

CS471 Project 5

Generated by Doxygen 1.8.13

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Namespace Documentation	9
5.1	cs471 Namespace Reference	9
5.2	fshop Namespace Reference	9
5.3	mdata Namespace Reference	10
5.4	util Namespace Reference	10
5.4.1	Function Documentation	10
5.4.1.1	allocArray()	10
5.4.1.2	allocMatrix()	11
5.4.1.3	copyArray()	12
5.4.1.4	initArray()	12
5.4.1.5	initMatrix()	13
5.4.1.6	loadMatrixFromFile()	14
5.4.1.7	outputMatrix()	15
5.4.1.8	releaseArray()	15
5.4.1.9	releaseMatrix()	16

6	Class Documentation	19
6.1	mdata::DataTable< T > Class Template Reference	19
6.1.1	Detailed Description	19
6.1.2	Constructor & Destructor Documentation	20
6.1.2.1	DataTable()	20
6.1.2.2	~DataTable()	21
6.1.3	Member Function Documentation	21
6.1.3.1	clearData()	21
6.1.3.2	exportCSV()	21
6.1.3.3	getColLabel()	22
6.1.3.4	getEntry()	23
6.1.3.5	setColLabel()	23
6.1.3.6	setEntry()	24
6.2	cs471::Experiment Class Reference	24
6.2.1	Detailed Description	25
6.2.2	Constructor & Destructor Documentation	25
6.2.2.1	Experiment()	25
6.2.2.2	~Experiment()	26
6.2.3	Member Function Documentation	26
6.2.3.1	runDebugSeq()	26
6.2.3.2	runNEH()	27
6.3	fshop::FlowshopBasic Class Reference	28
6.3.1	Detailed Description	30
6.3.2	Constructor & Destructor Documentation	30
6.3.2.1	FlowshopBasic() [1/3]	30
6.3.2.2	~FlowshopBasic()	30
6.3.2.3	FlowshopBasic() [2/3]	31
6.3.2.4	FlowshopBasic() [3/3]	31
6.3.3	Member Function Documentation	31
6.3.3.1	allocTimeMatrix()	31

6.3.3.2	calcObjective()	32
6.3.3.3	calcStartTimeCol()	32
6.3.3.4	calcTimeMatrix()	33
6.3.3.5	getCmax()	34
6.3.3.6	getFuncCallCounts()	34
6.3.3.7	getProcessingTime()	35
6.3.3.8	getTFT()	36
6.3.3.9	getTotalJobs()	37
6.3.3.10	getTotalMachines()	38
6.3.3.11	initTimeMatrix()	38
6.3.3.12	operator=() [1/2]	39
6.3.3.13	operator=() [2/2]	39
6.3.3.14	validateParams()	39
6.3.4	Member Data Documentation	40
6.3.4.1	funcCallCounter	40
6.3.4.2	procTimeMatrix	40
6.3.4.3	ptMatrixCols	40
6.3.4.4	ptMatrixRows	41
6.3.4.5	startTimeMatrix	41
6.4	fshop::FlowshopBlocking Class Reference	41
6.4.1	Detailed Description	42
6.4.2	Constructor & Destructor Documentation	42
6.4.2.1	FlowshopBlocking()	42
6.4.2.2	~FlowshopBlocking()	43
6.4.3	Member Function Documentation	43
6.4.3.1	calcTimeMatrix()	43
6.4.3.2	initTimeMatrix()	44
6.5	fshop::FlowshopNoWait Class Reference	45
6.5.1	Detailed Description	46
6.5.2	Constructor & Destructor Documentation	46

6.5.2.1	FlowshopNoWait()	46
6.5.2.2	~FlowshopNoWait()	46
6.5.3	Member Function Documentation	46
6.5.3.1	calcTimeMatrix()	46
6.5.3.2	initTimeMatrix()	47
6.6	fshop::FlowshopSolution Struct Reference	48
6.6.1	Detailed Description	49
6.6.2	Constructor & Destructor Documentation	49
6.6.2.1	FlowshopSolution() [1/3]	49
6.6.2.2	~FlowshopSolution()	50
6.6.2.3	FlowshopSolution() [2/3]	50
6.6.2.4	FlowshopSolution() [3/3]	51
6.6.3	Member Function Documentation	51
6.6.3.1	getDepartTimeMatrix()	51
6.6.3.2	getJobSeq()	51
6.6.3.3	getJobSeqAsString()	52
6.6.3.4	getStartTimeMatrix()	52
6.6.3.5	operator=() [1/2]	52
6.6.3.6	operator=() [2/2]	52
6.6.3.7	outputAll()	52
6.6.3.8	outputTimesCsv()	53
6.6.4	Member Data Documentation	54
6.6.4.1	cmax	54
6.6.4.2	numMachines	54
6.6.4.3	seqSize	55
6.6.4.4	totalFlowTime	55
6.7	util::IniReader Class Reference	55
6.7.1	Detailed Description	56
6.7.2	Constructor & Destructor Documentation	56
6.7.2.1	IniReader()	56

6.7.2.2	~IniReader()	56
6.7.3	Member Function Documentation	56
6.7.3.1	entryExists()	56
6.7.3.2	getEntry()	57
6.7.3.3	getEntryAs()	58
6.7.3.4	openFile()	58
6.7.3.5	sectionExists()	58
6.8	fshop::JobTimePair Struct Reference	59
6.8.1	Detailed Description	59
6.8.2	Constructor & Destructor Documentation	59
6.8.2.1	JobTimePair()	60
6.8.3	Member Data Documentation	60
6.8.3.1	job	60
6.8.3.2	time	60
6.9	fshop::NEH Class Reference	60
6.9.1	Detailed Description	61
6.9.2	Constructor & Destructor Documentation	61
6.9.2.1	NEH()	61
6.9.3	Member Function Documentation	61
6.9.3.1	run()	61
6.10	cs471::TestParams Struct Reference	62
6.10.1	Detailed Description	62
6.10.2	Member Data Documentation	63
6.10.2.1	algorithm	63
6.10.2.2	inputFilesDir	63
6.10.2.3	maxTestFile	63
6.10.2.4	minTestFile	63
6.10.2.5	numThreads	63
6.10.2.6	resultsFile	64
6.10.2.7	timesFile	64
6.11	ThreadPool Class Reference	64
6.11.1	Detailed Description	64
6.11.2	Constructor & Destructor Documentation	65
6.11.2.1	ThreadPool()	65
6.11.2.2	~ThreadPool()	65
6.11.3	Member Function Documentation	65
6.11.3.1	enqueue()	66
6.11.3.2	stopAndJoinAll()	66

7 File Documentation	67
7.1 include/datatable.h File Reference	67
7.1.1 Detailed Description	68
7.2 datatable.h	69
7.3 include/experiment.h File Reference	70
7.3.1 Detailed Description	71
7.4 experiment.h	72
7.5 include/flowshopbasic.h File Reference	72
7.5.1 Detailed Description	73
7.6 flowshopbasic.h	74
7.7 include/flowshopblocking.h File Reference	75
7.7.1 Detailed Description	76
7.8 flowshopblocking.h	76
7.9 include/flowshopnowait.h File Reference	77
7.9.1 Detailed Description	78
7.10 flowshopnowait.h	78
7.11 include/inireader.h File Reference	79
7.11.1 Detailed Description	80
7.12 inireader.h	80
7.13 include/mem.h File Reference	81
7.13.1 Detailed Description	82
7.14 mem.h	83
7.15 include/neh.h File Reference	85
7.15.1 Detailed Description	86
7.15.2 Typedef Documentation	87
7.15.2.1 fsSol	87
7.16 neh.h	87
7.17 include/stringutils.h File Reference	87
7.17.1 Detailed Description	88
7.18 stringutils.h	89

7.19	include/threadpool.h File Reference	89
7.20	threadpool.h	90
7.21	src/experiment.cpp File Reference	92
7.21.1	Detailed Description	92
7.21.2	Macro Definition Documentation	93
7.21.2.1	INI_TEST_ALGORITHM	93
7.21.2.2	INI_TEST_INPUTFILEDIR	93
7.21.2.3	INI_TEST_MAXFILE	93
7.21.2.4	INI_TEST_MINFILE	93
7.21.2.5	INI_TEST_NUMTHREADS	93
7.21.2.6	INI_TEST_RESULTSFILE	94
7.21.2.7	INI_TEST_SECTION	94
7.21.2.8	INI_TEST_TIMESFILE	94
7.22	experiment.cpp	94
7.23	src/flowshopbasic.cpp File Reference	98
7.23.1	Detailed Description	98
7.23.2	Function Documentation	99
7.23.2.1	max()	99
7.24	flowshopbasic.cpp	99
7.25	src/flowshopblocking.cpp File Reference	103
7.25.1	Detailed Description	104
7.25.2	Function Documentation	105
7.25.2.1	max()	105
7.26	flowshopblocking.cpp	105
7.27	src/flowshopnowait.cpp File Reference	106
7.27.1	Detailed Description	106
7.27.2	Function Documentation	107
7.27.2.1	max()	107
7.28	flowshopnowait.cpp	107
7.29	src/inireader.cpp File Reference	108

7.29.1 Detailed Description	109
7.30 inireader.cpp	109
7.31 src/main.cpp File Reference	110
7.31.1 Detailed Description	111
7.31.2 Function Documentation	111
7.31.2.1 main()	112
7.31.2.2 runDebugJobSeq()	112
7.32 main.cpp	113
7.33 src/neh.cpp File Reference	114
7.33.1 Detailed Description	115
7.33.2 Typedef Documentation	115
7.33.2.1 jList	115
7.33.2.2 jtList	115
7.34 neh.cpp	116
Index	119

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

cs471	9
fshop	9
mdata	10
util	10

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

mdata::DataTable< T >	19
cs471::Experiment	24
fshop::FlowshopBasic	28
fshop::FlowshopBlocking	41
fshop::FlowshopNoWait	45
fshop::FlowshopSolution	48
util::IniReader	55
fshop::JobTimePair	59
fshop::NEH	60
cs471::TestParams	62
ThreadPool	64

Chapter 3

Class Index

3.1 Class List

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

mdata::DataTable< T >	
The DataTable class is a simple table of values with labeled columns	19
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	24
fshop::FlowshopBasic	
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	28
fshop::FlowshopBlocking	
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	41
fshop::FlowshopNoWait	
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	45
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	48
util::IniReader	
Simple *.ini file reader and parser	55
fshop::JobTimePair	
Simple struct that pairs a job with it's total processing time. Used for sorting purposes	59
fshop::NEH	
Runs the NEH algorithm on the given flowshop objective function and attempts to optimize the job sequence that produces the smallest cmax value	60
cs471::TestParams	
Simple data structure that stores the test parameters for the experiment	62
ThreadPool	64

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

include/ datatable.h	Header file for the DataTable class, which represents a spreadsheet/table of values that can easily be exported to a *.csv file	67
include/ experiment.h	Contains the Experiment class which runs the cs471 lab 5 experiment	70
include/ flowshopbasic.h	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	72
include/ flowshopblocking.h	Contains the FlowshopBlocking class, which inherits FlowshopBasic and solves a flowshop with blocking problem for a specific job sequence	75
include/ flowshopnowait.h	Contains the FlowshopNoWait class, which inherits FlowshopBasic and solves a flowshop with no waiting problem for a specific job sequence	77
include/ inireader.h	Header file for the IniReader class, which can open and parse simple *.ini files	79
include/ mem.h	Header file for various memory utility functions	81
include/ neh.h	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	85
include/ stringutils.h	Contains various string manipulation helper functions	87
include/ threadpool.h	89
src/ experiment.cpp	Implementation file for the Experiment class	92
src/ flowshopbasic.cpp	Implementation file for the FlowshopBasic class	98
src/ flowshopblocking.cpp	Implementation file for the FlowshopBlocking class	103
src/ flowshopnowait.cpp	Implementation file for the FlowshopNoWait class	106
src/ inireader.cpp	Implementation file for the IniReader class, which can open and parse simple *.ini files	108

src/ main.cpp	
Program entry point, runs the cs471 project 5 experiment via experiment.h	110
src/ neh.cpp	
Implementation file for the NEH class	114

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 [FlowshopBasic](#).
- 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

<i>Data</i>	type of the array
-------------	-------------------

Parameters

<i>size</i>	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](#).

```
00122     {
00123         return new(std::nothrow) T[size];
00124     }
```

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

<i>Data</i>	type of the matrix entries
-------------	----------------------------

Parameters

<i>rows</i>	The number of rows
<i>cols</i>	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](#).

```
00136     {
00137         T** m = (T**)allocArray<T*>(rows);
00138         if (m == nullptr) return nullptr;
00139         for (size_t i = 0; i < rows; i++)
00140             {
00141                 }
```

```

00142         m[i] = allocArray<T>(cols);
00143         if (m[i] == nullptr)
00144         {
00145             releaseMatrix<T>(m, rows);
00146             return nullptr;
00147         }
00148     }
00149
00150     return m;
00151 }

```

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

<i>Data</i>	type of the array
-------------	-------------------

Parameters

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

Definition at line 255 of file [mem.h](#).

```

00256     {
00257         for (size_t i = 0; i < size; i++)
00258             dest[i] = src[i];
00259     }

```

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

<i>Data</i>	type of array
-------------	---------------

Parameters

<i>a</i>	Pointer to array
<i>size</i>	Size of the array
<i>val</i>	Value to initialize the array to

Definition at line 34 of file [mem.h](#).

Referenced by [initMatrix\(\)](#).

```

00035     {
00036         if (a == nullptr) return;
00037
00038         for (size_t i = 0; i < size; i++)
00039         {
00040             a[i] = val;
00041         }
00042     }

```

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

<i>Data</i>	type of matrix entries
-------------	------------------------

Parameters

<i>m</i>	Pointer to a matrix
<i>rows</i>	Number of rows in matrix
<i>cols</i>	Number of columns in matrix
<i>val</i>	Value to initialize the matrix to

Definition at line 54 of file [mem.h](#).

References [initArray\(\)](#).

```

00055     {
00056         if (m == nullptr) return;
00057
00058         for (size_t i = 0; i < rows; i++)
00059         {
00060             initArray(m[i], cols, val);
00061         }
00062     }

```

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](#).

```

00155     {
00156         outNumRows = 0;
00157         outNumCols = 0;
00158
00159         std::ifstream is(filePath);
00160         if (!is.good())
00161         {
00162             std::cerr << "Error loading matrix from file: Unable to open file." << std::endl;
00163             return nullptr;
00164         }
00165
00166         std::string line;
00167         if (!std::getline(is, line))
00168         {
00169             std::cerr << "Error loading matrix from file: File is empty or invalid." << std::endl;
00170             is.close();
00171             return nullptr;
00172         }
00173
00174         size_t rows = 0;
00175         size_t cols = 0;
00176
00177         std::stringstream ss(line);
00178         if (!(ss >> rows >> cols) || rows == 0 || cols == 0)
00179         {
00180             std::cerr << "Error loading matrix from file: Row or column size is zero." << std::endl;
00181             is.close();
00182             return nullptr;
00183         }
00184
00185         T** retMatrix = allocMatrix<T>(rows, cols);
00186         if (retMatrix == nullptr)
00187         {
00188             std::cerr << "Error loading matrix from file: Matrix memory allocation failed." << std::endl;
00189             is.close();
00190             return nullptr;
00191         }
00192
00193         for (size_t r = 0; r < rows; r++)
00194         {
00195             if (!std::getline(is, line))
00196             {
00197                 std::cerr << "Error loading matrix from file: EOF reached before reading all rows." <<
00198                 std::endl;
00199                 releaseMatrix<T>(retMatrix, rows);
00200                 is.close();
00201                 return nullptr;
00202             }
00203
00204             std::stringstream ss(line);
00205
00206             for (size_t c = 0; c < cols; c++)
00207             {
00208                 T entry = 0;

```



```

00209             if (!(ss >> entry))
00210             {
00211                 std::cerr << "Error loading matrix from file: EOL reached before reading all cols." <<
std::endl;
00212                 releaseMatrix<T>(retMatrix, rows);
00213                 is.close();
00214                 return nullptr;
00215             }
00216
00217             retMatrix[r][c] = entry;
00218         }
00219     }
00220
00221     is.close();
00222     outNumRows = rows;
00223     outNumCols = cols;
00224     return retMatrix;
00225 }

```

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\(\)](#).

```

00229     {
00230         if (matrix == nullptr)
00231             return;
00232
00233         for (size_t r = 0; r < rows; r++)
00234         {
00235             for (size_t c = 0; c < cols; c++)
00236             {
00237                 os << std::setw(3) << matrix[r][c];
00238                 if (c < cols - 1)
00239                     os << " ";
00240                 else
00241                     os << std::endl;
00242             }
00243         }
00244     }

```

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

<i>Data</i>	type of array
-------------	---------------

Parameters

<i>a</i>	Pointer to array
----------	------------------

Definition at line 71 of file [mem.h](#).

```

00072     {
00073         if (a == nullptr) return true;
00074
00075         try
00076         {
00077             delete[] a;
00078             a = nullptr;
00079             return true;
00080         }
00081         catch (...)
00082         {
00083             return false;
00084         }
00085     }

```

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

<i>Data</i>	type of the matrix
-------------	--------------------

Parameters

<i>m</i>	Pointer th the matrix
<i>rows</i>	The number of rows in the matrix

Definition at line 95 of file [mem.h](#).

Referenced by [mdata::DataTable< T >::~~DataTable\(\)](#).

```

00096     {
00097         if (m == nullptr) return;
00098
00099         for (size_t i = 0; i < rows; i++)
00100         {
00101             if (m[i] != nullptr)

```

```
00102         {
00103             // Release each row
00104             releaseArray<T>(m[i]);
00105         }
00106     }
00107
00108     // Release columns
00109     delete[] m;
00110     m = nullptr;
00111 }
```


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

<code>_rows</code>	Number of rows in table
<code>_cols</code>	Number of columns in table

Definition at line 60 of file [datatable.h](#).

```
00060                                     : rows(_rows), cols(_cols), dataMatrix(nullptr)
00061     {
00062         if (rows == 0)
00063             throw std::length_error("Table rows must be greater than 0.");
00064         else if (cols == 0)
00065             throw std::length_error("Table columns must be greater than 0.");
00066
00067         dataMatrix = util::allocMatrix<T>(rows, cols);
00068         if (dataMatrix == nullptr)
00069             throw std::bad_alloc();
00070
00071         colLabels.resize(_cols, std::string());
00072     }
```

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\(\)](#).

```
00078         {
00079             util::releaseMatrix(dataMatrix, rows);
00080         }
```

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](#).

```
00083         {
00084             util::initMatrix<T>(dataMatrix, rows, cols, 0);
00085         }
```

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

<i>filePath</i>	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\(\)](#).

```

00161     {
00162         if (dataMatrix == nullptr) return false;
00163
00164         using namespace std;
00165         ofstream outFile;
00166         outFile.open(filePath, ofstream::out | ofstream::trunc);
00167         if (!outFile.good()) return false;
00168
00169         // Print column labels
00170         for (unsigned int c = 0; c < cols; c++)
00171         {
00172             outFile << colLabels[c];
00173             if (c < cols - 1) outFile << ",";
00174         }
00175
00176         outFile << endl;
00177
00178         // Print data rows
00179         for (unsigned int r = 0; r < rows; r++)
00180         {
00181             for (unsigned int c = 0; c < cols; c++)
00182             {
00183                 outFile << std::setprecision(8) << dataMatrix[r][c];
00184                 if (c < cols - 1) outFile << ",";
00185             }
00186             outFile << endl;
00187         }
00188
00189         outFile.close();
00190         return true;
00191     }

```

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

<i>colIndex</i>	Index of the column
-----------------	---------------------

Returns

std::string String value of the column label

Definition at line 93 of file [datatable.h](#).

```

00094     {
00095         if (colIndex >= colLabels.size())
00096             throw std::out_of_range("Column index out of range");
00097
00098         return colLabels[colIndex];
00099     }

```


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

<i>row</i>	Row index of the table
<i>col</i>	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](#).

```
00123     {
00124         if (dataMatrix == nullptr)
00125             throw std::runtime_error("Data matrix not allocated");
00126         if (row >= rows)
00127             throw std::out_of_range("Table row out of range");
00128         else if (col >= cols)
00129             throw std::out_of_range("Table column out of range");
00130
00131         return dataMatrix[row][col];
00132     }
```

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

<i>colIndex</i>	Index of the column
<i>newLabel</i>	New string label for the column

Definition at line 107 of file [datatable.h](#).

Referenced by [cs471::Experiment::runNEH\(\)](#).

```
00108     {
00109         if (colIndex >= colLabels.size())
```

```

00110             throw std::out_of_range("Column index out of range");
00111
00112         colLabels[colIndex] = newLabel;
00113     }

```

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

<i>row</i>	Row index of the table
<i>col</i>	Column index of the table
<i>val</i>	New value for the entry

Definition at line 141 of file [datatable.h](#).

Referenced by [cs471::Experiment::runNEH\(\)](#).

```

00142     {
00143         if (dataMatrix == nullptr)
00144             throw std::runtime_error("Data matrix not allocated");
00145         if (row >= rows)
00146             throw std::out_of_range("Table row out of range");
00147         else if (col >= cols)
00148             throw std::out_of_range("Table column out of range");
00149
00150         dataMatrix[row][col] = val;
00151     }

```

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

<i>paramsFile</i>	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

<i>seq</i>	Job sequence to run flowshop objective functions with
<i>seqSize</i>	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)
```

```

00340     {
00341         cout << "Objective flowshop function encountered an error." << endl;
00342         return 1;
00343     }
00344
00345     result = objectiveFs->calcObjective(seq, seqSize);
00346     result->outputAll(std::cout);
00347
00348     delete objectiveFs;
00349
00350     cout << "===== " << endl;
00351 }
00352
00353 cout << "Debug objective function sequence tests completed." << endl;
00354
00355 return 0;
00356 }

```

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 >::setEntry\(\)](#), [ThreadPool::stopAndJoinAll\(\)](#), and [cs471::TestParams::timesFile](#).

Referenced by [main\(\)](#).

```

00065 {
00066     // Retrieve test parameters from ini file
00067     TestParams p = readTestParams();
00068
00069     // Construct data table to store experiment results
00070     mdata::DataTable<string> resultsTable(p.maxTestFile - p.
minTestFile + 1, 6);
00071
00072     // Initialize thread pool with a parameter-given number of threads
00073     ThreadPool tpool(p.numThreads);
00074
00075     // Initialize thread future vector, used for thread pool synchronization
00076     // and keeps track of the individual tasks being executed.
00077     vector<std::future<int>> futures;
00078
00079     cout << "Started " << p.numThreads << " worker threads ..." << endl;
00080
00081     if (p.algorithm == 1)
00082         cout << "Running NEH on Flow Shop with Blocking ..." << endl;
00083     else if (p.algorithm == 2)
00084         cout << "Running NEH on Flow Shop with No Wait ..." << endl;
00085     else
00086         cout << "Running NEH on Flow Shop Scheduling ..." << endl;
00087
00088     // Prepare results table column header labels
00089     resultsTable.setColLabel(0, "Data Set");
00090     resultsTable.setColLabel(1, "cMax");

```

```

00091     resultsTable.setColLabel(2, "TFT");
00092     resultsTable.setColLabel(3, "Func Calls");
00093     resultsTable.setColLabel(4, "Execution Time (ms)");
00094     resultsTable.setColLabel(5, "Sequence");
00095
00096     // Add all input test files as tasks in thread pool
00097     for (int i = p.minTestFile; i <= p.maxTestFile; i++)
00098     {
00099         string inputFile = std::to_string(i) + ".txt";
00100         futures.emplace_back(
00101             tpool.enqueue(&cs471::Experiment::runNEHThreaded, this, &p, inputFile, i, &resultsTable)
00102         );
00103     }
00104
00105     // const size_t totalFutures = futures.size();
00106
00107     // Join all thread pool tasks using futures vector
00108     // and get the return value for each
00109     for (int i = 0; i < futures.size(); i++)
00110     {
00111         int err = futures[i].get();
00112         if (err)
00113         {
00114             // Threaded task returned with an error code, bail
00115             tpool.stopAndJoinAll();
00116             return err;
00117         }
00118     }
00119
00120     // Output results table to a csv file
00121     if (!p.resultsFile.empty())
00122     {
00123         resultsTable.exportCSV(p.resultsFile.c_str());
00124         cout << "Results exported to: " << p.resultsFile << endl;
00125     }
00126
00127     return 0;
00128 }

```

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

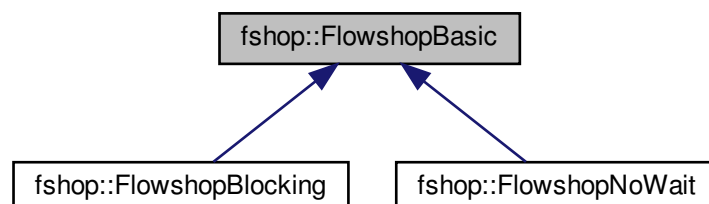
- [include/experiment.h](#)
- [src/experiment.cpp](#)

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

<i>procTimeMatrixFile</i>	File path to the file containing the job processing times matrix
---------------------------	--

Definition at line 235 of file [flowshopbasic.cpp](#).

References [procTimeMatrix](#), [ptMatrixCols](#), and [ptMatrixRows](#).

```
00236     : startTimeMatrix(nullptr), ptMatrixRows(0),
    ptMatrixCols(0), funcCallCounter(0)
00237 {
00238     // Attempt to load job processing times from the given file
00239     procTimeMatrix = util::loadMatrixFromFile<int>(procTimeMatrixFile,
    ptMatrixRows, ptMatrixCols);
00240     if (procTimeMatrix == nullptr)
00241     {
00242         std::string msg = "Error when loading matrix file: ";
00243         msg += procTimeMatrixFile;
00244         throw std::runtime_error(msg);
00245     }
00246 }
```

6.3.2.2 ~FlowshopBasic()

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

Destroys the [FlowshopBasic](#) object.

Definition at line 252 of file [flowshopbasic.cpp](#).

References [procTimeMatrix](#), and [ptMatrixRows](#).

```
00253 {
00254     util::releaseMatrix<int>(procTimeMatrix, ptMatrixRows);
00255 }
```


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

<i>rows</i>	Number of rows (machines)
<i>cols</i>	Number of columns (jobs)

Returns

Returns a pointer to the newly created matrix

Definition at line 378 of file [flowshopbasic.cpp](#).

Referenced by [calcObjective\(\)](#).

```
00379 {
00380     int** timeMatrix = util::allocMatrix<int>(rows, cols);
00381     if (timeMatrix == nullptr)
00382     {
00383         std::cerr << "Error allocating time matrix." << std::endl;
00384         throw std::bad_alloc();
00385     }
00386     util::initMatrix<int>(timeMatrix, rows, cols, 0);
00387     return timeMatrix;
00388 }
00389
00390 }
```

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

<i>seq</i>	Pointer to an int array containing the job sequence permutation
<i>seqSize</i>	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\(\)](#), [initTimeMatrix\(\)](#), [ptMatrixRows](#), [startTimeMatrix](#), and [validateParams\(\)](#).

Referenced by [fshop::NEH::run\(\)](#).

```
00319 {
00320     // Validate input parameters
00321     validateParams(seq, seqSize);
00322
00323     // Allocate completion (departure) time matrix and start time matrix
00324     auto compTimeMatrix = allocTimeMatrix(ptMatrixRows, seqSize);
00325     startTimeMatrix = allocTimeMatrix(ptMatrixRows, seqSize);
00326
00327     // Initialize completion time matrix and start time matrix
00328     initTimeMatrix(compTimeMatrix, seq, ptMatrixRows, seqSize);
00329     calcStartTimeCol(startTimeMatrix, compTimeMatrix, seq, 0,
00330                     ptMatrixRows, seqSize);
00330
00331     // Calculate all completion and start times
00332     calcTimeMatrix(compTimeMatrix, seq, ptMatrixRows, seqSize);
00333
00334     // Construct solution struct
00335     auto retVal = std::unique_ptr<FlowshopSolution>(new FlowshopSolution(
00336         startTimeMatrix, compTimeMatrix, ptMatrixRows, seq, seqSize,
00337         getCmax(compTimeMatrix, ptMatrixRows, seqSize),
00338         getTFT(compTimeMatrix, ptMatrixRows, seqSize)));
00337
00338     // Increment obj func call counter and return result
00339     funcCallCounter += 1;
00340     return std::move(retVal);
00341 }
```

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

<i>startTimeMatrix</i>	Pointer to start times matrix
<i>departTimeMatrix</i>	Pointer to departure (completion) times matrix
<i>seq</i>	Pointer to job sequence
<i>curCol</i>	Index of the column to be calculated
<i>rows</i>	Number of rows (machines) in the completion time matrix
<i>cols</i>	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::calcTimeMatrix\(\)](#), and [calcTimeMatrix\(\)](#).

```

00454 {
00455     for (size_t r = rows; r > 0; r--)
00456     {
00457         startTimeMatrix[r - 1][curCol] = departTimeMatrix[r - 1][curCol] -
00458         procTimeMatrix[r - 1][seq[curCol] - 1];
00459     }
00459 }
```

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

<i>compTimeMatrix</i>	Pointer to completion time matrix
<i>seq</i>	Pointer to job sequence
<i>rows</i>	Number of rows (machines) in the completion time matrix
<i>cols</i>	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\(\)](#).

```

00428 {
00429     for (size_t c = 1; c < cols; c++)
00430     {
00431         for (size_t r = 1; r < rows; r++)
00432         {
00433             int c1 = compTimeMatrix[r - 1][c];
00434             int c2 = compTimeMatrix[r][c - 1];
00435
00436             compTimeMatrix[r][c] = max(c1, c2) + procTimeMatrix[r][seq[c] - 1];
00437         }
00438
00439         FlowshopBasic::calcStartTimeCol(
00440             startTimeMatrix, compTimeMatrix, seq, c, rows, cols);
00441     }
00442 }

```

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

<i>compTimeMatrix</i>	Pointer to the completion time matrix
<i>rows</i>	Number of rows (machines) in the completion time matrix
<i>cols</i>	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\(\)](#).

```

00470 {
00471     return compTimeMatrix[rows - 1][cols - 1];
00472 }

```

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](#).

```
00306 {
00307     return funcCallCounter;
00308 }
```

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

<i>machine</i>	Number of machine [1-n]
<i>job</i>	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\(\)](#).

```
00265 {
00266     if (machine == 0 || job == 0)
00267     {
00268         std::string msg = "Error: Machine or job number cannot be zero";
00269         throw std::out_of_range(msg);
00270     }
00271     else if (machine > ptMatrixRows || job > ptMatrixCols)
00272     {
00273         std::string msg = "Error: Machine or job number out of range";
00274         throw std::out_of_range(msg);
00275     }
00276     return procTimeMatrix[machine - 1][job - 1];
00277 }
00278 }
```

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

<i>compTimeMatrix</i>	Pointer to the completion time matrix
<i>rows</i>	Number of rows (machines) in the completion time matrix
<i>cols</i>	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\(\)](#).

```

00483 {
00484     int sum = 0;
00485
00486     for (size_t c = 0; c < cols; c++)
00487     {
00488         sum += compTimeMatrix[rows - 1][c];
00489     }
00490
00491     return sum;
00492 }
```

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\(\)](#).

```

00286 {
00287     return ptMatrixCols;
00288 }
```

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\(\)](#).

```
00296 {
00297     return ptMatrixRows;
00298 }
```

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

<i>compTimeMatrix</i>	Pointer to completion time matrix
<i>seq</i>	Pointer to job sequence
<i>rows</i>	Number of rows (machines) in the completion time matrix
<i>cols</i>	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\(\)](#).

```
00402 {
00403     // Set first job, first machine
00404     compTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];
00405
00406     // Set first job for all machines
```



```

00407     for (size_t r = 1; r < rows; r++)
00408     {
00409         compTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + compTimeMatrix[r - 1][0];
00410     }
00411
00412     // Set first machine for all jobs
00413     for (size_t c = 1; c < cols; c++)
00414     {
00415         compTimeMatrix[0][c] = compTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
00416     }
00417 }

```

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

<i>seq</i>	Job permutation sequence array
<i>seqSize</i>	Size of job sequence array

Definition at line 349 of file [flowshopbasic.cpp](#).

References [ptMatrixCols](#).

Referenced by [calcObjective\(\)](#).

```

00350 {
00351     // Make sure job sequence is not empty, or too large
00352     if (seqSize == 0 || seqSize > ptMatrixCols)
00353     {
00354         std::string msg = "Error: seqSize cannot be larger than ptMatrixCols";

```

```

00355         throw std::out_of_range(msg);
00356     }
00357
00358     // Make sure all jobs in job sequence are within bounds of processing time matrix
00359     for (size_t i = 0; i < seqSize; i++)
00360     {
00361         if (seq[i] <= 0 || seq[i] > ptMatrixCols)
00362         {
00363             std::string msg = "Error: seq contains a job number out of range [1, ";
00364             msg += std::to_string(ptMatrixCols);
00365             msg += "]\n";
00366             throw std::out_of_range(msg);
00367         }
00368     }
00369 }

```

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::calcTimeMatrix\(\)](#), [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 [~FlowshopBasic\(\)](#).

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::calcTimeMatrix\(\)](#), and [calcTimeMatrix\(\)](#).

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

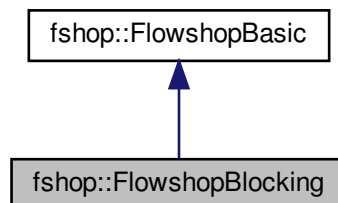
- [include/flowshopbasic.h](#)
- [src/flowshopbasic.cpp](#)

6.4 fshop::FlowshopBlocking Class Reference

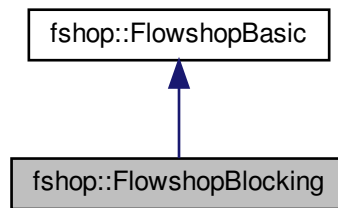
The [FlowshopBlocking](#) class runs the flowshop with blocking problem for a given job-machine processing time matrix that is read from a file. The `run()` method takes the specific job sequence being calculated. Inherits from [FlowshopBasic](#).

```
#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

<i>procTimeMatrixFile</i>	File path to the file containing the job processing times matrix
---------------------------	--

Definition at line 34 of file [flowshopblocking.cpp](#).

```
00035     : FlowshopBasic(procTimeMatrixFile)
00036 {
00037 }
```

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

<i>compTimeMatrix</i>	Pointer to completion time matrix
<i>seq</i>	Pointer to job sequence
<i>rows</i>	Number of rows (machines) in the completion time matrix
<i>cols</i>	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](#).

```
00069 {
00070     for (size_t c = 1; c < cols; c++)
00071     {
00072         int d1 = departTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
```

```

00073         int d2 = departTimeMatrix[1][c - 1];
00074
00075         departTimeMatrix[0][c] = max(d1, d2);
00076
00077         for (size_t r = 1; r < rows - 1; r++)
00078         {
00079             int d1 = departTimeMatrix[r - 1][c] + procTimeMatrix[r][seq[c] - 1];
00080             int d2 = departTimeMatrix[r + 1][c - 1];
00081
00082             departTimeMatrix[r][c] = max(d1, d2);
00083         }
00084
00085         departTimeMatrix[rows - 1][c] = departTimeMatrix[rows - 2][c] +
procTimeMatrix[rows - 1][seq[c] - 1];
00086
00087         FlowshopBasic::calcStartTimeCol(
startTimeMatrix, departTimeMatrix, seq, c, rows, cols);
00088     }
00089 }

```

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

<i>compTimeMatrix</i>	Pointer to completion time matrix
<i>seq</i>	Pointer to job sequence
<i>rows</i>	Number of rows (machines) in the completion time matrix
<i>cols</i>	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](#).

```

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 }

```

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

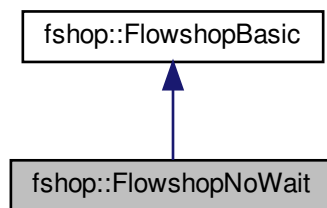
- [include/flowshopblocking.h](#)
- [src/flowshopblocking.cpp](#)

6.5 fshop::FlowshopNoWait Class Reference

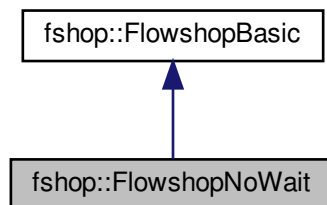
The [FlowshopNoWait](#) class runs the flowshop with no wait problem for a given job-machine processing time matrix that is read from a file. The `run()` method takes the specific job sequence being calculated. Inherits from [FlowshopBasic](#).

```
#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 [FlowshopBasic](#).

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

<i>procTimeMatrixFile</i>	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

<i>compTimeMatrix</i>	Pointer to completion time matrix
<i>seq</i>	Pointer to job sequence
<i>rows</i>	Number of rows (machines) in the completion time matrix
<i>cols</i>	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](#).

```

00072 {
00073     for (size_t c = 1; c < cols; c++)
00074     {
00075         departTimeMatrix[0][c] = departTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
00076
00077         for (size_t r = 1; r < rows; r++)
00078         {
00079             int d1 = departTimeMatrix[r - 1][c];
00080             int d2 = departTimeMatrix[r][c - 1];
00081
00082             if (d1 < d2)
00083             {
00084                 const int diff = d2 - d1;
00085                 for (size_t r2 = r + 1; r2 > 0; r2--)
00086                     departTimeMatrix[r2 - 1][c] += diff;
00087
00088                 d1 = departTimeMatrix[r - 1][c];
00089             }
00090
00091             departTimeMatrix[r][c] = d1 + procTimeMatrix[r][seq[c] - 1];
00092         }
00093
00094         FlowshopBasic::calcStartTimeCol(
00095             startTimeMatrix, departTimeMatrix, seq, c, rows, cols);
00096     }

```

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

<i>compTimeMatrix</i>	Pointer to completion time matrix
<i>seq</i>	Pointer to job sequence
<i>rows</i>	Number of rows (machines) in the completion time matrix
<i>cols</i>	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](#).

```
00053 {
00054     departTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];
00055
00056     for (size_t r = 1; r < rows; r++)
00057     {
00058         departTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + departTimeMatrix[r - 1][0];
00059     }
00060 }
```

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 **_startTimeMatrix, int **_departTimeMatrix, size_t _tMatrixRows, int *_jobSeq, size_t _t_seqSize, int _cmax, int _totalFlowTime)
Constructs a new FlowshopSolution 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 ** _startTimeMatrix,
    int ** _departTimeMatrix,
    size_t _tMatrixRows,
    int * _jobSeq,
    size_t _seqSize,
    int _cmax,
    int _totalFlowTime )
```

Constructs a new FlowshopSolution object.

Parameters

_startTimeMatrix	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(_startTimeMatrix), departTimeMatrix(_departTimeMatrix),
```

```

        numMachines(_tMatrixRows), seqSize(_seqSize), cmax(_cmax),
        totalFlowTime(_totalFlowTime)
00047 {
00048     if (_jobSeq == nullptr)
00049         throw std::invalid_argument("Error: _jobSeq cannot be nullptr");
00050     else if (_startTimeMatrix == nullptr)
00051         throw std::invalid_argument("Error: _startTimeMatrix cannot be nullptr");
00052     else if (_departTimeMatrix == nullptr)
00053         throw std::invalid_argument("Error: _departTimeMatrix cannot be nullptr");
00054     else if (seqSize == 0)
00055         throw std::invalid_argument("Error: _seqSize cannot be zero");
00056
00057     jobSequence = util::allocArray<int>(seqSize);
00058     for (size_t i = 0; i < seqSize; i++)
00059         jobSequence[i] = _jobSeq[i];
00060 }

```

6.6.2.2 ~FlowshopSolution()

FlowshopSolution::~~FlowshopSolution ()

Destroys the [FlowshopSolution](#) object.

Definition at line 65 of file [flowshopbasic.cpp](#).

References [numMachines](#).

```

00066 {
00067     util::releaseArray<int>(jobSequence);
00068     util::releaseMatrix<int>(startTimeMatrix, numMachines);
00069     util::releaseMatrix<int>(departTimeMatrix, numMachines);
00070 }

```

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](#).

```

00202     : startTimeMatrix(obj.startTimeMatrix), departTimeMatrix(obj.departTimeMatrix),
    numMachines(obj.numMachines), seqSize(obj.seqSize),
    cmax(obj.cmax), totalFlowTime(obj.totalFlowTime)
00203 {
00204     if (obj.jobSequence == nullptr)
00205         throw std::invalid_argument("Error: jobSequence cannot be nullptr");
00206     else if (seqSize == 0)
00207         throw std::invalid_argument("Error: seqSize cannot be zero");
00208
00209     jobSequence = util::allocArray<int>(seqSize);
00210     for (size_t i = 0; i < seqSize; i++)
00211         jobSequence[i] = obj.jobSequence[i];
00212 }

```

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](#).

```
00088 {
00089     std::string retStr = "[";
00090
00091     for (size_t i = 0; i < seqSize; i++)
00092     {
00093         retStr += std::to_string(jobSequence[i]);
00094         if (i < seqSize - 1) retStr += "-";
00095     }
00096
00097     retStr += "]";
00098     return retStr;
00099 }
```

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](#).

```
00107 {
00108     return const_cast<const int** const>(startTimeMatrix);
00109 }
```

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

<code>os</code>	The output stream to write to
-----------------	-------------------------------

Definition at line 173 of file `flowshopbasic.cpp`.

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

```

00174 {
00175     std::cout << "Input seq: ";
00176
00177     for (size_t i = 0; i < seqSize; i++)
00178     {
00179         std::cout << jobSequence[i];
00180         if (i < seqSize - 1)
00181             std::cout << ", ";
00182     }
00183
00184     std::cout << std::endl;
00185
00186     std::cout << "Cmax: " << cmax << std::endl;
00187     std::cout << "TFT: " << totalFlowTime << std::endl << std::endl;
00188
00189     std::cout << "Starting times matrix:" << std::endl;
00190     util::outputMatrix(std::cout, startTimeMatrix, numMachines, seqSize, 4);
00191     std::cout << std::endl;
00192
00193     std::cout << "Departure times matrix:" << std::endl;
00194     util::outputMatrix(std::cout, departTimeMatrix,
00195                       numMachines, seqSize, 4);
00195     std::cout << std::endl;
00196 }
```

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

<code>fileNamePrefix</code>	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`.

```

00129 {
00130     using namespace std;
00131
00132     // Create file name strings
00133     string startTimesFile = fileNamePrefix + "starttimes.csv";
00134     string departTimesFile = fileNamePrefix + "departtimes.csv";
00135 }
```

```

00136     // Open files
00137     ofstream startOs = ofstream(startTimesFile, ios::trunc | ios::out);
00138     if (!startOs.good()) return false;
00139
00140     ofstream departOs = ofstream(departTimesFile, ios::trunc | ios::out);
00141     if (!departOs.good()) return false;
00142
00143     // Output start times and departure times data to the files
00144     for (size_t m = 0; m < numMachines; m++)
00145     {
00146         for (size_t j = 0; j < seqSize; j++)
00147         {
00148             startOs << startTimeMatrix[m][j];
00149             departOs << departTimeMatrix[m][j];
00150
00151             if (j < seqSize - 1)
00152             {
00153                 startOs << ",";
00154                 departOs << ",";
00155             }
00156         }
00157
00158         startOs << endl;
00159         departOs << endl;
00160     }
00161
00162     // Close file handles and return
00163     startOs.close();
00164     departOs.close();
00165     return true;
00166 }

```

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](#).

```
00021             : file(""), iniMap()  
00022 {  
00023 }
```

6.7.2.2 ~IniReader()

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

Destroys the [IniReader](#) object.

Definition at line 28 of file [inireader.cpp](#).

```
00029 {  
00030     iniMap.clear();  
00031 }
```

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

<i>section</i>	std::string containing the section name
<i>entry</i>	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\(\)](#).

```
00068 {  
00069     auto it = iniMap.find(section);  
00070     if (it == iniMap.end()) return false;  
00071  
00072     return it->second.find(entry) != it->second.end();  
00073 }
```

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

<i>section</i>	std::string containing the section name
<i>entry</i>	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\(\)](#).

```
00085 {  
00086     if (!entryExists(section, entry)) return defVal;  
00087  
00088     return iniMap[section][entry];  
00089 }
```

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\(\)](#).

```
00057                                     {}
00058     {
00059         std::stringstream ss(getEntry(section, entry, std::to_string(defVal)));
00060         T retVal;
00061         ss >> retVal;
00062         return retVal;
00063     }
```

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

<i>filePath</i>	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](#).

```
00041 {
00042     file = filePath;
00043     if (!parseFile())
00044         return false;
00045
00046     return true;
00047 }
```

6.7.3.5 sectionExists()

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

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

Parameters

<i>section</i>	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](#).

```
00056 {  
00057     return iniMap.find(section) != iniMap.end();  
00058 }
```

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

<i>objectiveFs</i>	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\(\)](#).

```

00033 {
00034     jtList availJobsList;
00035     makeInitialAvailJobList(objectiveFs, availJobsList);
00036
00037     auto firstJob = availJobsList.front();
00038     availJobsList.pop_front();
00039
00040     fsSol bestSol = nullptr;
00041     jList* curJobSeq = new jList();
00042     jList* nextJobSeq = new jList();
00043     curJobSeq->push_back(firstJob.job);
00044
00045     while (availJobsList.size() > 0)
00046     {
00047         auto nextJob = availJobsList.front();
00048         availJobsList.pop_front();
00049
00050         bestSol.reset();
00051         bestSol = bestPermutation(objectiveFs, *curJobSeq, nextJob.job, *nextJobSeq);
00052
00053         auto tmp = curJobSeq;
00054         curJobSeq = nextJobSeq;
00055         nextJobSeq = tmp;
00056     }
00057
00058     delete curJobSeq;
00059     delete nextJobSeq;
00060
00061     return std::move(bestSol);
00062 }

```

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

Copyright (c) 2012 Jakob Progsch, Václav Zeman <https://github.com/progschj/ThreadPool>

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

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](#).

```
00065     :   stop(false)
00066 {
00067     for(size_t i = 0; i<threads; ++i)
00068         workers.emplace_back(
00069             [this]
00070             {
00071                 for(;;)
00072                 {
00073                     std::function<void()> task;
00074
00075                     {
00076                         std::unique_lock<std::mutex> lock(this->queue_mutex);
00077                         this->condition.wait(lock,
00078                             [this]{ return this->stop || !this->tasks.empty(); });
00079                         if(this->stop && this->tasks.empty())
00080                             return;
00081                         task = std::move(this->tasks.front());
00082                         this->tasks.pop();
00083                     }
00084
00085                     task();
00086                 }
00087             }
00088         );
00089 }
```

6.11.2.2 ~ThreadPool()

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

Definition at line 117 of file [threadpool.h](#).

References [stopAndJoinAll\(\)](#).

```
00118 {
00119     stopAndJoinAll();
00120 }
```

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\(\)](#).

```
00095 {
00096     using return_type = typename std::result_of<F(Args...)>::type;
00097
00098     auto task = std::make_shared< std::packaged_task<return_type()> >(
00099         std::bind(std::forward<F>(f), std::forward<Args>(args)...)
00100     );
00101
00102     std::future<return_type> res = task->get_future();
00103     {
00104         std::unique_lock<std::mutex> lock(queue_mutex);
00105
00106         // don't allow enqueueing after stopping the pool
00107         if(stop)
00108             throw std::runtime_error("enqueue on stopped ThreadPool");
00109
00110         tasks.emplace([task]() { (*task)(); });
00111     }
00112     condition.notify_one();
00113     return res;
00114 }
```

6.11.3.2 stopAndJoinAll()

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

Definition at line 122 of file [threadpool.h](#).

Referenced by [cs471::Experiment::runNEH\(\)](#), and [~ThreadPool\(\)](#).

```
00123 {
00124     {
00125         std::unique_lock<std::mutex> lock(queue_mutex);
00126         stop = true;
00127     }
00128
00129     condition.notify_all();
00130     for(std::thread &worker: workers)
00131         worker.join();
00132 }
```

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

- [include/threadpool.h](#)

Chapter 7

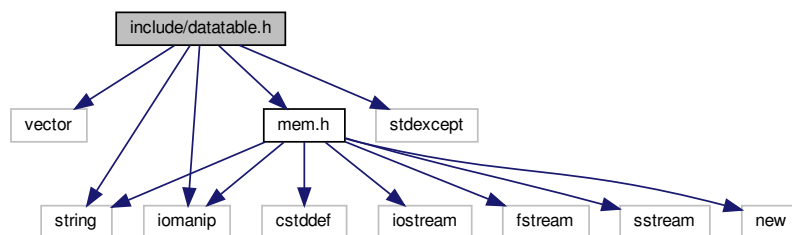
File Documentation

7.1 include/datatable.h File Reference

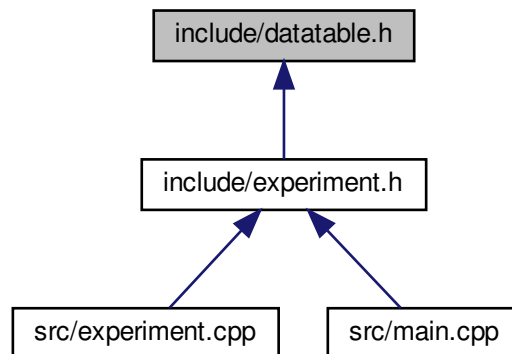
Header file for the DataTable class, which represents a spreadsheet/table of values that can easily be exported to a *.csv file.

```
#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

```

00001
00013 #ifndef __DATATABLE_H
00014 #define __DATATABLE_H
00015
00016 #include <vector>
00017 #include <string>
00018 #include <stdexcept>
00019 #include <iomanip>
00020 #include "mem.h"
00021
00022 namespace mdata
00023 {
00049     template <class T>
00050     class DataTable
00051     {
00052     public:
00060         DataTable(size_t _rows, size_t _cols) : rows(_rows), cols(_cols), dataMatrix(nullptr)
00061         {
00062             if (rows == 0)
00063                 throw std::length_error("Table rows must be greater than 0.");
00064             else if (cols == 0)
00065                 throw std::length_error("Table columns must be greater than 0.");
00066
00067             dataMatrix = util::allocMatrix<T>(rows, cols);
00068             if (dataMatrix == nullptr)
00069                 throw std::bad_alloc();
00070
00071             colLabels.resize(_cols, std::string());
00072         }
00073
00077         ~DataTable()
00078         {
00079             util::releaseMatrix(dataMatrix, rows);
00080         }
00081
00082         void clearData()
00083         {
00084             util::initMatrix<T>(dataMatrix, rows, cols, 0);
00085         }
00086
00093         std::string getColLabel(size_t colIndex)
00094         {
00095             if (colIndex >= colLabels.size())
00096                 throw std::out_of_range("Column index out of range");
00097
00098             return colLabels[colIndex];
00099         }
00100
00107         void setColLabel(size_t colIndex, std::string newLabel)
00108         {
00109             if (colIndex >= colLabels.size())
00110                 throw std::out_of_range("Column index out of range");
00111
00112             colLabels[colIndex] = newLabel;
00113         }
00114
00122         T getEntry(size_t row, size_t col)
00123         {
00124             if (dataMatrix == nullptr)
00125                 throw std::runtime_error("Data matrix not allocated");
00126             if (row >= rows)
00127                 throw std::out_of_range("Table row out of range");
00128             else if (col >= cols)
00129                 throw std::out_of_range("Table column out of range");
00130
00131             return dataMatrix[row][col];
00132         }
00133
00141         void setEntry(size_t row, size_t col, T val)
00142         {
00143             if (dataMatrix == nullptr)
00144                 throw std::runtime_error("Data matrix not allocated");
00145             if (row >= rows)
00146                 throw std::out_of_range("Table row out of range");
00147             else if (col >= cols)
00148                 throw std::out_of_range("Table column out of range");
00149
00150             dataMatrix[row][col] = val;
00151         }
00152
00160         bool exportCSV(const char* filePath)
00161         {
00162             if (dataMatrix == nullptr) return false;
00163

```

```

00164         using namespace std;
00165         ofstream outFile;
00166         outFile.open(filePath, ofstream::out | ofstream::trunc);
00167         if (!outFile.good()) return false;
00168
00169         // Print column labels
00170         for (unsigned int c = 0; c < cols; c++)
00171         {
00172             outFile << colLabels[c];
00173             if (c < cols - 1) outFile << ",";
00174         }
00175
00176         outFile << endl;
00177
00178         // Print data rows
00179         for (unsigned int r = 0; r < rows; r++)
00180         {
00181             for (unsigned int c = 0; c < cols; c++)
00182             {
00183                 outFile << std::setprecision(8) << dataMatrix[r][c];
00184                 if (c < cols - 1) outFile << ",";
00185             }
00186             outFile << endl;
00187         }
00188
00189         outFile.close();
00190         return true;
00191     }
00192 private:
00193     size_t rows;
00194     size_t cols;
00195     std::vector<std::string> colLabels;
00196     T** dataMatrix;
00197 };
00198 // mdata
00199 } // mdata
00200
00201 #endif
00202
00203 // =====
00204 // End of datatable.h
00205 // =====

```

7.3 include/experiment.h File Reference

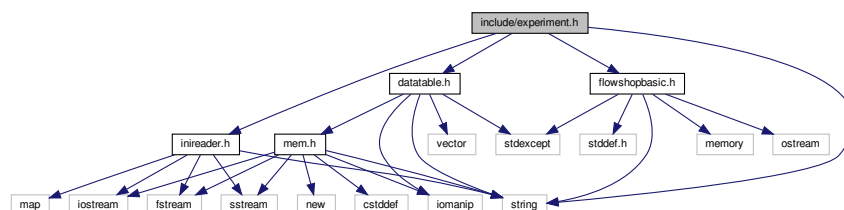
Contains the Experiment class which runs the [cs471](#) lab 5 experiment.

```

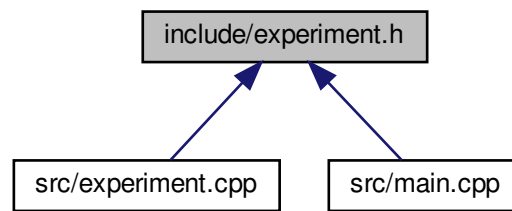
#include <string>
#include "inireader.h"
#include "datatable.h"
#include "flowshopbasic.h"

```

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

```

00001
00012 #ifndef __EXPERIMENT_H
00013 #define __EXPERIMENT_H
00014
00015 #include <string>
00016 #include "inireader.h"
00017 #include "datatable.h"
00018 #include "flowshopbasic.h"
00019
00020 namespace cs471
00021 {
00022     struct TestParams
00023     {
00024         int minTestFile;
00025         int maxTestFile;
00026         int numThreads;
00027         int algorithm;
00028         std::string inputFilesDir;
00029         std::string resultsFile;
00030         std::string timesFile;
00031     };
00032
00033     class Experiment
00034     {
00035     public:
00036         Experiment(std::string paramsFile);
00037         ~Experiment() = default;
00038
00039         int runNEH();
00040         int runDebugSeq(int* seq, size_t seqSize);
00041     private:
00042         util::IniReader iniParams;
00043
00044         int runNEHThreaded(TestParams* const p, const std::string inputFile, int testIndex,
00045                             mdata::DataTable<std::string>* resultsTable);
00046         fshop::FlowshopBasic* allocFlowShop(const char* inputFile, int alg);
00047         TestParams readTestParams();
00048     };
00049 }
00050
00051 #endif
00052
00053 // =====
00054 // End of experiment.h
00055 // =====

```

7.5 include/flowshopbasic.h File Reference

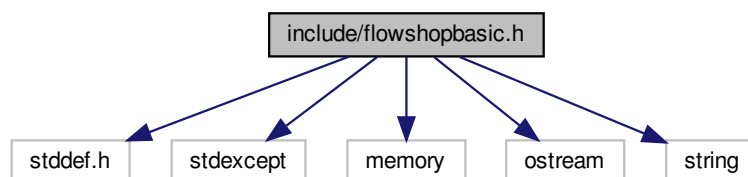
Contains the FlowshopBasic class which can solve a basic flowshop scheduling problem for a given job sequence. The FlowshopBasic class is also used as a base class for the FlowshopBlocking and FlowshopNoWait classes.

```

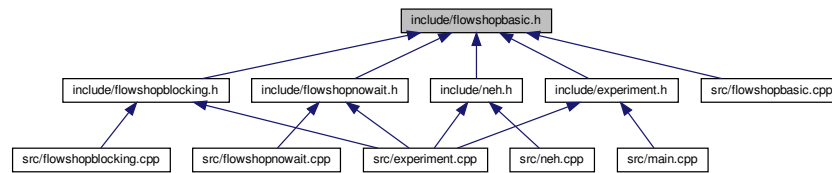
#include <stddef.h>
#include <stdexcept>
#include <memory>
#include <ostream>
#include <string>

```

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

```

00001
00015 #ifndef __FLOWSHOPBASIC_H
00016 #define __FLOWSHOPBASIC_H
00017
00018 #include <stddef.h>
00019 #include <stdexcept>
00020 #include <memory>
00021 #include <ostream>
00022 #include <string>
00023
00024 namespace fshop
00025 {
00032     struct FlowshopSolution
00033     {
00034         FlowshopSolution(int** _startTimeMatrix, int** _departTimeMatrix, size_t
_tMatrixRows, int* _jobSeq, size_t _seqSize, int _cmax, int _totalFlowTime);
00035         ~FlowshopSolution();
00036
00037         const size_t seqSize;
00038         const size_t numMachines;
00039         const int cmax;
00040         const int totalFlowTime;
00042         const int* const getJobSeq();
00043         std::string getJobSeqAsString();
00044         const int** const getStartTimeMatrix();
00045         const int** const getDepartTimeMatrix();
00046
00047         bool outputTimesCsv(const std::string& fileNamePrefix);
00048
00049         void outputAll(std::ostream& os);
00050
00051         // Copy constructor
00052         FlowshopSolution(const FlowshopSolution& obj);
00053
00054         // Move constructor
00055         FlowshopSolution(FlowshopSolution&& obj);
00056
00057         // Delete copy assignment
00058         FlowshopSolution& operator=(const
FlowshopSolution& obj) = delete;
00059
00060         // Delete move assignment
00061         FlowshopSolution& operator=(FlowshopSolution&& obj) =
delete;
00062     private:
00063         int* jobSequence;
00064         int** startTimeMatrix;
00065         int** departTimeMatrix;
00066     };
00067
00074     class FlowshopBasic
00075     {
00076     public:
00077         FlowshopBasic(const char* procTimeMatrixFile);
00078         virtual ~FlowshopBasic();
00079         virtual std::unique_ptr<FlowshopSolution> calcObjective(int* seq, size_t
seqSize);
00080
00081         virtual int getProcessingTime(size_t machine, size_t job);
00082         virtual size_t getTotalJobs();
00083         virtual size_t getTotalMachines();
00084         virtual size_t getFuncCallCounts();
00085
00086         // Delete copy/move constructors and assignments
00087         FlowshopBasic(const FlowshopBasic& o) = delete;
00088         FlowshopBasic(const FlowshopBasic&& o) = delete;
00089         FlowshopBasic& operator=(const FlowshopBasic& o) = delete;
00090         FlowshopBasic& operator=(const FlowshopBasic&& o) = delete;
00091     protected:
00092         int** procTimeMatrix;
00093         int** startTimeMatrix;
00094         size_t ptMatrixRows;
00095         size_t ptMatrixCols;
00096         size_t funcCallCounter;
00098         virtual void validateParams(int* seq, size_t seqSize);
00099         virtual int** allocTimeMatrix(size_t rows, size_t cols);
00100         virtual void initTimeMatrix(int** compTimeMatrix, int* seq, size_t rows, size_t cols);
00101         virtual void calcTimeMatrix(int** compTimeMatrix, int* seq, size_t rows, size_t cols);
00102         virtual void calcStartTimeCol(int** startTimeMatrix, int** departTimeMatrix, int* seq, size_t
curCol, size_t rows, size_t cols);
00103         virtual int getCmax(int** compTimeMatrix, size_t rows, size_t cols);
00104         virtual int getTFT(int** compTimeMatrix, size_t rows, size_t cols);
00105     };
00106 }

```

```

00107
00108 #endif
00109
00110 // =====
00111 // End of flowshopbasic.h
00112 // =====

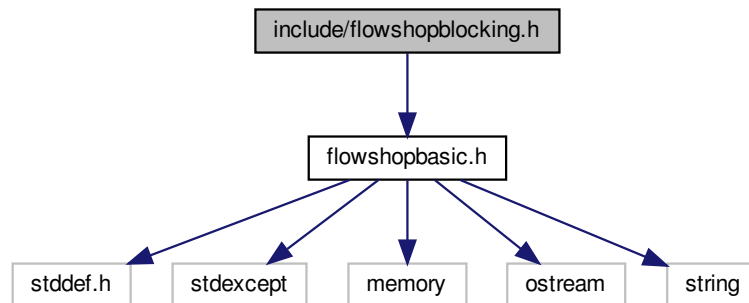
```

7.7 include/flowshopblocking.h File Reference

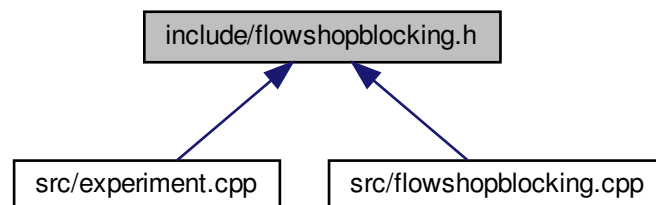
Contains the FlowshopBlocking class, which inherits FlowshopBasic and solves a flowshop with blocking problem for a specific job sequence.

```
#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 [FlowshopBasic](#).

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

```

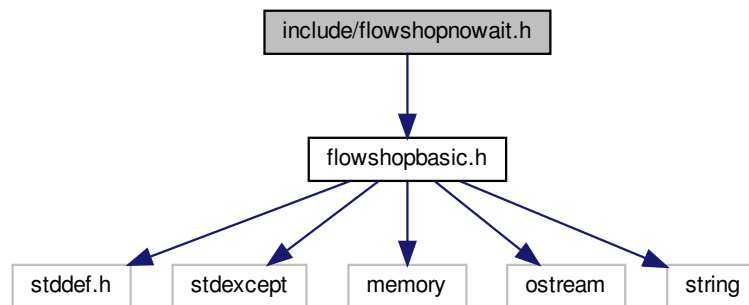
00001
00014 #ifndef __FLOWSHOPBLOCKING_H
00015 #define __FLOWSHOPBLOCKING_H
00016
00017 #include "flowshopbasic.h"
00018
00019 namespace fshop
00020 {
00026     class FlowshopBlocking : public fshop::FlowshopBasic
00027     {
00028     public:
00029         FlowshopBlocking(const char* procTimeMatrixFile);
00030         virtual ~FlowshopBlocking() = default;
00031     protected:
00032         virtual void initTimeMatrix(int** compTimeMatrix, int* seq, size_t rows, size_t cols)
00033         override;
00034         virtual void calcTimeMatrix(int** compTimeMatrix, int* seq, size_t rows, size_t cols)
00035         override;
00036     };
00037 #endif
00038
00039 // =====
00040 // End of flowshopblocking.h
00041 // =====

```

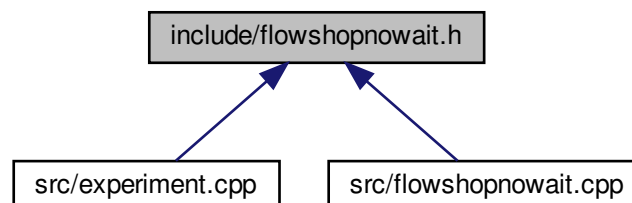
7.9 include/flowshopnowait.h File Reference

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

```

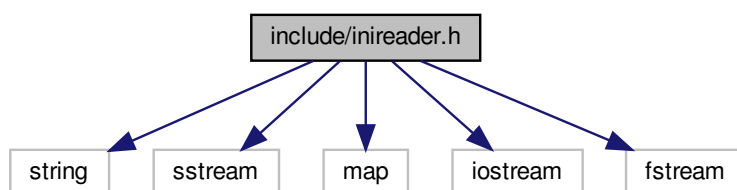
00001
00013 #ifndef __FLOWSHOPNOWAIT_H
00014 #define __FLOWSHOPNOWAIT_H
00015
00016 #include "flowshopbasic.h"
00017
00018 namespace fshop
00019 {
00025     class FlowshopNoWait : public fshop::FlowshopBasic
00026     {
00027     public:
00028         FlowshopNoWait(const char* procTimeMatrixFile);
00029         virtual ~FlowshopNoWait() = default;
00030     protected:
00031         virtual void initTimeMatrix(int** departTimeMatrix, int* seq, size_t rows, size_t
cols) override;
00032         virtual void calcTimeMatrix(int** departTimeMatrix, int* seq, size_t rows, size_t
cols) override;
00033     };
00034 }
00035
00036 #endif
00037
00038 // =====
00039 // End of flowshopnowait.h
00040 // =====

```

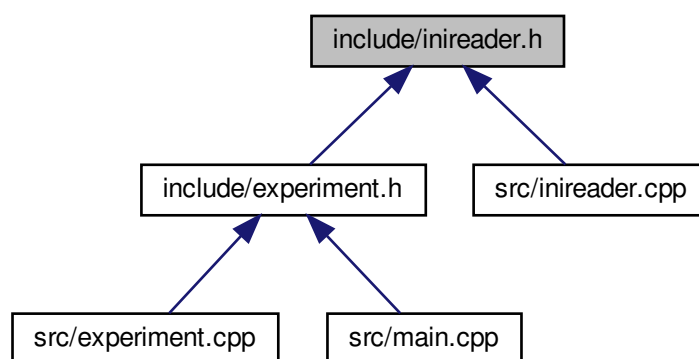

7.11 include/inireader.h File Reference

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

```

00001
00013 #ifndef __INIREADER_H
00014 #define __INIREADER_H
00015
00016 #include <string>
00017 #include <sstream>
00018 #include <map>
00019 #include <iostream>
00020 #include <fstream>
00021
00022 namespace util
00023 {
00046     class IniReader
00047     {
00048     public:
00049         IniReader();
00050         ~IniReader();
00051         bool openFile(std::string filePath);
00052         bool sectionExists(std::string section);
00053         bool entryExists(std::string section, std::string entry);
00054         std::string getEntry(std::string section, std::string entry, std::string defVal = "");
00055
00056         template <class T>
00057         T getEntryAs(std::string section, std::string entry, T defVal = {})
00058         {
00059             std::stringstream ss(getEntry(section, entry, std::to_string(defVal)));
00060             T retVal;
00061             ss >> retVal;
00062             return retVal;
00063         }
00064     private:
00065         std::string file;
00066         std::map<std::string, std::map<std::string, std::string>> iniMap;
00068         bool parseFile();
00069         void parseEntry(const std::string& sectionName, const std::string& entry);
00070     };
00071 }
00072
00073 #endif
00074
00075 // =====
00076 // End of inireader.h
00077 // =====

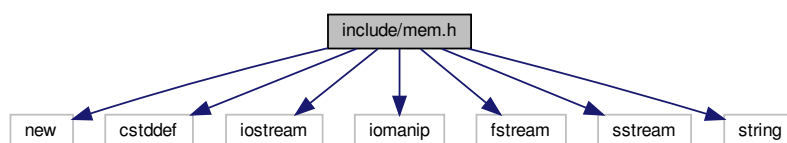
```

7.13 include/mem.h File Reference

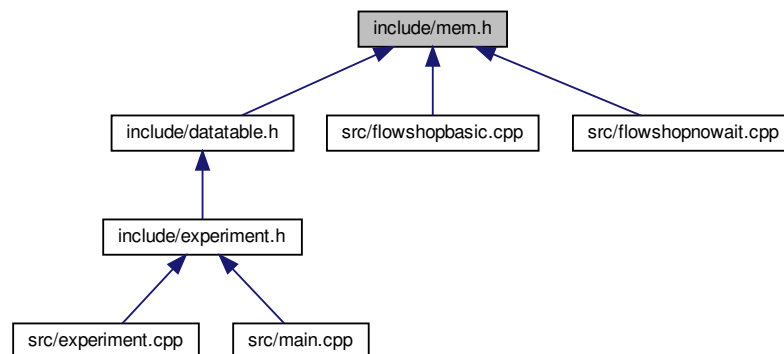
Header file for various memory utility functions.

```
#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

```

00001
00012 #ifndef __MEM_H
00013 #define __MEM_H
00014
00015 #include <new> // std::nothrow
00016 #include <cstdint> // size_t definition
00017 #include <iostream>
00018 #include <iomanip>
00019 #include <fstream>
00020 #include <sstream>
00021 #include <string>
00022
00023 namespace util
00024 {
00033     template <class T = double>
00034     inline void initArray(T* a, size_t size, T val)
00035     {
00036         if (a == nullptr) return;
00037
00038         for (size_t i = 0; i < size; i++)
00039         {
00040             a[i] = val;
00041         }
00042     }
00043
00053     template <class T = double>
00054     inline void initMatrix(T** m, size_t rows, size_t cols, T val)
00055     {
00056         if (m == nullptr) return;
00057
00058         for (size_t i = 0; i < rows; i++)
00059         {
00060             initArray(m[i], cols, val);
00061         }
00062     }
00063
00070     template <class T = double>
00071     bool releaseArray(T*& a)
00072     {
00073         if (a == nullptr) return true;
00074
00075         try
00076         {
00077             delete[] a;
00078             a = nullptr;
00079             return true;
00080         }
00081         catch(...)
00082         {
00083             return false;
00084         }
00085     }
00086
00094     template <class T = double>
00095     void releaseMatrix(T**& m, size_t rows)
00096     {
00097         if (m == nullptr) return;
00098
00099         for (size_t i = 0; i < rows; i++)
00100         {
00101             if (m[i] != nullptr)
00102             {
00103                 // Release each row
00104                 releaseArray<T>(m[i]);
00105             }
00106         }
00107
00108         // Release columns
00109         delete[] m;
00110         m = nullptr;
00111     }
00112
00120     template <class T = double>
00121     inline T* allocArray(size_t size)
00122     {
00123         return new(std::nothrow) T[size];
00124     }
00125
00134     template <class T = double>
00135     inline T** allocMatrix(size_t rows, size_t cols)
00136     {
00137         T** m = (T**)allocArray<T*>(rows);
00138         if (m == nullptr) return nullptr;
00139

```

```

00140         for (size_t i = 0; i < rows; i++)
00141         {
00142             m[i] = allocArray<T>(cols);
00143             if (m[i] == nullptr)
00144             {
00145                 releaseMatrix<T>(m, rows);
00146                 return nullptr;
00147             }
00148         }
00149         return m;
00150     }
00151 }
00152
00153 template <class T = double>
00154 inline T** loadMatrixFromFile(const char* filePath, size_t& outNumRows, size_t&
outNumCols)
00155 {
00156     outNumRows = 0;
00157     outNumCols = 0;
00158
00159     std::ifstream is(filePath);
00160     if (!is.good())
00161     {
00162         std::cerr << "Error loading matrix from file: Unable to open file." << std::endl;
00163         return nullptr;
00164     }
00165
00166     std::string line;
00167     if (!std::getline(is, line))
00168     {
00169         std::cerr << "Error loading matrix from file: File is empty or invalid." << std::endl;
00170         is.close();
00171         return nullptr;
00172     }
00173
00174     size_t rows = 0;
00175     size_t cols = 0;
00176
00177     std::stringstream ss(line);
00178     if (!(ss >> rows >> cols) || rows == 0 || cols == 0)
00179     {
00180         std::cerr << "Error loading matrix from file: Row or column size is zero." << std::endl;
00181         is.close();
00182         return nullptr;
00183     }
00184
00185     T** retMatrix = allocMatrix<T>(rows, cols);
00186     if (retMatrix == nullptr)
00187     {
00188         std::cerr << "Error loading matrix from file: Matrix memory allocation failed." << std::endl;
00189         is.close();
00190         return nullptr;
00191     }
00192
00193     for (size_t r = 0; r < rows; r++)
00194     {
00195         if (!std::getline(is, line))
00196         {
00197             std::cerr << "Error loading matrix from file: EOF reached before reading all rows." <<
std::endl;
00198             releaseMatrix<T>(retMatrix, rows);
00199             is.close();
00200             return nullptr;
00201         }
00202
00203         std::stringstream ss(line);
00204
00205         for (size_t c = 0; c < cols; c++)
00206         {
00207             T entry = 0;
00208             if (!(ss >> entry))
00209             {
00210                 std::cerr << "Error loading matrix from file: EOL reached before reading all cols." <<
std::endl;
00211                 releaseMatrix<T>(retMatrix, rows);
00212                 is.close();
00213                 return nullptr;
00214             }
00215
00216             retMatrix[r][c] = entry;
00217         }
00218     }
00219
00220     is.close();
00221     outNumRows = rows;
00222     outNumCols = cols;
00223 }

```

```

00224         return retMatrix;
00225     }
00226
00227     template <class T = double>
00228     inline void outputMatrix(std::ostream& os, T** matrix, size_t rows, size_t cols, int
colWidth = 3)
00229     {
00230         if (matrix == nullptr)
00231             return;
00232
00233         for (size_t r = 0; r < rows; r++)
00234         {
00235             for (size_t c = 0; c < cols; c++)
00236             {
00237                 os << std::setw(3) << matrix[r][c];
00238                 if (c < cols - 1)
00239                     os << " ";
00240                 else
00241                     os << std::endl;
00242             }
00243         }
00244     }
00245
00254     template <class T = double>
00255     inline void copyArray(T* src, T* dest, size_t size)
00256     {
00257         for (size_t i = 0; i < size; i++)
00258             dest[i] = src[i];
00259     }
00260 }
00261
00262 #endif
00263
00264 // =====
00265 // End of mem.h
00266 // =====

```

7.15 include/neh.h File Reference

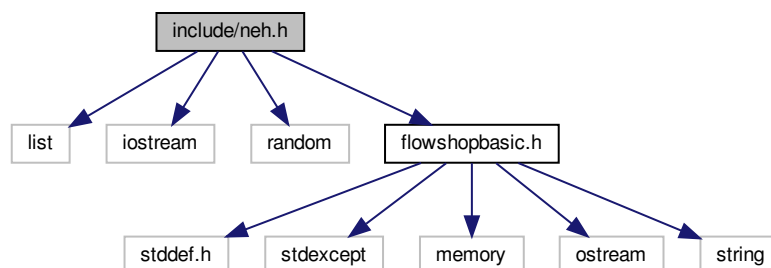
Contains the NEH class, which runs the NEH algorithm on a given flowshop problem. The NEH algorithm aims to optimize the job sequence such that it produces the smallest cMax value.

```

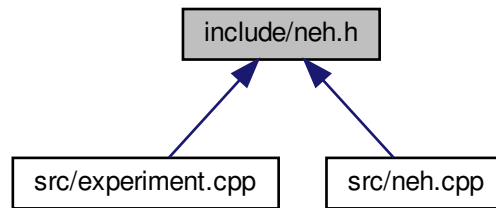
#include <list>
#include <iostream>
#include <random>
#include "flowshopbasic.h"

```

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,
00057             std::list<fshop::JobTimePair>& outList);
00057         fsSol bestPermutation(FlowshopBasic* const objectiveFs, const std::list<int>&
00058             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.

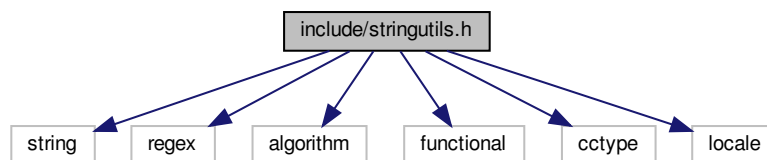
```

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

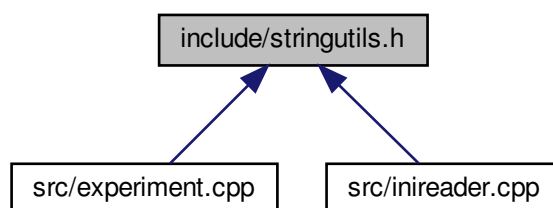
```

```
#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

```

00001
00008 #ifndef __STRINGUTILS_H
00009 #define __STRINGUTILS_H
00010
00011 #include <string>
00012 #include <regex>
00013 #include <algorithm>
00014 #include <functional>
00015 #include <cctype>
00016 #include <locale>
00017
00018 namespace util
00019 {
00028     static inline std::string s_replace(std::string input, std::string pattern, std::string replacement)
00029     {
00030         pattern = std::string("\\") + pattern;
00031         return std::regex_replace(input, std::regex(pattern), replacement);
00032     }
00033
00034     // =====
00035     // The string functions below were written by Evan Teran
00036     // from Stack Overflow:
00037     // https://stackoverflow.com/questions/216823/whats-the-best-way-to-trim-stdstring
00038     // =====
00039
00040     // trim from start (in place)
00041     static inline void s_ltrim(std::string &s) {
00042         s.erase(s.begin(), std::find_if(s.begin(), s.end(),
00043             std::not1(std::ptr_fun<int, int>(std::isspace))));
00044     }
00045
00046     // trim from end (in place)
00047     static inline void s_rtrim(std::string &s) {
00048         s.erase(std::find_if(s.rbegin(), s.rend(),
00049             std::not1(std::ptr_fun<int, int>(std::isspace))).base(), s.end());
00050     }
00051
00052     // trim from both ends (in place)
00053     static inline void s_trim(std::string &s) {
00054         s_ltrim(s);
00055         s_rtrim(s);
00056     }
00057
00058     // trim from start (copying)
00059     static inline std::string s_ltrim_copy(std::string s) {
00060         s_ltrim(s);
00061         return s;
00062     }
00063
00064     // trim from end (copying)
00065     static inline std::string s_rtrim_copy(std::string s) {
00066         s_rtrim(s);
00067         return s;
00068     }
00069
00070     // trim from both ends (copying)
00071     static inline std::string s_trim_copy(std::string s) {
00072         s_trim(s);
00073         return s;
00074     }
00075 }
00076 #endif
00077
00078 // =====
00079 // End of stringutils.h
00080 // =====

```

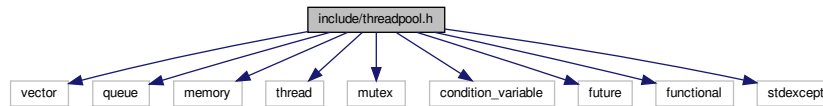
7.19 include/threadpool.h File Reference

```

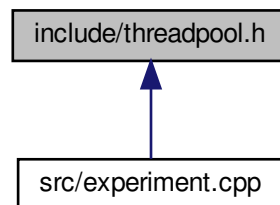
#include <vector>
#include <queue>
#include <memory>
#include <thread>
#include <mutex>
#include <condition_variable>

```

```
#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

```
00001
00029 #ifndef __THREADPOOL_H
00030 #define __THREADPOOL_H
00031
00032 #include <vector>
00033 #include <queue>
00034 #include <memory>
00035 #include <thread>
00036 #include <mutex>
00037 #include <condition_variable>
00038 #include <future>
00039 #include <functional>
00040 #include <stdexcept>
00041
00042 class ThreadPool {
00043 public:
00044     ThreadPool(size_t);
00045     template<class F, class... Args>
00046     auto enqueue(F&& f, Args&&... args)
00047         -> std::future<typename std::result_of<F(Args...)>::type>;
00048     ~ThreadPool();
00049
00050     void stopAndJoinAll();
00051 private:
00052     // need to keep track of threads so we can join them
```

```

00053     std::vector< std::thread > workers;
00054     // the task queue
00055     std::queue< std::function<void()> > tasks;
00056
00057     // synchronization
00058     std::mutex queue_mutex;
00059     std::condition_variable condition;
00060     bool stop;
00061 };
00062
00063 // the constructor just launches some amount of workers
00064 inline ThreadPool::ThreadPool(size_t threads)
00065     :   stop(false)
00066 {
00067     for(size_t i = 0; i<threads;++i)
00068         workers.emplace_back(
00069             [this]
00070             {
00071                 for(;;)
00072                 {
00073                     std::function<void()> task;
00074
00075                     {
00076                         std::unique_lock<std::mutex> lock(this->queue_mutex);
00077                         this->condition.wait(lock,
00078                             [this]{ return this->stop || !this->tasks.empty(); });
00079                         if(this->stop && this->tasks.empty())
00080                             return;
00081                         task = std::move(this->tasks.front());
00082                         this->tasks.pop();
00083                     }
00084
00085                     task();
00086                 }
00087             }
00088         );
00089 }
00090
00091 // add new work item to the pool
00092 template<class F, class... Args>
00093 auto ThreadPool::enqueue(F&& f, Args&&... args)
00094     -> std::future<typename std::result_of<F(Args...)>::type>
00095 {
00096     using return_type = typename std::result_of<F(Args...)>::type;
00097
00098     auto task = std::make_shared< std::packaged_task<return_type()> >(
00099         std::bind(std::forward<F>(f), std::forward<Args>(args)...)
00100     );
00101
00102     std::future<return_type> res = task->get_future();
00103     {
00104         std::unique_lock<std::mutex> lock(queue_mutex);
00105
00106         // don't allow enqueueing after stopping the pool
00107         if(stop)
00108             throw std::runtime_error("enqueue on stopped ThreadPool");
00109
00110         tasks.emplace([task]() { (*task)(); });
00111     }
00112     condition.notify_one();
00113     return res;
00114 }
00115
00116 // the destructor joins all threads
00117 inline ThreadPool::~ThreadPool()
00118 {
00119     stopAndJoinAll();
00120 }
00121
00122 inline void ThreadPool::stopAndJoinAll()
00123 {
00124     {
00125         std::unique_lock<std::mutex> lock(queue_mutex);
00126         stop = true;
00127     }
00128
00129     condition.notify_all();
00130     for(std::thread &worker: workers)
00131         worker.join();
00132 }
00133
00134 #endif
00135
00136 // =====
00137 // End of threadpool.h
00138 // =====

```


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     {
```



```

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 }
00056
00064 int Experiment::runNEH()
00065 {
00066     // Retrieve test parameters from ini file
00067     TestParams p = readTestParams();
00068
00069     // Construct data table to store experiment results
00070     mdata::DataTable<string> resultsTable(p.maxTestFile - p.
minTestFile + 1, 6);
00071
00072     // Initialize thread pool with a parameter-given number of threads
00073     ThreadPool tpool(p.numThreads);
00074
00075     // Initialize thread future vector, used for thread pool synchronization
00076     // and keeps track of the individual tasks being executed.
00077     vector<std::future<int>> futures;
00078
00079     cout << "Started " << p.numThreads << " worker threads ..." << endl;
00080
00081     if (p.algorithm == 1)
00082         cout << "Running NEH on Flow Shop with Blocking ..." << endl;
00083     else if (p.algorithm == 2)
00084         cout << "Running NEH on Flow Shop with No Wait ..." << endl;
00085     else
00086         cout << "Running NEH on Flow Shop Scheduling ..." << endl;
00087
00088     // Prepare results table column header labels
00089     resultsTable.setColLabel(0, "Data Set");
00090     resultsTable.setColLabel(1, "cMax");
00091     resultsTable.setColLabel(2, "TFT");
00092     resultsTable.setColLabel(3, "Func Calls");
00093     resultsTable.setColLabel(4, "Execution Time (ms)");
00094     resultsTable.setColLabel(5, "Sequence");
00095
00096     // Add all input test files as tasks in thread pool
00097     for (int i = p.minTestFile; i <= p.maxTestFile; i++)
00098     {
00099         string inputFile = std::to_string(i) + ".txt";
00100         futures.emplace_back(
00101             tpool.enqueue(&cs471::Experiment::runNEHThreaded, this, &p, inputFile, i, &resultsTable)
00102         );
00103     }
00104
00105     // const size_t totalFutures = futures.size();
00106
00107     // Join all thread pool tasks using futures vector
00108     // and get the return value for each
00109     for (int i = 0; i < futures.size(); i++)
00110     {
00111         int err = futures[i].get();
00112         if (err)
00113         {
00114             // Threaded task returned with an error code, bail
00115             tpool.stopAndJoinAll();
00116             return err;
00117         }
00118     }
00119
00120     // Output results table to a csv file
00121     if (!p.resultsFile.empty())
00122     {
00123         resultsTable.exportCSV(p.resultsFile.c_str());
00124         cout << "Results exported to: " << p.resultsFile << endl;
00125     }
00126
00127     return 0;
00128 }
00129
00140 int Experiment::runNEHThreaded(TestParams* const p, const std::string inputFile, int testIndex,
mdata::DataTable<std::string>* resultsTable)
00141 {
00142     string fullInputPath = p->inputFilesDir + inputFile;
00143
00144     // Get the flowshop objective function that we want to optimize
00145     auto objectiveFs = allocFlowShop(fullInputPath.c_str(), p->algorithm);
00146     if (objectiveFs == nullptr)
00147         return 1;
00148
00149     // Prepare pointer to results
00150     fsSol result = nullptr;

```

```

00151
00152 // Start recording execution time
00153 high_resolution_clock::time_point t_start = high_resolution_clock::now();
00154
00155 try
00156 {
00157     // Run the NEH algorithm on the objective flowshop function
00158     NEH neh;
00159     result = neh.run(objectiveFs);
00160 }
00161 catch(const std::exception& e)
00162 {
00163     std::cerr << "An exception occurred while running NEH:" << endl;
00164     std::cerr << e.what() << endl;
00165     std::cerr << "Input file: " << inputFile << endl;
00166     return 2;
00167 }
00168
00169 // Record execution time
00170 high_resolution_clock::time_point t_end = high_resolution_clock::now();
00171 double execTimeMs = static_cast<double>(duration_cast<nanoseconds>(t_end - t_start).count()) / 1000000.
0;
00172
00173 // Insert NEH results into results table at the correct row
00174 resultsTable->setEntry(testIndex, 0, std::to_string(testIndex));
00175 resultsTable->setEntry(testIndex, 1, std::to_string(result->cmax));
00176 resultsTable->setEntry(testIndex, 2, std::to_string(result->totalFlowTime));
00177 resultsTable->setEntry(testIndex, 3, std::to_string(objectiveFs->getFuncCallCounts()));
00178 resultsTable->setEntry(testIndex, 4, std::to_string(execTimeMs));
00179 resultsTable->setEntry(testIndex, 5, result->getJobSeqAsString());
00180
00181 // ===== GANTT STUFF =====
00182 /*
00183 const size_t numMachines = objectiveFs->getTotalMachines();
00184 const size_t numJobs = objectiveFs->getTotalJobs();
00185
00186 auto startTimeMatrix = result->getStartTimeMatrix();
00187 auto departTimeMatrix = result->getDepartTimeMatrix();
00188
00189 mdata::DataTable<std::string> ganttTable(numJobs * numMachines, 5);
00190
00191 ganttTable.setColLabel(0, "Item");
00192 ganttTable.setColLabel(1, "Machine");
00193 ganttTable.setColLabel(2, "Job");
00194 ganttTable.setColLabel(3, "Start");
00195 ganttTable.setColLabel(4, "End");
00196
00197 size_t row = 0;
00198
00199 for (size_t m = 0; m < numMachines; m++)
00200 {
00201     for (size_t j = 0; j < numJobs; j++)
00202     {
00203         ganttTable.setEntry(row, 0, std::to_string(row + 1));
00204         ganttTable.setEntry(row, 1, std::string("Machine ") + std::to_string(m + 1));
00205         ganttTable.setEntry(row, 2, std::string("Job ") + std::to_string(j + 1));
00206         ganttTable.setEntry(row, 3, std::to_string(startTimeMatrix[m][j]));
00207         ganttTable.setEntry(row, 4, std::to_string(departTimeMatrix[m][j]));
00208
00209         row++;
00210     }
00211 }
00212
00213 std::string ganttfile = "results/gantt/";
00214
00215 if (p->algorithm == 0)
00216     ganttfile += "fss/";
00217 else if (p->algorithm == 1)
00218     ganttfile += "fsb/";
00219 else
00220     ganttfile += "fsnw/";
00221
00222 ganttfile += std::to_string(testIndex);
00223 ganttfile += "-gantt.csv";
00224 ganttTable.exportCSV(ganttfile.c_str());
00225 */
00226 // =====
00227
00228 // Dump NEH results start and departure time matrices to a csv file
00229 if (!p->timesFile.empty())
00230     result->outputTimesCsv(util::s_replace(p->timesFile, "%TEST%", std::to_string(testIndex)))
;
00231
00232 // Clean up allocated memory
00233 delete objectiveFs;
00234
00235

```

```

00236
00237     return 0;
00238 }
00239
00240 FlowshopBasic* Experiment::allocFlowShop(const char* inputFile, int alg)
00241 {
00242     FlowshopBasic* objectiveFs = nullptr;
00243
00244     switch (alg)
00245     {
00246     case 0:
00247         objectiveFs = new FlowshopBasic(inputFile);
00248         break;
00249     case 1:
00250         objectiveFs = new FlowshopBlocking(inputFile);
00251         break;
00252     case 2:
00253         objectiveFs = new FlowshopNoWait(inputFile);
00254         break;
00255     }
00256
00257     return objectiveFs;
00258 }
00259
00260 TestParams Experiment::readTestParams()
00261 {
00262     TestParams p = { };
00263
00264     p.minTestFile = iniParams.getEntryAs<int>(INI_TEST_SECTION,
00265     INI_TEST_MINFILE, 0);
00266     p.maxTestFile = iniParams.getEntryAs<int>(INI_TEST_SECTION,
00267     INI_TEST_MAXFILE, 120);
00268     p.numThreads = iniParams.getEntryAs<int>(INI_TEST_SECTION,
00269     INI_TEST_NUMTHREADS, 1);
00270     p.algorithm = iniParams.getEntryAs<int>(INI_TEST_SECTION,
00271     INI_TEST_ALGORITHM, 0);
00272     p.inputFilesDir = iniParams.getEntry(INI_TEST_SECTION,
00273     INI_TEST_INPUTFILEDIR, "");
00274     p.resultsFile = iniParams.getEntry(INI_TEST_SECTION,
00275     INI_TEST_RESULTSFILE, "");
00276     p.timesFile = iniParams.getEntry(INI_TEST_SECTION,
00277     INI_TEST_TIMESFILE, "");
00278
00279     // Check bounds for numThreads
00280     if (p.numThreads < 1 || p.numThreads > 16)
00281     {
00282         cout << "Warning: Number of threads invalid. Defaulting to default 1 threads." << endl;
00283         p.numThreads = 1;
00284     }
00285
00286     // Check bounds for algorithm selection
00287     if (p.algorithm < 0 || p.algorithm > 2)
00288     {
00289         cout << "Warning: Algorithm selection invalid. Defaulting to algorithm 0." << endl;
00290         p.algorithm = 0;
00291     }
00292
00293     return p;
00294 }
00295
00296 int Experiment::runDebugSeq(int* seq, size_t seqSize)
00297 {
00298     // Retrieve test parameters from ini file
00299     TestParams p = readTestParams();
00300
00301     if (p.algorithm == 1)
00302         cout << "Running Flow Shop with Blocking ..." << endl;
00303     else if (p.algorithm == 2)
00304         cout << "Running Flow Shop with No Wait ..." << endl;
00305     else
00306         cout << "Running Flow Shop Scheduling ..." << endl;
00307
00308     cout << endl;
00309
00310     // Prepare pointer to results
00311     fsSol result = nullptr;
00312
00313     for (int i = p.minTestFile; i <= p.maxTestFile; i++)
00314     {
00315         string fullInputPath = p.inputFilesDir + std::to_string(i) + ".txt";
00316
00317         cout << "Input file: " << fullInputPath << endl;
00318
00319         // Get the flowshop objective function that we want to optimize
00320         auto objectiveFs = allocFlowShop(fullInputPath.c_str(), p.algorithm);
00321         if (objectiveFs == nullptr)
00322         {

```

```

00341         cout << "Objective flowshop function encountered an error." << endl;
00342         return 1;
00343     }
00344
00345     result = objectiveFs->calcObjective(seq, seqSize);
00346     result->outputAll(std::cout);
00347
00348     delete objectiveFs;
00349
00350     cout << "======" << endl;
00351 }
00352
00353 cout << "Debug objective function sequence tests completed." << endl;
00354
00355 return 0;
00356 }
00357
00358 // =====
00359 // End of experiment.cpp
00360 // =====

```

7.23 src/flowshopbasic.cpp File Reference

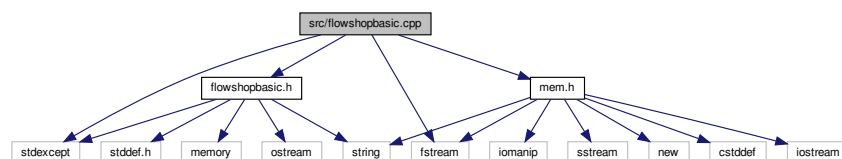
Implementation file for the FlowshopBasic class.

```

#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

<i>val1</i>	First integer
<i>val2</i>	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 {
```

```

00028     if (val1 >= val2) return val1;
00029     else return val2;
00030 }
00031
00032 // =====
00033
00045 FlowshopSolution::FlowshopSolution(int** _startTimeMatrix, int**
    _departTimeMatrix, size_t _tMatrixRows, int* _jobSeq, size_t _seqSize, int _cmax, int _totalFlowTime)
00046     : startTimeMatrix(_startTimeMatrix), departTimeMatrix(_departTimeMatrix), numMachines(_tMatrixRows),
    seqSize(_seqSize), cmax(_cmax), totalFlowTime(_totalFlowTime)
00047 {
00048     if (_jobSeq == nullptr)
00049         throw std::invalid_argument("Error: _jobSeq cannot be nullptr");
00050     else if (_startTimeMatrix == nullptr)
00051         throw std::invalid_argument("Error: _startTimeMatrix cannot be nullptr");
00052     else if (_departTimeMatrix == nullptr)
00053         throw std::invalid_argument("Error: _departTimeMatrix cannot be nullptr");
00054     else if (seqSize == 0)
00055         throw std::invalid_argument("Error: _seqSize cannot be zero");
00056
00057     jobSequence = util::allocArray<int>(seqSize);
00058     for (size_t i = 0; i < seqSize; i++)
00059         jobSequence[i] = _jobSeq[i];
00060 }
00061
00065 FlowshopSolution::~FlowshopSolution()
00066 {
00067     util::releaseArray<int>(jobSequence);
00068     util::releaseMatrix<int>(startTimeMatrix, numMachines);
00069     util::releaseMatrix<int>(departTimeMatrix, numMachines);
00070 }
00071
00077 const int* const FlowshopSolution::getJobSeq()
00078 {
00079     return const_cast<const int* const>(jobSequence);
00080 }
00081
00087 std::string FlowshopSolution::getJobSeqAsString()
00088 {
00089     std::string retStr = "[";
00090
00091     for (size_t i = 0; i < seqSize; i++)
00092     {
00093         retStr += std::to_string(jobSequence[i]);
00094         if (i < seqSize - 1) retStr += "-";
00095     }
00096
00097     retStr += "]";
00098     return retStr;
00099 }
00100
00106 const int** const FlowshopSolution::getStartTimeMatrix()
00107 {
00108     return const_cast<const int** const>(startTimeMatrix);
00109 }
00110
00116 const int** const FlowshopSolution::getDepartTimeMatrix()
00117 {
00118     return const_cast<const int** const>(departTimeMatrix);
00119 }
00120
00128 bool FlowshopSolution::outputTimesCsv(const std::string& fileNamePrefix)
00129 {
00130     using namespace std;
00131
00132     // Create file name strings
00133     string startTimesFile = fileNamePrefix + "starttimes.csv";
00134     string departTimesFile = fileNamePrefix + "departtimes.csv";
00135
00136     // Open files
00137     ofstream startOs = ofstream(startTimesFile, ios::trunc | ios::out);
00138     if (!startOs.good()) return false;
00139
00140     ofstream departOs = ofstream(departTimesFile, ios::trunc | ios::out);
00141     if (!departOs.good()) return false;
00142
00143     // Output start times and departure times data to the files
00144     for (size_t m = 0; m < numMachines; m++)
00145     {
00146         for (size_t j = 0; j < seqSize; j++)
00147         {
00148             startOs << startTimeMatrix[m][j];
00149             departOs << departTimeMatrix[m][j];
00150
00151             if (j < seqSize - 1)
00152             {
00153                 startOs << ",";

```

```

00154         departOs << ", ";
00155     }
00156 }
00157
00158     startOs << endl;
00159     departOs << endl;
00160 }
00161
00162 // Close file handles and return
00163 startOs.close();
00164 departOs.close();
00165 return true;
00166 }
00167
00173 void FlowshopSolution::outputAll(std::ostream& os)
00174 {
00175     std::cout << "Input seq: ";
00176
00177     for (size_t i = 0; i < seqSize; i++)
00178     {
00179         std::cout << jobSequence[i];
00180         if (i < seqSize - 1)
00181             std::cout << ", ";
00182     }
00183
00184     std::cout << std::endl;
00185
00186     std::cout << "Cmax: " << cmax << std::endl;
00187     std::cout << "TFT: " << totalFlowTime << std::endl << std::endl;
00188
00189     std::cout << "Starting times matrix:" << std::endl;
00190     util::outputMatrix(std::cout, startTimeMatrix, numMachines, seqSize, 4);
00191     std::cout << std::endl;
00192
00193     std::cout << "Departure times matrix:" << std::endl;
00194     util::outputMatrix(std::cout, departTimeMatrix,
00195         numMachines, seqSize, 4);
00196     std::cout << std::endl;
00197 }
00201 FlowshopSolution::FlowshopSolution(const
00202     FlowshopSolution& obj)
00203 : startTimeMatrix(obj.startTimeMatrix), departTimeMatrix(obj.departTimeMatrix),
00204     numMachines(obj.numMachines), seqSize(obj.seqSize),
00205     cmax(obj.cmax), totalFlowTime(obj.totalFlowTime)
00206 {
00207     if (obj.jobSequence == nullptr)
00208         throw std::invalid_argument("Error: jobSequence cannot be nullptr");
00209     else if (seqSize == 0)
00210         throw std::invalid_argument("Error: seqSize cannot be zero");
00211
00212     jobSequence = util::allocArray<int>(seqSize);
00213     for (size_t i = 0; i < seqSize; i++)
00214         jobSequence[i] = obj.jobSequence[i];
00215 }
00217 FlowshopSolution::FlowshopSolution(
00218     FlowshopSolution&& obj)
00219 : numMachines(obj.numMachines), seqSize(obj.seqSize), cmax(obj.cmax), totalFlowTime(obj.totalFlowTime)
00220 {
00221     jobSequence = obj.jobSequence;
00222     startTimeMatrix = obj.startTimeMatrix;
00223     departTimeMatrix = obj.departTimeMatrix;
00224     obj.jobSequence = nullptr;
00225     obj.startTimeMatrix = nullptr;
00226     obj.departTimeMatrix = nullptr;
00227 }
00228 // =====
00229
00235 FlowshopBasic::FlowshopBasic(const char* procTimeMatrixFile)
00236 : startTimeMatrix(nullptr), ptMatrixRows(0), ptMatrixCols(0), funcCallCounter(0)
00237 {
00238     // Attempt to load job processing times from the given file
00239     procTimeMatrix = util::loadMatrixFromFile<int>(procTimeMatrixFile,
00240         ptMatrixRows, ptMatrixCols);
00241     if (procTimeMatrix == nullptr)
00242     {
00243         std::string msg = "Error when loading matrix file: ";
00244         msg += procTimeMatrixFile;
00245         throw std::runtime_error(msg);
00246     }
00247 }
00252 FlowshopBasic::~FlowshopBasic()
00253 {
00254     util::releaseMatrix<int>(procTimeMatrix, ptMatrixRows);

```

```

00255 }
00256
00264 int FlowshopBasic::getProcessingTime(size_t machine, size_t job)
00265 {
00266     if (machine == 0 || job == 0)
00267     {
00268         std::string msg = "Error: Machine or job number cannot be zero";
00269         throw std::out_of_range(msg);
00270     }
00271     else if (machine > ptMatrixRows || job > ptMatrixCols)
00272     {
00273         std::string msg = "Error: Machine or job number out of range";
00274         throw std::out_of_range(msg);
00275     }
00276
00277     return procTimeMatrix[machine - 1][job - 1];
00278 }
00279
00285 size_t FlowshopBasic::getTotalJobs()
00286 {
00287     return ptMatrixCols;
00288 }
00289
00295 size_t FlowshopBasic::getTotalMachines()
00296 {
00297     return ptMatrixRows;
00298 }
00299
00305 size_t FlowshopBasic::getFuncCallCounts()
00306 {
00307     return funcCallCounter;
00308 }
00309
00318 std::unique_ptr<FlowshopSolution> FlowshopBasic::calcObjective(int* seq, size_t
seqSize)
00319 {
00320     // Validate input parameters
00321     validateParams(seq, seqSize);
00322
00323     // Allocate completion (departure) time matrix and start time matrix
00324     auto compTimeMatrix = allocTimeMatrix(ptMatrixRows, seqSize);
00325     startTimeMatrix = allocTimeMatrix(ptMatrixRows, seqSize);
00326
00327     // Initialize completion time matrix and start time matrix
00328     initTimeMatrix(compTimeMatrix, seq, ptMatrixRows, seqSize);
00329     calcStartTimeCol(startTimeMatrix, compTimeMatrix, seq, 0,
ptMatrixRows, seqSize);
00330
00331     // Calculate all completion and start times
00332     calcTimeMatrix(compTimeMatrix, seq, ptMatrixRows, seqSize);
00333
00334     // Construct solution struct
00335     auto retVal = std::unique_ptr<FlowshopSolution>(new FlowshopSolution(
startTimeMatrix, compTimeMatrix, ptMatrixRows, seq, seqSize,
getCmax(compTimeMatrix, ptMatrixRows, seqSize),
getTFT(compTimeMatrix, ptMatrixRows, seqSize)));
00336
00337     // Increment obj func call counter and return result
00338     funcCallCounter += 1;
00339     return std::move(retVal);
00340 }
00341
00342
00349 void FlowshopBasic::validateParams(int* seq, size_t seqSize)
00350 {
00351     // Make sure job sequence is not empty, or too large
00352     if (seqSize == 0 || seqSize > ptMatrixCols)
00353     {
00354         std::string msg = "Error: seqSize cannot be larger than ptMatrixCols";
00355         throw std::out_of_range(msg);
00356     }
00357
00358     // Make sure all jobs in job sequence are within bounds of processing time matrix
00359     for (size_t i = 0; i < seqSize; i++)
00360     {
00361         if (seq[i] <= 0 || seq[i] > ptMatrixCols)
00362         {
00363             std::string msg = "Error: seq contains a job number out of range [1, ";
00364             msg += std::to_string(ptMatrixCols);
00365             msg += "].";
00366             throw std::out_of_range(msg);
00367         }
00368     }
00369 }
00370
00378 int** FlowshopBasic::allocTimeMatrix(size_t rows, size_t cols)
00379 {
00380     int** timeMatrix = util::allocMatrix<int>(rows, cols);

```



```

00381     if (timeMatrix == nullptr)
00382     {
00383         std::cerr << "Error allocating time matrix." << std::endl;
00384         throw std::bad_alloc();
00385     }
00386
00387     util::initMatrix<int>(timeMatrix, rows, cols, 0);
00388
00389     return timeMatrix;
00390 }
00391
00401 void FlowshopBasic::initTimeMatrix(int** compTimeMatrix, int* seq, size_t rows
, size_t cols)
00402 {
00403     // Set first job, first machine
00404     compTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];
00405
00406     // Set first job for all machines
00407     for (size_t r = 1; r < rows; r++)
00408     {
00409         compTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + compTimeMatrix[r - 1][0];
00410     }
00411
00412     // Set first machine for all jobs
00413     for (size_t c = 1; c < cols; c++)
00414     {
00415         compTimeMatrix[0][c] = compTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
00416     }
00417 }
00418
00427 void FlowshopBasic::calcTimeMatrix(int** compTimeMatrix, int* seq, size_t rows
, size_t cols)
00428 {
00429     for (size_t c = 1; c < cols; c++)
00430     {
00431         for (size_t r = 1; r < rows; r++)
00432         {
00433             int c1 = compTimeMatrix[r - 1][c];
00434             int c2 = compTimeMatrix[r][c - 1];
00435
00436             compTimeMatrix[r][c] = max(c1, c2) + procTimeMatrix[r][seq[c] - 1];
00437         }
00438
00439         FlowshopBasic::calcStartTimeCol(
startTimeMatrix, compTimeMatrix, seq, c, rows, cols);
00440     }
00441 }
00442
00453 void FlowshopBasic::calcStartTimeCol(int**
startTimeMatrix, int** departTimeMatrix, int* seq, size_t curCol, size_t rows, size_t cols)
00454 {
00455     for (size_t r = rows; r > 0; r--)
00456     {
00457         startTimeMatrix[r - 1][curCol] = departTimeMatrix[r - 1][curCol] -
procTimeMatrix[r - 1][seq[curCol] - 1];
00458     }
00459 }
00460
00469 int FlowshopBasic::getCmax(int** compTimeMatrix, size_t rows, size_t cols)
00470 {
00471     return compTimeMatrix[rows - 1][cols - 1];
00472 }
00473
00482 int FlowshopBasic::getTFT(int** compTimeMatrix, size_t rows, size_t cols)
00483 {
00484     int sum = 0;
00485
00486     for (size_t c = 0; c < cols; c++)
00487     {
00488         sum += compTimeMatrix[rows - 1][c];
00489     }
00490
00491     return sum;
00492 }
00493
00494 // =====
00495 // End of flowshopbasic.cpp
00496 // =====

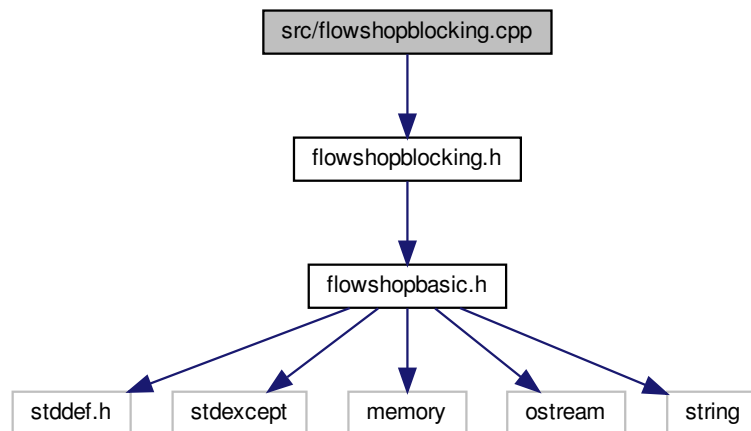
```

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

<i>val1</i>	First integer
<i>val2</i>	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)
```

```

00069 {
00070     for (size_t c = 1; c < cols; c++)
00071     {
00072         int d1 = departTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
00073         int d2 = departTimeMatrix[1][c - 1];
00074
00075         departTimeMatrix[0][c] = max(d1, d2);
00076
00077         for (size_t r = 1; r < rows - 1; r++)
00078         {
00079             int d1 = departTimeMatrix[r - 1][c] + procTimeMatrix[r][seq[c] - 1];
00080             int d2 = departTimeMatrix[r + 1][c - 1];
00081
00082             departTimeMatrix[r][c] = max(d1, d2);
00083         }
00084
00085         departTimeMatrix[rows - 1][c] = departTimeMatrix[rows - 2][c] +
00086         procTimeMatrix[rows - 1][seq[c] - 1];
00087
00088         FlowshopBasic::calcStartTimeCol(
00089             startTimeMatrix, departTimeMatrix, seq, c, rows, cols);
00090     }
00091 }
00092 // =====
00093 // End of flowshopblocking.cpp
00094 // =====

```

7.27 src/flowshopnowait.cpp File Reference

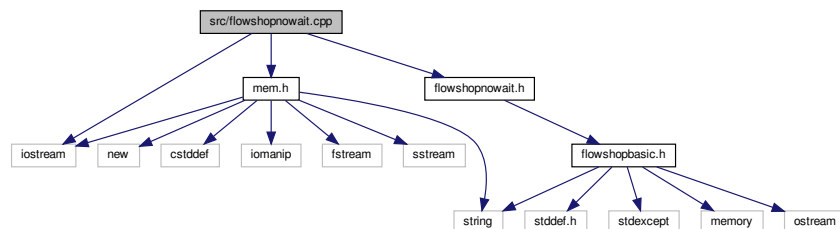
Implementation file for the FlowshopNoWait class.

```
#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

<i>val1</i>	First integer
<i>val2</i>	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
```

```

00017 using namespace fshop;
00018
00026 inline int max(int val1, int val2)
00027 {
00028     if (val1 >= val2) return val1;
00029     else return val2;
00030 }
00031
00037 FlowshopNoWait::FlowshopNoWait(const char* procTimeMatrixFile)
00038     : FlowshopBasic(procTimeMatrixFile)
00039 {
00040 }
00041
00052 void FlowshopNoWait::initTimeMatrix(int** departTimeMatrix, int* seq, size_t
rows, size_t cols)
00053 {
00054     departTimeMatrix[0][0] = procTimeMatrix[0][seq[0] - 1];
00055
00056     for (size_t r = 1; r < rows; r++)
00057     {
00058         departTimeMatrix[r][0] = procTimeMatrix[r][seq[0] - 1] + departTimeMatrix[r - 1][0];
00059     }
00060 }
00061
00071 void FlowshopNoWait::calcTimeMatrix(int** departTimeMatrix, int* seq, size_t
rows, size_t cols)
00072 {
00073     for (size_t c = 1; c < cols; c++)
00074     {
00075         departTimeMatrix[0][c] = departTimeMatrix[0][c - 1] + procTimeMatrix[0][seq[c] - 1];
00076
00077         for (size_t r = 1; r < rows; r++)
00078         {
00079             int d1 = departTimeMatrix[r - 1][c];
00080             int d2 = departTimeMatrix[r][c - 1];
00081
00082             if (d1 < d2)
00083             {
00084                 const int diff = d2 - d1;
00085                 for (size_t r2 = r + 1; r2 > 0; r2--)
00086                     departTimeMatrix[r2 - 1][c] += diff;
00087
00088                 d1 = departTimeMatrix[r - 1][c];
00089             }
00090
00091             departTimeMatrix[r][c] = d1 + procTimeMatrix[r][seq[c] - 1];
00092         }
00093     }
00094
00095     FlowshopBasic::calcStartTimeCol(
00096         startTimeMatrix, departTimeMatrix, seq, c, rows, cols);
00097 }
00098 // =====
00099 // End of flowshopnowait.cpp
00100 // =====

```

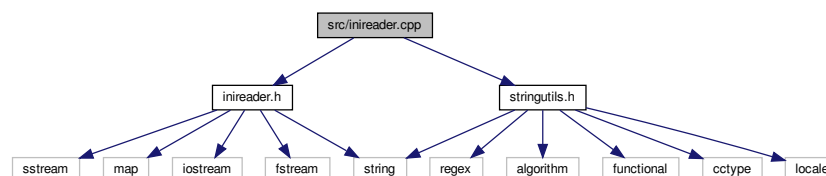
7.29 src/inireader.cpp File Reference

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

```

00001
00013 #include "inireader.h"
00014 #include "stringutils.h"
00015
00016 using namespace util;
00017
00021 IniReader::IniReader() : file(""), iniMap()
00022 {
00023 }
00024
00028 IniReader::~IniReader()
00029 {
00030     iniMap.clear();
00031 }
00032
00040 bool IniReader::openFile(std::string filePath)
00041 {
00042     file = filePath;
00043     if (!parseFile())
00044         return false;
00045     return true;
00046 }
00047
00048
00055 bool IniReader::sectionExists(std::string section)
00056 {
00057     return iniMap.find(section) != iniMap.end();
00058 }
00059
00067 bool IniReader::entryExists(std::string section, std::string entry)
00068 {
00069     auto it = iniMap.find(section);
00070     if (it == iniMap.end()) return false;
00071     return it->second.find(entry) != it->second.end();
00072 }
00073
00074
00084 std::string IniReader::getEntry(std::string section, std::string entry, std::string
defVal)
00085 {
00086     if (!entryExists(section, entry)) return defVal;
00087

```

```

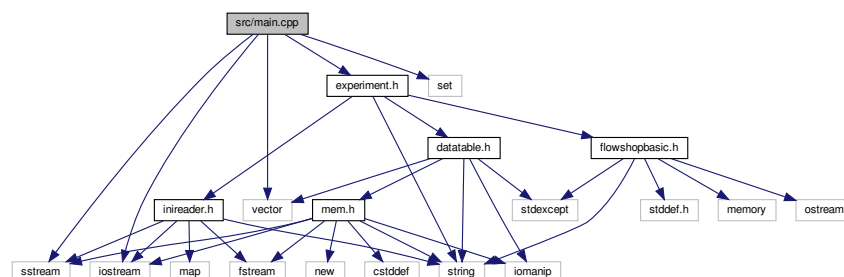
00088     return iniMap[section][entry];
00089 }
00090
00097 bool IniReader::parseFile()
00098 {
00099     iniMap.clear();
00100
00101     using namespace std;
00102
00103     ifstream inputF(file, ifstream::in);
00104     if (!inputF.good()) return false;
00105
00106     string curSection;
00107     string line;
00108
00109     while (getline(inputF, line))
00110     {
00111         // Trim whitespace on both ends of the line
00112         s_trim(line);
00113
00114         // Ignore empty lines and comments
00115         if (line.empty() || line.front() == '#')
00116         {
00117             continue;
00118         }
00119         else if (line.front() == '[' && line.back() == ']')
00120         {
00121             // Line is a section definition
00122             // Erase brackets and trim to get section name
00123             line.erase(0, 1);
00124             line.erase(line.length() - 1, 1);
00125             s_trim(line);
00126             curSection = line;
00127         }
00128         else if (!curSection.empty())
00129         {
00130             // Line is an entry, parse the key and value
00131             parseEntry(curSection, line);
00132         }
00133     }
00134
00135     // Close input file
00136     inputF.close();
00137     return true;
00138 }
00139
00144 void IniReader::parseEntry(const std::string& sectionName, const std::string& entry)
00145 {
00146     using namespace std;
00147
00148     // Split string around equals sign character
00149     const string delim = "=";
00150     string entryName;
00151     string entryValue;
00152
00153     // Find index of '='
00154     auto delimPos = entry.find(delim);
00155
00156     if (delimPos == string::npos || delimPos >= entry.length() - 1)
00157         return; // '=' is missing, or is last char in string
00158
00159     // Extract entry name/key and value
00160     entryName = entry.substr((size_t)0, delimPos);
00161     entryValue = entry.substr(delimPos + 1, entry.length());
00162
00163     // Remove leading and trailing whitespace
00164     s_trim(entryName);
00165     s_trim(entryValue);
00166
00167     // We cannot have entries with empty keys
00168     if (entryName.empty()) return;
00169
00170     // Add entry to cache
00171     iniMap[sectionName][entryName] = entryValue;
00172 }
00173
00174 // =====
00175 // End of inireader.cpp
00176 // =====

```

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\(\)](#).

```
00024 {
00025     // Make sure we have enough command line args
00026     if (argc <= 1)
00027     {
00028         cout << "Error: Missing command line parameter." << endl;
00029         cout << "Proper usage: " << argv[0] << " [param file] \"[Debug Job Sequence]\"" << endl;
00030         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;
00031         return EXIT_FAILURE;
00032     }
00033
00034     try
00035     {
00036         if (argc > 2)
00037         {
00038             return runDebugJobSeq(argv[1], argv[2]);
00039         }
00040         else
00041         {
00042             // Run experiment and return error code
00043             cs471::Experiment ex(argv[1]);
00044             return ex.runNEH();
00045         }
00046     }
00047
00048     catch(const std::exception& e)
00049     {
00050         std::cerr << "An exception occurred:" << endl;
00051         std::cerr << e.what() << endl;
00052         return 3;
00053     }
00054
00055     return 0;
00056 }
```

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\(\)](#).

```
00059 {
00060     string strSeq = seq;
00061     vector<int> jobSeq;
00062
00063     stringstream ss(strSeq);
00064     int val;
00065
00066     while (ss >> val)
00067     {
00068         jobSeq.push_back(val);
```

```

00069     }
00070
00071     if (jobSeq.size() == 0)
00072     {
00073         cerr << "Error: debug job sequence is missing or invalid." << endl;
00074         return 1;
00075     }
00076
00077     set<int> permCheckSet(jobSeq.begin(), jobSeq.end());
00078     if (permCheckSet.size() != jobSeq.size())
00079     {
00080         cerr << "Error: debug job sequence has duplicate jobs in permutation." << endl;
00081         return 2;
00082     }
00083
00084     cout << "Running debug sequence: " << seq << endl;
00085     cout << "===== " << endl;
00086
00087     // Run experiment and return error code
00088     cs471::Experiment ex(paramsFile);
00089     return ex.runDebugSeq(&jobSeq[0], jobSeq.size());
00090
00091     return 0;
00092 }

```

7.32 main.cpp

```

00001
00012 #include <iostream>
00013 #include <sstream>
00014 #include <vector>
00015 #include <set>
00016
00017 #include "experiment.h"
00018
00019 using namespace std;
00020
00021 int runDebugJobSeq(const char* paramsFile, const char* seq);
00022
00023 int main(int argc, char** argv)
00024 {
00025     // Make sure we have enough command line args
00026     if (argc <= 1)
00027     {
00028         cout << "Error: Missing command line parameter." << endl;
00029         cout << "Proper usage: " << argv[0] << " [param file] \"[Debug Job Sequence]\"" << endl;
00030         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;
00031         return EXIT_FAILURE;
00032     }
00033
00034     try
00035     {
00036         if (argc > 2)
00037         {
00038             return runDebugJobSeq(argv[1], argv[2]);
00039         }
00040         else
00041         {
00042             // Run experiment and return error code
00043             cs471::Experiment ex(argv[1]);
00044             return ex.runNEH();
00045         }
00046     }
00047
00048     catch(const std::exception& e)
00049     {
00050         std::cerr << "An exception occurred:" << endl;
00051         std::cerr << e.what() << endl;
00052         return 3;
00053     }
00054
00055     return 0;
00056 }
00057
00058 int runDebugJobSeq(const char* paramsFile, const char* seq)
00059 {
00060     string strSeq = seq;
00061     vector<int> jobSeq;
00062
00063     stringstream ss(strSeq);
00064     int val;

```

```

00065
00066     while (ss >> val)
00067     {
00068         jobSeq.push_back(val);
00069     }
00070
00071     if (jobSeq.size() == 0)
00072     {
00073         cerr << "Error: debug job sequence is missing or invalid." << endl;
00074         return 1;
00075     }
00076
00077     set<int> permCheckSet(jobSeq.begin(), jobSeq.end());
00078     if (permCheckSet.size() != jobSeq.size())
00079     {
00080         cerr << "Error: debug job sequence has duplicate jobs in permutation." << endl;
00081         return 2;
00082     }
00083
00084     cout << "Running debug sequence: " << seq << endl;
00085     cout << "===== " << endl;
00086
00087     // Run experiment and return error code
00088     cs471::Experiment ex(paramsFile);
00089     return ex.runDebugSeq(&jobSeq[0], jobSeq.size());
00090
00091     return 0;
00092 }
00093
00094 // =====
00095 // End of main.cpp
00096 // =====

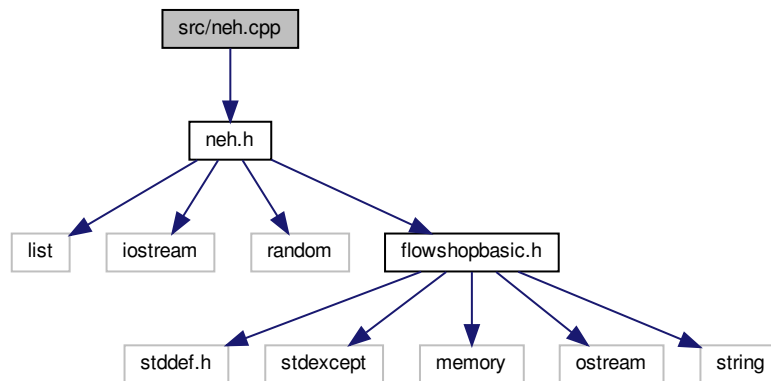
```

7.33 src/neh.cpp File Reference

Implementation file for the NEH class.

```
#include "neh.h"
```

Include dependency graph for neh.cpp:



Typedefs

- using `jList` = `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

```

00001
00012 #include "neh.h"
00013
00014 // Type alias
00015 using jtList = std::list<fshop::JobTimePair>;
00016 using jList = std::list<int>;
00017
00022 fshop::NEH::NEH()
00023 : rd(), randEngine(rd()), randChance(0, 1)
00024 { }
00025
00032 fsSol fshop::NEH::run(FlowshopBasic* const objectiveFs)
00033 {
00034     jtList availJobsList;
00035     makeInitialAvailJobList(objectiveFs, availJobsList);
00036
00037     auto firstJob = availJobsList.front();
00038     availJobsList.pop_front();
00039
00040     fsSol bestSol = nullptr;
00041     jList* curJobSeq = new jList();
00042     jList* nextJobSeq = new jList();
00043     curJobSeq->push_back(firstJob.job);
00044
00045     while (availJobsList.size() > 0)
00046     {
00047         auto nextJob = availJobsList.front();
00048         availJobsList.pop_front();
00049
00050         bestSol.reset();
00051         bestSol = bestPermutation(objectiveFs, *curJobSeq, nextJob.job, *nextJobSeq);
00052
00053         auto tmp = curJobSeq;
00054         curJobSeq = nextJobSeq;
00055         nextJobSeq = tmp;
00056     }
00057
00058     delete curJobSeq;
00059     delete nextJobSeq;
00060
00061     return std::move(bestSol);
00062 }
00063
00070 void fshop::NEH::makeInitialAvailJobList(FlowshopBasic* const objectiveFs,
jtList& outList)
00071 {
00072     const size_t numMachines = objectiveFs->getTotalMachines();
00073     const size_t numJobs = objectiveFs->getTotalJobs();
00074
00075     for (size_t j = 1; j <= numJobs; j++)
00076     {
00077         int sum = 0;
00078         for (size_t m = 1; m <= numMachines; m++)
00079             sum += objectiveFs->getProcessingTime(m, j);
00080
00081         outList.emplace_back(
00082             JobTimePair(j, sum)
00083         );
00084     }
00085
00086     // Sort in decreasing order of time
00087     outList.sort([](JobTimePair& lhs, JobTimePair& rhs) { return lhs.
time >= rhs.time; });
00088 }
00089
00099 fsSol fshop::NEH::bestPermutation(FlowshopBasic* const objectiveFs, const
jList& baseList, int jobInsert, jList& outBestSeq)
00100 {
00101     fsSol bestSol = nullptr;
00102     outBestSeq.clear();
00103
00104     jList bufferList;
00105     int* seqArr = new int[baseList.size() + 1];
00106
00107     for (size_t i = 0; i <= baseList.size(); i++)
00108     {
00109         bufferList = jList(baseList.begin(), baseList.end());
00110         auto it = bufferList.begin();
00111         std::advance(it, i);
00112
00113         bufferList.insert(it, jobInsert);
00114
00115         size_t index = 0;
00116

```

```
00117         for (auto i = bufferList.begin(); i != bufferList.end(); i++)
00118         {
00119             seqArr[index] = *i;
00120             index++;
00121         }
00122
00123         auto result = objectiveFs->calcObjective(seqArr, bufferList.size());
00124         if (bestSol == nullptr || result->cmax < bestSol->cmax ||
00125             (result->cmax == bestSol->cmax && randChance(randEngine) >= 0.5))
00126         {
00127             bestSol.reset();
00128             bestSol = std::move(result);
00129             outBestSeq = jList(bufferList.begin(), bufferList.end());
00130         }
00131     }
00132
00133     delete[] seqArr;
00134     return std::move(bestSol);
00135 }
00136
00137 // =====
00138 // End of neh.cpp
00139 // =====
```


Index

- ~DataTable
 - mdata::DataTable, [20](#)
- ~Experiment
 - cs471::Experiment, [25](#)
- ~FlowshopBasic
 - fshop::FlowshopBasic, [30](#)
- ~FlowshopBlocking
 - fshop::FlowshopBlocking, [43](#)
- ~FlowshopNoWait
 - fshop::FlowshopNoWait, [46](#)
- ~FlowshopSolution
 - fshop::FlowshopSolution, [50](#)
- ~IniReader
 - util::IniReader, [56](#)
- ~ThreadPool
 - ThreadPool, [65](#)
- algorithm
 - cs471::TestParams, [63](#)
- allocArray
 - util, [10](#)
- allocMatrix
 - util, [11](#)
- allocTimeMatrix
 - fshop::FlowshopBasic, [31](#)
- calcObjective
 - fshop::FlowshopBasic, [31](#)
- calcStartTimeCol
 - fshop::FlowshopBasic, [32](#)
- calcTimeMatrix
 - fshop::FlowshopBasic, [33](#)
 - fshop::FlowshopBlocking, [43](#)
 - fshop::FlowshopNoWait, [46](#)
- clearData
 - mdata::DataTable, [21](#)
- cmax
 - fshop::FlowshopSolution, [54](#)
- copyArray
 - util, [12](#)
- cs471, [9](#)
- cs471::Experiment, [24](#)
 - ~Experiment, [25](#)
 - Experiment, [25](#)
 - runDebugSeq, [26](#)
 - runNEH, [27](#)
- cs471::TestParams, [62](#)
 - algorithm, [63](#)
 - inputFilesDir, [63](#)
 - maxTestFile, [63](#)
 - minTestFile, [63](#)
 - numThreads, [63](#)
 - resultsFile, [63](#)
 - timesFile, [64](#)
- DataTable
 - mdata::DataTable, [20](#)
- enqueue
 - ThreadPool, [65](#)
- entryExists
 - util::IniReader, [56](#)
- Experiment
 - cs471::Experiment, [25](#)
- experiment.cpp
 - INI_TEST_ALGORITHM, [93](#)
 - INI_TEST_INPUTFILEDIR, [93](#)
 - INI_TEST_MAXFILE, [93](#)
 - INI_TEST_MINFILE, [93](#)
 - INI_TEST_NUMTHREADS, [93](#)
 - INI_TEST_RESULTSFILE, [93](#)
 - INI_TEST_SECTION, [94](#)
 - INI_TEST_TIMESFILE, [94](#)
- exportCSV
 - mdata::DataTable, [21](#)
- FlowshopBasic
 - fshop::FlowshopBasic, [30, 31](#)
- FlowshopBlocking
 - fshop::FlowshopBlocking, [42](#)
- FlowshopNoWait
 - fshop::FlowshopNoWait, [46](#)
- FlowshopSolution
 - fshop::FlowshopSolution, [49, 50](#)
- flowshopbasic.cpp
 - max, [99](#)
- flowshopblocking.cpp
 - max, [105](#)
- flowshopnowait.cpp
 - max, [107](#)
- fsSol
 - neh.h, [87](#)
- fshop, [9](#)
- fshop::FlowshopBasic, [28](#)
 - ~FlowshopBasic, [30](#)
 - allocTimeMatrix, [31](#)
 - calcObjective, [31](#)
 - calcStartTimeCol, [32](#)
 - calcTimeMatrix, [33](#)
 - FlowshopBasic, [30, 31](#)

- funcCallCounter, 40
- getCmax, 34
- getFuncCallCounts, 34
- getProcessingTime, 35
- getTFT, 35
- getTotalJobs, 37
- getTotalMachines, 37
- initTimeMatrix, 38
- operator=, 39
- procTimeMatrix, 40
- ptMatrixCols, 40
- ptMatrixRows, 40
- startTimeMatrix, 41
- validateParams, 39
- fshop::FlowshopBlocking, 41
 - ~FlowshopBlocking, 43
 - calcTimeMatrix, 43
 - FlowshopBlocking, 42
 - initTimeMatrix, 44
- fshop::FlowshopNoWait, 45
 - ~FlowshopNoWait, 46
 - calcTimeMatrix, 46
 - FlowshopNoWait, 46
 - initTimeMatrix, 47
- fshop::FlowshopSolution, 48
 - ~FlowshopSolution, 50
 - cmax, 54
 - FlowshopSolution, 49, 50
 - getDepartTimeMatrix, 51
 - getJobSeq, 51
 - getJobSeqAsString, 51
 - getStartTimeMatrix, 52
 - numMachines, 54
 - operator=, 52
 - outputAll, 52
 - outputTimesCsv, 53
 - seqSize, 54
 - totalFlowTime, 55
- fshop::JobTimePair, 59
 - job, 60
 - JobTimePair, 59
 - time, 60
- fshop::NEH, 60
 - NEH, 61
 - run, 61
- funcCallCounter
 - fshop::FlowshopBasic, 40
- getCmax
 - fshop::FlowshopBasic, 34
- getColLabel
 - mdata::DataTable, 22
- getDepartTimeMatrix
 - fshop::FlowshopSolution, 51
- getEntry
 - mdata::DataTable, 22
 - util::IniReader, 57
- getEntryAs
 - util::IniReader, 57
- getFuncCallCounts
 - fshop::FlowshopBasic, 34
- getJobSeq
 - fshop::FlowshopSolution, 51
- getJobSeqAsString
 - fshop::FlowshopSolution, 51
- getProcessingTime
 - fshop::FlowshopBasic, 35
- getStartTimeMatrix
 - fshop::FlowshopSolution, 52
- getTFT
 - fshop::FlowshopBasic, 35
- getTotalJobs
 - fshop::FlowshopBasic, 37
- getTotalMachines
 - fshop::FlowshopBasic, 37
- INI_TEST_ALGORITHM
 - experiment.cpp, 93
- INI_TEST_INPUTFILEDIR
 - experiment.cpp, 93
- INI_TEST_MAXFILE
 - experiment.cpp, 93
- INI_TEST_MINFILE
 - experiment.cpp, 93
- INI_TEST_NUMTHREADS
 - experiment.cpp, 93
- INI_TEST_RESULTSFILE
 - experiment.cpp, 93
- INI_TEST_SECTION
 - experiment.cpp, 94
- INI_TEST_TIMESFILE
 - experiment.cpp, 94
- include/datatable.h, 67, 69
- include/experiment.h, 70, 72
- include/flowshopbasic.h, 72, 74
- include/flowshopblocking.h, 75, 76
- include/flowshopnowait.h, 77, 78
- include/inireader.h, 79, 80
- include/mem.h, 81, 83
- include/neh.h, 85, 87
- include/stringutils.h, 87, 89
- include/threadpool.h, 89, 90
- IniReader
 - util::IniReader, 56
- initArray
 - util, 12
- initMatrix
 - util, 13
- initTimeMatrix
 - fshop::FlowshopBasic, 38
 - fshop::FlowshopBlocking, 44
 - fshop::FlowshopNoWait, 47
- inputFilesDir
 - cs471::TestParams, 63
- jList
 - neh.cpp, 115
- job

- fshop::JobTimePair, 60
- JobTimePair
 - fshop::JobTimePair, 59
- jtList
 - neh.cpp, 115
- loadMatrixFromFile
 - util, 14
- main
 - main.cpp, 111
- main.cpp
 - main, 111
 - runDebugJobSeq, 112
- max
 - flowshopbasic.cpp, 99
 - flowshopblocking.cpp, 105
 - flowshopnowait.cpp, 107
- maxTestFile
 - cs471::TestParams, 63
- mdata, 10
- mdata::DataTable
 - ~DataTable, 20
 - clearData, 21
 - DataTable, 20
 - exportCSV, 21
 - getColLabel, 22
 - getEntry, 22
 - setColLabel, 23
 - setEntry, 24
- mdata::DataTable< T >, 19
- minTestFile
 - cs471::TestParams, 63
- NEH
 - fshop::NEH, 61
- neh.cpp
 - jList, 115
 - jtList, 115
- neh.h
 - fsSol, 87
- numMachines
 - fshop::FlowshopSolution, 54
- numThreads
 - cs471::TestParams, 63
- openFile
 - util::IniReader, 58
- operator=
 - fshop::FlowshopBasic, 39
 - fshop::FlowshopSolution, 52
- outputAll
 - fshop::FlowshopSolution, 52
- outputMatrix
 - util, 15
- outputTimesCsv
 - fshop::FlowshopSolution, 53
- procTimeMatrix
 - fshop::FlowshopBasic, 40
- ptMatrixCols
 - fshop::FlowshopBasic, 40
- ptMatrixRows
 - fshop::FlowshopBasic, 40
- releaseArray
 - util, 15
- releaseMatrix
 - util, 16
- resultsFile
 - cs471::TestParams, 63
- run
 - fshop::NEH, 61
- runDebugJobSeq
 - main.cpp, 112
- runDebugSeq
 - cs471::Experiment, 26
- runNEH
 - cs471::Experiment, 27
- sectionExists
 - util::IniReader, 58
- seqSize
 - fshop::FlowshopSolution, 54
- setColLabel
 - mdata::DataTable, 23
- setEntry
 - mdata::DataTable, 24
- src/experiment.cpp, 92, 94
- src/flowshopbasic.cpp, 98, 99
- src/flowshopblocking.cpp, 103, 105
- src/flowshopnowait.cpp, 106, 107
- src/inireader.cpp, 108, 109
- src/main.cpp, 110, 113
- src/neh.cpp, 114, 116
- startTimeMatrix
 - fshop::FlowshopBasic, 41
- stopAndJoinAll
 - ThreadPool, 66
- ThreadPool, 64
 - ~ThreadPool, 65
 - enqueue, 65
 - stopAndJoinAll, 66
 - ThreadPool, 65
- time
 - fshop::JobTimePair, 60
- timesFile
 - cs471::TestParams, 64
- totalFlowTime
 - fshop::FlowshopSolution, 55
- util, 10
 - allocArray, 10
 - allocMatrix, 11
 - copyArray, 12
 - initArray, 12
 - initMatrix, 13

- loadMatrixFromFile, [14](#)
- outputMatrix, [15](#)
- releaseArray, [15](#)
- releaseMatrix, [16](#)
- util::IniReader, [55](#)
 - ~IniReader, [56](#)
 - entryExists, [56](#)
 - getEntry, [57](#)
 - getEntryAs, [57](#)
 - IniReader, [56](#)
 - openFile, [58](#)
 - sectionExists, [58](#)
- validateParams
 - fshop::FlowshopBasic, [39](#)