)
                throw std::length_error("Table rows must be greater than 0.");
            else if (cols == 0)
                throw std::length_error("Table columns must be greater than 0.");

            dataMatrix = util::allocMatrix<T>(rows, cols);
            if (dataMatrix == nullptr)
                throw std::bad_alloc();

            colLabels.resize(_cols, std::string());
        }
```

6.1.2.2 ∼DataTable()

```
template<class T>
mdata::DataTable< T >::∼DataTable ( )  [inline]
```

Destroy the Data Table object.

Definition at line 77 of file datatable.h.

References util::releaseMatrix().

```
{
    util::releaseMatrix(dataMatrix, rows);
}
```

6.1.3 Member Function Documentation

6.1.3.1 clearData()

```
template<class T>
void mdata::DataTable< T >::clearData ( )  [inline]
```

Definition at line 82 of file datatable.h.

```
{
    util::initMatrix<T>(dataMatrix, rows, cols, 0);
}
```

6.1.3.2 exportCSV()

```
template<class T>
bool mdata::DataTable< T >::exportCSV (
             const char * filePath )  [inline]
```

Exports the contents of this DataTable to a .csv file.

Parameters

filePath	Path to the file that will be filled with this table's values

Returns

true If the file was successfully written to
false If there was an error opening the file

Definition at line 160 of file datatable.h.

Referenced by cs471::Experiment::runNEH().

```
        {
            if (dataMatrix == nullptr) return false;

            using namespace std;
            ofstream outFile;
            outFile.open(filePath, ofstream::out | ofstream::trunc);
            if (!outFile.good()) return false;

            // Print column labels
            for (unsigned int c = 0; c < cols; c++)
            {
                outFile << colLabels[c];
                if (c < cols - 1) outFile << ",";
            }

            outFile << endl;

            // Print data rows
            for (unsigned int r = 0; r < rows; r++)
            {
                for (unsigned int c = 0; c < cols; c++)
                {
                    outFile << std::setprecision(8) << dataMatrix[r][c];
                    if (c < cols - 1) outFile << ",";
                }
                outFile << endl;
            }

            outFile.close();
            return true;
        }
```

6.1.3.3 getColLabel()

```
template<class T>
std::string mdata::DataTable< T >::getColLabel (
              size_t colIndex )  [inline]
```

Gets the string label for the column with the given index.

Parameters

colIndex	Index of the column

Returns

std::string String value of the column label

Definition at line 93 of file datatable.h.

```
        {
            if (colIndex >= colLabels.size())
                throw std::out_of_range("Column index out of range");

            return colLabels[colIndex];
        }
```

6.1.3.4 getEntry()

```
template<class T>
T mdata::DataTable< T >::getEntry (
              size_t row,
              size_t col ) [inline]
```

Returns the value in the table at the given row and column.

Parameters

row	Row index of the table
col	Column index of the table

Returns

T Value of the entry at the given row and column

Definition at line 122 of file datatable.h.

```
{
    if (dataMatrix == nullptr)
        throw std::runtime_error("Data matrix not allocated");
    if (row >= rows)
        throw std::out_of_range("Table row out of range");
    else if (col >= cols)
        throw std::out_of_range("Table column out of range");

    return dataMatrix[row][col];
}
```

6.1.3.5 setColLabel()

```
template<class T>
void mdata::DataTable< T >::setColLabel (
              size_t colIndex,
              std::string newLabel ) [inline]
```

Sets the string label for the column with the given index.

Parameters

colIndex	Index of the column
newLabel	New string label for the column

Definition at line 107 of file datatable.h.

Referenced by cs471::Experiment::runNEH().

```
{
     if (colIndex >= colLabels.size())
```

```
                throw std::out_of_range("Column index out of range");

            colLabels[colIndex] = newLabel;
        }
```

6.1.3.6 setEntry()

```
template<class T>
void mdata::DataTable< T >::setEntry (
            size_t row,
            size_t col,
            T val )  [inline]
```

Set the value for the table entry at the given row and column.

Parameters

row	Row index of the table
col	Column index of the table
val	New value for the entry

Definition at line 141 of file datatable.h.

Referenced by cs471::Experiment::runNEH().

```
        {
            if (dataMatrix == nullptr)
                throw std::runtime_error("Data matrix not allocated");
            if (row >= rows)
                throw std::out_of_range("Table row out of range");
            else if (col >= cols)
                throw std::out_of_range("Table column out of range");

            dataMatrix[row][col] = val;
        }
```

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

- include/datatable.h

6.2 cs471::Experiment Class Reference

The experiment class runs takes a given ini file path, opens it, parses the parameters, then runs the NEH algorithm with the given parameters.

```
#include <experiment.h>
```

Public Member Functions

- Experiment (std::string paramsFile)

  Construct a Experiment object.
- ∼Experiment ()=default
- int runNEH ()

  Runs the cs471 lab 5 experiment, which involves executing the NEH algorithm for a specific flowshop objective function that is specified in the input parameters file.
- int runDebugSeq (int ∗seq, size_t seqSize)

  Used for debugging the objective flowshop functions. This method runs the objective function specified in the parameters file with the specified input processing time files and then prints the results of the flowshop objective function with the given job sequence.

6.2.1 Detailed Description

The experiment class runs takes a given ini file path, opens it, parses the parameters, then runs the NEH algorithm with the given parameters.

Definition at line 42 of file experiment.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Experiment()

```
Experiment::Experiment (
            std::string paramsFile )
```

Construct a Experiment object.

Parameters

paramsFile	File path to the input ini paramater file

Definition at line 44 of file experiment.cpp.

```
00045 {
00046     // Attempt to open parameters file
00047     if (!iniParams.openFile(paramsFile))
00048     {
00049         string msg = "Error opening ini file: ";
00050         msg += paramsFile;
00051         throw std::runtime_error(msg);
00052     }
00053
00054     cout << "Loaded parameters file: " << paramsFile << endl;
00055 }
```

6.2.2.2 ∼Experiment()

```
cs471::Experiment::∼Experiment ( )  [default]
```

6.2.3 Member Function Documentation

6.2.3.1 runDebugSeq()

```
int Experiment::runDebugSeq (
              int * seq,
              size_t seqSize )
```

Used for debugging the objective flowshop functions. This method runs the objective function specified in the parameters file with the specified input processing time files and then prints the results of the flowshop objective function with the given job sequence.

Parameters

seq	Job sequence to run flowshop objective functions with
seqSize	Size of the job sequence array

Returns

Returns a non-zero error code on failure, otherwise zero.

Definition at line 314 of file experiment.cpp.

References cs471::TestParams::algorithm, cs471::TestParams::inputFilesDir, cs471::TestParams::maxTestFile, and cs471::TestParams::minTestFile.

Referenced by runDebugJobSeq().

```
00315 {
00316     // Retrieve test parameters from ini file
00317     TestParams p = readTestParams();
00318 
00319     if (p.algorithm == 1)
00320         cout << "Running Flow Shop with Blocking ..." << endl;
00321     else if (p.algorithm == 2)
00322         cout << "Running Flow Shop with No Wait ..." << endl;
00323     else
00324         cout << "Running Flow Shop Scheduling ..." << endl;
00325 
00326     cout << endl;
00327 
00328     // Prepare pointer to results
00329     fsSol result = nullptr;
00330 
00331     for (int i = p.minTestFile; i <= p.maxTestFile; i++)
00332     {
00333         string fullInputPath = p.inputFilesDir + std::to_string(i) + ".txt";
00334 
00335         cout << "Input file: " << fullInputPath << endl;
00336 
00337         // Get the flowshop objective function that we want to optimize
00338         auto objectiveFs = allocFlowShop(fullInputPath.c_str(), p.algorithm);
00339         if (objectiveFs == nullptr)
```

```
        {
            cout << "Objective flowshop function encountered an error." << endl;
            return 1;
        }

        result = objectiveFs->calcObjective(seq, seqSize);
        result->outputAll(std::cout);

        delete objectiveFs;

        cout << "========================================" << endl;
    }

    cout << "Debug objective function sequence tests completed." << endl;

    return 0;
}
```

6.2.3.2 runNEH()

```
int Experiment::runNEH ( )
```

Runs the cs471 lab 5 experiment, which involves executing the NEH algorithm for a specific flowshop objective function that is specified in the input parameters file.

Returns

int Returns a non-zero error code on failure. Otherwise returns zero.

Definition at line 64 of file experiment.cpp.

References cs471::TestParams::algorithm, ThreadPool::enqueue(), mdata::DataTable< T >::exportCSV(), INI_T↵EST_ALGORITHM, INI_TEST_INPUTFILEDIR, INI_TEST_MAXFILE, INI_TEST_MINFILE, INI_TEST_NUMTHR↵EADS, INI_TEST_RESULTSFILE, INI_TEST_SECTION, INI_TEST_TIMESFILE, cs471::TestParams::inputFiles↵Dir, cs471::TestParams::maxTestFile, cs471::TestParams::minTestFile, cs471::TestParams::numThreads, cs471::↵TestParams::resultsFile, fshop::NEH::run(), mdata::DataTable< T >::setColLabel(), mdata::DataTable< T >::set↵Entry(), ThreadPool::stopAndJoinAll(), and cs471::TestParams::timesFile.

Referenced by main().

```
{
    // Retrieve test parameters from ini file
    TestParams p = readTestParams();

    // Construct data table to store experiment results
    mdata::DataTable<string> resultsTable(p.maxTestFile - p.
minTestFile + 1, 6);

    // Initialize thread pool with a parameter-given number of threads
    ThreadPool tpool(p.numThreads);

    // Initialize thread future vector, used for thread pool synchronization
    // and keeps track of the individual tasks being executed.
    vector<std::future<int>> futures;

    cout << "Started " << p.numThreads << " worker threads ..." << endl;

    if (p.algorithm == 1)
        cout << "Running NEH on Flow Shop with Blocking ..." << endl;
    else if (p.algorithm == 2)
        cout << "Running NEH on Flow Shop with No Wait ..." << endl;
    else
        cout << "Running NEH on Flow Shop Scheduling ..." << endl;

    // Prepare results table column header labels
    resultsTable.setColLabel(0, "Data Set");
    resultsTable.setColLabel(1, "cMax");
    resultsTable.setColLabel(2, "TFT");
    resultsTable.setColLabel(3, "Func Calls");
    resultsTable.setColLabel(4, "Execution Time (ms)");
    resultsTable.setColLabel(5, "Sequence");

    // Add all input test files as tasks in thread pool
    for (int i = p.minTestFile; i <= p.maxTestFile; i++)
    {
        string inputFile = std::to_string(i) + ".txt";
        futures.emplace_back(
            tpool.enqueue(&cs471::Experiment::runNEHThreaded, this, &p, inputFile, i, &resultsTable)
        );
    }

    // const size_t totalFutures = futures.size();

    // Join all thread pool tasks using futures vector
    // and get the return value for each
    for (int i = 0; i < futures.size(); i++)
    {
        int err = futures[i].get();
        if (err)
        {
            // Threaded task returned with an error code, bail
            tpool.stopAndJoinAll();
            return err;
        }
    }

    // Output results table to a csv file
    if (!p.resultsFile.empty())
    {
        resultsTable.exportCSV(p.resultsFile.c_str());
        cout << "Results exported to: " << p.resultsFile << endl;
    }

    return 0;
}
```

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

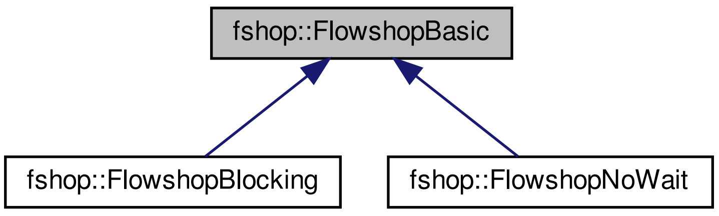
- include/experiment.h
- src/experiment.cpp

6.3 fshop::FlowshopBasic Class Reference

The FlowshopBasic class runs the standard flowshop scheduling problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. This class also serves as a base class for the Flowshop with Blocking and Flowshop with No Wait problem variants.

```
#include <flowshopbasic.h>
```

Inheritance diagram for fshop::FlowshopBasic:



Public Member Functions

- FlowshopBasic (const char ∗procTimeMatrixFile)

  Constructs a new FlowshopBasic object.
- virtual ∼FlowshopBasic ()

  Destroys the FlowshopBasic object.
- virtual std::unique_ptr< FlowshopSolution > calcObjective (int ∗seq, size_t seqSize)

  Calculates the objective flowshop scheduling problem result using the given job sequence.
- virtual int getProcessingTime (size_t machine, size_t job)

  Returns the processing time for the given job on the given machine.
- virtual size_t getTotalJobs ()

  Returns the total number of jobs in the jobs processing time matrix.
- virtual size_t getTotalMachines ()

  Returns the total number of machines in the jobs processing time matrix.
- virtual size_t getFuncCallCounts ()

  Returns the number of times the current flowshop objective function has been executed.
- FlowshopBasic (const FlowshopBasic &o)=delete
- FlowshopBasic (const FlowshopBasic &&o)=delete
- FlowshopBasic & operator= (const FlowshopBasic &o)=delete
- FlowshopBasic & operator= (const FlowshopBasic &&o)=delete

Protected Member Functions

- virtual void validateParams (int ∗seq, size_t seqSize)

  Validates the flowshop input parameters, and throws an exception on error.
- virtual int ∗∗ allocTimeMatrix (size_t rows, size_t cols)

  Allocates the start times and completion time matrices.
- virtual void initTimeMatrix (int ∗∗compTimeMatrix, int ∗seq, size_t rows, size_t cols)

  Initializes the completion time matrix (first row and first column) so that it is ready to be completed with the main algorithm.
- virtual void calcTimeMatrix (int ∗∗compTimeMatrix, int ∗seq, size_t rows, size_t cols)

  Calculates all remaining start and completion times for the current flowshop problem.
- virtual void calcStartTimeCol (int ∗∗startTimeMatrix, int ∗∗departTimeMatrix, int ∗seq, size_t curCol, size_t rows, size_t cols)

  Calculates the start times for a single column. Depends on values in completion time matrix.
- virtual int getCmax (int ∗∗compTimeMatrix, size_t rows, size_t cols)

  Returns the cmax value for a given completion time matrix.
- virtual int getTFT (int ∗∗compTimeMatrix, size_t rows, size_t cols)

  Returns the total flow time value for a given completion time matrix.

Protected Attributes

- int ∗∗ procTimeMatrix
- int ∗∗ startTimeMatrix
- size_t ptMatrixRows
- size_t ptMatrixCols
- size_t funcCallCounter

6.3.1 Detailed Description

The FlowshopBasic class runs the standard flowshop scheduling problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. This class also serves as a base class for the Flowshop with Blocking and Flowshop with No Wait problem variants.

Definition at line 74 of file flowshopbasic.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 FlowshopBasic() [1/3]

```
FlowshopBasic::FlowshopBasic (
            const char * procTimeMatrixFile )
```

Constructs a new FlowshopBasic object.

Parameters

procTimeMatrixFile	File path to the file containing the job processing times matrix

Definition at line 235 of file flowshopbasic.cpp.

References procTimeMatrix, ptMatrixCols, and ptMatrixRows.

```
    : startTimeMatrix(nullptr), ptMatrixRows(0), ptMatrixCols(0), funcCallCounter(0)
{
    // Attempt to load job processing times from the given file
    procTimeMatrix = util::loadMatrixFromFile<int>(procTimeMatrixFile, ptMatrixRows, ptMatrixCols);
    if (procTimeMatrix == nullptr)
    {
        std::string msg = "Error when loading matrix file: ";
        msg += procTimeMatrixFile;
        throw std::runtime_error(msg);
    }
}
```

6.3.2.2 ∼FlowshopBasic()

```
FlowshopBasic::∼FlowshopBasic ( )  [virtual]
```

Destroys the FlowshopBasic object.

Definition at line 252 of file flowshopbasic.cpp.

References procTimeMatrix, and ptMatrixRows.

```
{
    util::releaseMatrix<int>(procTimeMatrix, ptMatrixRows);
}
```

6.3.2.3 FlowshopBasic() [2/3]

```
fshop::FlowshopBasic::FlowshopBasic (
             const FlowshopBasic & o )  [delete]
```

6.3.2.4 FlowshopBasic() [3/3]

```
fshop::FlowshopBasic::FlowshopBasic (
             const FlowshopBasic && o )  [delete]
```

6.3.3 Member Function Documentation

6.3.3.1 allocTimeMatrix()

```
int ** FlowshopBasic::allocTimeMatrix (
             size_t rows,
             size_t cols )  [protected], [virtual]
```

Allocates the start times and completion time matrices.

Parameters

rows	Number of rows (machines)
cols	Number of columns (jobs)

Returns

Returns a pointer to the newly created matrix

Definition at line 378 of file flowshopbasic.cpp.

Referenced by calcObjective().

```
{
    int** timeMatrix = util::allocMatrix<int>(rows, cols);
    if (timeMatrix == nullptr)
    {
        std::cerr << "Error allocating time matrix." << std::endl;
        throw std::bad_alloc();
    }

    util::initMatrix<int>(timeMatrix, rows, cols, 0);

    return timeMatrix;
}
```

6.3.3.2 calcObjective()

```
std::unique_ptr< FlowshopSolution > FlowshopBasic::calcObjective (
            int * seq,
            size_t seqSize )  [virtual]
```

Calculates the objective flowshop scheduling problem result using the given job sequence.

Parameters

seq	Pointer to an int array containing the job sequence permutation
seqSize	Size of the job sequence array

Returns

Returns a unique_ptr to a FlowshopSolution object that contains all solution results

Definition at line 318 of file flowshopbasic.cpp.

References allocTimeMatrix(), calcStartTimeCol(), calcTimeMatrix(), funcCallCounter, getCmax(), getTFT(), init←TimeMatrix(), ptMatrixRows, startTimeMatrix, and validateParams().

Referenced by fshop::NEH::run().

```
{
    // Validate input parameters
    validateParams(seq, seqSize);

    // Allocate completion (departure) time matrix and start time matrix
    auto compTimeMatrix = allocTimeMatrix(ptMatrixRows, seqSize);
    startTimeMatrix = allocTimeMatrix(ptMatrixRows, seqSize);

    // Initialize completion time matrix and start time matrix
    initTimeMatrix(compTimeMatrix, seq, ptMatrixRows, seqSize);
    calcStartTimeCol(startTimeMatrix, compTimeMatrix, seq, 0, ptMatrixRows, seqSize);

    // Calculate all completion and start times
    calcTimeMatrix(compTimeMatrix, seq, ptMatrixRows, seqSize);

    // Construct solution struct
    auto retVal = std::unique_ptr<FlowshopSolution>(new FlowshopSolution(startTimeMatrix, compTimeMatrix, ptMatrixRows, seq, seqSize,
          getCmax(compTimeMatrix, ptMatrixRows, seqSize), getTFT(compTimeMatrix, ptMatrixRows, seqSize)));

    // Increment obj func call counter and return result
    funcCallCounter += 1;
    return std::move(retVal);
}
```

6.3.3.3 calcStartTimeCol()

```
void FlowshopBasic::calcStartTimeCol (
            int ** startTimeMatrix,
            int ** departTimeMatrix,
            int * seq,
            size_t curCol,
            size_t rows,
            size_t cols )  [protected], [virtual]
```

Calculates the start times for a single column. Depends on values in completion time matrix.

Parameters

startTimeMatrix	Pointer to start times matrix
departTimeMatrix	Pointer to departure (completion) times matrix
seq	Pointer to job sequence
curCol	Index of the column to be calculated
rows	Number of rows (machines) in the completion time matrix
cols	Number of columns (jobs) in the completion time matrix

Definition at line 453 of file flowshopbasic.cpp.

References procTimeMatrix.

Referenced by calcObjective(), fshop::FlowshopNoWait::calcTimeMatrix(), fshop::FlowshopBlocking::calcTime↩ Matrix(), and calcTimeMatrix().

```
{
    for (size_t r = rows; r > 0; r--)
    {
        startTimeMatrix[r - 1][curCol] = departTimeMatrix[r - 1][curCol] -
    procTimeMatrix[r - 1][seq[curCol] - 1];
    }
}
```

6.3.3.4 calcTimeMatrix()

```
void FlowshopBasic::calcTimeMatrix (
            int ** compTimeMatrix,
            int * seq,
            size_t rows,
            size_t cols )  [protected], [virtual]
```

Calculates all remaining start and completion times for the current flowshop problem.

Parameters

compTimeMatrix	Pointer to completion time matrix
seq	Pointer to job sequence
rows	Number of rows (machines) in the completion time matrix
cols	Number of columns (jobs) in the completion time matrix

Reimplemented in fshop::FlowshopBlocking, and fshop::FlowshopNoWait.

Definition at line 427 of file flowshopbasic.cpp.

References calcStartTimeCol(), max(), procTimeMatrix, and startTimeMatrix.

Referenced by calcObjective().

```
{
    for (size_t c = 1; c < cols; c++)
    {
        for (size_t r = 1; r < rows; r++)
        {
            int c1 = compTimeMatrix[r - 1][c];
            int c2 = compTimeMatrix[r][c - 1];

            compTimeMatrix[r][c] = max(c1, c2) + procTimeMatrix[r][seq[c] - 1];
        }

        FlowshopBasic::calcStartTimeCol(
   startTimeMatrix, compTimeMatrix, seq, c, rows, cols);
    }
}
```

6.3.3.5 getCmax()

```
int FlowshopBasic::getCmax (
            int ** compTimeMatrix,
            size_t rows,
            size_t cols )  [protected], [virtual]
```

Returns the cmax value for a given completion time matrix.

Parameters

compTimeMatrix	Pointer to the completion time matrix
rows	Number of rows (machines) in the completion time matrix
cols	Number of columns (jobs) in the completion time matrix

Returns

Returns the cmax value (last row, last column) in the completion time matrix

Definition at line 469 of file flowshopbasic.cpp.

Referenced by calcObjective().

```
{
    return compTimeMatrix[rows - 1][cols - 1];
}
```

6.3.3.6 getFuncCallCounts()

```
size_t FlowshopBasic::getFuncCallCounts ( )  [virtual]
```

Returns the number of times the current flowshop objective function has been executed.

Returns

Returns the number of times the current flowshop objective function has been executed

Definition at line 305 of file flowshopbasic.cpp.

References funcCallCounter.

```
{
    return funcCallCounter;
}
```

6.3.3.7 getProcessingTime()

```
int FlowshopBasic::getProcessingTime (
            size_t machine,
            size_t job )  [virtual]
```

Returns the processing time for the given job on the given machine.

Parameters

machine	Number of machine [1-n]
job	Number of job [1-n]

Returns

Returns the processing time

Definition at line 264 of file flowshopbasic.cpp.

References procTimeMatrix, ptMatrixCols, and ptMatrixRows.

Referenced by fshop::NEH::run().

```
{
    if (machine == 0 || job == 0)
    {
        std::string msg = "Error: Machine or job number cannot be zero";
        throw std::out_of_range(msg);
    }
    else if (machine > ptMatrixRows || job > ptMatrixCols)
    {
        std::string msg = "Error: Machine or job number out of range";
        throw std::out_of_range(msg);
    }

    return procTimeMatrix[machine - 1][job - 1];
}
```

6.3.3.8 getTFT()

```
int FlowshopBasic::getTFT (
            int ** compTimeMatrix,
            size_t rows,
            size_t cols )  [protected], [virtual]
```

Returns the total flow time value for a given completion time matrix.

Parameters

compTimeMatrix	Pointer to the completion time matrix
rows	Number of rows (machines) in the completion time matrix
cols	Number of columns (jobs) in the completion time matrix

Returns

Returns the TFT value (sum of last row) in the completion time matrix

Definition at line 482 of file flowshopbasic.cpp.

Referenced by calcObjective().

```
{
    int sum = 0;

    for (size_t c = 0; c < cols; c++)
    {
        sum += compTimeMatrix[rows - 1][c];
    }

    return sum;
}
```

6.3.3.9 getTotalJobs()

```
size_t FlowshopBasic::getTotalJobs ( )  [virtual]
```

Returns the total number of jobs in the jobs processing time matrix.

Returns

Returns the total number of jobs

Definition at line 285 of file flowshopbasic.cpp.

References ptMatrixCols.

Referenced by fshop::NEH::run().

```
{
    return ptMatrixCols;
}
```

6.3.3.10 getTotalMachines()

```
size_t FlowshopBasic::getTotalMachines ( ) [virtual]
```

Returns the total number of machines in the jobs processing time matrix.

Returns

Returns the total number of machines

Definition at line 295 of file flowshopbasic.cpp.

References ptMatrixRows.

Referenced by fshop::NEH::run().

```
{
    return ptMatrixRows;
}
```

6.3.3.11 initTimeMatrix()

```
void FlowshopBasic::initTimeMatrix (
             int ** compTimeMatrix,
             int * seq,
             size_t rows,
             size_t cols ) [protected], [virtual]
```

Initializes the completion time matrix (first row and first column) so that it is ready to be completed with the main algorithm.

Parameters

compTimeMatrix	Pointer to completion time matrix
seq	Pointer to job sequence
rows	Number of rows (machines) in the completion time matrix
cols	Number of columns (jobs) in the completion time matrix

Reimplemented in fshop::FlowshopBlocking, and fshop::FlowshopNoWait.

Definition at line 401 of file flowshopbasic.cpp.

References procTimeMatrix.

Referenced by calcObjective().

```
{
    // Set first job, first machine
    compTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];

    // Set first job for all machines
    for (size_t r = 1; r < rows; r++)
    {
        compTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + compTimeMatrix[r - 1][0];
    }

    // Set first machine for all jobs
    for (size_t c = 1; c < cols; c++)
    {
        compTimeMatrix[0][c] = compTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
    }
}
```

6.3.3.12 operator=() [1/2]

```
FlowshopBasic& fshop::FlowshopBasic::operator= (
            const FlowshopBasic & o )  [delete]
```

6.3.3.13 operator=() [2/2]

```
FlowshopBasic& fshop::FlowshopBasic::operator= (
            const FlowshopBasic && o )  [delete]
```

6.3.3.14 validateParams()

```
void FlowshopBasic::validateParams (
            int * seq,
            size_t seqSize )  [protected], [virtual]
```

Validates the flowshop input parameters, and throws an exception on error.

Keeps track of the number of times run() is called

Parameters

seq	Job permutation sequence array
seqSize	Size of job sequence array

Definition at line 349 of file flowshopbasic.cpp.

References ptMatrixCols.

Referenced by calcObjective().

```
{
    // Make sure job sequence is not empty, or too large
    if (seqSize == 0 || seqSize > ptMatrixCols)
    {
        std::string msg = "Error: seqSize cannot be larger than ptMatrixCols";
        throw std::out_of_range(msg);
    }

    // Make sure all jobs in job sequence are within bounds of processing time matrix
    for (size_t i = 0; i < seqSize; i++)
    {
        if (seq[i] <= 0 || seq[i] > ptMatrixCols)
        {
            std::string msg = "Error: seq contains a job number out of range [1, ";
            msg += std::to_string(ptMatrixCols);
            msg += "]";
            throw std::out_of_range(msg);
        }
    }
}
```

6.3.4 Member Data Documentation

6.3.4.1 funcCallCounter

`size_t fshop::FlowshopBasic::funcCallCounter [protected]`

The number of columns (jobs) in the processing time matrix

Definition at line 96 of file flowshopbasic.h.

Referenced by calcObjective(), and getFuncCallCounts().

6.3.4.2 procTimeMatrix

`int** fshop::FlowshopBasic::procTimeMatrix [protected]`

Definition at line 92 of file flowshopbasic.h.

Referenced by calcStartTimeCol(), fshop::FlowshopNoWait::calcTimeMatrix(), fshop::FlowshopBlocking::calc←TimeMatrix(), calcTimeMatrix(), FlowshopBasic(), getProcessingTime(), fshop::FlowshopNoWait::initTimeMatrix(), fshop::FlowshopBlocking::initTimeMatrix(), initTimeMatrix(), and ∼FlowshopBasic().

6.3.4.3 ptMatrixCols

`size_t fshop::FlowshopBasic::ptMatrixCols [protected]`

The number of rows (machines) in the processing time matrix

Definition at line 95 of file flowshopbasic.h.

Referenced by FlowshopBasic(), getProcessingTime(), getTotalJobs(), and validateParams().

6.3.4.4 ptMatrixRows

```
size_t fshop::FlowshopBasic::ptMatrixRows  [protected]
```

The job start times matrix

Definition at line 94 of file flowshopbasic.h.

Referenced by calcObjective(), FlowshopBasic(), getProcessingTime(), getTotalMachines(), and ∼Flowshop↩
Basic().

6.3.4.5 startTimeMatrix

```
int** fshop::FlowshopBasic::startTimeMatrix  [protected]
```

The job processing time matrix, which is read from a file

Definition at line 93 of file flowshopbasic.h.

Referenced by calcObjective(), fshop::FlowshopNoWait::calcTimeMatrix(), fshop::FlowshopBlocking::calcTime↩
Matrix(), and calcTimeMatrix().

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

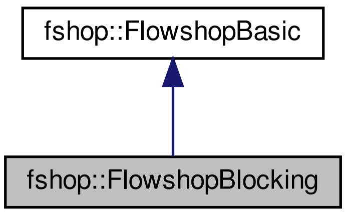
- include/flowshopbasic.h
- src/flowshopbasic.cpp

6.4 fshop::FlowshopBlocking Class Reference

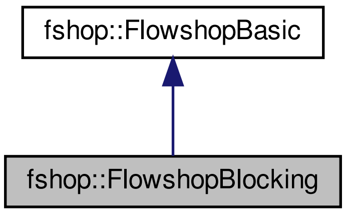
The FlowshopBlocking class runs the flowshop with blocking problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. Inherits from FlowshopBasic.

```
#include <flowshopblocking.h>
```

Inheritance diagram for fshop::FlowshopBlocking:



Collaboration diagram for fshop::FlowshopBlocking:



Public Member Functions

- FlowshopBlocking (const char ∗procTimeMatrixFile)

  Construct a new FlowshopBlocking object.
- virtual ∼FlowshopBlocking ()=default

Protected Member Functions

- virtual void initTimeMatrix (int ∗∗compTimeMatrix, int ∗seq, size_t rows, size_t cols) override

  Initializes the completion time matrix (first column) so that it is ready to be completed with the main algorithm. Overrides method in base class.
- virtual void calcTimeMatrix (int ∗∗compTimeMatrix, int ∗seq, size_t rows, size_t cols) override

  Calculates all remaining start and completion times for the current flowshop problem. Overrides method in base class.

Additional Inherited Members

6.4.1 Detailed Description

The FlowshopBlocking class runs the flowshop with blocking problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. Inherits from FlowshopBasic.

Definition at line 26 of file flowshopblocking.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 FlowshopBlocking()

```
FlowshopBlocking::FlowshopBlocking (
              const char ∗ procTimeMatrixFile )
```

Construct a new FlowshopBlocking object.

Parameters

procTimeMatrixFile	File path to the file containing the job processing times matrix

Definition at line 34 of file flowshopblocking.cpp.

```
    : FlowshopBasic(procTimeMatrixFile)
{
}
```

6.4.2.2 ∼FlowshopBlocking()

```
virtual fshop::FlowshopBlocking::∼FlowshopBlocking ( ) [virtual], [default]
```

6.4.3 Member Function Documentation

6.4.3.1 calcTimeMatrix()

```
void FlowshopBlocking::calcTimeMatrix (
            int ** departTimeMatrix,
            int * seq,
            size_t rows,
            size_t cols ) [override], [protected], [virtual]
```

Calculates all remaining start and completion times for the current flowshop problem. Overrides method in base class.

Parameters

compTimeMatrix	Pointer to completion time matrix
seq	Pointer to job sequence
rows	Number of rows (machines) in the completion time matrix
cols	Number of columns (jobs) in the completion time matrix

Reimplemented from fshop::FlowshopBasic.

Definition at line 68 of file flowshopblocking.cpp.

References fshop::FlowshopBasic::calcStartTimeCol(), max(), fshop::FlowshopBasic::procTimeMatrix, and fshop::FlowshopBasic::startTimeMatrix.

```
{
    for (size_t c = 1; c < cols; c++)
    {
        int d1 = departTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
        int d2 = departTimeMatrix[1][c - 1];

        departTimeMatrix[0][c] = max(d1, d2);

        for (size_t r = 1; r < rows - 1; r++)
        {
            int d1 = departTimeMatrix[r - 1][c] + procTimeMatrix[r][seq[c] - 1];
            int d2 = departTimeMatrix[r + 1][c - 1];

            departTimeMatrix[r][c] = max(d1, d2);
        }

        departTimeMatrix[rows - 1][c] = departTimeMatrix[rows - 2][c] + procTimeMatrix[rows - 1][seq[c] - 1];

        FlowshopBasic::calcStartTimeCol(startTimeMatrix, departTimeMatrix, seq, c, rows, cols);
    }
}
```

6.4.3.2 initTimeMatrix()

```
void FlowshopBlocking::initTimeMatrix (
            int ** departTimeMatrix,
            int * seq,
            size_t rows,
            size_t cols ) [override], [protected], [virtual]
```

Initializes the completion time matrix (first column) so that it is ready to be completed with the main algorithm. Overrides method in base class.

Parameters

compTimeMatrix	Pointer to completion time matrix
seq	Pointer to job sequence
rows	Number of rows (machines) in the completion time matrix
cols	Number of columns (jobs) in the completion time matrix

Reimplemented from fshop::FlowshopBasic.

Definition at line 49 of file flowshopblocking.cpp.

References fshop::FlowshopBasic::procTimeMatrix.

```
{
    departTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];

    for (size_t r = 1; r < rows; r++)
    {
        departTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + departTimeMatrix[r - 1][0];
    }
}
```

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

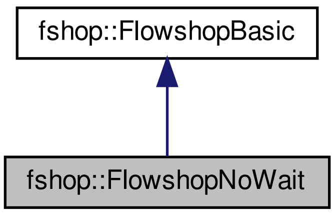
- include/flowshopblocking.h
- src/flowshopblocking.cpp

6.5 fshop::FlowshopNoWait Class Reference

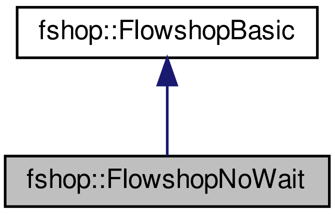
The FlowshopNoWait class runs the flowshop with no wait problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. Inherits from Flowshop←Basic.

```
#include <flowshopnowait.h>
```

Inheritance diagram for fshop::FlowshopNoWait:



Collaboration diagram for fshop::FlowshopNoWait:



Public Member Functions

- FlowshopNoWait (const char ∗procTimeMatrixFile)

  Construct a new FlowshopNoWait object.
- virtual ∼FlowshopNoWait ()=default

Protected Member Functions

- virtual void initTimeMatrix (int ∗∗departTimeMatrix, int ∗seq, size_t rows, size_t cols) override

  Initializes the completion time matrix (first column) so that it is ready to be completed with the main algorithm. Overrides method in base class.
- virtual void calcTimeMatrix (int ∗∗departTimeMatrix, int ∗seq, size_t rows, size_t cols) override

  Calculates all remaining start and completion times for the current flowshop problem. Overrides method in base class.

Additional Inherited Members

6.5.1 Detailed Description

The FlowshopNoWait class runs the flowshop with no wait problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. Inherits from Flowshop↩ Basic.

Definition at line 25 of file flowshopnowait.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 FlowshopNoWait()

```
FlowshopNoWait::FlowshopNoWait (
              const char * procTimeMatrixFile )
```

Construct a new FlowshopNoWait object.

Parameters

procTimeMatrixFile	File path to the file containing the job processing times matrix

Definition at line 37 of file flowshopnowait.cpp.

```
00038     : FlowshopBasic(procTimeMatrixFile)
00039 {
00040 }
```

6.5.2.2 ∼FlowshopNoWait()

```
virtual fshop::FlowshopNoWait::~FlowshopNoWait ( )  [virtual], [default]
```

6.5.3 Member Function Documentation

6.5.3.1 calcTimeMatrix()

```
void FlowshopNoWait::calcTimeMatrix (
              int ** departTimeMatrix,
              int * seq,
              size_t rows,
              size_t cols )  [override], [protected], [virtual]
```

Calculates all remaining start and completion times for the current flowshop problem. Overrides method in base class.

Parameters

compTimeMatrix	Pointer to completion time matrix
seq	Pointer to job sequence
rows	Number of rows (machines) in the completion time matrix
cols	Number of columns (jobs) in the completion time matrix

Reimplemented from fshop::FlowshopBasic.

Definition at line 71 of file flowshopnowait.cpp.

References fshop::FlowshopBasic::calcStartTimeCol(), fshop::FlowshopBasic::procTimeMatrix, and fshop::↩ FlowshopBasic::startTimeMatrix.

```
{
    for (size_t c = 1; c < cols; c++)
    {
        departTimeMatrix[0][c] = departTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];

        for (size_t r = 1; r < rows; r++)
        {
            int d1 = departTimeMatrix[r - 1][c];
            int d2 = departTimeMatrix[r][c - 1];

            if (d1 < d2)
            {
                const int diff = d2 - d1;
                for (size_t r2 = r + 1; r2 > 0; r2--)
                    departTimeMatrix[r2 - 1][c] += diff;

                d1 = departTimeMatrix[r - 1][c];
            }

            departTimeMatrix[r][c] = d1 + procTimeMatrix[r][seq[c] - 1];
        }

        FlowshopBasic::calcStartTimeCol(
startTimeMatrix, departTimeMatrix, seq, c, rows, cols);
    }
}
```

6.5.3.2 initTimeMatrix()

```
void FlowshopNoWait::initTimeMatrix (
              int ** departTimeMatrix,
              int * seq,
              size_t rows,
              size_t cols ) [override], [protected], [virtual]
```

Initializes the completion time matrix (first column) so that it is ready to be completed with the main algorithm. Overrides method in base class.

Parameters

compTimeMatrix	Pointer to completion time matrix
seq	Pointer to job sequence
rows	Number of rows (machines) in the completion time matrix
cols	Number of columns (jobs) in the completion time matrix

Reimplemented from fshop::FlowshopBasic.

Definition at line 52 of file flowshopnowait.cpp.

References fshop::FlowshopBasic::procTimeMatrix.

```
{
    departTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];

    for (size_t r = 1; r < rows; r++)
    {
        departTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + departTimeMatrix[r - 1][0];
    }
}
```

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

- include/flowshopnowait.h
- src/flowshopnowait.cpp

6.6 fshop::FlowshopSolution Struct Reference

The FlowshopSolution struct houses all solution data returned from calculating the objective flowshop function. This includes the cmax value, total flow time, job sequence used, start time matrix, and departure time matrix.

`#include <flowshopbasic.h>`

Public Member Functions

- FlowshopSolution (int ∗∗_startimeMatrix, int ∗∗_departTimeMatrix, size_t _tMatrixRows, int ∗_jobSeq, size←
  _t _seqSize, int _cmax, int _totalFlowTime)

    Constructs a new SlowshopSolution object.
- ∼FlowshopSolution ()

    Destroys the FlowshopSolution object.
- const int ∗const getJobSeq ()

    Returns a const pointer to the job sequence array.
- std::string getJobSeqAsString ()

    Returns the job sequence array as a string.
- const int ∗∗const getStartTimeMatrix ()

    Returns a const pointer to the start times matrix.
- const int ∗∗const getDepartTimeMatrix ()

    Returns a const pointer to the departure times matrix.
- bool outputTimesCsv (const std::string &fileNamePrefix)

    Dumps the start times and departure times matrices to a csv file that starts with the given fileNamePrefix.
- void outputAll (std::ostream &os)

    Outputs all results data to the given stream in a human readable format.
- FlowshopSolution (const FlowshopSolution &obj)

    Copy constructor for the FlowshopSolution class.
- FlowshopSolution (FlowshopSolution &&obj)

    Move constructor for the FlowshopSolution class.
- FlowshopSolution & operator= (const FlowshopSolution &obj)=delete
- FlowshopSolution & operator= (FlowshopSolution &&obj)=delete

Public Attributes

- const size_t seqSize
- const size_t numMachines
- const int cmax
- const int totalFlowTime

6.6.1 Detailed Description

The FlowshopSolution struct houses all solution data returned from calculating the objective flowshop function. This includes the cmax value, total flow time, job sequence used, start time matrix, and departure time matrix.

Definition at line 32 of file flowshopbasic.h.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 FlowshopSolution() [1/3]

```
FlowshopSolution::FlowshopSolution (
              int ** _startimeMatrix,
              int ** _departTimeMatrix,
              size_t _tMatrixRows,
              int * _jobSeq,
              size_t _seqSize,
              int _cmax,
              int _totalFlowTime )
```

Constructs a new SlowshopSolution object.

Parameters

_startimeMatrix	Pointer to the start times matrix. This class takes ownership of the pointer and will destroy it.
_departTimeMatrix	Pointer to the departure times matrix. This class takes ownership of the pointer and will destroy it.
_tMatrixRows	Number of rows (machines) in the start and departure time matrices
_jobSeq	Pointer to the job sequence array. This class takes ownership of the pointer and will destroy it.
_seqSize	Size of the job sequence array
_cmax	Cmax value of the flowshop result
_totalFlowTime	Total flow time of the flowshop result

Definition at line 45 of file flowshopbasic.cpp.

References seqSize.

```
00046     : startTimeMatrix(_startimeMatrix), departTimeMatrix(_departTimeMatrix),
      numMachines(_tMatrixRows), seqSize(_seqSize), cmax(_cmax),
      totalFlowTime(_totalFlowTime)
{
    if (_jobSeq == nullptr)
        throw std::invalid_argument("Error: _jobSeq cannot be nullptr");
    else if (_startimeMatrix == nullptr)
        throw std::invalid_argument("Error: _startimeMatrix cannot be nullptr");
    else if (_departTimeMatrix == nullptr)
        throw std::invalid_argument("Error: _departTimeMatrix cannot be nullptr");
    else if (seqSize == 0)
        throw std::invalid_argument("Error: _seqSize cannot be zero");

    jobSequence = util::allocArray<int>(seqSize);
    for (size_t i = 0; i < seqSize; i++)
        jobSequence[i] = _jobSeq[i];
}
```

6.6.2.2 ∼FlowshopSolution()

```
FlowshopSolution::∼FlowshopSolution ( )
```

Destroys the FlowshopSolution object.

Definition at line 65 of file flowshopbasic.cpp.

References numMachines.

```
{
    util::releaseArray<int>(jobSequence);
    util::releaseMatrix<int>(startTimeMatrix, numMachines);
    util::releaseMatrix<int>(departTimeMatrix, numMachines);
}
```

6.6.2.3 FlowshopSolution() [2/3]

```
FlowshopSolution::FlowshopSolution (
              const FlowshopSolution & obj )
```

Copy constructor for the FlowshopSolution class.

Definition at line 201 of file flowshopbasic.cpp.

References seqSize.

```
    : startTimeMatrix(obj.startTimeMatrix), departTimeMatrix(obj.departTimeMatrix),
      numMachines(obj.numMachines), seqSize(obj.seqSize),
      cmax(obj.cmax), totalFlowTime(obj.totalFlowTime)
{
    if (obj.jobSequence == nullptr)
        throw std::invalid_argument("Error: jobSequence cannot be nullptr");
    else if (seqSize == 0)
        throw std::invalid_argument("Error: seqSize cannot be zero");

    jobSequence = util::allocArray<int>(seqSize);
    for (size_t i = 0; i < seqSize; i++)
        jobSequence[i] = obj.jobSequence[i];
}
```

6.6.2.4 FlowshopSolution() [3/3]

```
FlowshopSolution::FlowshopSolution (
              FlowshopSolution && obj )
```

Move constructor for the FlowshopSolution class.

Definition at line 217 of file flowshopbasic.cpp.

```
00218     : numMachines(obj.numMachines), seqSize(obj.
      seqSize), cmax(obj.cmax), totalFlowTime(obj.
      totalFlowTime)
00219 {
00220     jobSequence = obj.jobSequence;
00221     startTimeMatrix = obj.startTimeMatrix;
00222     departTimeMatrix = obj.departTimeMatrix;
00223     obj.jobSequence = nullptr;
00224     obj.startTimeMatrix = nullptr;
00225     obj.departTimeMatrix = nullptr;
00226 }
```

6.6.3 Member Function Documentation

6.6.3.1 getDepartTimeMatrix()

```
const int **const FlowshopSolution::getDepartTimeMatrix ( )
```

Returns a const pointer to the departure times matrix.

Returns

Returns a const pointer to the departure times matrix.

Definition at line 116 of file flowshopbasic.cpp.

```
00117 {
00118     return const_cast<const int** const>(departTimeMatrix);
00119 }
```

6.6.3.2 getJobSeq()

```
const int *const FlowshopSolution::getJobSeq ( )
```

Returns a const pointer to the job sequence array.

Flowshop total flow time value, which is the sum of all departure times on the last machine

Returns

Returns a const pointer to the job sequence array

Definition at line 77 of file flowshopbasic.cpp.

```
00078 {
00079     return const_cast<const int* const>(jobSequence);
00080 }
```

6.6.3.3 getJobSeqAsString()

```
std::string FlowshopSolution::getJobSeqAsString ( )
```

Returns the job sequence array as a string.

Returns

Returns the job sequence array as a string

Definition at line 87 of file flowshopbasic.cpp.

References seqSize.

```
{
    std::string retStr = "[";

    for (size_t i = 0; i < seqSize; i++)
    {
        retStr += std::to_string(jobSequence[i]);
        if (i < seqSize - 1) retStr += "-";
    }

    retStr += "]";
    return retStr;
}
```

6.6.3.4 getStartTimeMatrix()

```
const int **const FlowshopSolution::getStartTimeMatrix ( )
```

Returns a const pointer to the start times matrix.

Returns

Returns a const pointer to the start times matrix.

Definition at line 106 of file flowshopbasic.cpp.

```
{
    return const_cast<const int** const>(startTimeMatrix);
}
```

6.6.3.5 operator=() [1/2]

```
FlowshopSolution& fshop::FlowshopSolution::operator= (
              const FlowshopSolution & obj )   [delete]
```

6.6.3.6 operator=() [2/2]

```
FlowshopSolution& fshop::FlowshopSolution::operator= (
              FlowshopSolution && obj )   [delete]
```

6.6.3.7 outputAll()

```
void FlowshopSolution::outputAll (
              std::ostream & os )
```

Outputs all results data to the given stream in a human readable format.

Parameters

os	The output stream to write to

Definition at line 173 of file flowshopbasic.cpp.

References cmax, numMachines, util::outputMatrix(), seqSize, and totalFlowTime.

```
{
    std::cout << "Input seq: ";

    for (size_t i = 0; i < seqSize; i++)
    {
        std::cout << jobSequence[i];
        if (i < seqSize - 1)
            std::cout << ", ";
    }

    std::cout << std::endl;

    std::cout << "Cmax: " << cmax << std::endl;
    std::cout << "TFT: " << totalFlowTime << std::endl << std::endl;

    std::cout << "Starting times matrix:" << std::endl;
    util::outputMatrix(std::cout, startTimeMatrix, numMachines, seqSize, 4);
    std::cout << std::endl;

    std::cout << "Departure times matrix:" << std::endl;
    util::outputMatrix(std::cout, departTimeMatrix,
 numMachines, seqSize, 4);
    std::cout << std::endl;
}
```

6.6.3.8 outputTimesCsv()

```
bool FlowshopSolution::outputTimesCsv (
            const std::string & fileNamePrefix )
```

Dumps the start times and departure times matrices to a csv file that starts with the given fileNamePrefix.

Parameters

fileNamePrefix	Start of the path/file which will contain the data

Returns

Returns true on success. Otherwise false.

Definition at line 128 of file flowshopbasic.cpp.

References numMachines, and seqSize.

```
{
    using namespace std;

    // Create file name strings
    string startTimesFile = fileNamePrefix + "starttimes.csv";
    string departTimesFile = fileNamePrefix + "departtimes.csv";

    // Open files
    ofstream startOs = ofstream(startTimesFile, ios::trunc | ios::out);
    if (!startOs.good()) return false;

    ofstream departOs = ofstream(departTimesFile, ios::trunc | ios::out);
    if (!departOs.good()) return false;

    // Output start times and departure times data to the files
    for (size_t m = 0; m < numMachines; m++)
    {
        for (size_t j = 0; j < seqSize; j++)
        {
            startOs << startTimeMatrix[m][j];
            departOs << departTimeMatrix[m][j];

            if (j < seqSize - 1)
            {
                startOs << ",";
                departOs << ",";
            }
        }

        startOs << endl;
        departOs << endl;
    }

    // Close file handles and return
    startOs.close();
    departOs.close();
    return true;
}
```

6.6.4 Member Data Documentation

6.6.4.1 cmax

`const int fshop::FlowshopSolution::cmax`

Number of machines executing jobs

Definition at line 39 of file flowshopbasic.h.

Referenced by outputAll().

6.6.4.2 numMachines

`const size_t fshop::FlowshopSolution::numMachines`

Number of jobs in job sequence

Definition at line 38 of file flowshopbasic.h.

Referenced by outputAll(), outputTimesCsv(), and ∼FlowshopSolution().

6.6.4.3 seqSize

```
const size_t fshop::FlowshopSolution::seqSize
```

Definition at line 37 of file flowshopbasic.h.

Referenced by FlowshopSolution(), getJobSeqAsString(), outputAll(), and outputTimesCsv().

6.6.4.4 totalFlowTime

```
const int fshop::FlowshopSolution::totalFlowTime
```

Flowshop cmax value, which is the departure time of the last job

Definition at line 40 of file flowshopbasic.h.

Referenced by outputAll().

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

- include/flowshopbasic.h
- src/flowshopbasic.cpp

6.7 util::IniReader Class Reference

The IniReader class is a simple ∗.ini file reader and parser.

```
#include <inireader.h>
```

Public Member Functions

- IniReader ()

    Construct a new IniReader object.
- ∼IniReader ()

    Destroys the IniReader object.
- bool openFile (std::string filePath)

    Opens the given ini file and parses all sections/entries. The all file data is stored in memory and the file is closed.
- bool sectionExists (std::string section)

    Returns true if the given section exists in the current ini file.
- bool entryExists (std::string section, std::string entry)

    Returns true if the given section and entry key exists in the current ini file.
- std::string getEntry (std::string section, std::string entry, std::string defVal="")

    Returns the value for the entry that has the given entry key within the given section.
- template<class T >
  T getEntryAs (std::string section, std::string entry, T defVal={})

6.7.1 Detailed Description

The IniReader class is a simple ∗.ini file reader and parser.

– Initialize an IniReader object:

IniReader ini;

Open and parse an ∗.ini file:

ini.openFile("my_ini_file.ini");

Note that the file is immediately closed after parsing, and the file data is retained in memory.

Retrieve an entry from the ini file:

std::string value = ini.getEntry("My Section", "entryKey");

Definition at line 46 of file inireader.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 IniReader()

```
IniReader::IniReader ( )
```

Construct a new IniReader object.

Definition at line 21 of file inireader.cpp.

```
                       : file(""), iniMap()
{
}
```

6.7.2.2 ∼IniReader()

```
IniReader::~IniReader ( )
```

Destroys the IniReader object.

Definition at line 28 of file inireader.cpp.

```
{
    iniMap.clear();
}
```

6.7.3 Member Function Documentation

6.7.3.1 entryExists()

```
bool IniReader::entryExists (
            std::string section,
            std::string entry )
```

Returns true if the given section and entry key exists in the current ini file.

Parameters

section	std::string containing the section name
entry	std::string containing the entry key name

Returns

Returns true if the section and entry key exist in the ini file, otherwise false.

Definition at line 67 of file inireader.cpp.

Referenced by getEntry().

```
{
    auto it = iniMap.find(section);
    if (it == iniMap.end()) return false;

    return it->second.find(entry) != it->second.end();
}
```

6.7.3.2 getEntry()

```
std::string IniReader::getEntry (
             std::string section,
             std::string entry,
             std::string defVal = "" )
```

Returns the value for the entry that has the given entry key within the given section.

Parameters

section	std::string containing the section name
entry	std::string containing the entry key name

Returns

The value of the entry with the given entry key and section. Returns an empty string if the entry does not exist.

Definition at line 84 of file inireader.cpp.

References entryExists().

Referenced by getEntryAs().

```
{
    if (!entryExists(section, entry)) return defVal;

    return iniMap[section][entry];
}
```

6.7.3.3 getEntryAs()

```
template<class T >
T util::IniReader::getEntryAs (
            std::string section,
            std::string entry,
            T defVal = {} )  [inline]
```

Definition at line 57 of file inireader.h.

References getEntry().

```
                                                                    {})
        {
            std::stringstream ss(getEntry(section, entry, std::to_string(defVal)));
            T retVal;
            ss >> retVal;
            return retVal;
        }
```

6.7.3.4 openFile()

```
bool IniReader::openFile (
            std::string filePath )
```

Opens the given ini file and parses all sections/entries. The all file data is stored in memory and the file is closed.

Parameters

filePath	Path to the ini file you wish to open

Returns

Returns true if the file was succesfully opened and parsed. Otherwise false.

Definition at line 40 of file inireader.cpp.

```
{
    file = filePath;
    if (!parseFile())
        return false;

    return true;
}
```

6.7.3.5 sectionExists()

```
bool IniReader::sectionExists (
            std::string section )
```

Returns true if the given section exists in the current ini file.

Parameters

section	std::string containing the section name

Returns

Returns true if the section exists in the ini file, otherwise false.

Definition at line 55 of file inireader.cpp.

```
{
    return iniMap.find(section) != iniMap.end();
}
```

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

- include/inireader.h
- src/inireader.cpp

6.8 fshop::JobTimePair Struct Reference

Simple struct that pairs a job with it's total processing time. Used for sorting purposes.

```
#include <neh.h>
```

Public Member Functions

- JobTimePair (int _job, int _time)

Public Attributes

- const int job
- const int time

6.8.1 Detailed Description

Simple struct that pairs a job with it's total processing time. Used for sorting purposes.

Definition at line 29 of file neh.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 JobTimePair()

```
fshop::JobTimePair::JobTimePair (
            int _job,
            int _time )  [inline]
```

Definition at line 34 of file neh.h.

```
00035            : job(_job), time(_time)
00036        {
00037        }
```

6.8.3 Member Data Documentation

6.8.3.1 job

```
const int fshop::JobTimePair::job
```

Definition at line 31 of file neh.h.

6.8.3.2 time

```
const int fshop::JobTimePair::time
```

Definition at line 32 of file neh.h.

Referenced by fshop::NEH::run().

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

- include/neh.h

6.9 fshop::NEH Class Reference

The NEH class runs the NEH algorithm on the given flowshop objective function and attempts to optimize the job sequence that produces the smallest cmax value.

```
#include <neh.h>
```

Public Member Functions

- NEH ()
  Construct a new NEH object.
- fsSol run (FlowshopBasic ∗const objectiveFs)
  Runs the NEH algorithm on the given flowshop objective function.

6.9.1 Detailed Description

The NEH class runs the NEH algorithm on the given flowshop objective function and attempts to optimize the job sequence that produces the smallest cmax value.

Definition at line 46 of file neh.h.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 NEH()

```
fshop::NEH::NEH ( )
```

Construct a new NEH object.

Definition at line 22 of file neh.cpp.

```
00023     : rd(), randEngine(rd()), randChance(0, 1)
00024 { }
```

6.9.3 Member Function Documentation

6.9.3.1 run()

```
fsSol fshop::NEH::run (
            FlowshopBasic *const objectiveFs )
```

Runs the NEH algorithm on the given flowshop objective function.

Parameters

objectiveFs	Pointer to the flowshop objective function being optimized

Returns

Returns a unique_ptr to a FlowshopSolution object that contains the best solution found.

Definition at line 32 of file neh.cpp.

References fshop::FlowshopBasic::calcObjective(), fshop::FlowshopBasic::getProcessingTime(), fshop::↩
FlowshopBasic::getTotalJobs(), fshop::FlowshopBasic::getTotalMachines(), and fshop::JobTimePair::time.

Referenced by cs471::Experiment::runNEH().

```
{
    jtList availJobsList;
    makeInitialAvailJobList(objectiveFs, availJobsList);

    auto firstJob = availJobsList.front();
    availJobsList.pop_front();

    fsSol bestSol = nullptr;
    jList* curJobSeq = new jList();
    jList* nextJobSeq = new jList();
    curJobSeq->push_back(firstJob.job);

    while (availJobsList.size() > 0)
    {
        auto nextJob = availJobsList.front();
        availJobsList.pop_front();

        bestSol.reset();
        bestSol = bestPermutation(objectiveFs, *curJobSeq, nextJob.job, *nextJobSeq);

        auto tmp = curJobSeq;
        curJobSeq = nextJobSeq;
        nextJobSeq = tmp;
    }

    delete curJobSeq;
    delete nextJobSeq;

    return std::move(bestSol);
}
```

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

- include/neh.h
- src/neh.cpp

6.10 cs471::TestParams Struct Reference

Simple data structure that stores the test parameters for the experiment.

```
#include <experiment.h>
```

Public Attributes

- int minTestFile
- int maxTestFile
- int numThreads
- int algorithm
- std::string inputFilesDir
- std::string resultsFile
- std::string timesFile

6.10.1 Detailed Description

Simple data structure that stores the test parameters for the experiment.

Definition at line 26 of file experiment.h.

6.10.2 Member Data Documentation

6.10.2.1 algorithm

```
int cs471::TestParams::algorithm
```

Definition at line 31 of file experiment.h.

Referenced by cs471::Experiment::runDebugSeq(), and cs471::Experiment::runNEH().

6.10.2.2 inputFilesDir

```
std::string cs471::TestParams::inputFilesDir
```

Definition at line 32 of file experiment.h.

Referenced by cs471::Experiment::runDebugSeq(), and cs471::Experiment::runNEH().

6.10.2.3 maxTestFile

```
int cs471::TestParams::maxTestFile
```

Definition at line 29 of file experiment.h.

Referenced by cs471::Experiment::runDebugSeq(), and cs471::Experiment::runNEH().

6.10.2.4 minTestFile

```
int cs471::TestParams::minTestFile
```

Definition at line 28 of file experiment.h.

Referenced by cs471::Experiment::runDebugSeq(), and cs471::Experiment::runNEH().

6.10.2.5 numThreads

```
int cs471::TestParams::numThreads
```

Definition at line 30 of file experiment.h.

Referenced by cs471::Experiment::runNEH().

6.10.2.6 resultsFile

```
std::string cs471::TestParams::resultsFile
```

Definition at line 33 of file experiment.h.

Referenced by cs471::Experiment::runNEH().

6.10.2.7 timesFile

```
std::string cs471::TestParams::timesFile
```

Definition at line 34 of file experiment.h.

Referenced by cs471::Experiment::runNEH().

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

- include/experiment.h

6.11 ThreadPool Class Reference

```
#include <threadpool.h>
```

Public Member Functions

- ThreadPool (size_t)
- template<class F , class... Args>
  auto enqueue (F &&f, Args &&... args) -> std::future< typename std::result_of< F(Args...)>::type >
- ~ThreadPool ()
- void stopAndJoinAll ()

6.11.1 Detailed Description

This source file has been modified slightly by Andrew Dunn

Definition at line 42 of file threadpool.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 ThreadPool()

```
ThreadPool::ThreadPool (
            size_t threads )  [inline]
```

Definition at line 64 of file threadpool.h.

```
    :   stop(false)
{
    for(size_t i = 0;i<threads;++i)
        workers.emplace_back(
            [this]
            {
                for(;;)
                {
                    std::function<void()> task;

                    {
                        std::unique_lock<std::mutex> lock(this->queue_mutex);
                        this->condition.wait(lock,
                            [this]{ return this->stop || !this->tasks.empty(); });
                        if(this->stop && this->tasks.empty())
                            return;
                        task = std::move(this->tasks.front());
                        this->tasks.pop();
                    }

                    task();
                }
            }
        );
}
```

6.11.2.2 ∼ThreadPool()

```
ThreadPool::∼ThreadPool ( )  [inline]
```

Definition at line 117 of file threadpool.h.

References stopAndJoinAll().

```
{
    stopAndJoinAll();
}
```

6.11.3 Member Function Documentation

6.11.3.1 enqueue()

```
template<class F , class... Args>
auto ThreadPool::enqueue (
            F && f,
            Args &&... args ) -> std::future<typename std::result_of<F(Args...)>::type>
```

Definition at line 93 of file threadpool.h.

Referenced by cs471::Experiment::runNEH().

```
{
    using return_type = typename std::result_of<F(Args...)>::type;

    auto task = std::make_shared< std::packaged_task<return_type()> >(
            std::bind(std::forward<F>(f), std::forward<Args>(args)...)
        );

    std::future<return_type> res = task->get_future();
    {
        std::unique_lock<std::mutex> lock(queue_mutex);

        // don't allow enqueueing after stopping the pool
        if(stop)
            throw std::runtime_error("enqueue on stopped ThreadPool");

        tasks.emplace([task](){ (*task)(); });
    }
    condition.notify_one();
    return res;
}
```

6.11.3.2 stopAndJoinAll()

```
void ThreadPool::stopAndJoinAll ( ) [inline]
```

Definition at line 122 of file threadpool.h.

Referenced by cs471::Experiment::runNEH(), and ~ThreadPool().

```
{
    {
        std::unique_lock<std::mutex> lock(queue_mutex);
        stop = true;
    }

    condition.notify_all();
    for(std::thread &worker: workers)
        worker.join();
}
```

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

- include/threadpool.h

Chapter 7

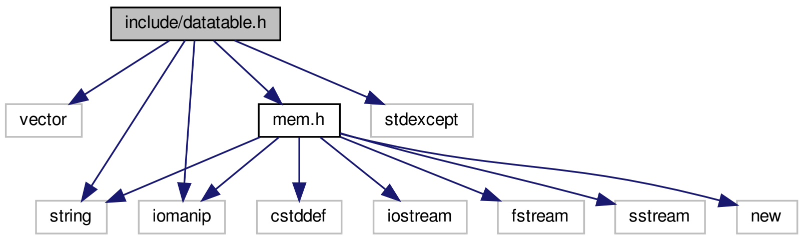
File Documentation

7.1 include/datatable.h File Reference

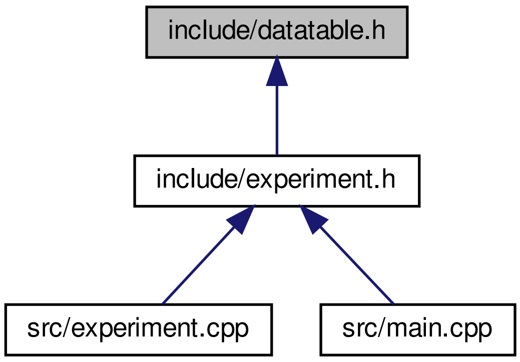
Header file for the DataTable class, which represents a spreadsheet/table of values that can easily be exported to a ∗.csv file.

```
#include <vector>
#include <string>
#include <stdexcept>
#include <iomanip>
#include "mem.h"
```

Include dependency graph for datatable.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mdata::DataTable< T >

  The DataTable class is a simple table of values with labeled columns.

Namespaces

- mdata

7.1.1 Detailed Description

Header file for the DataTable class, which represents a spreadsheet/table of values that can easily be exported to a ∗.csv file.

Author

Andrew Dunn (`Andrew.Dunn@cwu.edu`)

Version

0.2

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file datatable.h.

7.2 datatable.h

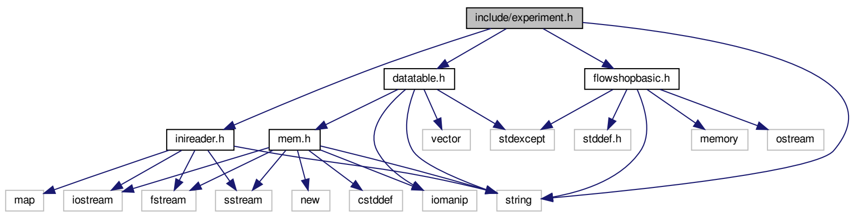
```
#ifndef __DATATABLE_H
#define __DATATABLE_H

#include <vector>
#include <string>
#include <stdexcept>
#include <iomanip>
#include "mem.h"

namespace mdata
{
    template <class T>
    class DataTable
    {
    public:
        DataTable(size_t _rows, size_t _cols) : rows(_rows), cols(_cols), dataMatrix(nullptr)
        {
            if (rows == 0)
                throw std::length_error("Table rows must be greater than 0.");
            else if (cols == 0)
                throw std::length_error("Table columns must be greater than 0.");

            dataMatrix = util::allocMatrix<T>(rows, cols);
            if (dataMatrix == nullptr)
                throw std::bad_alloc();

            colLabels.resize(_cols, std::string());
        }

        ~DataTable()
        {
            util::releaseMatrix(dataMatrix, rows);
        }

        void clearData()
        {
            util::initMatrix<T>(dataMatrix, rows, cols, 0);
        }

        std::string getColLabel(size_t colIndex)
        {
            if (colIndex >= colLabels.size())
                throw std::out_of_range("Column index out of range");

            return colLabels[colIndex];
        }

        void setColLabel(size_t colIndex, std::string newLabel)
        {
             if (colIndex >= colLabels.size())
                throw std::out_of_range("Column index out of range");

            colLabels[colIndex] = newLabel;
        }

        T getEntry(size_t row, size_t col)
        {
            if (dataMatrix == nullptr)
                throw std::runtime_error("Data matrix not allocated");
            if (row >= rows)
                throw std::out_of_range("Table row out of range");
            else if (col >= cols)
                throw std::out_of_range("Table column out of range");

            return dataMatrix[row][col];
        }

        void setEntry(size_t row, size_t col, T val)
        {
            if (dataMatrix == nullptr)
                throw std::runtime_error("Data matrix not allocated");
            if (row >= rows)
                throw std::out_of_range("Table row out of range");
            else if (col >= cols)
                throw std::out_of_range("Table column out of range");

            dataMatrix[row][col] = val;
        }

        bool exportCSV(const char* filePath)
        {
            if (dataMatrix == nullptr) return false;

            using namespace std;
            ofstream outFile;
            outFile.open(filePath, ofstream::out | ofstream::trunc);
            if (!outFile.good()) return false;

            // Print column labels
            for (unsigned int c = 0; c < cols; c++)
            {
                outFile << colLabels[c];
                if (c < cols - 1) outFile << ",";
            }

            outFile << endl;

            // Print data rows
            for (unsigned int r = 0; r < rows; r++)
            {
                for (unsigned int c = 0; c < cols; c++)
                {
                    outFile << std::setprecision(8) << dataMatrix[r][c];
                    if (c < cols - 1) outFile << ",";
                }
                outFile << endl;
            }

            outFile.close();
            return true;
        }
    private:
        size_t rows;
        size_t cols;
        std::vector<std::string> colLabels;
        T** dataMatrix;
    };
} // mdata

#endif

// =========================
// End of datatable.h
// =========================
```

7.3 include/experiment.h File Reference

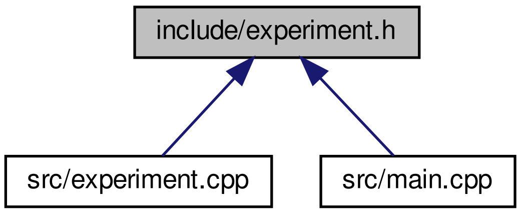
Contains the Experiment class which runs the cs471 lab 5 experiment.

```
#include <string>
#include "inireader.h"
#include "datatable.h"
#include "flowshopbasic.h"
```

Include dependency graph for experiment.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct cs471::TestParams

  Simple data structure that stores the test parameters for the experiment.
- class cs471::Experiment

  The experiment class runs takes a given ini file path, opens it, parses the parameters, then runs the NEH algorithm with the given parameters.

Namespaces

- cs471

7.3.1 Detailed Description

Contains the Experiment class which runs the cs471 lab 5 experiment.

Author

Andrew Dunn (`Andrew.Dunn@cwu.edu`)

Version

0.1

Date

2019-05-26

Copyright

Copyright (c) 2019

Definition in file experiment.h.

7.4 experiment.h

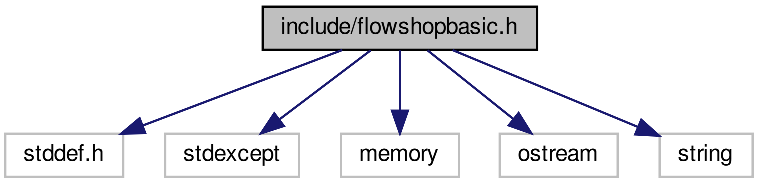
```
#ifndef __EXPERIMENT_H
#define __EXPERIMENT_H

#include <string>
#include "inireader.h"
#include "datatable.h"
#include "flowshopbasic.h"

namespace cs471
{
    struct TestParams
    {
        int minTestFile;
        int maxTestFile;
        int numThreads;
        int algorithm;
        std::string inputFilesDir;
        std::string resultsFile;
        std::string timesFile;
    };

    class Experiment
    {
    public:
        Experiment(std::string paramsFile);
        ~Experiment() = default;

        int runNEH();
        int runDebugSeq(int* seq, size_t seqSize);
    private:
        util::IniReader iniParams;

        int runNEHThreaded(TestParams* const p, const std::string inputFile, int testIndex,
 mdata::DataTable<std::string>* resultsTable);
        fshop::FlowshopBasic* allocFlowShop(const char* inputFile, int alg);
        TestParams readTestParams();
    };
}

#endif

// =========================
// End of experiment.h
// =========================
```

7.5 include/flowshopbasic.h File Reference

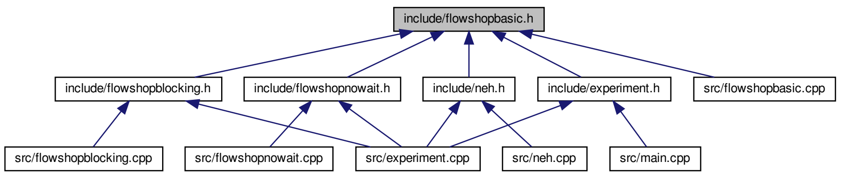
Contains the FlowshopBasic class which can solve a basic flowshop scheduling problem for a given job sequence. The FlowshopBasic class is also used as a base class for the FlowshopBlocking and FlowshopNoWait classes.

```
#include <stddef.h>
#include <stdexcept>
#include <memory>
#include <ostream>
#include <string>
```

Include dependency graph for flowshopbasic.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct fshop::FlowshopSolution

  The FlowshopSolution struct houses all solution data returned from calculating the objective flowshop function. This includes the cmax value, total flow time, job sequence used, start time matrix, and departure time matrix.
- class fshop::FlowshopBasic

  The FlowshopBasic class runs the standard flowshop scheduling problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. This class also serves as a base class for the Flowshop with Blocking and Flowshop with No Wait problem variants.

Namespaces

- fshop

7.5.1 Detailed Description

Contains the FlowshopBasic class which can solve a basic flowshop scheduling problem for a given job sequence. The FlowshopBasic class is also used as a base class for the FlowshopBlocking and FlowshopNoWait classes.

Author

Andrew Dunn (`Andrew.Dunn@cwu.edu`)

Version

0.1

Date

2019-05-24

Copyright

Copyright (c) 2019

Definition in file flowshopbasic.h.

7.6 flowshopbasic.h

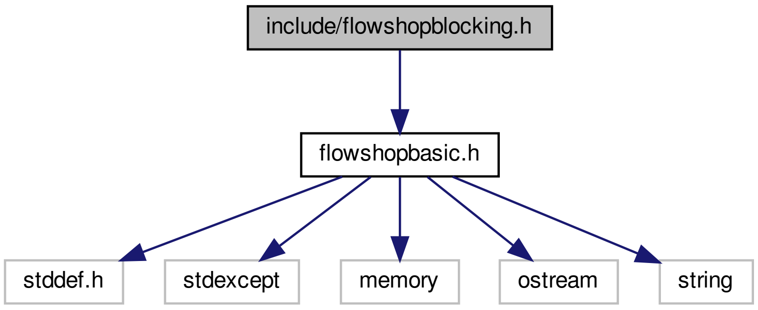
```cpp

#ifndef __FLOWSHOPBASIC_H
#define __FLOWSHOPBASIC_H

#include <stddef.h>
#include <stdexcept>
#include <memory>
#include <ostream>
#include <string>

namespace fshop
{
    struct FlowshopSolution
    {
        FlowshopSolution(int** _startimeMatrix, int** _departTimeMatrix, size_t _tMatrixRows, int* _jobSeq, size_t _seqSize, int _cmax, int _totalFlowTime);
        ~FlowshopSolution();

        const size_t seqSize;
        const size_t numMachines;
        const int cmax;
        const int totalFlowTime;
        const int* const getJobSeq();
        std::string getJobSeqAsString();
        const int** const getStartTimeMatrix();
        const int** const getDepartTimeMatrix();

        bool outputTimesCsv(const std::string& fileNamePrefix);

        void outputAll(std::ostream& os);

        // Copy constructor
        FlowshopSolution(const FlowshopSolution& obj);

        // Move constructor
        FlowshopSolution(FlowshopSolution&& obj);

        // Delete copy assignment
        FlowshopSolution& operator=(const FlowshopSolution& obj) = delete;

        // Delete move assignment
        FlowshopSolution& operator=(FlowshopSolution&& obj) = delete;
    private:
        int* jobSequence;
        int** startTimeMatrix;
        int** departTimeMatrix;
    };

    class FlowshopBasic
    {
    public:
        FlowshopBasic(const char* procTimeMatrixFile);
        virtual ~FlowshopBasic();
        virtual std::unique_ptr<FlowshopSolution> calcObjective(int* seq, size_t seqSize);

        virtual int getProcessingTime(size_t machine, size_t job);
        virtual size_t getTotalJobs();
        virtual size_t getTotalMachines();
        virtual size_t getFuncCallCounts();

        // Delete copy/move constructors and assignments
        FlowshopBasic(const FlowshopBasic& o) = delete;
        FlowshopBasic(const FlowshopBasic&& o) = delete;
        FlowshopBasic& operator=(const FlowshopBasic& o) = delete;
        FlowshopBasic& operator=(const FlowshopBasic&& o) = delete;
    protected:
        int** procTimeMatrix;
        int** startTimeMatrix;
        size_t ptMatrixRows;
        size_t ptMatrixCols;
        size_t funcCallCounter;
        virtual void validateParams(int* seq, size_t seqSize);
        virtual int** allocTimeMatrix(size_t rows, size_t cols);
        virtual void initTimeMatrix(int** compTimeMatrix, int* seq, size_t rows, size_t cols);
        virtual void calcTimeMatrix(int** compTimeMatrix, int* seq, size_t rows, size_t cols);
        virtual void calcStartTimeCol(int** startTimeMatrix, int** departTimeMatrix, int* seq, size_t curCol, size_t rows, size_t cols);
        virtual int getCmax(int** compTimeMatrix, size_t rows, size_t cols);
        virtual int getTFT(int** compTimeMatrix, size_t rows, size_t cols);
    };
}
```

```
#endif

// =========================
// End of flowshopbasic.h
// =========================
```

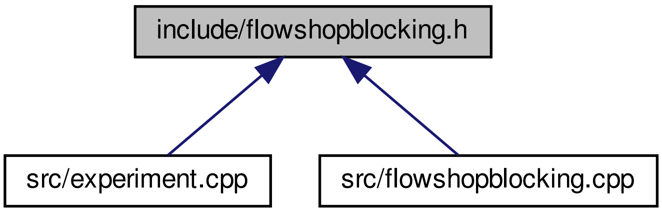
7.7 include/flowshopblocking.h File Reference

Contains the FlowshopBlocking class, which inherits FlowshopBasic and solves a flowshop with blocking problem for a specific job sequence.

`#include "flowshopbasic.h"`
Include dependency graph for flowshopblocking.h:



This graph shows which files directly or indirectly include this file:



Classes

- class fshop::FlowshopBlocking

  The FlowshopBlocking class runs the flowshop with blocking problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. Inherits from Flowshop↩ Basic.

Namespaces

- fshop

7.7.1 Detailed Description

Contains the FlowshopBlocking class, which inherits FlowshopBasic and solves a flowshop with blocking problem for a specific job sequence.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-05-25

Copyright

Copyright (c) 2019

Definition in file flowshopblocking.h.

7.8 flowshopblocking.h

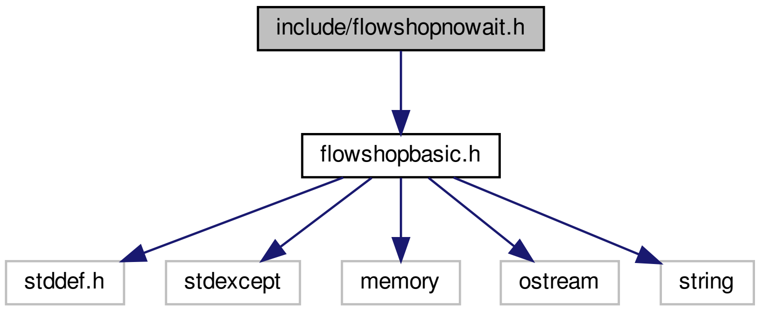
```
#ifndef __FLOWSHOPBLOCKING_H
#define __FLOWSHOPBLOCKING_H

#include "flowshopbasic.h"

namespace fshop
{
    class FlowshopBlocking : public fshop::FlowshopBasic
    {
    public:
        FlowshopBlocking(const char* procTimeMatrixFile);
        virtual ~FlowshopBlocking() = default;
    protected:
        virtual void initTimeMatrix(int** compTimeMatrix, int* seq, size_t rows, size_t cols)
 override;
        virtual void calcTimeMatrix(int** compTimeMatrix, int* seq, size_t rows, size_t cols)
 override;
    };
}

#endif

// ==========================
// End of flowshopblocking.h
// ==========================
```

7.9 include/flowshopnowait.h File Reference

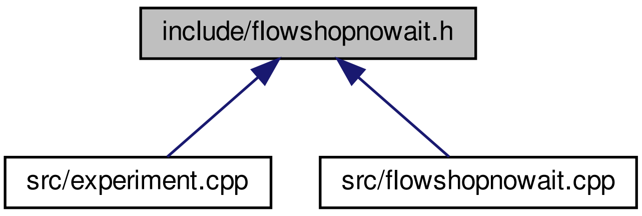
Contains the FlowshopNoWait class, which inherits FlowshopBasic and solves a flowshop with no waiting problem for a specific job sequence.

```
#include "flowshopbasic.h"
```

Include dependency graph for flowshopnowait.h:



This graph shows which files directly or indirectly include this file:



Classes

- class fshop::FlowshopNoWait

  The FlowshopNoWait class runs the flowshop with no wait problem for a given job-machine processing time matrix that is read from a file. The run() method takes the specific job sequence being calculated. Inherits from FlowshopBasic.

Namespaces

- fshop

7.9.1 Detailed Description

Contains the FlowshopNoWait class, which inherits FlowshopBasic and solves a flowshop with no waiting problem for a specific job sequence.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-05-26

Copyright

Copyright (c) 2019

Definition in file flowshopnowait.h.

7.10 flowshopnowait.h

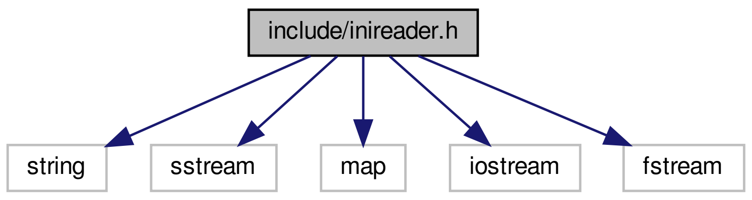
```

#ifndef __FLOWSHOPNOWAIT_H
#define __FLOWSHOPNOWAIT_H

#include "flowshopbasic.h"

namespace fshop
{
    class FlowshopNoWait : public fshop::FlowshopBasic
    {
    public:
        FlowshopNoWait(const char* procTimeMatrixFile);
        virtual ~FlowshopNoWait() = default;
    protected:
        virtual void initTimeMatrix(int** departTimeMatrix, int* seq, size_t rows, size_t
cols) override;
        virtual void calcTimeMatrix(int** departTimeMatrix, int* seq, size_t rows, size_t
cols) override;
    };
}

#endif

// ==========================
// End of flowshopnowait.h
// ==========================
```

7.11 include/inireader.h File Reference

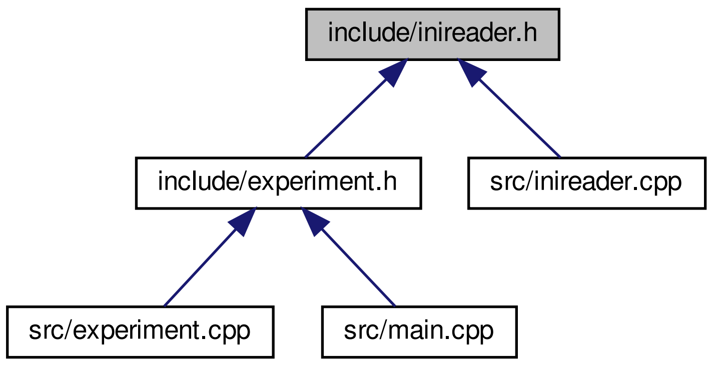
Header file for the IniReader class, which can open and parse simple ∗.ini files.

```
#include <string>
#include <sstream>
#include <map>
#include <iostream>
#include <fstream>
```

Include dependency graph for inireader.h:



This graph shows which files directly or indirectly include this file:



Classes

- class util::IniReader

    The IniReader class is a simple ∗.ini file reader and parser.

Namespaces

- util

7.11.1 Detailed Description

Header file for the IniReader class, which can open and parse simple ∗.ini files.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file inireader.h.

7.12 inireader.h

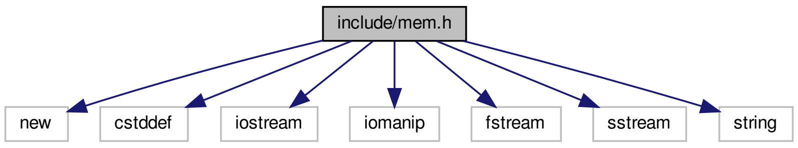
```
#ifndef __INIREADER_H
#define __INIREADER_H

#include <string>
#include <sstream>
#include <map>
#include <iostream>
#include <fstream>

namespace util
{
    class IniReader
    {
    public:
        IniReader();
        ~IniReader();
        bool openFile(std::string filePath);
        bool sectionExists(std::string section);
        bool entryExists(std::string section, std::string entry);
        std::string getEntry(std::string section, std::string entry, std::string defVal = "");

        template <class T>
        T getEntryAs(std::string section, std::string entry, T defVal = {})
        {
            std::stringstream ss(getEntry(section, entry, std::to_string(defVal)));
            T retVal;
            ss >> retVal;
            return retVal;
        }
    private:
        std::string file;
        std::map<std::string, std::map<std::string, std::string>> iniMap;
        bool parseFile();
        void parseEntry(const std::string& sectionName, const std::string& entry);
    };
}

#endif

// ========================
// End of inireader.h
// ========================
```

7.13 include/mem.h File Reference

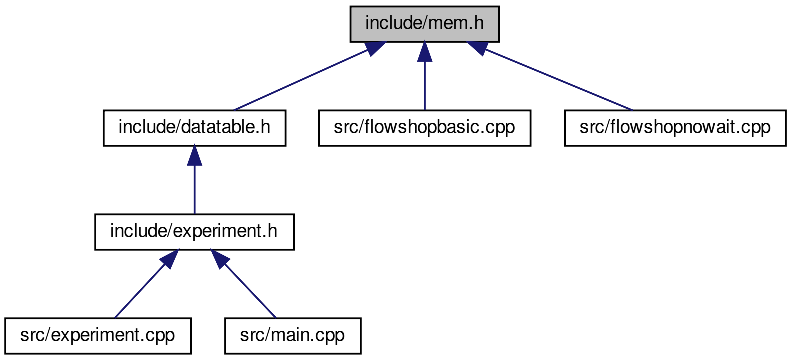
Header file for various memory utility functions.

```
#include <new>
#include <cstddef>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <string>
```

Include dependency graph for mem.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- util

Functions

- template<class T = double>
  void util::initArray (T ∗a, size_t size, T val)

  Initializes an array with some set value.

- template<class T = double>
  void util::initMatrix (T **m, size_t rows, size_t cols, T val)

  Initializes a matrix with a set value for each entry.
- template<class T = double>
  bool util::releaseArray (T *&a)

  Releases an allocated array's memory and sets the pointer to nullptr.
- template<class T = double>
  void util::releaseMatrix (T **&m, size_t rows)

  Releases an allocated matrix's memory and sets the pointer to nullptr.
- template<class T = double>
  T * util::allocArray (size_t size)

  Allocates a new array of the given data type.
- template<class T = double>
  T ** util::allocMatrix (size_t rows, size_t cols)

  Allocates a new matrix of the given data type.
- template<class T = double>
  T ** util::loadMatrixFromFile (const char *filePath, size_t &outNumRows, size_t &outNumCols)
- template<class T = double>
  void util::outputMatrix (std::ostream &os, T **matrix, size_t rows, size_t cols, int colWidth=3)
- template<class T = double>
  void util::copyArray (T *src, T *dest, size_t size)

  Copies the elements from one equal-sized array to another.

7.13.1 Detailed Description

Header file for various memory utility functions.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.2

Date

2019-04-02

Copyright

Copyright (c) 2019

Definition in file mem.h.

7.14 mem.h

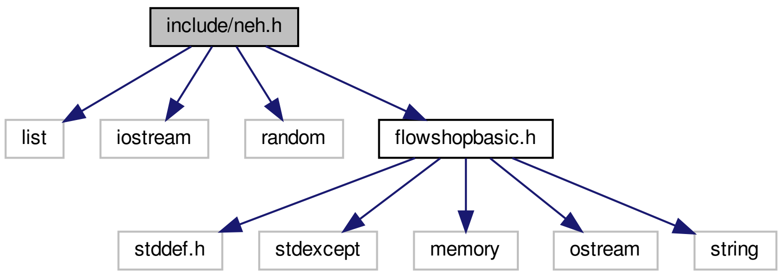
```

#ifndef __MEM_H
#define __MEM_H

#include <new> // std::nothrow
#include <cstddef> // size_t definition
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <string>

namespace util
{
    template <class T = double>
    inline void initArray(T* a, size_t size, T val)
    {
        if (a == nullptr) return;

        for (size_t i = 0; i < size; i++)
        {
            a[i] = val;
        }
    }

    template <class T = double>
    inline void initMatrix(T** m, size_t rows, size_t cols, T val)
    {
        if (m == nullptr) return;

        for (size_t i = 0; i < rows; i++)
        {
            initArray(m[i], cols, val);
        }
    }

    template <class T = double>
    bool releaseArray(T*& a)
    {
        if (a == nullptr) return true;

        try
        {
            delete[] a;
            a = nullptr;
            return true;
        }
        catch(...)
        {
            return false;
        }
    }

    template <class T = double>
    void releaseMatrix(T**& m, size_t rows)
    {
        if (m == nullptr) return;

        for (size_t i = 0; i < rows; i++)
        {
            if (m[i] != nullptr)
            {
                // Release each row
                releaseArray<T>(m[i]);
            }
        }

        // Release columns
        delete[] m;
        m = nullptr;
    }

    template <class T = double>
    inline T* allocArray(size_t size)
    {
        return new(std::nothrow) T[size];
    }

    template <class T = double>
    inline T** allocMatrix(size_t rows, size_t cols)
    {
        T** m = (T**)allocArray<T*>(rows);
        if (m == nullptr) return nullptr;

        for (size_t i = 0; i < rows; i++)
        {
            m[i] = allocArray<T>(cols);
            if (m[i] == nullptr)
            {
                releaseMatrix<T>(m, rows);
                return nullptr;
            }
        }

        return m;
    }

    template <class T = double>
    inline T** loadMatrixFromFile(const char* filePath, size_t& outNumRows, size_t& outNumCols)
    {
        outNumRows = 0;
        outNumCols = 0;

        std::ifstream is(filePath);
        if (!is.good())
        {
            std::cerr << "Error loading matrix from file: Unable to open file." << std::endl;
            return nullptr;
        }

        std::string line;
        if (!std::getline(is, line))
        {
            std::cerr << "Error loading matrix from file: File is empty or invalid." << std::endl;
            is.close();
            return nullptr;
        }

        size_t rows = 0;
        size_t cols = 0;

        std::stringstream ss(line);
        if (!(ss >> rows >> cols) || rows == 0 || cols == 0)
        {
            std::cerr << "Error loading matrix from file: Row or column size is zero." << std::endl;
            is.close();
            return nullptr;
        }


        T** retMatrix = allocMatrix<T>(rows, cols);
        if (retMatrix == nullptr)
        {
            std::cerr << "Error loading matrix from file: Matrix memory allocation failed." << std::endl;
            is.close();
            return nullptr;
        }

        for (size_t r = 0; r < rows; r++)
        {
            if (!std::getline(is, line))
            {
                std::cerr << "Error loading matrix from file: EOF reached before reading all rows." << std::endl;
                releaseMatrix<T>(retMatrix, rows);
                is.close();
                return nullptr;
            }

            std::stringstream ss(line);

            for (size_t c = 0; c < cols; c++)
            {
                T entry = 0;
                if (!(ss >> entry))
                {
                    std::cerr << "Error loading matrix from file: EOL reached before reading all cols." << std::endl;
                    releaseMatrix<T>(retMatrix, rows);
                    is.close();
                    return nullptr;
                }

                retMatrix[r][c] = entry;
            }
        }

        is.close();
        outNumRows = rows;
        outNumCols = cols;
        return retMatrix;
    }

    template <class T = double>
    inline void outputMatrix(std::ostream& os, T** matrix, size_t rows, size_t cols, int colWidth = 3)
    {
        if (matrix == nullptr)
            return;

        for (size_t r = 0; r < rows; r++)
        {
            for (size_t c = 0; c < cols; c++)
            {
                os << std::setw(3) << matrix[r][c];
                if (c < cols - 1)
                    os << " ";
                else
                    os << std::endl;
            }
        }
    }

    template <class T = double>
    inline void copyArray(T* src, T* dest, size_t size)
    {
        for (size_t i = 0; i < size; i++)
            dest[i] = src[i];
    }
}

#endif

// =========================
// End of mem.h
// =========================
```

7.15 include/neh.h File Reference

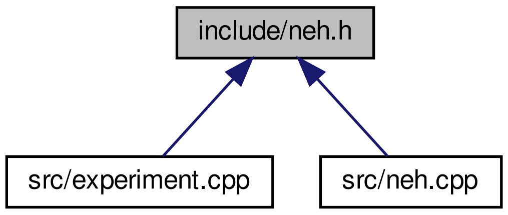
Contains the NEH class, which runs the NEH algorithm on a given flowshop problem. The NEH algorithm aims to optimize the job sequence such that it produces the smallest cMax value.

```
#include <list>
#include <iostream>
#include <random>
#include "flowshopbasic.h"
```

Include dependency graph for neh.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct fshop::JobTimePair

    Simple struct that pairs a job with it's total processing time. Used for sorting purposes.
- class fshop::NEH

    The NEH class runs the NEH algorithm on the given flowshop objective function and attempts to optimize the job sequence that produces the smallest cmax value.

Namespaces

- fshop

Typedefs

- using fsSol = std::unique_ptr< fshop::FlowshopSolution >

7.15.1 Detailed Description

Contains the NEH class, which runs the NEH algorithm on a given flowshop problem. The NEH algorithm aims to optimize the job sequence such that it produces the smallest cMax value.

Author

Andrew Dunn (`Andrew.Dunn@cwu.edu`)

Version

0.1

Date

2019-05-27

Copyright

Copyright (c) 2019

Definition in file neh.h.

7.15.2 Typedef Documentation

7.15.2.1 fsSol

```
using fsSol = std::unique_ptr<fshop::FlowshopSolution>
```

Definition at line 21 of file neh.h.

7.16 neh.h

```
00001
00013 #ifndef __NEH_H
00014 #define __NEH_H
00015
00016 #include <list>
00017 #include <iostream>
00018 #include <random>
00019 #include "flowshopbasic.h"
00020
00021 using fsSol = std::unique_ptr<fshop::FlowshopSolution>;
00022
00023 namespace fshop
00024 {
00029     struct JobTimePair
00030     {
00031         const int job;
00032         const int time;
00033
00034         JobTimePair(int _job, int _time)
00035             : job(_job), time(_time)
00036         {
00037         }
00038     };
00039
00046     class NEH
00047     {
00048     public:
00049         NEH();
00050         fsSol run(FlowshopBasic* const objectiveFs);
00051     private:
00052         std::random_device rd;
00053         std::mt19937 randEngine;
00054         std::uniform_real_distribution<float> randChance;
00055
00056         void makeInitialAvailJobList(FlowshopBasic* const objectiveFs,
      std::list<fshop::JobTimePair>& outList);
00057         fsSol bestPermutation(FlowshopBasic* const objectiveFs, const std::list<int>&
      baseList, int jobInsert, std::list<int>& outBestSeq);
00058     };
00059 }
00060
00061 #endif
00062
00063 // =========================
00064 // End of neh.h
00065 // =========================
```

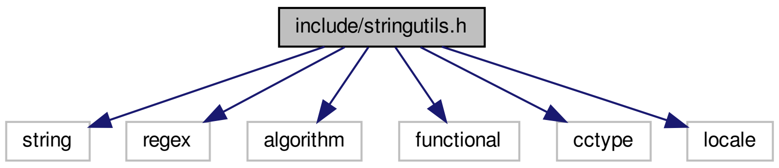
7.17 include/stringutils.h File Reference

Contains various string manipulation helper functions.

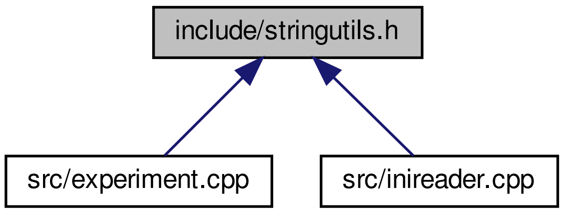
```
#include <string>
#include <regex>
#include <algorithm>
```

```
#include <functional>
#include <cctype>
#include <locale>
```

Include dependency graph for stringutils.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- util

7.17.1 Detailed Description

Contains various string manipulation helper functions.

Author

Evan Teran (`https://github.com/eteran`) and Andrew Dunn

Date

2019-04-01

Definition in file stringutils.h.

7.18 stringutils.h

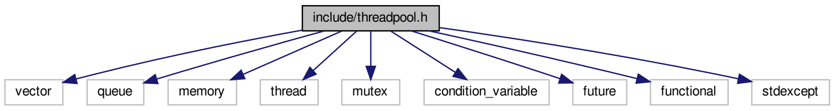
```

#ifndef __STRINGUTILS_H
#define __STRINGUTILS_H

#include <string>
#include <regex>
#include <algorithm>
#include <functional>
#include <cctype>
#include <locale>

namespace util
{
    static inline std::string s_replace(std::string input, std::string pattern, std::string replacement)
    {
        pattern = std::string("\\") + pattern;
        return std::regex_replace(input, std::regex(pattern), replacement);
    }

    // =======================================================
    // The string functions below were written by Evan Teran
    // from Stack Overflow:
    // https://stackoverflow.com/questions/216823/whats-the-best-way-to-trim-stdstring
    // =======================================================

    // trim from start (in place)
    static inline void s_ltrim(std::string &s) {
        s.erase(s.begin(), std::find_if(s.begin(), s.end(),
                std::not1(std::ptr_fun<int, int>(std::isspace))));
    }

    // trim from end (in place)
    static inline void s_rtrim(std::string &s) {
        s.erase(std::find_if(s.rbegin(), s.rend(),
                std::not1(std::ptr_fun<int, int>(std::isspace))).base(), s.end());
    }

    // trim from both ends (in place)
    static inline void s_trim(std::string &s) {
        s_ltrim(s);
        s_rtrim(s);
    }

    // trim from start (copying)
    static inline std::string s_ltrim_copy(std::string s) {
        s_ltrim(s);
        return s;
    }

    // trim from end (copying)
    static inline std::string s_rtrim_copy(std::string s) {
        s_rtrim(s);
        return s;
    }

    // trim from both ends (copying)
    static inline std::string s_trim_copy(std::string s) {
        s_trim(s);
        return s;
    }
}
#endif

// =========================
// End of stringutils.h
// =========================
```

7.19 include/threadpool.h File Reference

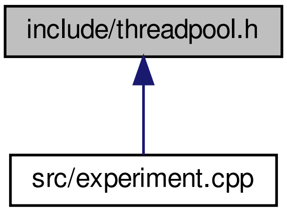
```
#include <vector>
#include <queue>
#include <memory>
#include <thread>
#include <mutex>
#include <condition_variable>
```

```
#include <future>
#include <functional>
#include <stdexcept>
```

Include dependency graph for threadpool.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ThreadPool

7.20 threadpool.h

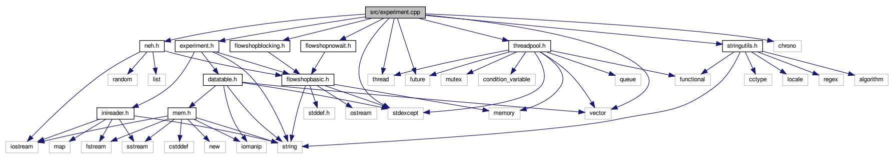
```
#ifndef __THREADPOOL_H
#define __THREADPOOL_H

#include <vector>
#include <queue>
#include <memory>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <future>
#include <functional>
#include <stdexcept>

class ThreadPool {
public:
    ThreadPool(size_t);
    template<class F, class... Args>
    auto enqueue(F&& f, Args&&... args)
        -> std::future<typename std::result_of<F(Args...)>::type>;
    ~ThreadPool();

    void stopAndJoinAll();
private:
    // need to keep track of threads so we can join them
    std::vector< std::thread > workers;
    // the task queue
    std::queue< std::function<void()> > tasks;

    // synchronization
    std::mutex queue_mutex;
    std::condition_variable condition;
    bool stop;
};

// the constructor just launches some amount of workers
inline ThreadPool::ThreadPool(size_t threads)
    :   stop(false)
{
    for(size_t i = 0;i<threads;++i)
        workers.emplace_back(
            [this]
            {
                for(;;)
                {
                    std::function<void()> task;

                    {
                        std::unique_lock<std::mutex> lock(this->queue_mutex);
                        this->condition.wait(lock,
                            [this]{ return this->stop || !this->tasks.empty(); });
                        if(this->stop && this->tasks.empty())
                            return;
                        task = std::move(this->tasks.front());
                        this->tasks.pop();
                    }

                    task();
                }
            }
        );
}

// add new work item to the pool
template<class F, class... Args>
auto ThreadPool::enqueue(F&& f, Args&&... args)
    -> std::future<typename std::result_of<F(Args...)>::type>
{
    using return_type = typename std::result_of<F(Args...)>::type;

    auto task = std::make_shared< std::packaged_task<return_type()> >(
            std::bind(std::forward<F>(f), std::forward<Args>(args)...)
        );

    std::future<return_type> res = task->get_future();
    {
        std::unique_lock<std::mutex> lock(queue_mutex);

        // don't allow enqueueing after stopping the pool
        if(stop)
            throw std::runtime_error("enqueue on stopped ThreadPool");

        tasks.emplace([task](){ (*task)(); });
    }
    condition.notify_one();
    return res;
}

// the destructor joins all threads
inline ThreadPool::~ThreadPool()
{
    stopAndJoinAll();
}

inline void ThreadPool::stopAndJoinAll()
{
    {
        std::unique_lock<std::mutex> lock(queue_mutex);
        stop = true;
    }

    condition.notify_all();
    for(std::thread &worker: workers)
        worker.join();
}

#endif

// ========================
// End of threadpool.h
// ========================
```

7.21 src/experiment.cpp File Reference

Implementation file for the Experiment class.

```
#include <stdexcept>
#include <vector>
#include <thread>
#include <future>
#include <chrono>
#include "experiment.h"
#include "threadpool.h"
#include "stringutils.h"
#include "flowshopblocking.h"
#include "flowshopnowait.h"
#include "neh.h"
```

Include dependency graph for experiment.cpp:



Macros

- #define INI_TEST_SECTION "test"
- #define INI_TEST_MINFILE "minTestFile"
- #define INI_TEST_MAXFILE "maxTestFile"
- #define INI_TEST_NUMTHREADS "numThreads"
- #define INI_TEST_ALGORITHM "algorithm"
- #define INI_TEST_INPUTFILEDIR "inputFilesDir"
- #define INI_TEST_RESULTSFILE "resultsFile"
- #define INI_TEST_TIMESFILE "timesFile"

7.21.1 Detailed Description

Implementation file for the Experiment class.

Author

Andrew Dunn (`Andrew.Dunn@cwu.edu`)

Version

0.1

Date

2019-05-26

Copyright

Copyright (c) 2019

Definition in file experiment.cpp.

7.21.2 Macro Definition Documentation

7.21.2.1 INI_TEST_ALGORITHM

```
#define INI_TEST_ALGORITHM "algorithm"
```

Definition at line 28 of file experiment.cpp.

Referenced by cs471::Experiment::runNEH().

7.21.2.2 INI_TEST_INPUTFILEDIR

```
#define INI_TEST_INPUTFILEDIR "inputFilesDir"
```

Definition at line 29 of file experiment.cpp.

Referenced by cs471::Experiment::runNEH().

7.21.2.3 INI_TEST_MAXFILE

```
#define INI_TEST_MAXFILE "maxTestFile"
```

Definition at line 26 of file experiment.cpp.

Referenced by cs471::Experiment::runNEH().

7.21.2.4 INI_TEST_MINFILE

```
#define INI_TEST_MINFILE "minTestFile"
```

Definition at line 25 of file experiment.cpp.

Referenced by cs471::Experiment::runNEH().

7.21.2.5 INI_TEST_NUMTHREADS

```
#define INI_TEST_NUMTHREADS "numThreads"
```

Definition at line 27 of file experiment.cpp.

Referenced by cs471::Experiment::runNEH().

7.21.2.6 INI_TEST_RESULTSFILE

```
#define INI_TEST_RESULTSFILE "resultsFile"
```

Definition at line 30 of file experiment.cpp.

Referenced by cs471::Experiment::runNEH().

7.21.2.7 INI_TEST_SECTION

```
#define INI_TEST_SECTION "test"
```

Definition at line 24 of file experiment.cpp.

Referenced by cs471::Experiment::runNEH().

7.21.2.8 INI_TEST_TIMESFILE

```
#define INI_TEST_TIMESFILE "timesFile"
```

Definition at line 31 of file experiment.cpp.

Referenced by cs471::Experiment::runNEH().

7.22 experiment.cpp

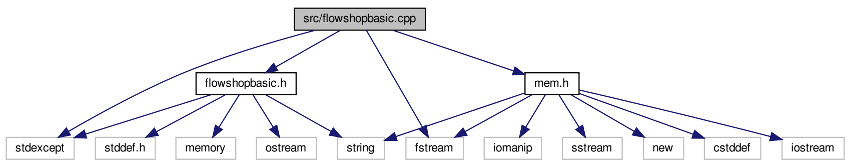
```
00001
00012 #include <stdexcept>
00013 #include <vector>
00014 #include <thread>
00015 #include <future>
00016 #include <chrono>
00017 #include "experiment.h"
00018 #include "threadpool.h"
00019 #include "stringutils.h"
00020 #include "flowshopblocking.h"
00021 #include "flowshopnowait.h"
00022 #include "neh.h"
00023
00024 #define INI_TEST_SECTION      "test"
00025 #define INI_TEST_MINFILE      "minTestFile"
00026 #define INI_TEST_MAXFILE      "maxTestFile"
00027 #define INI_TEST_NUMTHREADS   "numThreads"
00028 #define INI_TEST_ALGORITHM    "algorithm"
00029 #define INI_TEST_INPUTFILEDIR "inputFilesDir"
00030 #define INI_TEST_RESULTSFILE  "resultsFile"
00031 #define INI_TEST_TIMESFILE    "timesFile"
00032
00033 using namespace cs471;
00034 using namespace fshop;
00035 using namespace util;
00036 using namespace std;
00037 using namespace chrono;
00038
00044 Experiment::Experiment(string paramsFile)
00045 {
00046     // Attempt to open parameters file
00047     if (!iniParams.openFile(paramsFile))
00048     {
        string msg = "Error opening ini file: ";
        msg += paramsFile;
        throw std::runtime_error(msg);
    }

    cout << "Loaded parameters file: " << paramsFile << endl;
}

int Experiment::runNEH()
{
    // Retrieve test parameters from ini file
    TestParams p = readTestParams();

    // Construct data table to store experiment results
    mdata::DataTable<string> resultsTable(p.maxTestFile - p.minTestFile + 1, 6);

    // Initialize thread pool with a parameter-given number of threads
    ThreadPool tpool(p.numThreads);

    // Initialize thread future vector, used for thread pool synchronization
    // and keeps track of the individual tasks being executed.
    vector<std::future<int>> futures;

    cout << "Started " << p.numThreads << " worker threads ..." << endl;

    if (p.algorithm == 1)
        cout << "Running NEH on Flow Shop with Blocking ..." << endl;
    else if (p.algorithm == 2)
        cout << "Running NEH on Flow Shop with No Wait ..." << endl;
    else
        cout << "Running NEH on Flow Shop Scheduling ..." << endl;

    // Prepare results table column header labels
    resultsTable.setColLabel(0, "Data Set");
    resultsTable.setColLabel(1, "cMax");
    resultsTable.setColLabel(2, "TFT");
    resultsTable.setColLabel(3, "Func Calls");
    resultsTable.setColLabel(4, "Execution Time (ms)");
    resultsTable.setColLabel(5, "Sequence");

    // Add all input test files as tasks in thread pool
    for (int i = p.minTestFile; i <= p.maxTestFile; i++)
    {
        string inputFile = std::to_string(i) + ".txt";
        futures.emplace_back(
            tpool.enqueue(&cs471::Experiment::runNEHThreaded, this, &p, inputFile, i, &resultsTable)
        );
    }

    // const size_t totalFutures = futures.size();

    // Join all thread pool tasks using futures vector
    // and get the return value for each
    for (int i = 0; i < futures.size(); i++)
    {
        int err = futures[i].get();
        if (err)
        {
            // Threaded task returned with an error code, bail
            tpool.stopAndJoinAll();
            return err;
        }
    }

    // Output results table to a csv file
    if (!p.resultsFile.empty())
    {
        resultsTable.exportCSV(p.resultsFile.c_str());
        cout << "Results exported to: " << p.resultsFile << endl;
    }

    return 0;
}

int Experiment::runNEHThreaded(TestParams* const p, const std::string inputFile, int testIndex, mdata::DataTable<std::string>* resultsTable)
{
    string fullInputPath = p->inputFilesDir + inputFile;

    // Get the flowshop objective function that we want to optimize
    auto objectiveFs = allocFlowShop(fullInputPath.c_str(), p->algorithm);
    if (objectiveFs == nullptr)
        return 1;

    // Prepare pointer to results
    fsSol result = nullptr;

    // Start recording execution time
    high_resolution_clock::time_point t_start = high_resolution_clock::now();

    try
    {
        // Run the NEH algorithm on the objective flowshop function
        NEH neh;
        result = neh.run(objectiveFs);
    }
    catch(const std::exception& e)
    {
        std::cerr << "An exception occurred while running NEH:" << endl;
        std::cerr << e.what() << endl;
        std::cerr << "Input file: " << inputFile << endl;
        return 2;
    }

    // Record execution time
    high_resolution_clock::time_point t_end = high_resolution_clock::now();
    double execTimeMs = static_cast<double>(duration_cast<nanoseconds>(t_end - t_start).count()) / 1000000.
0;

    // Insert NEH results into results table at the correct row
    resultsTable->setEntry(testIndex, 0, std::to_string(testIndex));
    resultsTable->setEntry(testIndex, 1, std::to_string(result->cmax));
    resultsTable->setEntry(testIndex, 2, std::to_string(result->totalFlowTime));
    resultsTable->setEntry(testIndex, 3, std::to_string(objectiveFs->getFuncCallCounts()));
    resultsTable->setEntry(testIndex, 4, std::to_string(execTimeMs));
    resultsTable->setEntry(testIndex, 5, result->getJobSeqAsString());


    // ======= GANTT STUFF =======
    /*
    const size_t numMachines = objectiveFs->getTotalMachines();
    const size_t numJobs = objectiveFs->getTotalJobs();

    auto startTimeMatrix = result->getStartTimeMatrix();
    auto departTimeMatrix = result->getDepartTimeMatrix();

    mdata::DataTable<std::string> ganttTable(numJobs * numMachines, 5);

    ganttTable.setColLabel(0, "Item");
    ganttTable.setColLabel(1, "Machine");
    ganttTable.setColLabel(2, "Job");
    ganttTable.setColLabel(3, "Start");
    ganttTable.setColLabel(4, "End");

    size_t row = 0;

    for (size_t m = 0; m < numMachines; m++)
    {
        for (size_t j = 0; j < numJobs; j++)
        {
            ganttTable.setEntry(row, 0, std::to_string(row + 1));
            ganttTable.setEntry(row, 1, std::string("Machine ") + std::to_string(m + 1));
            ganttTable.setEntry(row, 2, std::string("Job ") + std::to_string(j + 1));
            ganttTable.setEntry(row, 3, std::to_string(startTimeMatrix[m][j]));
            ganttTable.setEntry(row, 4, std::to_string(departTimeMatrix[m][j]));

            row++;
        }
    }

    std::string ganttfile = "results/gantt/";

    if (p->algorithm == 0)
        ganttfile += "fss/";
    else if (p->algorithm == 1)
        ganttfile += "fsb/";
    else
        ganttfile += "fsnw/";

    ganttfile += std::to_string(testIndex);
    ganttfile += "-gantt.csv";
    ganttTable.exportCSV(ganttfile.c_str());
    */
    // ===========================


    // Dump NEH results start and departure time matrices to a csv file
    if (!p->timesFile.empty())
        result->outputTimesCsv(util::s_replace(p->timesFile, "%TEST%", std::to_string(testIndex)))
;

    // Clean up allocated memory
    delete objectiveFs;

    return 0;
}

FlowshopBasic* Experiment::allocFlowShop(const char* inputFile, int alg)
{
    FlowshopBasic* objectiveFs = nullptr;

    switch (alg)
    {
        case 0:
            objectiveFs = new FlowshopBasic(inputFile);
            break;
        case 1:
            objectiveFs = new FlowshopBlocking(inputFile);
            break;
        case 2:
            objectiveFs = new FlowshopNoWait(inputFile);
            break;
    }

    return objectiveFs;
}

TestParams Experiment::readTestParams()
{
    TestParams p = { };

    p.minTestFile = iniParams.getEntryAs<int>(INI_TEST_SECTION, INI_TEST_MINFILE, 0);
    p.maxTestFile = iniParams.getEntryAs<int>(INI_TEST_SECTION, INI_TEST_MAXFILE, 120);
    p.numThreads = iniParams.getEntryAs<int>(INI_TEST_SECTION, INI_TEST_NUMTHREADS, 1);
    p.algorithm = iniParams.getEntryAs<int>(INI_TEST_SECTION, INI_TEST_ALGORITHM, 0);
    p.inputFilesDir = iniParams.getEntry(INI_TEST_SECTION, INI_TEST_INPUTFILEDIR, "");
    p.resultsFile = iniParams.getEntry(INI_TEST_SECTION, INI_TEST_RESULTSFILE, "");
    p.timesFile = iniParams.getEntry(INI_TEST_SECTION, INI_TEST_TIMESFILE, "");

    // Check bounds for numThreads
    if (p.numThreads < 1 || p.numThreads > 16)
    {
        cout << "Warning: Number of threads invalid. Defaulting to default 1 threads." << endl;
        p.numThreads = 1;
    }

    // Check bounds for algorithm selection
    if (p.algorithm < 0 || p.algorithm > 2)
    {
        cout << "Warning: Algorithm selection invalid. Defaulting to algorithm 0." << endl;
        p.algorithm = 0;
    }

    return p;
}

int Experiment::runDebugSeq(int* seq, size_t seqSize)
{
    // Retrieve test parameters from ini file
    TestParams p = readTestParams();

    if (p.algorithm == 1)
        cout << "Running Flow Shop with Blocking ..." << endl;
    else if (p.algorithm == 2)
        cout << "Running Flow Shop with No Wait ..." << endl;
    else
        cout << "Running Flow Shop Scheduling ..." << endl;

    cout << endl;

    // Prepare pointer to results
    fsSol result = nullptr;

    for (int i = p.minTestFile; i <= p.maxTestFile; i++)
    {
        string fullInputPath = p.inputFilesDir + std::to_string(i) + ".txt";

        cout << "Input file: " << fullInputPath << endl;

        // Get the flowshop objective function that we want to optimize
        auto objectiveFs = allocFlowShop(fullInputPath.c_str(), p.algorithm);
        if (objectiveFs == nullptr)
        {
            cout << "Objective flowshop function encountered an error." << endl;
            return 1;
        }

        result = objectiveFs->calcObjective(seq, seqSize);
        result->outputAll(std::cout);

        delete objectiveFs;

        cout << "========================================" << endl;
    }

    cout << "Debug objective function sequence tests completed." << endl;

    return 0;
}

// =========================
// End of experiment.cpp
// =========================
```

7.23 src/flowshopbasic.cpp File Reference

Implementation file for the FlowshopBasic class.

```
#include <stdexcept>
#include <fstream>
#include "flowshopbasic.h"
#include "mem.h"
```

Include dependency graph for flowshopbasic.cpp:



Functions

- int max (int val1, int val2)

  Simple inline helper function that returns the max of two integers.

7.23.1 Detailed Description

Implementation file for the FlowshopBasic class.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-05-24

Copyright

Copyright (c) 2019

Definition in file flowshopbasic.cpp.

7.23.2 Function Documentation

7.23.2.1 max()

```
int max (
            int val1,
            int val2 )  [inline]
```

Simple inline helper function that returns the max of two integers.

Parameters

val1	First integer
val2	Second integer

Returns

Returns the maximum of the two integers

Definition at line 26 of file flowshopbasic.cpp.

Referenced by fshop::FlowshopBasic::calcTimeMatrix().

```
00027 {
00028     if (val1 >= val2) return val1;
00029     else return val2;
00030 }
```

7.24 flowshopbasic.cpp

```
00001
00012 #include <stdexcept>
00013 #include <fstream>
00014 #include "flowshopbasic.h"
00015 #include "mem.h"
00016
00017 using namespace fshop;
00018
00026 inline int max(int val1, int val2)
00027 {
```

```cpp
    if (val1 >= val2) return val1;
    else return val2;
}

// ============================================================

FlowshopSolution::FlowshopSolution(int** _startimeMatrix, int** _departTimeMatrix, size_t _tMatrixRows, int* _jobSeq, size_t _seqSize, int _cmax, int _totalFlowTime)
    : startTimeMatrix(_startimeMatrix), departTimeMatrix(_departTimeMatrix), numMachines(_tMatrixRows), seqSize(_seqSize), cmax(_cmax), totalFlowTime(_totalFlowTime)
{
    if (_jobSeq == nullptr)
        throw std::invalid_argument("Error: _jobSeq cannot be nullptr");
    else if (_startimeMatrix == nullptr)
        throw std::invalid_argument("Error: _startimeMatrix cannot be nullptr");
    else if (_departTimeMatrix == nullptr)
        throw std::invalid_argument("Error: _departTimeMatrix cannot be nullptr");
    else if (seqSize == 0)
        throw std::invalid_argument("Error: _seqSize cannot be zero");

    jobSequence = util::allocArray<int>(seqSize);
    for (size_t i = 0; i < seqSize; i++)
        jobSequence[i] = _jobSeq[i];
}

FlowshopSolution::~FlowshopSolution()
{
    util::releaseArray<int>(jobSequence);
    util::releaseMatrix<int>(startTimeMatrix, numMachines);
    util::releaseMatrix<int>(departTimeMatrix, numMachines);
}

const int* const FlowshopSolution::getJobSeq()
{
    return const_cast<const int* const>(jobSequence);
}

std::string FlowshopSolution::getJobSeqAsString()
{
    std::string retStr = "[";

    for (size_t i = 0; i < seqSize; i++)
    {
        retStr += std::to_string(jobSequence[i]);
        if (i < seqSize - 1) retStr += "-";
    }

    retStr += "]";
    return retStr;
}

const int** const FlowshopSolution::getStartTimeMatrix()
{
    return const_cast<const int** const>(startTimeMatrix);
}

const int** const FlowshopSolution::getDepartTimeMatrix()
{
    return const_cast<const int** const>(departTimeMatrix);
}

bool FlowshopSolution::outputTimesCsv(const std::string& fileNamePrefix)
{
    using namespace std;

    // Create file name strings
    string startTimesFile = fileNamePrefix + "starttimes.csv";
    string departTimesFile = fileNamePrefix + "departtimes.csv";

    // Open files
    ofstream startOs = ofstream(startTimesFile, ios::trunc | ios::out);
    if (!startOs.good()) return false;

    ofstream departOs = ofstream(departTimesFile, ios::trunc | ios::out);
    if (!departOs.good()) return false;

    // Output start times and departure times data to the files
    for (size_t m = 0; m < numMachines; m++)
    {
        for (size_t j = 0; j < seqSize; j++)
        {
            startOs << startTimeMatrix[m][j];
            departOs << departTimeMatrix[m][j];

            if (j < seqSize - 1)
            {
                startOs << ",";
                departOs << ",";
            }
        }

        startOs << endl;
        departOs << endl;
    }

    // Close file handles and return
    startOs.close();
    departOs.close();
    return true;
}

void FlowshopSolution::outputAll(std::ostream& os)
{
    std::cout << "Input seq: ";

    for (size_t i = 0; i < seqSize; i++)
    {
        std::cout << jobSequence[i];
        if (i < seqSize - 1)
            std::cout << ", ";
    }

    std::cout << std::endl;

    std::cout << "Cmax: " << cmax << std::endl;
    std::cout << "TFT: " << totalFlowTime << std::endl << std::endl;

    std::cout << "Starting times matrix:" << std::endl;
    util::outputMatrix(std::cout, startTimeMatrix, numMachines, seqSize, 4);
    std::cout << std::endl;

    std::cout << "Departure times matrix:" << std::endl;
    util::outputMatrix(std::cout, departTimeMatrix, numMachines, seqSize, 4);
    std::cout << std::endl;
}

FlowshopSolution::FlowshopSolution(const FlowshopSolution& obj)
    : startTimeMatrix(obj.startTimeMatrix), departTimeMatrix(obj.departTimeMatrix), numMachines(obj.numMachines), seqSize(obj.seqSize), cmax(obj.cmax), totalFlowTime(obj.totalFlowTime)
{
    if (obj.jobSequence == nullptr)
        throw std::invalid_argument("Error: jobSequence cannot be nullptr");
    else if (seqSize == 0)
        throw std::invalid_argument("Error: seqSize cannot be zero");

    jobSequence = util::allocArray<int>(seqSize);
    for (size_t i = 0; i < seqSize; i++)
        jobSequence[i] = obj.jobSequence[i];
}

FlowshopSolution::FlowshopSolution(FlowshopSolution&& obj)
    : numMachines(obj.numMachines), seqSize(obj.seqSize), cmax(obj.cmax), totalFlowTime(obj.totalFlowTime)
{
    jobSequence = obj.jobSequence;
    startTimeMatrix = obj.startTimeMatrix;
    departTimeMatrix = obj.departTimeMatrix;
    obj.jobSequence = nullptr;
    obj.startTimeMatrix = nullptr;
    obj.departTimeMatrix = nullptr;
}

// ============================================================

FlowshopBasic::FlowshopBasic(const char* procTimeMatrixFile)
    : startTimeMatrix(nullptr), ptMatrixRows(0), ptMatrixCols(0), funcCallCounter(0)
{
    // Attempt to load job processing times from the given file
    procTimeMatrix = util::loadMatrixFromFile<int>(procTimeMatrixFile, ptMatrixRows, ptMatrixCols);
    if (procTimeMatrix == nullptr)
    {
        std::string msg = "Error when loading matrix file: ";
        msg += procTimeMatrixFile;
        throw std::runtime_error(msg);
    }
}

FlowshopBasic::~FlowshopBasic()
{
    util::releaseMatrix<int>(procTimeMatrix, ptMatrixRows);
```

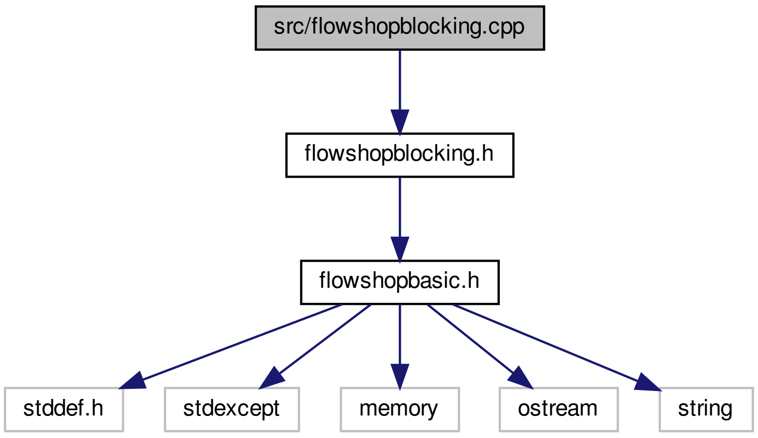
```
}

int FlowshopBasic::getProcessingTime(size_t machine, size_t job)
{
    if (machine == 0 || job == 0)
    {
        std::string msg = "Error: Machine or job number cannot be zero";
        throw std::out_of_range(msg);
    }
    else if (machine > ptMatrixRows || job > ptMatrixCols)
    {
        std::string msg = "Error: Machine or job number out of range";
        throw std::out_of_range(msg);
    }

    return procTimeMatrix[machine - 1][job - 1];
}

size_t FlowshopBasic::getTotalJobs()
{
    return ptMatrixCols;
}

size_t FlowshopBasic::getTotalMachines()
{
    return ptMatrixRows;
}

size_t FlowshopBasic::getFuncCallCounts()
{
    return funcCallCounter;
}

std::unique_ptr<FlowshopSolution> FlowshopBasic::calcObjective(int* seq, size_t
 seqSize)
{
    // Validate input parameters
    validateParams(seq, seqSize);

    // Allocate completion (departure) time matrix and start time matrix
    auto compTimeMatrix = allocTimeMatrix(ptMatrixRows, seqSize);
    startTimeMatrix = allocTimeMatrix(ptMatrixRows, seqSize);

    // Initialize completion time matrix and start time matrix
    initTimeMatrix(compTimeMatrix, seq, ptMatrixRows, seqSize);
    calcStartTimeCol(startTimeMatrix, compTimeMatrix, seq, 0,
ptMatrixRows, seqSize);

    // Calculate all completion and start times
    calcTimeMatrix(compTimeMatrix, seq, ptMatrixRows, seqSize);

    // Construct solution struct
    auto retVal = std::unique_ptr<FlowshopSolution>(new FlowshopSolution(
startTimeMatrix, compTimeMatrix, ptMatrixRows, seq, seqSize,
        getCmax(compTimeMatrix, ptMatrixRows, seqSize),
getTFT(compTimeMatrix, ptMatrixRows, seqSize)));

    // Increment obj func call counter and return result
    funcCallCounter += 1;
    return std::move(retVal);
}

void FlowshopBasic::validateParams(int* seq, size_t seqSize)
{
    // Make sure job sequence is not empty, or too large
    if (seqSize == 0 || seqSize > ptMatrixCols)
    {
        std::string msg = "Error: seqSize cannot be larger than ptMatrixCols";
        throw std::out_of_range(msg);
    }

    // Make sure all jobs in job sequence are within bounds of processing time matrix
    for (size_t i = 0; i < seqSize; i++)
    {
        if (seq[i] <= 0 || seq[i] > ptMatrixCols)
        {
            std::string msg = "Error: seq contains a job number out of range [1, ";
            msg += std::to_string(ptMatrixCols);
            msg += "]";
            throw std::out_of_range(msg);
        }
    }
}

int** FlowshopBasic::allocTimeMatrix(size_t rows, size_t cols)
{
    int** timeMatrix = util::allocMatrix<int>(rows, cols);
    if (timeMatrix == nullptr)
    {
        std::cerr << "Error allocating time matrix." << std::endl;
        throw std::bad_alloc();
    }

    util::initMatrix<int>(timeMatrix, rows, cols, 0);

    return timeMatrix;
}

void FlowshopBasic::initTimeMatrix(int** compTimeMatrix, int* seq, size_t rows
, size_t cols)
{
    // Set first job, first machine
    compTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];

    // Set first job for all machines
    for (size_t r = 1; r < rows; r++)
    {
        compTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + compTimeMatrix[r - 1][0];
    }

    // Set first machine for all jobs
    for (size_t c = 1; c < cols; c++)
    {
        compTimeMatrix[0][c] = compTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
    }
}

void FlowshopBasic::calcTimeMatrix(int** compTimeMatrix, int* seq, size_t rows
, size_t cols)
{
    for (size_t c = 1; c < cols; c++)
    {
        for (size_t r = 1; r < rows; r++)
        {
            int c1 = compTimeMatrix[r - 1][c];
            int c2 = compTimeMatrix[r][c - 1];

            compTimeMatrix[r][c] = max(c1, c2) + procTimeMatrix[r][seq[c] - 1];
        }

        FlowshopBasic::calcStartTimeCol(
startTimeMatrix, compTimeMatrix, seq, c, rows, cols);
    }
}

void FlowshopBasic::calcStartTimeCol(int**
startTimeMatrix, int** departTimeMatrix, int* seq, size_t curCol, size_t rows, size_t cols)
{
    for (size_t r = rows; r > 0; r--)
    {
        startTimeMatrix[r - 1][curCol] = departTimeMatrix[r - 1][curCol] -
procTimeMatrix[r - 1][seq[curCol] - 1];
    }
}

int FlowshopBasic::getCmax(int** compTimeMatrix, size_t rows, size_t cols)
{
    return compTimeMatrix[rows - 1][cols - 1];
}

int FlowshopBasic::getTFT(int** compTimeMatrix, size_t rows, size_t cols)
{
    int sum = 0;

    for (size_t c = 0; c < cols; c++)
    {
        sum += compTimeMatrix[rows - 1][c];
    }

    return sum;
}

// =========================
// End of flowshopbasic.cpp
// =========================
```

7.25 src/flowshopblocking.cpp File Reference

Implementation file for the FlowshopBlocking class.

```
#include "flowshopblocking.h"
```

Include dependency graph for flowshopblocking.cpp:



Functions

- int max (int val1, int val2)

    Simple inline helper function that returns the max of two integers.

7.25.1 Detailed Description

Implementation file for the FlowshopBlocking class.

Author

Andrew Dunn (`Andrew.Dunn@cwu.edu`)

Version

0.1

Date

2019-05-25

Copyright

Copyright (c) 2019

Definition in file flowshopblocking.cpp.

7.25.2 Function Documentation

7.25.2.1 max()

```
int max (
            int val1,
            int val2 )  [inline]
```

Simple inline helper function that returns the max of two integers.

Parameters

val1	First integer
val2	Second integer

Returns

Returns the maximum of the two integers

Definition at line 23 of file flowshopblocking.cpp.

Referenced by fshop::FlowshopBlocking::calcTimeMatrix().

```
00024 {
00025     if (val1 >= val2) return val1;
00026     else return val2;
00027 }
```

7.26 flowshopblocking.cpp

```
00001
00012 #include "flowshopblocking.h"
00013
00014 using namespace fshop;
00015
00023 inline int max(int val1, int val2)
00024 {
00025     if (val1 >= val2) return val1;
00026     else return val2;
00027 }
00028
00034 FlowshopBlocking::FlowshopBlocking(const char* procTimeMatrixFile)
00035     : FlowshopBasic(procTimeMatrixFile)
00036 {
00037 }
00038
00049 void FlowshopBlocking::initTimeMatrix(int** departTimeMatrix, int* seq,
      size_t rows, size_t cols)
00050 {
00051     departTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];
00052
00053     for (size_t r = 1; r < rows; r++)
00054     {
00055         departTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + departTimeMatrix[r - 1][0];
00056     }
00057 }
00058
00068 void FlowshopBlocking::calcTimeMatrix(int** departTimeMatrix, int* seq,
      size_t rows, size_t cols)
```

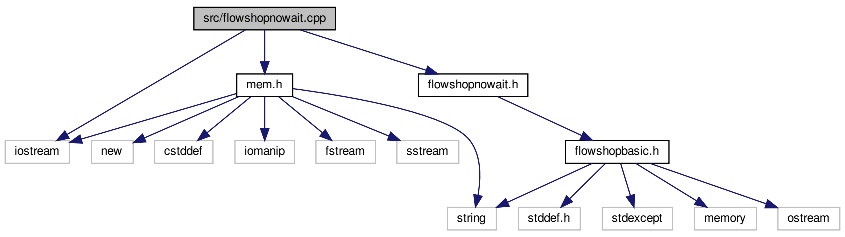
```
{
    for (size_t c = 1; c < cols; c++)
    {
        int d1 = departTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
        int d2 = departTimeMatrix[1][c - 1];

        departTimeMatrix[0][c] = max(d1, d2);

        for (size_t r = 1; r < rows - 1; r++)
        {
            int d1 = departTimeMatrix[r - 1][c] + procTimeMatrix[r][seq[c] - 1];
            int d2 = departTimeMatrix[r + 1][c - 1];

            departTimeMatrix[r][c] = max(d1, d2);
        }

        departTimeMatrix[rows - 1][c] = departTimeMatrix[rows - 2][c] +
procTimeMatrix[rows - 1][seq[c] - 1];

        FlowshopBasic::calcStartTimeCol(
startTimeMatrix, departTimeMatrix, seq, c, rows, cols);
    }
}

// ==========================
// End of flowshopblocking.cpp
// ==========================
```

7.27 src/flowshopnowait.cpp File Reference

Implementation file for the FlowshopNoWait class.

```
#include <iostream>
#include <mem.h>
#include "flowshopnowait.h"
```

Include dependency graph for flowshopnowait.cpp:



Functions

- int max (int val1, int val2)

  Simple inline helper function that returns the max of two integers.

7.27.1 Detailed Description

Implementation file for the FlowshopNoWait class.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-05-26

Copyright

Copyright (c) 2019

Definition in file flowshopnowait.cpp.

7.27.2 Function Documentation

7.27.2.1 max()

```
int max (
            int val1,
            int val2 )  [inline]
```

Simple inline helper function that returns the max of two integers.

Parameters

val1	First integer
val2	Second integer

Returns

Returns the maximum of the two integers

Definition at line 26 of file flowshopnowait.cpp.

```
00027 {
00028     if (val1 >= val2) return val1;
00029     else return val2;
00030 }
```

7.28 flowshopnowait.cpp

```
00001
00012 #include <iostream>
00013 #include <mem.h>
00014 #include "flowshopnowait.h"
00015 #include "mem.h"
00016
```

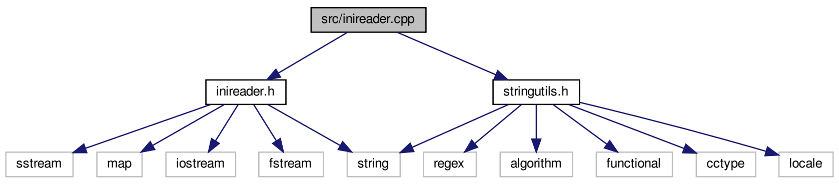
```
using namespace fshop;

inline int max(int val1, int val2)
{
    if (val1 >= val2) return val1;
    else return val2;
}

FlowshopNoWait::FlowshopNoWait(const char* procTimeMatrixFile)
    : FlowshopBasic(procTimeMatrixFile)
{
}

void FlowshopNoWait::initTimeMatrix(int** departTimeMatrix, int* seq, size_t
rows, size_t cols)
{
    departTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];

    for (size_t r = 1; r < rows; r++)
    {
        departTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + departTimeMatrix[r - 1][0];
    }
}

void FlowshopNoWait::calcTimeMatrix(int** departTimeMatrix, int* seq, size_t
rows, size_t cols)
{
    for (size_t c = 1; c < cols; c++)
    {
        departTimeMatrix[0][c] = departTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];

        for (size_t r = 1; r < rows; r++)
        {
            int d1 = departTimeMatrix[r - 1][c];
            int d2 = departTimeMatrix[r][c - 1];

            if (d1 < d2)
            {
                const int diff = d2 - d1;
                for (size_t r2 = r + 1; r2 > 0; r2--)
                    departTimeMatrix[r2 - 1][c] += diff;

                d1 = departTimeMatrix[r - 1][c];
            }

            departTimeMatrix[r][c] = d1 + procTimeMatrix[r][seq[c] - 1];
        }

        FlowshopBasic::calcStartTimeCol(
startTimeMatrix, departTimeMatrix, seq, c, rows, cols);
    }
}

// ========================
// End of flowshopnowait.cpp
// ========================
```

7.29 src/inireader.cpp File Reference

Implementation file for the IniReader class, which can open and parse simple *.ini files.

```
#include "inireader.h"
#include "stringutils.h"
```

Include dependency graph for inireader.cpp:



7.29.1 Detailed Description

Implementation file for the IniReader class, which can open and parse simple ∗.ini files.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file inireader.cpp.

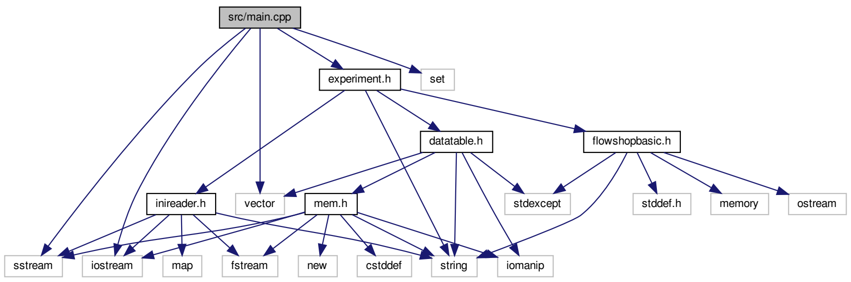
7.30 inireader.cpp

```
#include "inireader.h"
#include "stringutils.h"

using namespace util;

IniReader::IniReader() : file(""), iniMap()
{
}

IniReader::~IniReader()
{
    iniMap.clear();
}

bool IniReader::openFile(std::string filePath)
{
    file = filePath;
    if (!parseFile())
        return false;

    return true;
}

bool IniReader::sectionExists(std::string section)
{
    return iniMap.find(section) != iniMap.end();
}

bool IniReader::entryExists(std::string section, std::string entry)
{
    auto it = iniMap.find(section);
    if (it == iniMap.end()) return false;

    return it->second.find(entry) != it->second.end();
}

std::string IniReader::getEntry(std::string section, std::string entry, std::string
defVal)
{
    if (!entryExists(section, entry)) return defVal;

    return iniMap[section][entry];
}

bool IniReader::parseFile()
{
    iniMap.clear();

    using namespace std;

    ifstream inputF(file, ifstream::in);
    if (!inputF.good()) return false;

    string curSection;
    string line;

    while (getline(inputF, line))
    {
        // Trim whitespace on both ends of the line
        s_trim(line);

        // Ignore empty lines and comments
        if (line.empty() || line.front() == '#')
        {
            continue;
        }
        else if (line.front() == '[' && line.back() == ']')
        {
            // Line is a section definition
            // Erase brackets and trim to get section name
            line.erase(0, 1);
            line.erase(line.length() - 1, 1);
            s_trim(line);
            curSection = line;
        }
        else if (!curSection.empty())
        {
            // Line is an entry, parse the key and value
            parseEntry(curSection, line);
        }
    }

    // Close input file
    inputF.close();
    return true;
}

void IniReader::parseEntry(const std::string& sectionName, const std::string& entry)
{
    using namespace std;

    // Split string around equals sign character
    const string delim = "=";
    string entryName;
    string entryValue;

    // Find index of '='
    auto delimPos = entry.find(delim);

    if (delimPos == string::npos || delimPos >= entry.length() - 1)
        return; // '=' is missing, or is last char in string

    // Extract entry name/key and value
    entryName = entry.substr((size_t)0, delimPos);
    entryValue = entry.substr(delimPos + 1, entry.length());

    // Remove leading and trailing whitespace
    s_trim(entryName);
    s_trim(entryValue);

    // We cannot have entries with empty keys
    if (entryName.empty()) return;

    // Add entry to cache
    iniMap[sectionName][entryName] = entryValue;
}

// =========================
// End of inireader.cpp
// =========================
```

7.31 src/main.cpp File Reference

Program entry point, runs the cs471 project 5 experiment via experiment.h.

```
#include <iostream>
#include <sstream>
#include <vector>
#include <set>
#include "experiment.h"
```
Include dependency graph for main.cpp:



Functions

- int runDebugJobSeq (const char ∗paramsFile, const char ∗seq)
- int main (int argc, char ∗∗argv)

7.31.1 Detailed Description

Program entry point, runs the cs471 project 5 experiment via experiment.h.

Author

Andrew Dunn (`Andrew.Dunn@cwu.edu`)

Version

0.1

Date

2019-05-23

Copyright

Copyright (c) 2019

Definition in file main.cpp.

7.31.2 Function Documentation

7.31.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

Definition at line 23 of file main.cpp.

References runDebugJobSeq(), and cs471::Experiment::runNEH().

```
{
    // Make sure we have enough command line args
    if (argc <= 1)
    {
        cout << "Error: Missing command line parameter." << endl;
        cout << "Proper usage: " << argv[0] << " [param file] \"[Debug Job Sequence]\"" << endl;
        cout << "The debug job sequence is optional, and must be passed in the form \"1 2 3 4 5\" as a
      single argument, where the values are the jobs separated by spaces." << endl;
        return EXIT_FAILURE;
    }

    try
    {
        if (argc > 2)
        {
            return runDebugJobSeq(argv[1], argv[2]);
        }
        else
        {
            // Run experiment and return error code
            cs471::Experiment ex(argv[1]);
            return ex.runNEH();

        }
    }
    catch(const std::exception& e)
    {
        std::cerr << "An exception occurred:" << endl;
        std::cerr << e.what() << endl;
        return 3;
    }

    return 0;
}
```

7.31.2.2 runDebugJobSeq()

```
int runDebugJobSeq (
            const char * paramsFile,
            const char * seq )
```

Definition at line 58 of file main.cpp.

References cs471::Experiment::runDebugSeq().

Referenced by main().

```
{
    string strSeq = seq;
    vector<int> jobSeq;

    stringstream ss(strSeq);
    int val;

    while (ss >> val)
    {
        jobSeq.push_back(val);
    }

    if (jobSeq.size() == 0)
    {
        cerr << "Error: debug job sequence is missing or invalid." << endl;
        return 1;
    }

    set<int> permCheckSet(jobSeq.begin(), jobSeq.end());
    if (permCheckSet.size() != jobSeq.size())
    {
        cerr << "Error: debug job sequence has duplicate jobs in permutation." << endl;
        return 2;
    }

    cout << "Running debug sequence: " << seq << endl;
    cout << "========================================" << endl;

    // Run experiment and return error code
    cs471::Experiment ex(paramsFile);
    return ex.runDebugSeq(&jobSeq[0], jobSeq.size());

    return 0;
}
```

7.32 main.cpp

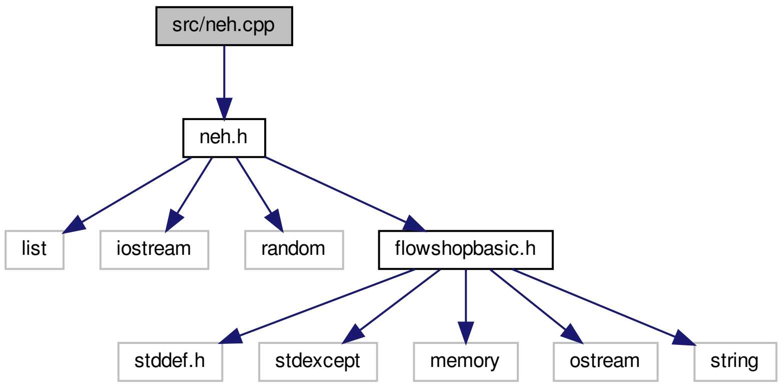
```

#include <iostream>
#include <sstream>
#include <vector>
#include <set>

#include "experiment.h"

using namespace std;

int runDebugJobSeq(const char* paramsFile, const char* seq);

int main(int argc, char** argv)
{
    // Make sure we have enough command line args
    if (argc <= 1)
    {
        cout << "Error: Missing command line parameter." << endl;
        cout << "Proper usage: " << argv[0] << " [param file] \"[Debug Job Sequence]\"" << endl;
        cout << "The debug job sequence is optional, and must be passed in the form \"1 2 3 4 5\" as a
 single argument, where the values are the jobs separated by spaces." << endl;
        return EXIT_FAILURE;
    }

    try
    {
        if (argc > 2)
        {
            return runDebugJobSeq(argv[1], argv[2]);
        }
        else
        {
            // Run experiment and return error code
            cs471::Experiment ex(argv[1]);
            return ex.runNEH();

        }
    }
    catch(const std::exception& e)
    {
        std::cerr << "An exception occurred:" << endl;
        std::cerr << e.what() << endl;
        return 3;
    }

    return 0;
}

int runDebugJobSeq(const char* paramsFile, const char* seq)
{
    string strSeq = seq;
    vector<int> jobSeq;

    stringstream ss(strSeq);
    int val;

    while (ss >> val)
    {
        jobSeq.push_back(val);
    }

    if (jobSeq.size() == 0)
    {
        cerr << "Error: debug job sequence is missing or invalid." << endl;
        return 1;
    }

    set<int> permCheckSet(jobSeq.begin(), jobSeq.end());
    if (permCheckSet.size() != jobSeq.size())
    {
        cerr << "Error: debug job sequence has duplicate jobs in permutation." << endl;
        return 2;
    }

    cout << "Running debug sequence: " << seq << endl;
    cout << "=======================================" << endl;

    // Run experiment and return error code
    cs471::Experiment ex(paramsFile);
    return ex.runDebugSeq(&jobSeq[0], jobSeq.size());

    return 0;
}

// ========================
// End of main.cpp
// ========================
```

7.33 src/neh.cpp File Reference

Implementation file for the NEH class.

`#include "neh.h"`

Include dependency graph for neh.cpp:



Typedefs

- using jtList = std::list< fshop::JobTimePair >
- using jList = std::list< int >

7.33.1 Detailed Description

Implementation file for the NEH class.

Author

Andrew Dunn (`Andrew.Dunn@cwu.edu`)

Version

0.1

Date

2019-05-27

Copyright

Copyright (c) 2019

Definition in file neh.cpp.

7.33.2 Typedef Documentation

7.33.2.1 jList

```
using jList = std::list<int>
```

Definition at line 16 of file neh.cpp.

7.33.2.2 jtList

```
using jtList = std::list<fshop::JobTimePair>
```

Definition at line 15 of file neh.cpp.

7.34 neh.cpp

```cpp

#include "neh.h"

// Type alias
using jtList = std::list<fshop::JobTimePair>;
using jList = std::list<int>;

fshop::NEH::NEH()
    : rd(), randEngine(rd()), randChance(0, 1)
{ }

fsSol fshop::NEH::run(FlowshopBasic* const objectiveFs)
{
    jtList availJobsList;
    makeInitialAvailJobList(objectiveFs, availJobsList);

    auto firstJob = availJobsList.front();
    availJobsList.pop_front();

    fsSol bestSol = nullptr;
    jList* curJobSeq = new jList();
    jList* nextJobSeq = new jList();
    curJobSeq->push_back(firstJob.job);

    while (availJobsList.size() > 0)
    {
        auto nextJob = availJobsList.front();
        availJobsList.pop_front();

        bestSol.reset();
        bestSol = bestPermutation(objectiveFs, *curJobSeq, nextJob.job, *nextJobSeq);

        auto tmp = curJobSeq;
        curJobSeq = nextJobSeq;
        nextJobSeq = tmp;
    }

    delete curJobSeq;
    delete nextJobSeq;

    return std::move(bestSol);
}

void fshop::NEH::makeInitialAvailJobList(FlowshopBasic* const objectiveFs,
jtList& outList)
{
    const size_t numMachines = objectiveFs->getTotalMachines();
    const size_t numJobs = objectiveFs->getTotalJobs();

    for (size_t j = 1; j <= numJobs; j++)
    {
        int sum = 0;
        for (size_t m = 1; m <= numMachines; m++)
            sum += objectiveFs->getProcessingTime(m, j);

        outList.emplace_back(
            JobTimePair(j, sum)
        );
    }

    // Sort in decreasing order of time
    outList.sort([](JobTimePair& lhs, JobTimePair& rhs) { return lhs.
time >= rhs.time; });
}

fsSol fshop::NEH::bestPermutation(FlowshopBasic* const objectiveFs, const
jList& baseList, int jobInsert, jList& outBestSeq)
{
    fsSol bestSol = nullptr;
    outBestSeq.clear();

    jList bufferList;
    int* seqArr = new int[baseList.size() + 1];

    for (size_t i = 0; i <= baseList.size(); i++)
    {
        bufferList = jList(baseList.begin(), baseList.end());
        auto it = bufferList.begin();
        std::advance(it, i);

        bufferList.insert(it, jobInsert);

        size_t index = 0;

        for (auto i = bufferList.begin(); i != bufferList.end(); i++)
        {
            seqArr[index] = *i;
            index++;
        }

        auto result = objectiveFs->calcObjective(seqArr, bufferList.size());
        if (bestSol == nullptr || result->cmax < bestSol->cmax ||
            (result->cmax == bestSol->cmax && randChance(randEngine) >= 0.5))
        {
            bestSol.reset();
            bestSol = std::move(result);
            outBestSeq = jList(bufferList.begin(), bufferList.end());
        }
    }

    delete[] seqArr;
    return std::move(bestSol);
}

// =========================
// End of neh.cpp
// =========================
```

Index