

pamsi

0.5

Wygenerowano przez Doxygen 1.8.11

Spis treści

1	Strona główna	1
1.1	Dokumentacja klas w repozytorium pamsi.	1
1.2	Przykład uruchomienia testu	1
1.3	Inne przykłady	1
1.3.1	Test sortowania bąbelkowego	1
1.3.2	Test obsługi wyjątków	2
1.3.3	Obsługa stosu	2
2	Indeks hierarchiczny	3
2.1	Hierarchia klas	3
3	Indeks klas	5
3.1	Lista klas	5
4	Indeks plików	7
4.1	Lista plików	7
5	Dokumentacja klas	9
5.1	Dokumentacja szablonu klasy <code>Asoc< T, T2 ></code>	9
5.1.1	Opis szczegółowy	10
5.1.2	Dokumentacja konstruktora i destruktoru	10
5.1.2.1	<code>Asoc(int nOBuckets)</code>	10
5.1.2.2	<code>~Asoc()</code>	10
5.1.3	Dokumentacja funkcji składowych	11
5.1.3.1	<code>add(T key, T2 val)</code>	11

5.1.3.2	find(T position)	11
5.1.3.3	findOne(T position)	11
5.2	Dokumentacja klasy asoc_test	11
5.2.1	Opis szczegółowy	12
5.2.2	Dokumentacja konstruktora i destruktora	12
5.2.2.1	asoc_test()	12
5.2.2.2	asoc_test(int sizeOfTest)	12
5.2.2.3	~asoc_test()	12
5.2.3	Dokumentacja funkcji składowych	13
5.2.3.1	prepare(int sOT)	13
5.2.3.2	run()	13
5.3	Dokumentacja szablonu klasy Bucket< T, T2 >	14
5.3.1	Opis szczegółowy	15
5.3.2	Dokumentacja konstruktora i destruktora	15
5.3.2.1	Bucket()	15
5.3.2.2	Bucket(int ID)	15
5.3.2.3	~Bucket()	15
5.3.3	Dokumentacja funkcji składowych	15
5.3.3.1	add(entry< T, T2 > Ent)	15
5.3.3.2	getID(void)	16
5.3.3.3	lookup(T position)	16
5.3.3.4	lookupWhole(T position)	16
5.3.3.5	printAllElements()	17
5.3.3.6	printFoundElements(void)	17
5.3.3.7	remove(T position)	17
5.3.4	Dokumentacja atrybutów składowych	18
5.3.4.1	temp	18
5.4	Dokumentacja klasy ContinueException	18
5.4.1	Opis szczegółowy	19
5.4.2	Dokumentacja konstruktora i destruktora	19

5.4.2.1	ContinueException()	19
5.4.2.2	ContinueException(std::string description)	19
5.4.3	Dokumentacja funkcji składowych	19
5.4.3.1	Throw()	19
5.5	Dokumentacja klasy CriticalException	20
5.5.1	Opis szczegółowy	20
5.5.2	Dokumentacja konstruktora i destruktora	21
5.5.2.1	CriticalException()	21
5.5.2.2	CriticalException(std::string description)	21
5.5.3	Dokumentacja funkcji składowych	21
5.5.3.1	Throw()	21
5.6	Dokumentacja szablonu klasy entry< T, T2 >	21
5.6.1	Opis szczegółowy	22
5.6.2	Dokumentacja konstruktora i destruktora	22
5.6.2.1	entry()	22
5.6.2.2	entry(T entryKey, T2 entryData)	22
5.6.3	Dokumentacja funkcji składowych	22
5.6.3.1	getKey(void)	22
5.6.3.2	getVal(void)	22
5.6.3.3	operator=(const entry< T, T2 > &read)	23
5.6.4	Dokumentacja przyjaciół i funkcji związanych	23
5.6.4.1	operator<	23
5.6.4.2	operator<<	23
5.6.4.3	operator<=	23
5.6.4.4	operator==	23
5.6.4.5	operator>	23
5.6.4.6	operator>=	23
5.6.4.7	operator>>	23
5.7	Dokumentacja klasy ExceptionBase	24
5.7.1	Opis szczegółowy	24

5.7.2	Dokumentacja konstruktora i destruktora	24
5.7.2.1	ExceptionBase()	24
5.7.2.2	ExceptionBase(std::string description)	25
5.7.3	Dokumentacja funkcji składowych	25
5.7.3.1	Throw()	25
5.7.4	Dokumentacja przyjaciół i funkcji związanych	25
5.7.4.1	operator<<	25
5.7.5	Dokumentacja atrybutów składowych	25
5.7.5.1	cause	25
5.8	Dokumentacja klasy Graph	25
5.8.1	Opis szczegółowy	26
5.8.2	Dokumentacja konstruktora i destruktora	26
5.8.2.1	Graph()	26
5.8.2.2	~Graph()	27
5.8.3	Dokumentacja funkcji składowych	27
5.8.3.1	areAdjacent(int index1, int index2)	27
5.8.3.2	BFS(void)	27
5.8.3.3	DFS(void)	28
5.8.3.4	getNeighbours(int index)	29
5.8.3.5	insertEdge(int index1, int index2)	30
5.8.3.6	insertVertex()	30
5.8.3.7	isEmpty(void)	31
5.8.3.8	numberOfEdges(void)	31
5.9	Dokumentacja szablonu klasy ISoc< T, T2 >	31
5.9.1	Opis szczegółowy	32
5.9.2	Dokumentacja konstruktora i destruktora	32
5.9.2.1	~ISoc()	32
5.9.3	Dokumentacja funkcji składowych	32
5.9.3.1	add(T, T2)=0	32
5.9.3.2	find(T)=0	33

5.9.3.3	findOne(T)=0	33
5.9.4	Dokumentacja przyjaciół i funkcji związanych	33
5.9.4.1	operator<<	33
5.10	Dokumentacja szablonu klasy IBucket< T, T2 >	34
5.10.1	Opis szczegółowy	35
5.10.2	Dokumentacja konstruktora i destruktora	35
5.10.2.1	~IBucket()	35
5.10.3	Dokumentacja funkcji składowych	35
5.10.3.1	add(entry< T, T2 >)=0	35
5.10.3.2	getID(void)=0	35
5.10.3.3	lookup(T)=0	35
5.10.3.4	lookupWhole(T)=0	35
5.10.3.5	printAllElements()=0	35
5.10.3.6	printFoundElements(void)=0	35
5.10.3.7	remove(T)=0	36
5.11	Dokumentacja klasy IGraph	36
5.11.1	Opis szczegółowy	36
5.11.2	Dokumentacja konstruktora i destruktora	37
5.11.2.1	~IGraph()	37
5.11.3	Dokumentacja funkcji składowych	37
5.11.3.1	areAdjacent(int, int)=0	37
5.11.3.2	BFS(void)=0	38
5.11.3.3	DFS(void)=0	38
5.11.3.4	getNeighbours(int)=0	39
5.11.3.5	insertEdge(int, int)=0	39
5.11.3.6	insertVertex(void)=0	40
5.11.3.7	isEmpty(void)=0	40
5.11.3.8	numberOfEdges(void)=0	41
5.12	Dokumentacja szablonu klasy IKolejka< T >	41
5.12.1	Opis szczegółowy	42

5.12.2	Dokumentacja konstruktora i destruktora	42
5.12.2.1	~IKolejka()	42
5.12.3	Dokumentacja funkcji składowych	43
5.12.3.1	dequeue(void)=0	43
5.12.3.2	enqueue(T)=0	43
5.12.3.3	get(void)=0	43
5.12.3.4	isEmpty(void)=0	44
5.12.4	Dokumentacja przyjaciół i funkcji związanych	44
5.12.4.1	operator<<	44
5.13	Dokumentacja szablonu klasy ILista< T >	45
5.13.1	Opis szczegółowy	45
5.13.2	Dokumentacja konstruktora i destruktora	46
5.13.2.1	~ILista()	46
5.13.3	Dokumentacja funkcji składowych	46
5.13.3.1	add(T, int)=0	46
5.13.3.2	add(T)=0	46
5.13.3.3	get(int)=0	46
5.13.3.4	isEmpty(void)=0	47
5.13.3.5	qs(int, int)=0	47
5.13.3.6	remove(int)=0	48
5.13.3.7	remove(void)=0	48
5.13.3.8	size(void)=0	48
5.14	Dokumentacja szablonu klasy IQueue< T >	49
5.14.1	Opis szczegółowy	50
5.14.2	Dokumentacja konstruktora i destruktora	50
5.14.2.1	~IQueue()	50
5.14.3	Dokumentacja funkcji składowych	50
5.14.3.1	dequeue(void)=0	50
5.14.3.2	enqueue(T)=0	50
5.14.3.3	get(void)=0	51

5.14.3.4	isEmpty(void)=0	51
5.14.4	Dokumentacja przyjaciół i funkcji związanych	52
5.14.4.1	operator<<	52
5.15	Dokumentacja klasy IRunnable	52
5.15.1	Opis szczegółowy	53
5.15.2	Dokumentacja konstruktora i destruktora	53
5.15.2.1	~IRunnable()	53
5.15.3	Dokumentacja funkcji składowych	53
5.15.3.1	prepare(int)=0	53
5.15.3.2	run()=0	53
5.16	Dokumentacja klasy IStoper	53
5.16.1	Opis szczegółowy	54
5.16.2	Dokumentacja konstruktora i destruktora	54
5.16.2.1	~IStoper()	54
5.16.3	Dokumentacja funkcji składowych	54
5.16.3.1	getElapsedTimeMs(void)=0	54
5.16.3.2	start(void)=0	55
5.16.3.3	stop(void)=0	55
5.17	Dokumentacja szablonu klasy IStos< T >	55
5.17.1	Opis szczegółowy	56
5.17.2	Dokumentacja konstruktora i destruktora	56
5.17.2.1	~IStos()	56
5.17.3	Dokumentacja funkcji składowych	56
5.17.3.1	get(void)=0	56
5.17.3.2	isEmpty(void)=0	57
5.17.3.3	pop(void)=0	57
5.17.3.4	push(T)=0	58
5.17.4	Dokumentacja przyjaciół i funkcji związanych	59
5.17.4.1	operator<<	59
5.18	Dokumentacja szablonu klasy Itabn< T >	59

5.18.1	Opis szczegółowy	60
5.18.2	Dokumentacja konstruktora i destruktora	60
5.18.2.1	~ltabn()	60
5.18.3	Dokumentacja funkcji składowych	61
5.18.3.1	add(T)=0	61
5.18.3.2	add(T, int)=0	61
5.18.3.3	aSize(void)=0	62
5.18.3.4	bubblesort()=0	62
5.18.3.5	isEmpty(void)=0	63
5.18.3.6	maxIndex(void)=0	63
5.18.3.7	nOE(void)=0	64
5.18.3.8	operator[](int)=0	64
5.18.3.9	operator[](int) const =0	64
5.18.3.10	remove()=0	65
5.18.3.11	remove(int)=0	65
5.18.3.12	search(T)=0	65
5.18.3.13	searchIndex(T)=0	66
5.18.3.14	searchObject(T)=0	67
5.18.3.15	show(int) const =0	67
5.18.3.16	showElems(void)=0	68
5.18.4	Dokumentacja przyjaciół i funkcji związanych	69
5.18.4.1	operator<<	69
5.19	Dokumentacja szablonu klasy ITreeRB< T >	69
5.19.1	Opis szczegółowy	70
5.19.2	Dokumentacja konstruktora i destruktora	70
5.19.2.1	~ITreeRB()	70
5.19.3	Dokumentacja funkcji składowych	70
5.19.3.1	insert(T)=0	70
5.19.3.2	insert(T, nodeRB< T > *)=0	71
5.19.3.3	leftRot(nodeRB< T > *)=0	71

5.19.3.4	retRoot(void)=0	71
5.19.3.5	rightRot(nodeRB< T > *)=0	71
5.19.3.6	search(T)=0	71
5.19.4	Dokumentacja przyjaciół i funkcji związanych	71
5.19.4.1	operator<<	71
5.20	Dokumentacja szablonu klasy Kolejka< T >	72
5.20.1	Opis szczegółowy	73
5.20.2	Dokumentacja konstruktora i destruktor	73
5.20.2.1	Kolejka()	73
5.20.2.2	~Kolejka()	73
5.20.3	Dokumentacja funkcji składowych	73
5.20.3.1	dequeue(void)	73
5.20.3.2	enqueue(T)	74
5.20.3.3	get(void)	74
5.20.3.4	isEmpty(void)	75
5.21	Dokumentacja szablonu klasy Lista< T >	75
5.21.1	Opis szczegółowy	76
5.21.2	Dokumentacja konstruktora i destruktor	77
5.21.2.1	Lista()	77
5.21.2.2	~Lista()	77
5.21.3	Dokumentacja funkcji składowych	77
5.21.3.1	add(T, int)	77
5.21.3.2	add(T)	78
5.21.3.3	get(int position)	78
5.21.3.4	isEmpty(void)	78
5.21.3.5	qs(int, int)	79
5.21.3.6	remove(int position)	79
5.21.3.7	remove(void)	80
5.21.3.8	size(void)	80
5.22	Dokumentacja klasy lista_test	81

5.22.1	Opis szczegółowy	82
5.22.2	Dokumentacja konstruktora i destruktora	82
5.22.2.1	lista_test()	82
5.22.2.2	~lista_test()	82
5.22.3	Dokumentacja funkcji składowych	82
5.22.3.1	prepare(int sizeOfTest)	82
5.22.3.2	run()	83
5.23	Dokumentacja szablonu klasy node< T >	83
5.23.1	Opis szczegółowy	84
5.23.2	Dokumentacja konstruktora i destruktora	84
5.23.2.1	node(T o)	84
5.23.2.2	node(void)	84
5.23.3	Dokumentacja funkcji składowych	84
5.23.3.1	operator=(const node< T > &read)	84
5.23.4	Dokumentacja przyjaciół i funkcji związanych	85
5.23.4.1	operator<	85
5.23.4.2	operator<<	85
5.23.4.3	operator<=	85
5.23.4.4	operator==	85
5.23.4.5	operator>	85
5.23.4.6	operator>=	85
5.23.5	Dokumentacja atrybutów składowych	85
5.23.5.1	next	85
5.23.5.2	previous	85
5.23.5.3	value	85
5.24	Dokumentacja szablonu klasy nodeRB< T >	86
5.24.1	Opis szczegółowy	87
5.24.2	Dokumentacja konstruktora i destruktora	87
5.24.2.1	nodeRB(T addKey, Colour col=red, nodeRB< T > *addUp=NULL, nodeRB< T > *addLeft=NULL, nodeRB< T > *addRight=NULL)	87
5.24.3	Dokumentacja funkcji składowych	87

5.24.3.1	getColour(void)	87
5.24.3.2	getKey(void)	88
5.24.3.3	getLeft(void)	88
5.24.3.4	getLeftKey(void)	88
5.24.3.5	getParent(void)	89
5.24.3.6	getParentKey(void)	89
5.24.3.7	getRight(void)	89
5.24.3.8	getRightKey(void)	90
5.24.3.9	operator=(const nodeRB< T > &read)	90
5.24.3.10	setColour(Colour colourToSet)	90
5.24.3.11	setKey(T keyToSet)	90
5.24.3.12	setLeft(nodeRB< T > *leftDescendant)	91
5.24.3.13	setParent(nodeRB< T > *up)	91
5.24.3.14	setRight(nodeRB< T > *rightDescendant)	91
5.24.4	Dokumentacja przyjaciół i funkcji związanych	91
5.24.4.1	operator<	91
5.24.4.2	operator<<	92
5.24.4.3	operator<=	92
5.24.4.4	operator==	92
5.24.4.5	operator>	92
5.24.4.6	operator>=	92
5.24.5	Dokumentacja atrybutów składowych	92
5.24.5.1	colour	92
5.24.5.2	key	92
5.24.5.3	left	92
5.24.5.4	right	93
5.24.5.5	up	93
5.25	Dokumentacja szablonu klasy Queue< T >	93
5.25.1	Opis szczegółowy	94
5.25.2	Dokumentacja konstruktora i destruktoru	94

5.25.2.1	Queue(void)	94
5.25.2.2	~Queue()	94
5.25.3	Dokumentacja funkcji składowych	94
5.25.3.1	dequeue(void)	94
5.25.3.2	enqueue(T element)	95
5.25.3.3	get(void)	95
5.25.3.4	isEmpty(void)	95
5.26	Dokumentacja klasy Stoper	95
5.26.1	Opis szczegółowy	96
5.26.2	Dokumentacja funkcji składowych	96
5.26.2.1	getElapsedTimeMs(void)	96
5.26.2.2	start(void)	97
5.26.2.3	stop(void)	97
5.27	Dokumentacja szablonu klasy Stos< T >	97
5.27.1	Opis szczegółowy	98
5.27.2	Dokumentacja konstruktora i destruktor	99
5.27.2.1	Stos()	99
5.27.2.2	~Stos()	99
5.27.3	Dokumentacja funkcji składowych	99
5.27.3.1	get(void)	99
5.27.3.2	isEmpty(void)	100
5.27.3.3	pop(void)	100
5.27.3.4	push(T)	101
5.28	Dokumentacja szablonu klasy tabn< T >	101
5.28.1	Opis szczegółowy	103
5.28.2	Dokumentacja konstruktora i destruktor	103
5.28.2.1	tabn()	103
5.28.2.2	~tabn()	103
5.28.3	Dokumentacja funkcji składowych	104
5.28.3.1	add(T)	104

5.28.3.2	<code>add(T, int)</code>	105
5.28.3.3	<code>aSize(void)</code>	105
5.28.3.4	<code>bubblesort(void)</code>	105
5.28.3.5	<code>isEmpty(void)</code>	106
5.28.3.6	<code>maxIndex(void)</code>	106
5.28.3.7	<code>nOE(void)</code>	106
5.28.3.8	<code>operator[](int index)</code>	107
5.28.3.9	<code>operator[](int index) const</code>	107
5.28.3.10	<code>remove()</code>	108
5.28.3.11	<code>remove(int)</code>	108
5.28.3.12	<code>search(T)</code>	109
5.28.3.13	<code>searchIndex(T)</code>	109
5.28.3.14	<code>searchObject(T)</code>	110
5.28.3.15	<code>show(int) const</code>	110
5.28.3.16	<code>showElems(void)</code>	111
5.29	Dokumentacja klasy <code>tabn_test</code>	111
5.29.1	Opis szczegółowy	112
5.29.2	Dokumentacja konstruktora i destruktor	112
5.29.2.1	<code>tabn_test()</code>	112
5.29.2.2	<code>~tabn_test()</code>	112
5.29.3	Dokumentacja funkcji składowych	112
5.29.3.1	<code>prepare(int sizeOfTest)</code>	112
5.29.3.2	<code>run()</code>	113
5.30	Dokumentacja klasy <code>test_graph_BFS</code>	114
5.30.1	Opis szczegółowy	114
5.30.2	Dokumentacja konstruktora i destruktor	115
5.30.2.1	<code>test_graph_BFS()</code>	115
5.30.3	Dokumentacja funkcji składowych	115
5.30.3.1	<code>prepare(int testSize)</code>	115
5.30.3.2	<code>run(void)</code>	115

5.31	Dokumentacja klasy <code>test_graph_DFS</code>	116
5.31.1	Opis szczegółowy	116
5.31.2	Dokumentacja konstruktora i destruktora	117
5.31.2.1	<code>test_graph_DFS()</code>	117
5.31.3	Dokumentacja funkcji składowych	117
5.31.3.1	<code>prepare(int testSize)</code>	117
5.31.3.2	<code>run(void)</code>	117
5.32	Dokumentacja klasy <code>tree_test</code>	118
5.32.1	Opis szczegółowy	119
5.32.2	Dokumentacja konstruktora i destruktora	119
5.32.2.1	<code>tree_test()</code>	119
5.32.2.2	<code>~tree_test()</code>	119
5.32.3	Dokumentacja funkcji składowych	119
5.32.3.1	<code>prepare(int sizeOfTest)</code>	119
5.32.3.2	<code>run()</code>	120
5.33	Dokumentacja szablonu klasy <code>TreeRB< T ></code>	120
5.33.1	Opis szczegółowy	121
5.33.2	Dokumentacja konstruktora i destruktora	121
5.33.2.1	<code>TreeRB()</code>	121
5.33.2.2	<code>~TreeRB()</code>	122
5.33.3	Dokumentacja funkcji składowych	122
5.33.3.1	<code>insert(T element)</code>	122
5.33.3.2	<code>insert(T element, nodeRB< T > *node)</code>	122
5.33.3.3	<code>leftRot(nodeRB< T > *nd)</code>	122
5.33.3.4	<code>retRoot(void)</code>	123
5.33.3.5	<code>rightRot(nodeRB< T > *nd)</code>	123
5.33.3.6	<code>search(T k)</code>	123

6 Dokumentacja plików	125
6.1 Dokumentacja pliku asoc.cpp	125
6.2 Dokumentacja pliku asoc.hh	125
6.3 Dokumentacja pliku except.cpp	126
6.4 Dokumentacja pliku except.hh	127
6.4.1 Opis szczegółowy	128
6.4.2 Dokumentacja funkcji	128
6.4.2.1 what(ExceptT &except)	128
6.5 Dokumentacja pliku graph.cpp	129
6.6 Dokumentacja pliku graph.hh	129
6.7 Dokumentacja pliku graph2.cpp	131
6.8 Dokumentacja pliku graph2.hh	131
6.9 Dokumentacja pliku hash.cpp	132
6.10 Dokumentacja pliku hash.hh	132
6.11 Dokumentacja pliku kolejka.cpp	134
6.12 Dokumentacja pliku kolejka.hh	134
6.13 Dokumentacja pliku lista.cpp	136
6.14 Dokumentacja pliku lista.hh	136
6.15 Dokumentacja pliku main.cpp	138
6.15.1 Opis szczegółowy	138
6.15.2 Dokumentacja funkcji	138
6.15.2.1 dumpToFile(string nameOfFile, unsigned int testsize, IStoper *stoper)	138
6.15.2.2 main(void)	139
6.15.2.3 printOnscreen(unsigned int testsize, IStoper *stoper)	139
6.16 Dokumentacja pliku main.hh	140
6.16.1 Dokumentacja funkcji	141
6.16.1.1 dumpToFile(std::string, unsigned int, IStoper *)	141
6.16.1.2 printOnscreen(unsigned int, IStoper *)	141
6.17 Dokumentacja pliku run.cpp	142
6.18 Dokumentacja pliku run.hh	142

6.18.1	Opis szczegółowy	143
6.19	Dokumentacja pliku stoper.cpp	144
6.20	Dokumentacja pliku stoper.hh	144
6.21	Dokumentacja pliku stos.cpp	145
6.22	Dokumentacja pliku stos.hh	146
6.23	Dokumentacja pliku tabl.cpp	148
6.24	Dokumentacja pliku tabl.hh	148
6.24.1	Opis szczegółowy	150
6.24.2	Dokumentacja definicji	150
6.24.2.1	SIZE	150
6.25	Dokumentacja pliku tree.cpp	150
6.25.1	Dokumentacja funkcji	151
6.25.1.1	operator<<(std::ostream &output, Colour col)	151
6.26	Dokumentacja pliku tree.hh	151
6.26.1	Dokumentacja typów wyliczanych	152
6.26.1.1	Colour	152
6.26.2	Dokumentacja funkcji	153
6.26.2.1	operator<<(std::ostream &, Colour)	153
Indeks		155

Rozdział 1

Strona główna

1.1 Dokumentacja klas w repozytorium pamsi.

Ten dokument zawiera dokumentację klas znajdujących się w plikach repozytorium pamsi.

1.2 Przykład uruchomienia testu

```
//Poniższy test wymaga, aby w folderze projektu znajdował się słownik o nazwie zadanej w metodzie virtual
bool lista_test::prepare(int) . Należy dokonać edycji w/w metody w celu zmian. Trawją prace nad rozwiązaniem
problemu.
IRunnable * runner = new lista_test;
IStoper * stoper = new Stoper;
unsigned int testSize = 100;
string outputFile = "file123";
try {
    runner->prepare(testSize);
    stoper->start();
    runner->run();
    stoper->stop();
    printOnscreen(testSize, stoper);
    dumpToFile(outputFile, testSize, stoper);
}
catch (ContinueException &cex) {
    std::cout << "Exception: " << cex.getError() << std::endl;
}
catch (CriticalException &crit_ex) {
    std::cout << "Critical: " << crit_ex.getError() << std::endl;
    delete stoper;
    delete runner;
    return -1;
}
catch (...) {
    std::cerr << "Unexpected exception!" << std::endl;
    delete stoper;
    delete runner;
    return -1;
}
delete stoper;
delete runner;
```

1.3 Inne przykłady

1.3.1 Test sortowania bąbelkowego

```
Itabn<int> * tablica = new tabn<int>;
tablica->add(7);
```

```

tablica->add(4);
tablica->add(1);
tablica->add(9);
tablica->add(10);
tablica->add(94);
tablica->add(-4);
tablica->add(5);
tablica->add(15);
tablica->add(8);
tablica->add(9);
tablica->add(17);
tablica->add(19);
tablica->showElems();
tablica->bubblesort();
tablica->showElems();
delete tablica;

```

1.3.2 Test obsługi wyjątków

W poniższym teście powinien wystąpić wyjątek, związany z próbą dodania elementu o indeksie 10, gdy tablica dynamicznie rozszerzalna ma 3 elementy (czyli gdy maksymalny indeks to 2).

```

Itabn<int> * tablica = new tabn<int>;
try {
    tablica->add(1,0);
    tablica->add(2,1);
    tablica->add(6,1);
    tablica->add(10,10);
}
catch (ContinueException &cex) {
    std::cout << "Exception: " << cex.getError() << std::endl;
    delete tablica;
    exit(-1);
}
catch (CriticalException &crit_ex) {
    std::cout << "Critical: " << crit_ex.getError() << std::endl;
    delete tablica;
    exit(-2);
}
catch (...) {
    std::cerr << "Unexpected exception!" << std::endl;
    delete tablica;
    exit(-3);
}
delete tablica;
return 0;

```

1.3.3 Obsługa stosu

```

//Wykorzystanie stosu
IStos<int> * stos = new Stos<int>;
try{
    stos->push(4);
    stos->push(3);
    cout << "TOP: " << stos->pop() << endl; //Powinno być 3
    cout << "TOP: " << stos->get() << endl; //Powinno być 4
    stos->pop();
    if (stos->isEmpty()) cout << "Stos pusty!" << endl; //wykona się
    cout << "-----" << endl;
    stos->pop(); //Wyrzuci wyjątek
}
catch (ContinueException &cex) {
    std::cout << "Exception: " << cex.getError() << std::endl;
    delete stos;
    exit(-1);
}
catch (CriticalException &crit_ex) {
    std::cout << "Critical: " << crit_ex.getError() << std::endl;
    delete stos;
    exit(-2);
}
catch (...) {
    std::cerr << "Unexpected exception!" << std::endl;
    delete stos;
    exit(-3);
}
delete stos;
return 0;

```

Rozdział 2

Indeks hierarchiczny

2.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

entry< T, T2 >	21
IBucket< T, T2 >	34
Bucket< T, T2 >	14
ExceptionBase	24
ContinueException	18
CriticalException	20
IAsoc< T, T2 >	31
Asoc< T, T2 >	9
IAsoc< std::string, int >	31
IGraph	36
Graph	25
IKolejka< T >	41
Kolejka< T >	72
ILista< T >	45
Lista< T >	75
ILista< std::string >	45
IQueue< T >	49
Queue< T >	93
IRunnable	52
asoc_test	11
lista_test	81
tabn_test	111
test_graph_BFS	114
test_graph_DFS	116
tree_test	118
IStoper	53
Stoper	95
IStos< T >	55
Stos< T >	97
Itabn< T >	59
tabn< T >	101
Itabn< Bucket< T, T2 > >	59

ltabn< entry< T, T2 > >	59
ltabn< int >	59
ltabn< ltabn< int > * >	59
ltabn< T2 >	59
ITreeRB< T >	69
TreeRB< T >	120
ITreeRB< int >	69
node< T >	83
nodeRB< T >	86

Rozdział 3

Indeks klas

3.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Asoc< T, T2 >	9
asoc_test	11
Bucket< T, T2 >	14
ContinueException	
Wyjątek, który może spowodować nieprzewidziane działanie programu, ale program mógłby dalej działać	18
CriticalException	
Wyjątek krytyczny, wymagający zamknięcia programu	20
entry< T, T2 >	
Klasa definiująca obiekt typu wpis	21
ExceptionBase	
Ogólny wyjątek	24
Graph	
Klasa implementująca interfejs grafu	25
IAsoc< T, T2 >	31
IBucket< T, T2 >	34
IGraph	
Interfejs grafu	36
IKolejka< T >	
Interfejs klasy Kolejka Definiuje operacje dostępne dla klasy Kolejka	41
ILista< T >	
Interfejs listy	45
IQueue< T >	49
IRunnable	
Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm	52
IStoper	
Plik definiuje stoper, obliczający czas wykonywania badanych funkcji	53
IStos< T >	
Interfejs stosu	55
Itabn< T >	
Interfejs klasy tabn	59
ITreeRB< T >	
Interfejs klasy drzewa czerwono-czarnego	69
Kolejka< T >	
Klasa modeluje kolejkę	72

Lista< T >	
Klasa lista	75
lista_test	
Definiuje sposób testowania wypełniania listy	81
node< T >	
Węzeł kolejki	83
nodeRB< T >	86
Queue< T >	
Kolejka oparta na węzłach	93
Stoper	
Klasa stoper implementująca interfejs IStoper	95
Stos< T >	
Klasa Stos	97
tabn< T >	
Modeluje tablicę dynamicznie rozszerzalną	101
tabn_test	
Definiuje sposób testowania wypełniania tablicy tabn	111
test_graph_BFS	114
test_graph_DFS	116
tree_test	
Klasa testująca drzewo czerwono-czarne	118
TreeRB< T >	
Klasa implementująca interfejs drzewa czerwono-czarnego	120

Rozdział 4

Indeks plików

4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

asoc.cpp	125
asoc.hh	125
except.cpp	126
except.hh	
Plik zawiera definicje wyjątków	127
graph.cpp	129
graph.hh	129
graph2.cpp	131
graph2.hh	131
hash.cpp	132
hash.hh	132
kolejka.cpp	134
kolejka.hh	134
lista.cpp	136
lista.hh	136
main.cpp	
Główny plik programu	138
main.hh	140
run.cpp	142
run.hh	
Plik definiuje interfejs IRunnable , ujednolicający klasy umożliwiające badanie algorytmów	142
stoper.cpp	144
stoper.hh	144
stos.cpp	145
stos.hh	146
tabl.cpp	148
tabl.hh	
Definicja interfejsu Itabn , klasy tabn oraz klasy tabn_test	148
tree.cpp	150
tree.hh	151

Rozdział 5

Dokumentacja klas

5.1 Dokumentacja szablonu klasy Asoc< T, T2 >

```
#include <asoc.hh>
```

Diagram dziedziczenia dla Asoc< T, T2 >

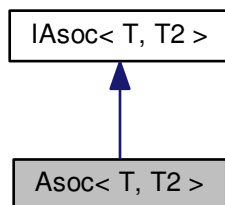
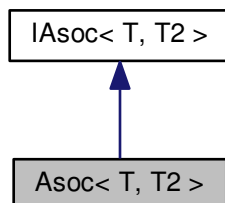


Diagram współpracy dla Asoc< T, T2 >:



Metody publiczne

- [Asoc](#) (int nOBuckets)
- [~Asoc](#) ()
- virtual void [add](#) (T key, T2 val)
- virtual [ltabn](#)< T2 > * [find](#) (T position)
- virtual T2 [findOne](#) (T position)

5.1.1 Opis szczegółowy

```
template<class T, class T2>  
class Asoc< T, T2 >
```

Definicja w linii 35 pliku asoc.hh.

5.1.2 Dokumentacja konstruktora i destruktor

5.1.2.1 `template<class T, class T2> Asoc< T, T2 >::Asoc (int nOBuckets) [inline]`

Definicja w linii 40 pliku asoc.hh.

Oto graf wywołań dla tej funkcji:



5.1.2.2 `template<class T, class T2> Asoc< T, T2 >::~~Asoc () [inline]`

Definicja w linii 48 pliku asoc.hh.

Oto graf wywołań dla tej funkcji:



5.1.3 Dokumentacja funkcji składowych

5.1.3.1 `template<class T, class T2> virtual void Asoc< T, T2 >::add (T key, T2 val) [inline],[virtual]`

Implementuje [IASoc< T, T2 >](#).

Definicja w linii 65 pliku asoc.hh.

Oto graf wywołań dla tej funkcji:



5.1.3.2 `template<class T, class T2> virtual ltabn<T2>* Asoc< T, T2 >::find (T position) [inline],[virtual]`

Implementuje [IASoc< T, T2 >](#).

Definicja w linii 70 pliku asoc.hh.

5.1.3.3 `template<class T, class T2> virtual T2 Asoc< T, T2 >::findOne (T position) [inline],[virtual]`

Implementuje [IASoc< T, T2 >](#).

Definicja w linii 74 pliku asoc.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [asoc.hh](#)

5.2 Dokumentacja klasy asoc_test

```
#include <asoc.hh>
```

Diagram dziedziczenia dla asoc_test

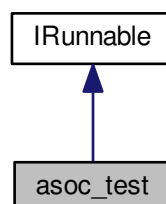
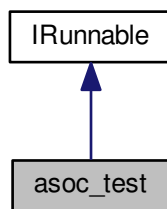


Diagram współpracy dla `asoc_test`:



Metody publiczne

- `asoc_test ()`
- `asoc_test (int sizeofTest)`
- `~asoc_test ()`
- virtual bool `prepare (int sOT)`
Przygotowanie badań
- virtual bool `run ()`
Przeprowadzanie badań

5.2.1 Opis szczegółowy

Definicja w linii 83 pliku `asoc.hh`.

5.2.2 Dokumentacja konstruktora i destruktora

5.2.2.1 `asoc_test::asoc_test () [inline]`

Definicja w linii 92 pliku `asoc.hh`.

5.2.2.2 `asoc_test::asoc_test (int sizeofTest) [inline]`

Definicja w linii 95 pliku `asoc.hh`.

5.2.2.3 `asoc_test::~~asoc_test () [inline]`

Definicja w linii 100 pliku `asoc.hh`.

5.2.3 Dokumentacja funkcji składowych

5.2.3.1 `virtual bool asoc_test::prepare (int) [inline],[virtual]`

Przygotowanie badań

Implementuje [IRunnable](#).

Definicja w linii 106 pliku `asoc.hh`.

Oto graf wywołań dla tej funkcji:



5.2.3.2 `virtual bool asoc_test::run (void) [inline],[virtual]`

Przeprowadzanie badań

Implementuje [IRunnable](#).

Definicja w linii 136 pliku `asoc.hh`.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [asoc.hh](#)

5.3 Dokumentacja szablonu klasy `Bucket< T, T2 >`

```
#include <hash.hh>
```

Diagram dziedziczenia dla `Bucket< T, T2 >`

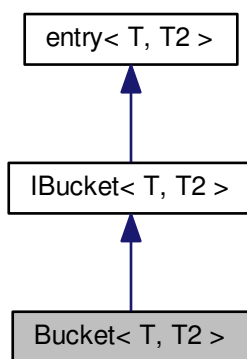
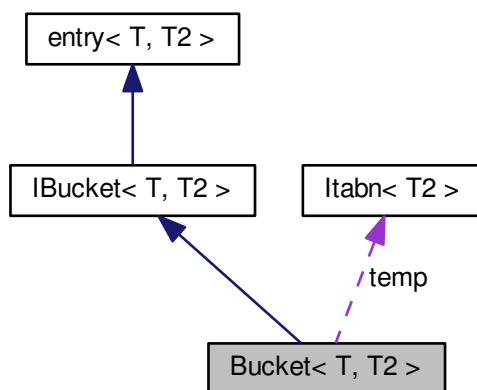


Diagram współpracy dla `Bucket< T, T2 >`:



Metody publiczne

- `Bucket ()`
- `Bucket (int ID)`
- `~Bucket ()`
- `virtual int getID (void)`

- virtual void `printAllElements` ()
- virtual void `printFoundElements` (void)
- virtual void `add` (`entry< T, T2 > Ent`)
- virtual `T2` `remove` (T position)
- virtual `T2` `lookup` (T position)
- virtual `Itabn< T2 > *` `lookupWhole` (T position)

Atrybuty publiczne

- `Itabn< T2 > *` `temp`

5.3.1 Opis szczegółowy

```
template<class T, class T2>
class Bucket< T, T2 >
```

Definicja w linii 111 pliku `hash.hh`.

5.3.2 Dokumentacja konstruktora i destruktor

5.3.2.1 `template<class T, class T2> Bucket< T, T2 >::Bucket ()` `[inline]`

Definicja w linii 120 pliku `hash.hh`.

5.3.2.2 `template<class T, class T2> Bucket< T, T2 >::Bucket (int ID)` `[inline]`

Definicja w linii 124 pliku `hash.hh`.

5.3.2.3 `template<class T, class T2> Bucket< T, T2 >::~~Bucket ()` `[inline]`

Definicja w linii 128 pliku `hash.hh`.

5.3.3 Dokumentacja funkcji składowych

5.3.3.1 `template<class T, class T2> virtual void Bucket< T, T2 >::add (entry< T, T2 > Ent)` `[inline]`,
`[virtual]`

Implementuje `IBucket< T, T2 >`.

Definicja w linii 145 pliku `hash.hh`.

Oto graf wywołań dla tej funkcji:



5.3.3.2 `template<class T, class T2> virtual int Bucket< T, T2 >::getID (void) [inline],[virtual]`

Implementuje `IBucket< T, T2 >`.

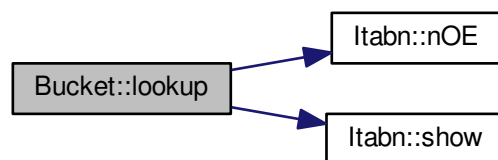
Definicja w linii 133 pliku hash.hh.

5.3.3.3 `template<class T, class T2> virtual T2 Bucket< T, T2 >::lookup (T position) [inline],[virtual]`

Implementuje `IBucket< T, T2 >`.

Definicja w linii 172 pliku hash.hh.

Oto graf wywołań dla tej funkcji:

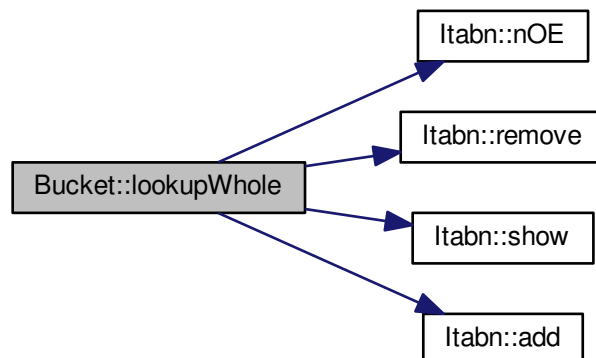


5.3.3.4 `template<class T, class T2> virtual Itabn<T2>* Bucket< T, T2 >::lookupWhole (T position) [inline],[virtual]`

Implementuje `IBucket< T, T2 >`.

Definicja w linii 186 pliku hash.hh.

Oto graf wywołań dla tej funkcji:



5.3.3.5 `template<class T, class T2> virtual void Bucket< T, T2 >::printAllElements () [inline], [virtual]`

Implementuje `IBucket< T, T2 >`.

Definicja w linii 137 pliku `hash.hh`.

Oto graf wywołań dla tej funkcji:

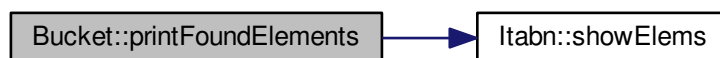


5.3.3.6 `template<class T, class T2> virtual void Bucket< T, T2 >::printFoundElements (void) [inline], [virtual]`

Implementuje `IBucket< T, T2 >`.

Definicja w linii 141 pliku `hash.hh`.

Oto graf wywołań dla tej funkcji:

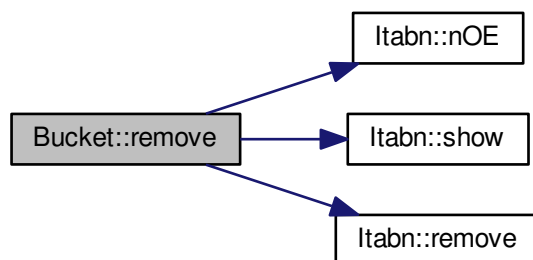


5.3.3.7 `template<class T, class T2> virtual T2 Bucket< T, T2 >::remove (T position) [inline], [virtual]`

Implementuje `IBucket< T, T2 >`.

Definicja w linii 154 pliku `hash.hh`.

Oto graf wywołań dla tej funkcji:



5.3.4 Dokumentacja atrybutów składowych

5.3.4.1 `template<class T, class T2> Itabn<T2>* Bucket< T, T2 >::temp`

Definicja w linii 118 pliku `hash.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [hash.hh](#)

5.4 Dokumentacja klasy `ContinueException`

Wyjątek, który może spowodować nieprzewidziane działanie programu, ale program mógłby dalej działać.

```
#include <except.hh>
```

Diagram dziedziczenia dla `ContinueException`

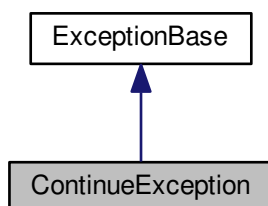
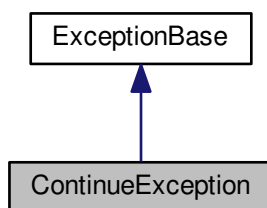


Diagram współpracy dla `ContinueException`:



Metody publiczne

- [ContinueException](#) ()
- [ContinueException](#) (std::string description)
- virtual void [Throw](#) ()

Dodatkowe Dziedziczone Składowe

5.4.1 Opis szczegółowy

Wyjątek, który może spowodować nieprzewidziane działanie programu, ale program mógłby dalej działać.

Definicja w linii 56 pliku `except.hh`.

5.4.2 Dokumentacja konstruktora i destruktor

5.4.2.1 `ContinueException::ContinueException ()` [inline]

Definicja w linii 59 pliku `except.hh`.

5.4.2.2 `ContinueException::ContinueException (std::string description)` [inline]

Definicja w linii 62 pliku `except.hh`.

5.4.3 Dokumentacja funkcji składowych

5.4.3.1 `virtual void ContinueException::Throw ()` [inline],[virtual]

Reimplementowana z [ExceptionBase](#).

Definicja w linii 65 pliku `except.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [except.hh](#)

5.5 Dokumentacja klasy `CriticalException`

Wyjątek krytyczny, wymagający zamknięcia programu.

```
#include <except.hh>
```

Diagram dziedziczenia dla `CriticalException`

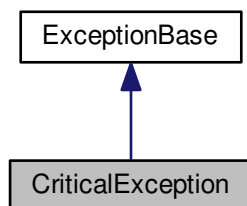
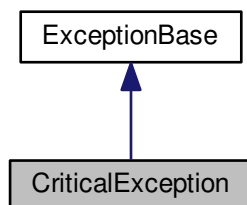


Diagram współpracy dla `CriticalException`:



Metody publiczne

- [CriticalException](#) ()
- [CriticalException](#) (std::string description)
- virtual void [Throw](#) ()

Dodatkowe Dziedziczone Składowe

5.5.1 Opis szczegółowy

Wyjątek krytyczny, wymagający zamknięcia programu.

Definicja w linii 38 pliku `except.hh`.

5.5.2 Dokumentacja konstruktora i destruktora

5.5.2.1 `CriticalException::CriticalException ()` `[inline]`

Definicja w linii 41 pliku `except.hh`.

5.5.2.2 `CriticalException::CriticalException (std::string description)` `[inline]`

Definicja w linii 44 pliku `except.hh`.

5.5.3 Dokumentacja funkcji składowych

5.5.3.1 `virtual void CriticalException::Throw ()` `[inline]`, `[virtual]`

Reimplementowana z [ExceptionBase](#).

Definicja w linii 47 pliku `except.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

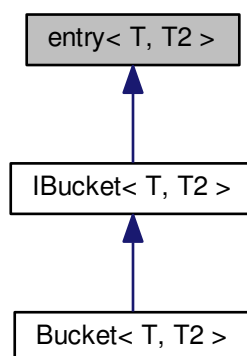
- [except.hh](#)

5.6 Dokumentacja szablonu klasy `entry< T, T2 >`

Klasa definiująca obiekt typu wpis.

```
#include <hash.hh>
```

Diagram dziedziczenia dla `entry< T, T2 >`



Metody publiczne

- `entry()`
- `entry(T entryKey, T2 entryData)`
- `T2 getVal()` (void)
- `T getKey()` (void)
- `entry< T, T2 > & operator= (const entry< T, T2 > &read)`

Przyjaciele

- `bool operator< (entry< T, T2 > one, entry< T, T2 > two)`
- `bool operator> (entry< T, T2 > one, entry< T, T2 > two)`
- `bool operator<= (entry< T, T2 > one, entry< T, T2 > two)`
- `bool operator>= (entry< T, T2 > one, entry< T, T2 > two)`
- `bool operator== (entry< T, T2 > one, entry< T, T2 > two)`
- `std::ostream & operator<< (std::ostream &output, const entry< T, T2 > &to)`
- `std::istream & operator>> (std::istream &input, const entry< T, T2 > &to)`

5.6.1 Opis szczegółowy

```
template<class T, class T2>
class entry< T, T2 >
```

Klasa definiująca obiekt typu wpis.

Definicja w linii 13 pliku hash.hh.

5.6.2 Dokumentacja konstruktora i destruktor

5.6.2.1 `template<class T, class T2> entry< T, T2 >::entry () [inline]`

Definicja w linii 19 pliku hash.hh.

5.6.2.2 `template<class T, class T2> entry< T, T2 >::entry (T entryKey, T2 entryData) [inline]`

Definicja w linii 22 pliku hash.hh.

5.6.3 Dokumentacja funkcji składowych

5.6.3.1 `template<class T, class T2> T entry< T, T2 >::getKey (void) [inline]`

Definicja w linii 34 pliku hash.hh.

5.6.3.2 `template<class T, class T2> T2 entry< T, T2 >::getVal (void) [inline]`

Definicja w linii 30 pliku hash.hh.


```
5.6.3.3  template<class T, class T2> entry<T,T2>& entry< T, T2 >::operator= ( const entry< T, T2 > & read )  
        [inline]
```

Definicja w linii 38 pliku `hash.hh`.

5.6.4 Dokumentacja przyjaciół i funkcji związanych

```
5.6.4.1  template<class T, class T2> bool operator< ( entry< T, T2 > one, entry< T, T2 > two ) [friend]
```

Definicja w linii 44 pliku `hash.hh`.

```
5.6.4.2  template<class T, class T2> std::ostream& operator<< ( std::ostream & output, const entry< T, T2 > & to )  
        [friend]
```

Definicja w linii 69 pliku `hash.hh`.

```
5.6.4.3  template<class T, class T2> bool operator<= ( entry< T, T2 > one, entry< T, T2 > two ) [friend]
```

Definicja w linii 54 pliku `hash.hh`.

```
5.6.4.4  template<class T, class T2> bool operator== ( entry< T, T2 > one, entry< T, T2 > two ) [friend]
```

Definicja w linii 64 pliku `hash.hh`.

```
5.6.4.5  template<class T, class T2> bool operator> ( entry< T, T2 > one, entry< T, T2 > two ) [friend]
```

Definicja w linii 49 pliku `hash.hh`.

```
5.6.4.6  template<class T, class T2> bool operator>= ( entry< T, T2 > one, entry< T, T2 > two ) [friend]
```

Definicja w linii 59 pliku `hash.hh`.

```
5.6.4.7  template<class T, class T2> std::istream& operator>> ( std::istream & input, const entry< T, T2 > & to )  
        [friend]
```

Definicja w linii 74 pliku `hash.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

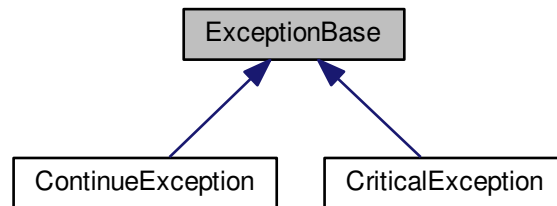
- [hash.hh](#)

5.7 Dokumentacja klasy `ExceptionBase`

Ogólny wyjątek.

```
#include <except.hh>
```

Diagram dziedziczenia dla `ExceptionBase`



Metody publiczne

- `ExceptionBase()`
- `ExceptionBase(std::string description)`
- `virtual void Throw()`

Atrybuty publiczne

- `std::string cause`

Przyjaciele

- `std::ostream & operator<< (std::ostream &output, const ExceptionBase &to)`

5.7.1 Opis szczegółowy

Ogólny wyjątek.

Definicja w linii 15 pliku `except.hh`.

5.7.2 Dokumentacja konstruktora i destruktor

5.7.2.1 `ExceptionBase::ExceptionBase()` `[inline]`

Definicja w linii 19 pliku `except.hh`.

5.7.2.2 `ExceptionBase::ExceptionBase (std::string description) [inline]`

Definicja w linii 22 pliku `except.hh`.

5.7.3 Dokumentacja funkcji składowych

5.7.3.1 `virtual void ExceptionBase::Throw () [inline],[virtual]`

Reimplementowana w [ContinueException](#) i [CriticalException](#).

Definicja w linii 25 pliku `except.hh`.

5.7.4 Dokumentacja przyjaciół i funkcji związanych

5.7.4.1 `std::ostream& operator<< (std::ostream & output, const ExceptionBase & to) [friend]`

Definicja w linii 29 pliku `except.hh`.

5.7.5 Dokumentacja atrybutów składowych

5.7.5.1 `std::string ExceptionBase::cause`

Definicja w linii 17 pliku `except.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [except.hh](#)

5.8 Dokumentacja klasy Graph

Klasa implementująca interfejs grafu.

```
#include <graph.hh>
```

Diagram dziedziczenia dla Graph

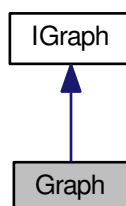
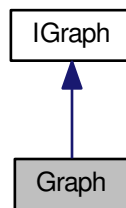


Diagram współpracy dla Graph:



Metody publiczne

- `Graph ()`
Konstruktor klasy `Graph`.
- `virtual ~Graph ()`
Destruktor klasy `Graph`.
- `virtual bool isEmpty (void)`
- `virtual void insertVertex ()`
Dodaje element do grafu.
- `virtual void insertEdge (int index1, int index2)`
Dodaje powiązanie między dwoma elementami.
- `virtual bool areAdjacent (int index1, int index2)`
Sprawdza, czy dwa elementy mają krawędź między sobą
- `virtual ltabn< int > * getNeighbours (int index)`
Zwraca wskaźnik na tablicę elementów będących zbiorem sąsiadów wierzchołka `index`.
- `virtual int numberOfEdges (void)`
Zwraca ilość krawędzi drzewa.
- `virtual ltabn< int > * DFS (void)`
Przeszukuje graf włąb.
- `virtual ltabn< int > * BFS (void)`
Przeszukuje graf wszerz.

5.8.1 Opis szczegółowy

Klasa implementująca interfejs grafu.

Definicja w linii 36 pliku `graph.hh`.

5.8.2 Dokumentacja konstruktora i destruktora

5.8.2.1 `Graph::Graph ()` `[inline]`

Konstruktor klasy `Graph`.

Definicja w linii 51 pliku `graph.hh`.

5.8.2.2 virtual Graph::~~Graph () [inline],[virtual]

Destruktor klasy [Graph](#).

Definicja w linii 61 pliku graph.hh.

5.8.3 Dokumentacja funkcji składowych

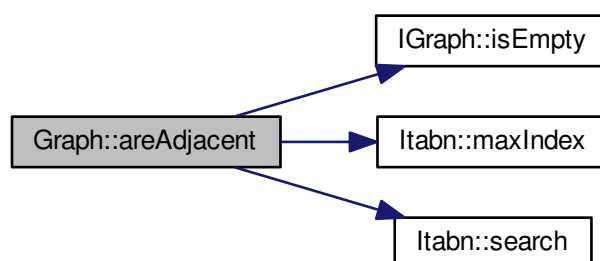
5.8.3.1 virtual bool Graph::areAdjacent (int *index1*, int *index2*) [inline],[virtual]

Sprawdza, czy dwa elementy mają krawędź między sobą

Implementuje [IGraph](#).

Definicja w linii 103 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



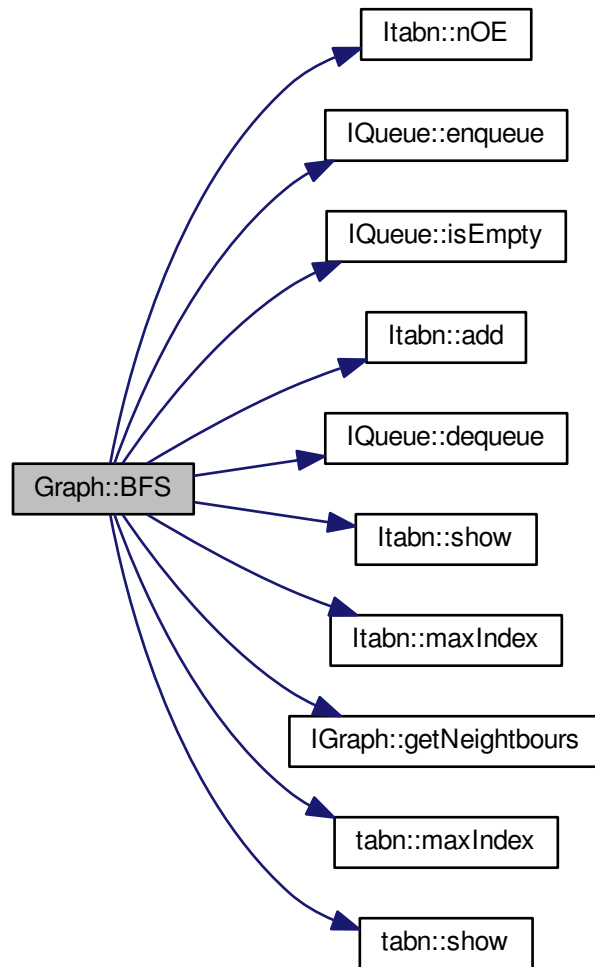
5.8.3.2 virtual Itabn<int>* Graph::BFS (void) [inline],[virtual]

Przeszukuje graf wszerek.

Implementuje [IGraph](#).

Definicja w linii 174 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



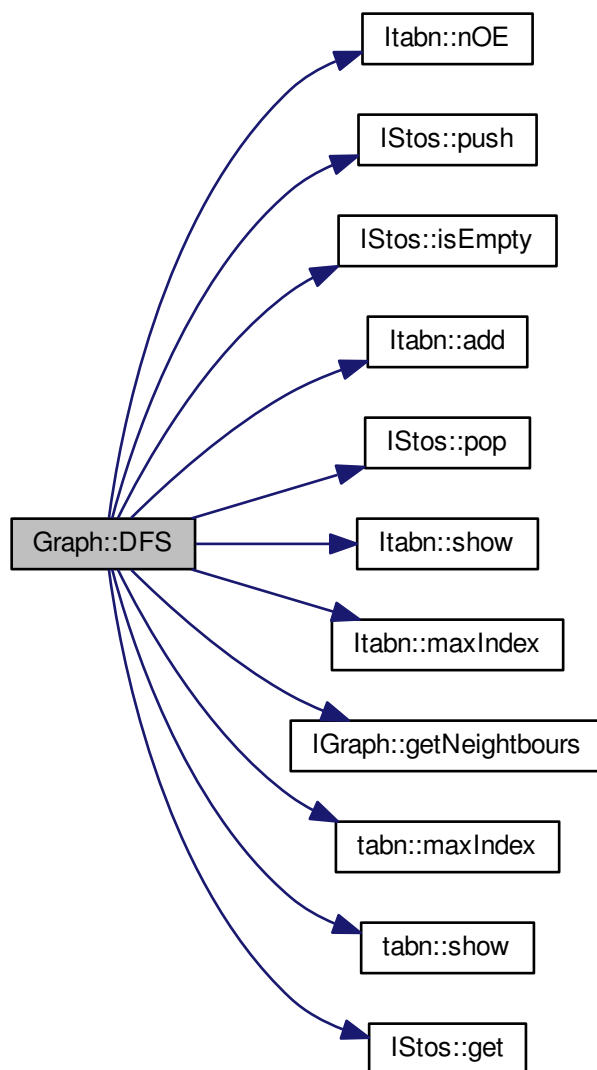
```
5.8.3.3 virtual Itabn<int>* Graph::DFS ( void ) [inline],[virtual]
```

Przeszukuje graf włąb.

Implementuje [IGraph](#).

Definicja w linii 134 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



5.8.3.4 `virtual Itabn<int>* Graph::getNeighbours (int index) [inline],[virtual]`

Zwraca wskaźnik na tablicę elementów będących zbiorem sąsiadów wierzchołka `index`.

Implementuje [IGraph](#).

Definicja w linii 120 pliku `graph.hh`.

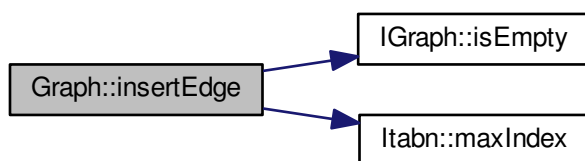
5.8.3.5 `virtual void Graph::insertEdge (int index1, int index2) [inline],[virtual]`

Dodaje powiązanie między dwoma elementami.

Implementuje [IGraph](#).

Definicja w linii 87 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



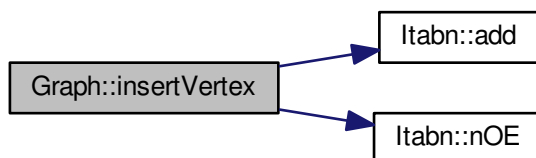
5.8.3.6 `virtual void Graph::insertVertex (void) [inline],[virtual]`

Dodaje element do grafu.

Implementuje [IGraph](#).

Definicja w linii 74 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



5.8.3.7 `virtual bool Graph::isEmpty (void) [inline],[virtual]`

Implementuje [IGraph](#).

Definicja w linii 66 pliku `graph.hh`.

Oto graf wywołań dla tej funkcji:



5.8.3.8 `virtual int Graph::numberOfEdges (void) [inline],[virtual]`

Zwraca ilość krawędzi drzewa.

Implementuje [IGraph](#).

Definicja w linii 127 pliku `graph.hh`.

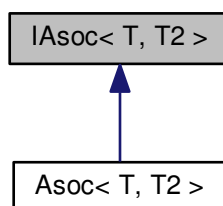
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [graph.hh](#)

5.9 Dokumentacja szablonu klasy `ISoc< T, T2 >`

```
#include <asoc.hh>
```

Diagram dziedziczenia dla `ISoc< T, T2 >`



Metody publiczne

- virtual void `add` (T, T2)=0
- virtual `~IASoc` ()
- virtual `ltabn`< T2 > * `find` (T)=0
- virtual T2 `findOne` (T)=0

Przyjaciele

- std::ostream & `operator<<` (std::ostream &output, `IASoc` *to)

5.9.1 Opis szczegółowy

```
template<class T, class T2>
class IAsoc< T, T2 >
```

Definicja w linii 15 pliku asoc.hh.

5.9.2 Dokumentacja konstruktora i destruktor

5.9.2.1 `template<class T, class T2> virtual IAsoc< T, T2 >::~~IASoc () [inline],[virtual]`

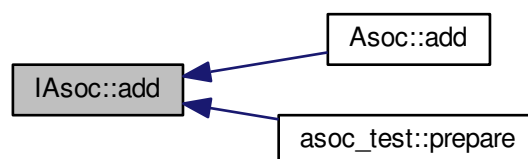
Definicja w linii 24 pliku asoc.hh.

5.9.3 Dokumentacja funkcji składowych

5.9.3.1 `template<class T, class T2> virtual void IAsoc< T, T2 >::add (T , T2) [pure virtual]`

Implementowany w `Asoc< T, T2 >`.

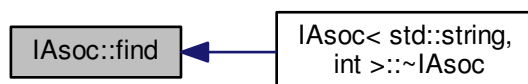
Oto graf wywoływań tej funkcji:



5.9.3.2 `template<class T, class T2> virtual ltabn<T2>* IAsoc< T, T2 >::find (T) [pure virtual]`

Implementowany w `Asoc< T, T2 >`.

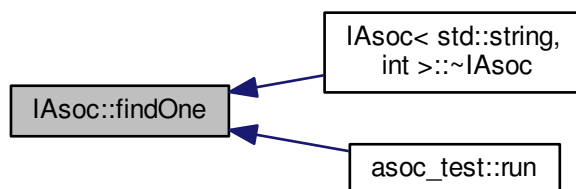
Oto graf wywoływań tej funkcji:



5.9.3.3 `template<class T, class T2> virtual T2 IAsoc< T, T2 >::findOne (T) [pure virtual]`

Implementowany w `Asoc< T, T2 >`.

Oto graf wywoływań tej funkcji:



5.9.4 Dokumentacja przyjaciół i funkcji związanych

5.9.4.1 `template<class T, class T2> std::ostream& operator<< (std::ostream & output, IAsoc< T, T2 > * to) [friend]`

Definicja w linii 28 pliku `asoc.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [asoc.hh](#)

5.10 Dokumentacja szablonu klasy IBucket< T, T2 >

```
#include <hash.hh>
```

Diagram dziedziczenia dla IBucket< T, T2 >

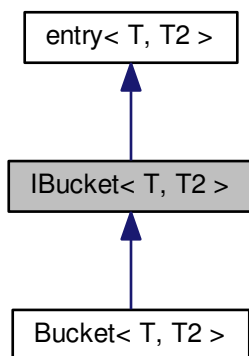
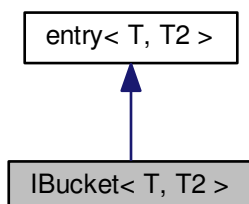


Diagram współpracy dla IBucket< T, T2 >:



Metody publiczne

- virtual void [add](#) ([entry](#) < T, T2 >)=0
- virtual T2 [remove](#) (T)=0
- virtual T2 [lookup](#) (T)=0
- virtual [Itabn](#) < T2 > * [lookupWhole](#) (T)=0
- virtual [~IBucket](#) ()
- virtual void [printAllElements](#) ()=0
- virtual int [getID](#) (void)=0
- virtual void [printFoundElements](#) (void)=0

5.10.1 Opis szczegółowy

```
template<class T, class T2>  
class IBucket< T, T2 >
```

Definicja w linii 92 pliku hash.hh.

5.10.2 Dokumentacja konstruktora i destruktor

5.10.2.1 `template<class T , class T2 > virtual IBucket< T, T2 >::~IBucket () [inline],[virtual]`

Definicja w linii 99 pliku hash.hh.

5.10.3 Dokumentacja funkcji składowych

5.10.3.1 `template<class T , class T2 > virtual void IBucket< T, T2 >::add (entry< T, T2 >) [pure virtual]`

Implementowany w [Bucket< T, T2 >](#).

5.10.3.2 `template<class T , class T2 > virtual int IBucket< T, T2 >::getID (void) [pure virtual]`

Implementowany w [Bucket< T, T2 >](#).

5.10.3.3 `template<class T , class T2 > virtual T2 IBucket< T, T2 >::lookup (T) [pure virtual]`

Implementowany w [Bucket< T, T2 >](#).

5.10.3.4 `template<class T , class T2 > virtual Itabn<T2>* IBucket< T, T2 >::lookupWhole (T) [pure virtual]`

Implementowany w [Bucket< T, T2 >](#).

5.10.3.5 `template<class T , class T2 > virtual void IBucket< T, T2 >::printAllElements () [pure virtual]`

Implementowany w [Bucket< T, T2 >](#).

5.10.3.6 `template<class T , class T2 > virtual void IBucket< T, T2 >::printFoundElements (void) [pure virtual]`

Implementowany w [Bucket< T, T2 >](#).

5.10.3.7 `template<class T, class T2> virtual T2 IBucket< T, T2>::remove (T) [pure virtual]`

Implementowany w `Bucket< T, T2 >`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

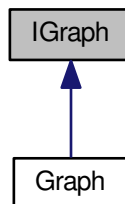
- [hash.hh](#)

5.11 Dokumentacja klasy IGraph

Interfejs grafu.

```
#include <graph.hh>
```

Diagram dziedziczenia dla IGraph



Metody publiczne

- virtual `~IGraph ()`
- virtual bool `isEmpty ()=0`
- virtual void `insertVertex ()=0`
- virtual void `insertEdge (int, int)=0`
- virtual `ltabn< int > * getNeighbours (int)=0`
- virtual bool `areAdjacent (int, int)=0`
- virtual int `numberOfEdges ()=0`
- virtual `ltabn< int > * DFS ()=0`
- virtual `ltabn< int > * BFS ()=0`

5.11.1 Opis szczegółowy

Interfejs grafu.

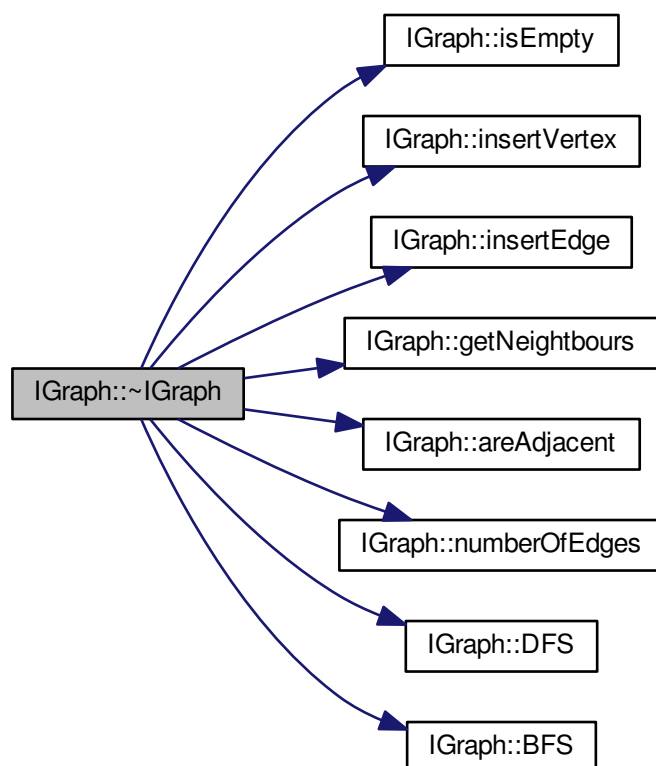
Definicja w linii 13 pliku graph.hh.

5.11.2 Dokumentacja konstruktora i destruktora

5.11.2.1 `virtual IGraph::~IGraph () [inline],[virtual]`

Definicja w linii 16 pliku graph.hh.

Oto graf wywołań dla tej funkcji:

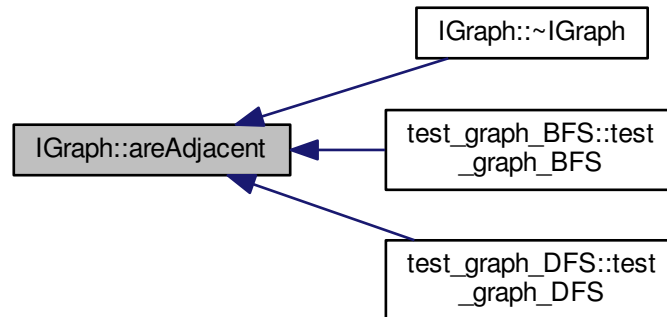


5.11.3 Dokumentacja funkcji składowych

5.11.3.1 `virtual bool IGraph::areAdjacent (int , int) [pure virtual]`

Implementowany w [Graph](#).

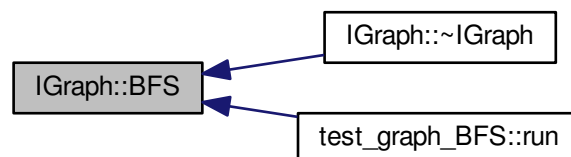
Oto graf wywoływań tej funkcji:



5.11.3.2 `virtual Itabn<int>* IGraph::BFS (void) [pure virtual]`

Implementowany w [Graph](#).

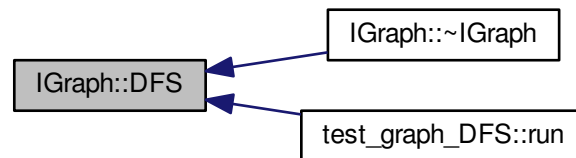
Oto graf wywoływań tej funkcji:



5.11.3.3 `virtual Itabn<int>* IGraph::DFS (void) [pure virtual]`

Implementowany w [Graph](#).

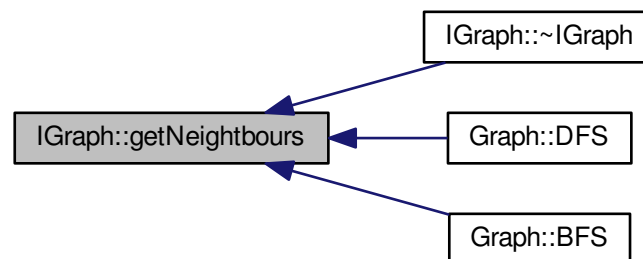
Oto graf wywoływań tej funkcji:



5.11.3.4 `virtual Itabn<int>* IGraph::getNeighbours (int) [pure virtual]`

Implementowany w [Graph](#).

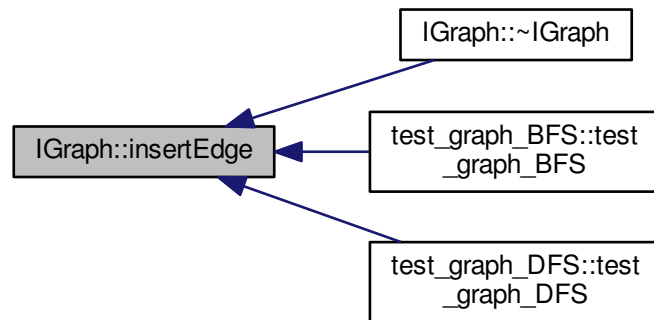
Oto graf wywoływań tej funkcji:



5.11.3.5 `virtual void IGraph::insertEdge (int , int) [pure virtual]`

Implementowany w [Graph](#).

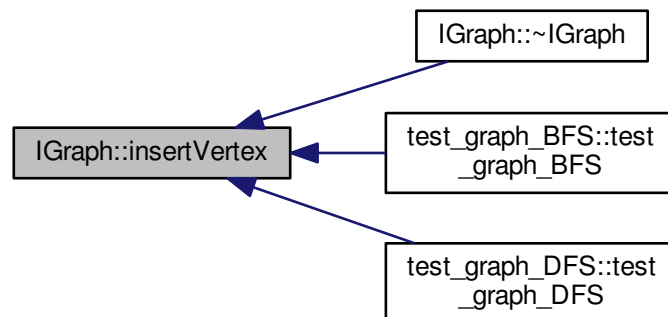
Oto graf wywoływań tej funkcji:



5.11.3.6 `virtual void IGraph::insertVertex (void) [pure virtual]`

Implementowany w [Graph](#).

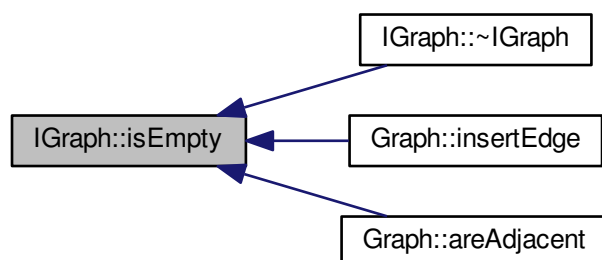
Oto graf wywoływań tej funkcji:



5.11.3.7 `virtual bool IGraph::isEmpty (void) [pure virtual]`

Implementowany w [Graph](#).

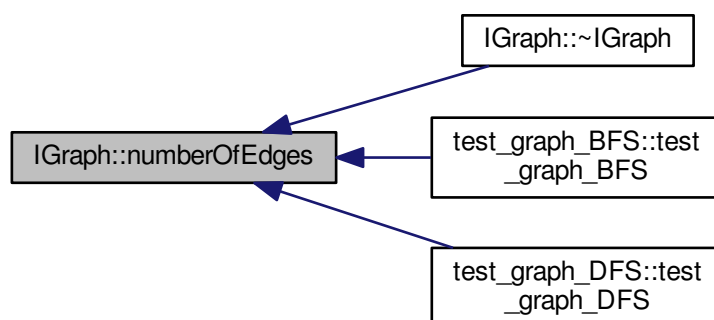
Oto graf wywołań tej funkcji:



5.11.3.8 `virtual int IGraph::numberOfEdges (void) [pure virtual]`

Implementowany w [Graph](#).

Oto graf wywołań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

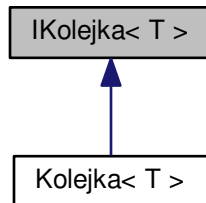
- [graph.hh](#)

5.12 Dokumentacja szablonu klasy IKolejka< T >

Interfejs klasy [Kolejka](#) Definiuje operacje dostępne dla klasy [Kolejka](#).

```
#include <kolejka.hh>
```

Diagram dziedziczenia dla IKolejka< T >



Metody publiczne

- virtual void `enqueue` (T)=0
Dodaje element na koniec kolejki.
- virtual T `dequeue` (void)=0
Usuwa i zwraca element z początku kolejki.
- virtual bool `isEmpty` (void)=0
Sprawdza, czy kolejka nie jest pusta.
- virtual T `get` (void)=0
Zwraca element z początku kolejki bez usuwania.
- virtual `~IKolejka` ()
Destruktor wirtualny interfejsu.

Przyjaciele

- std::ostream & `operator<<` (std::ostream &output, `IKolejka` *to)
Przeciążenie operatora <<.

5.12.1 Opis szczegółowy

```
template<class T>
class IKolejka< T >
```

Interfejs klasy `Kolejka` Definiuje operacje dostępne dla klasy `Kolejka`.

Definicja w linii 15 pliku `kolejka.hh`.

5.12.2 Dokumentacja konstruktora i destruktora

5.12.2.1 `template<class T> virtual IKolejka< T >::~~IKolejka () [inline],[virtual]`

Destruktor wirtualny interfejsu.

Definicja w linii 47 pliku `kolejka.hh`.

5.12.3 Dokumentacja funkcji składowych

5.12.3.1 `template<class T> virtual T IKolejka< T >::dequeue (void) [pure virtual]`

Usuwa i zwraca element z początku kolejki.

Zwracane wartości

<i>T</i>	element z początku kolejki
----------	----------------------------

Implementowany w [Kolejka< T >](#).

Oto graf wywoływań tej funkcji:



5.12.3.2 `template<class T> virtual void IKolejka< T >::enqueue (T) [pure virtual]`

Dodaje element na koniec kolejki.

Parametry

<i>element</i>	- element do umieszczenia w kolejce
----------------	-------------------------------------

Implementowany w [Kolejka< T >](#).

Oto graf wywoływań tej funkcji:



5.12.3.3 `template<class T> virtual T IKolejka< T >::get (void) [pure virtual]`

Zwraca element z początku kolejki bez usuwania.

Ostrzeżenie

Uwaga! Próba podglądu elementu z pustej kolejki spowoduje wyrzucenie wyjątku
Sprawdź dokumentację metody [Kolejka<T>::get\(void\)](#).

Implementowany w [Kolejka< T >](#).

5.12.3.4 `template<class T > virtual bool IKolejka< T >::isEmpty (void) [pure virtual]`

Sprawdza, czy kolejka nie jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementowany w [Kolejka< T >](#).

Oto graf wywoływań tej funkcji:

**5.12.4 Dokumentacja przyjaciół i funkcji związanych**

5.12.4.1 `template<class T > std::ostream& operator<< (std::ostream & output, IKolejka< T > * to) [friend]`

Przeciążenie operatora <<.

Definicja w linii 60 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

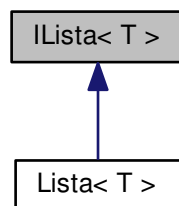
- [kolejka.hh](#)

5.13 Dokumentacja szablonu klasy ILista< T >

Interfejs listy.

```
#include <lista.hh>
```

Diagram dziedziczenia dla ILista< T >



Metody publiczne

- virtual void `add` (T, int)=0
Dodaje element do zadanego miejsca listy.
- virtual void `add` (T)=0
Dodaje element na koniec listy.
- virtual T `remove` (int)=0
Usuwa element z zadanego miejsca listy.
- virtual T `remove` (void)=0
Usuwa element z końca listy.
- virtual bool `isEmpty` (void)=0
Sprawdza, czy lista jest pusta.
- virtual T `get` (int)=0
Zwraca element z zadanego miejsca bez usunięcia.
- virtual int `size` (void)=0
Zwraca ilość elementów w liście.
- virtual void `qs` (int, int)=0
- virtual `~ILista` ()
Destruktor wirtualny interfejsu ILista.

5.13.1 Opis szczegółowy

```
template<class T>  
class ILista< T >
```

Interfejs listy.

Definiuje dostępne operacje na klasie `Lista`

Definicja w linii 17 pliku lista.hh.

5.13.2 Dokumentacja konstruktora i destruktora

5.13.2.1 `template<class T> virtual ILista< T >::~~ILista () [inline],[virtual]`

Destruktor wirtualny interfejsu [ILista](#).

Definicja w linii 75 pliku lista.hh.

5.13.3 Dokumentacja funkcji składowych

5.13.3.1 `template<class T> virtual void ILista< T >::add (T, int) [pure virtual]`

Dodaje element do zadanego miejsca listy.

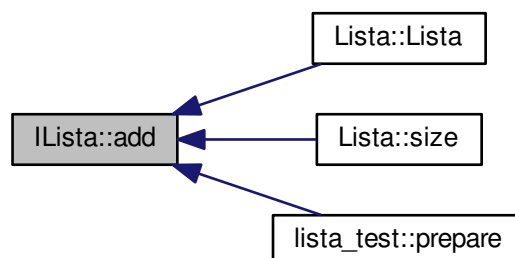
Jeśli następuje próba dodania elementu w miejscu istniejącego, następuje przesunięcie następujących po nim elementów na następne pozycje

Nota

Próba dodania elementu na miejsce dalsze niż pierwsze następujące po obecnie istniejącym spowoduje wyrzucenie wyjątku.

Implementowany w [Lista< T >](#).

Oto graf wywoływań tej funkcji:



5.13.3.2 `template<class T> virtual void ILista< T >::add (T) [pure virtual]`

Dodaje element na koniec listy.

Implementowany w [Lista< T >](#).

5.13.3.3 `template<class T> virtual T ILista< T >::get (int) [pure virtual]`

Zwraca element z zadanego miejsca bez usunięcia.

Zwracane wartości

<i>T</i>	element w zadanym miejscu
----------	---------------------------

Ostrzeżenie

Próba podglądu elementu nieistniejącego spowoduje wyrzucenie wyjątku.
Sprawdź dokumentację metody [Lista<T>::get\(int\)](#).

Implementowany w [Lista< T >](#).

Oto graf wywoływań tej funkcji:

5.13.3.4 `template<class T> virtual bool ILista< T >::isEmpty (void) [pure virtual]`

Sprawdza, czy lista jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

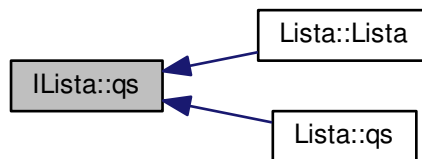
Implementowany w [Lista< T >](#).

Oto graf wywoływań tej funkcji:

5.13.3.5 `template<class T> virtual void ILista< T >::qs (int , int) [pure virtual]`

Implementowany w [Lista< T >](#).

Oto graf wywoływań tej funkcji:



5.13.3.6 `template<class T> virtual T ILista<T>::remove (int) [pure virtual]`

Usuwa element z zadanego miejsca listy.

Jeśli usunięcie następuje w środku listy, następujące po usuwanym elementy są przesuwane o jedną pozycję wcześniej.

Zwracane wartości

<code>T</code>	Usunięty element
----------------	------------------

Ostrzeżenie

Próba usunięcia elementu nieistniejącego lub z pustej listy spowoduje wyrzucenie wyjątku.
Sprawdź dokumentację metody [Lista<T>::remove\(int\)](#).

Implementowany w [Lista<T>](#).

5.13.3.7 `template<class T> virtual T ILista<T>::remove (void) [pure virtual]`

Usuwa element z końca listy.

Implementowany w [Lista<T>](#).

5.13.3.8 `template<class T> virtual int ILista<T>::size (void) [pure virtual]`

Zwraca ilość elementów w liście.

Zwracane wartości

<code>int</code>	ilość elementów
------------------	-----------------

Implementowany w [Lista<T>](#).

Oto graf wywoływań tej funkcji:



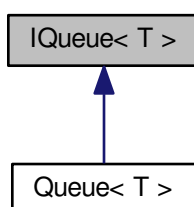
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [lista.hh](#)

5.14 Dokumentacja szablonu klasy IQueue< T >

```
#include <kolejka.hh>
```

Diagram dziedziczenia dla IQueue< T >



Metody publiczne

- virtual void [enqueue](#) (T)=0
Dodaje element na koniec kolejki.
- virtual T [dequeue](#) (void)=0
Usuwa i zwraca element z początku kolejki.
- virtual bool [isEmpty](#) (void)=0
Sprawdza, czy kolejka nie jest pusta.
- virtual T [get](#) (void)=0
Zwraca element z początku kolejki bez usuwania.
- virtual [~IQueue](#) ()
Destruktor wirtualny interfejsu.

Przyjaciele

- `std::ostream & operator<< (std::ostream &output, IQueue *to)`
Przeciążenie operatora `<<`.

5.14.1 Opis szczegółowy

```
template<class T>
class IQueue< T >
```

Definicja w linii 227 pliku kolejka.hh.

5.14.2 Dokumentacja konstruktora i destruktora

5.14.2.1 `template<class T> virtual IQueue< T >::~IQueue () [inline],[virtual]`

Destruktor wirtualny interfejsu.

Definicja w linii 259 pliku kolejka.hh.

5.14.3 Dokumentacja funkcji składowych

5.14.3.1 `template<class T> virtual T IQueue< T >::dequeue (void) [pure virtual]`

Usuwa i zwraca element z początku kolejki.

Zwracane wartości

<code>T</code>	element z początku kolejki
----------------	----------------------------

Implementowany w `Queue< T >`.

Oto graf wywoływań tej funkcji:



5.14.3.2 `template<class T> virtual void IQueue< T >::enqueue (T) [pure virtual]`

Dodaje element na koniec kolejki.

Parametry

<i>element</i>	- element do umieszczenia w kolejce
----------------	-------------------------------------

Implementowany w [Queue< T >](#).

Oto graf wywoływań tej funkcji:



5.14.3.3 `template<class T> virtual T IQueue< T >::get (void) [pure virtual]`

Zwraca element z początku kolejki bez usuwania.

Ostrzeżenie

Uwaga! Próba podglądu elementu z pustej kolejki spowoduje wyrzucenie wyjątku
Sprawdź dokumentację metody [Kolejka<T>::get\(void\)](#).

Implementowany w [Queue< T >](#).

5.14.3.4 `template<class T> virtual bool IQueue< T >::isEmpty (void) [pure virtual]`

Sprawdza, czy kolejka nie jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementowany w [Queue< T >](#).

Oto graf wywoływań tej funkcji:



5.14.4 Dokumentacja przyjaciół i funkcji związanych

5.14.4.1 `template<class T> std::ostream& operator<< (std::ostream & output, IQueue< T > * to) [friend]`

Przeciążenie operatora <<.

Definicja w linii 272 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

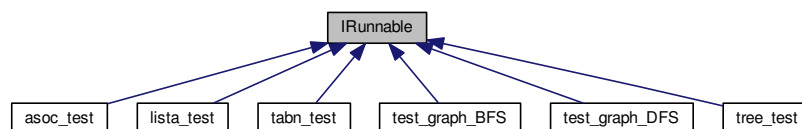
- [kolejka.hh](#)

5.15 Dokumentacja klasy IRunnable

Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.

```
#include <run.hh>
```

Diagram dziedziczenia dla IRunnable



Metody publiczne

- virtual bool `prepare` (int)=0
Przygotowanie badań
- virtual bool `run` ()=0
Przeprowadzanie badań
- virtual `~IRunnable` ()
Destruktor wirtualny IRunnable.

5.15.1 Opis szczegółowy

Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.

Definicja w linii 16 pliku run.hh.

5.15.2 Dokumentacja konstruktora i destruktora

5.15.2.1 `virtual IRunnable::~~IRunnable () [inline],[virtual]`

Destruktor wirtualny [IRunnable](#).

Definicja w linii 31 pliku run.hh.

5.15.3 Dokumentacja funkcji składowych

5.15.3.1 `virtual bool IRunnable::prepare (int) [pure virtual]`

Przygotowanie badań

Implementowany w [tabn_test](#), [tree_test](#), [test_graph_DFS](#), [lista_test](#), [test_graph_BFS](#) i [asoc_test](#).

5.15.3.2 `virtual bool IRunnable::run () [pure virtual]`

Przeprowadzanie badań

Implementowany w [tabn_test](#), [tree_test](#), [test_graph_DFS](#), [lista_test](#), [test_graph_BFS](#) i [asoc_test](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

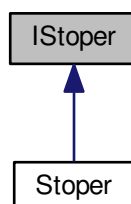
- [run.hh](#)

5.16 Dokumentacja klasy IStoper

Plik definiuje stoper, obliczający czas wykonywania badanych funkcji.

```
#include <stoper.hh>
```

Diagram dziedziczenia dla IStoper



Metody publiczne

- virtual void `start` (void)=0
- virtual void `stop` (void)=0
- virtual long double `getElapsedTimeMs` (void)=0
- virtual `~IStoper` ()

5.16.1 Opis szczegółowy

Plik definiuje stoper, obliczający czas wykonywania badanych funkcji.

Obliczanie czasu działania fragmentu programu na podstawie przykładu: <http://en.cppreference.com/w/cpp/chrono>

Interfejs `IStoper`

Definicja w linii 20 pliku `stoper.hh`.

5.16.2 Dokumentacja konstruktora i destruktor

5.16.2.1 `virtual IStoper::~IStoper () [inline],[virtual]`

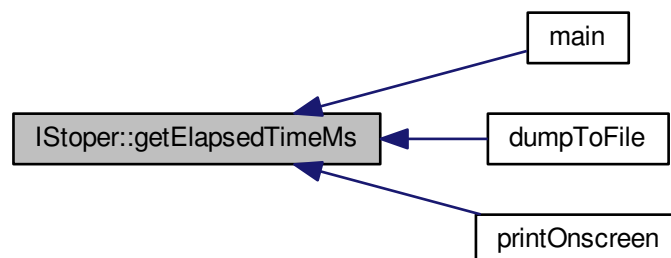
Definicja w linii 25 pliku `stoper.hh`.

5.16.3 Dokumentacja funkcji składowych

5.16.3.1 `virtual long double IStoper::getElapsedTimeMs (void) [pure virtual]`

Implementowany w `Stoper`.

Oto graf wywoływań tej funkcji:



5.16.3.2 `virtual void IStoper::start (void) [pure virtual]`

Implementowany w [Stoper](#).

Oto graf wywołań tej funkcji:



5.16.3.3 `virtual void IStoper::stop (void) [pure virtual]`

Implementowany w [Stoper](#).

Oto graf wywołań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

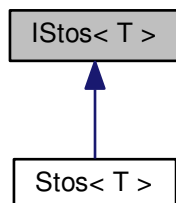
- [stoper.hh](#)

5.17 Dokumentacja szablonu klasy IStos< T >

Interfejs stosu.

```
#include <stos.hh>
```

Diagram dziedziczenia dla IStos< T >



Metody publiczne

- virtual void `push` (T)=0
Umieszcza element na szczycie stosu.
- virtual T `pop` (void)=0
Zdejmuje element ze szczytu stosu.
- virtual bool `isEmpty` (void)=0
Sprawdza, czy stos jest pusty.
- virtual T `get` (void)=0
Zwraca element ze szczytu stosu bez jego usuwania.
- virtual `~IStos` ()
Destruktor wirtualny `IStos`.

Przyjaciele

- std::ostream & `operator<<` (std::ostream &output, `IStos` *to)
Przeciążenie operatora <<.

5.17.1 Opis szczegółowy

```
template<class T>
class IStos< T >
```

Interfejs stosu.

Definiuje dostępne operacje na klasie `Stos`

Definicja w linii 16 pliku stos.hh.

5.17.2 Dokumentacja konstruktora i destruktora

5.17.2.1 `template<class T> virtual IStos< T >::~~IStos () [inline],[virtual]`

Destruktor wirtualny `IStos`.

Definicja w linii 53 pliku stos.hh.

5.17.3 Dokumentacja funkcji składowych

5.17.3.1 `template<class T> virtual T IStos< T >::get (void) [pure virtual]`

Zwraca element ze szczytu stosu bez jego usuwania.

Zwracane wartości

<code>T</code>	element ze szczytu stosu
----------------	--------------------------

Ostrzeżenie

Uwaga! Próba odczytania elementu z pustego stosu spowoduje wyrzucenie wyjątku
Sprawdź dokumentację metody `Stos<T>::get(void)`.

Implementowany w `Stos< T >`.

Oto graf wywoływań tej funkcji:



5.17.3.2 `template<class T> virtual bool IStos< T >::isEmpty (void) [pure virtual]`

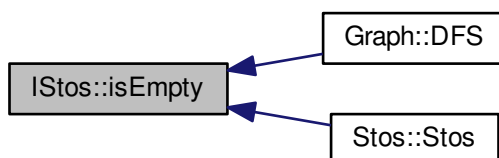
Sprawdza, czy stos jest pusty.

Zwracane wartości

0	gdy niepusty
1	gdy pusty

Implementowany w `Stos< T >`.

Oto graf wywoływań tej funkcji:



5.17.3.3 `template<class T> virtual T IStos< T >::pop (void) [pure virtual]`

Zdejmuje element ze szczytu stosu.

Zwracane wartości

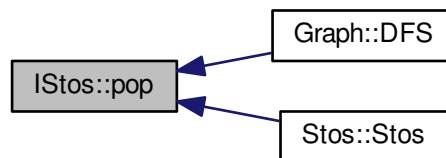
<i>T</i>	element ze szczytu stosu
----------	--------------------------

Ostrzeżenie

Uwaga! Próba zdjęcia elementu z pustego stosu spowoduje wyrzucenie wyjątku
Sprawdź dokumentację metody [Stos<T>::pop\(void\)](#).

Implementowany w [Stos< T >](#).

Oto graf wywoływań tej funkcji:



5.17.3.4 `template<class T> virtual void Istos< T >::push (T) [pure virtual]`

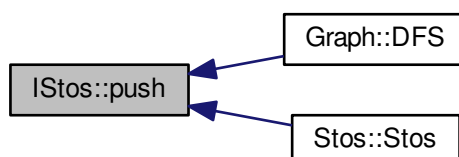
Umieszcza element na szczycie stosu.

Parametry

<i>element</i>	- element do umieszczenia na stosie
----------------	-------------------------------------

Implementowany w [Stos< T >](#).

Oto graf wywoływań tej funkcji:



5.17.4 Dokumentacja przyjaciół i funkcji związanych

5.17.4.1 `template<class T> std::ostream& operator<< (std::ostream & output, IStos< T > * to) [friend]`

Przeciążenie operatora <<.

Definicja w linii 66 pliku stos.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

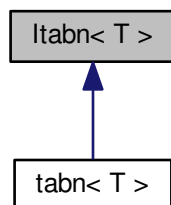
- [stos.hh](#)

5.18 Dokumentacja szablonu klasy Itabn< T >

Interfejs klasy tabn.

```
#include <tabl.hh>
```

Diagram dziedziczenia dla Itabn< T >



Metody publiczne

- virtual bool `isEmpty` (void)=0
Sprawdza, czy tablica jest pusta.
- virtual void `add` (T)=0
Dodaje element na koniec tablicy.
- virtual void `add` (T, int)=0
Dodaje element w dane miejsce do tablicy, przesuwając wszystkie następne elementy o miejsce w prawo.
- virtual T `remove` ()=0
Usuwa i zwraca element z końca tablicy.
- virtual T `remove` (int)=0
Usuwa i zwraca wybrany element z tablicy.
- virtual T `show` (int) const =0
Zwraca żądany element, o ile istnieje.
- virtual void `showElems` (void)=0
Wyświetla elementy tablicy.

- virtual int `nOE` (void)=0
Zwraca liczbę elementów w tablicy.
- virtual int `maxIndex` (void)=0
Zwraca index ostatniego elementu.
- virtual int `aSize` (void)=0
Zwraca ilość miejsca w tablicy.
- virtual T & `operator[]` (int)=0
Pozwala na dostęp do dowolnego elementu.
- virtual T `operator[]` (int) const =0
Pozwala na dostęp do dowolnego elementu.
- virtual `~Itabn` ()
Destruktor wirtualny interfejsu.
- virtual void `bubblesort` ()=0
Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.
- virtual bool `search` (T)=0
znajduje element w tablicy
- virtual int `searchIndex` (T)=0
znajduje element w tablicy
- virtual T `searchObject` (T)=0
znajduje element w tablicy

Przyjaciele

- std::ostream & `operator<<` (std::ostream &output, `Itabn`< T > *to)

5.18.1 Opis szczegółowy

```
template<class T>
class Itabn< T >
```

Interfejs klasy tabn.

Definiuje jednolity sposób dostępu do tablicy rozszerzalnej.

Definicja w linii 20 pliku tabl.hh.

5.18.2 Dokumentacja konstruktora i destruktora

5.18.2.1 `template<class T> virtual Itabn< T >::~~Itabn () [inline], [virtual]`

Destruktor wirtualny interfejsu.

Definicja w linii 88 pliku tabl.hh.

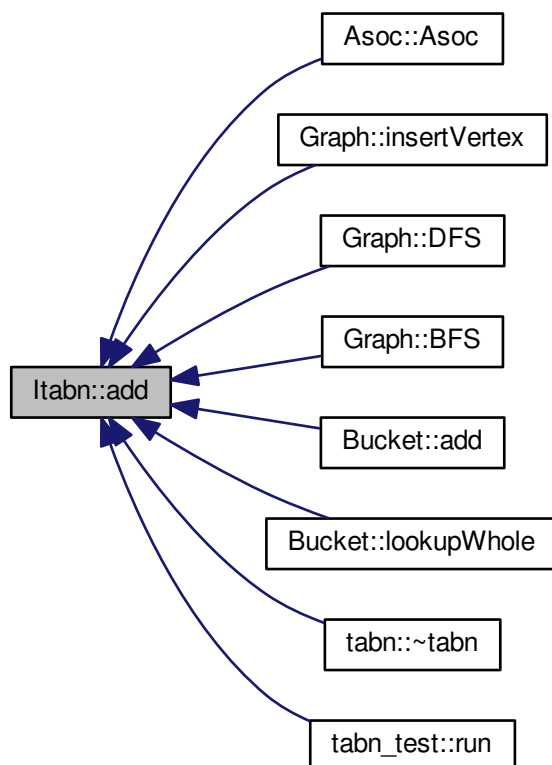
5.18.3 Dokumentacja funkcji składowych

5.18.3.1 `template<class T> virtual void Itabn< T >::add (T) [pure virtual]`

Dodaje element na koniec tablicy.

Implementowany w `tabn< T >`.

Oto graf wywołań tej funkcji:



5.18.3.2 `template<class T> virtual void Itabn< T >::add (T, int) [pure virtual]`

Dodaje element w dane miejsce do tablicy, przesuwając wszystkie następne elementy o miejsce w prawo.

Parametry

<i>element</i>	wstawiany element
<i>position</i>	indeks pola, w które ma być wstawiony element.

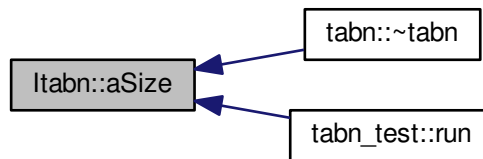
Implementowany w `tabn< T >`.

5.18.3.3 `template<class T> virtual int Itabn< T >::aSize (void) [pure virtual]`

Zwraca ilość miejsca w tablicy.

Implementowany w [tabn< T >](#).

Oto graf wywoływań tej funkcji:



5.18.3.4 `template<class T> virtual void Itabn< T >::bubblesort () [pure virtual]`

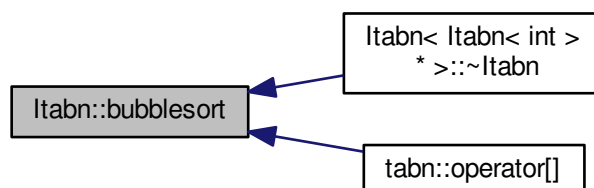
Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.

Ostrzeżenie

Wymaga typu danych ze zdefiniowanym operatorem porównania "większe od"

Implementowany w [tabn< T >](#).

Oto graf wywoływań tej funkcji:



5.18.3.5 `template<class T> virtual bool Itabn< T >::isEmpty (void) [pure virtual]`

Sprawdza, czy tablica jest pusta.

Implementowany w `tabn< T >`.

Oto graf wywołań tej funkcji:

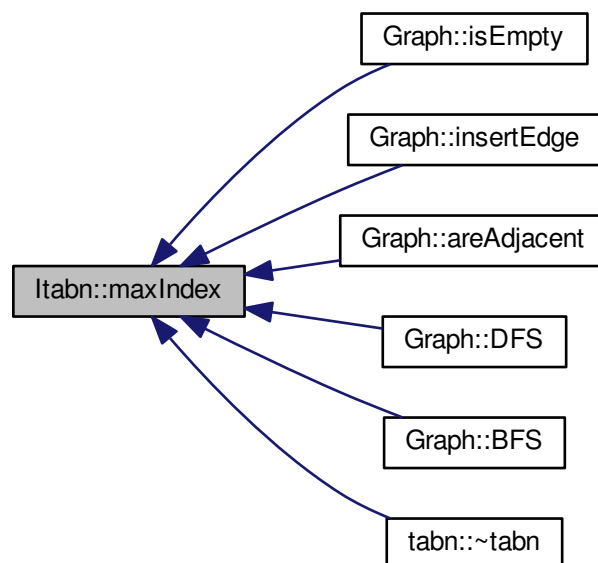


5.18.3.6 `template<class T> virtual int Itabn< T >::maxIndex (void) [pure virtual]`

Zwraca index ostatniego elementu.

Implementowany w `tabn< T >`.

Oto graf wywołań tej funkcji:

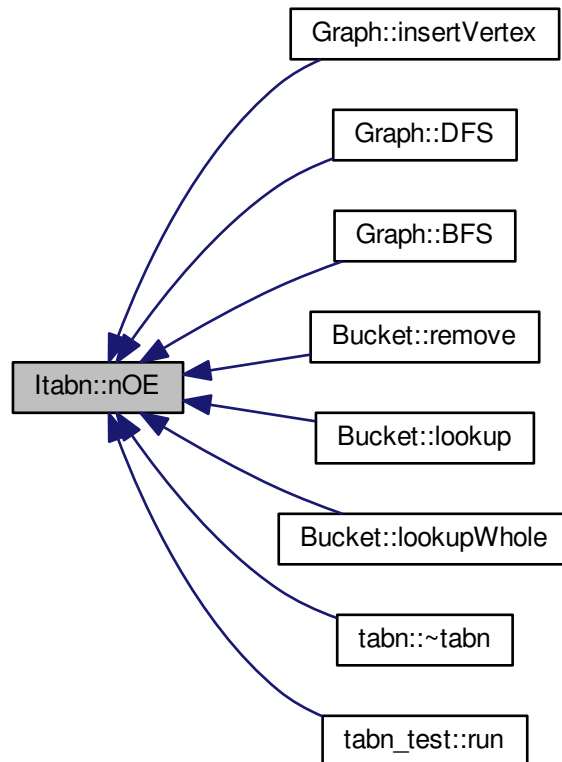


5.18.3.7 `template<class T> virtual int Itabn<T>::nOE (void) [pure virtual]`

Zwraca liczbę elementów w tablicy.

Implementowany w [tabn<T>](#).

Oto graf wywoływań tej funkcji:



5.18.3.8 `template<class T> virtual T& Itabn<T>::operator[] (int) [pure virtual]`

Pozwala na dostęp do dowolnego elementu.

Implementowany w [tabn<T>](#).

5.18.3.9 `template<class T> virtual T Itabn<T>::operator[] (int) const [pure virtual]`

Pozwala na dostęp do dowolnego elementu.

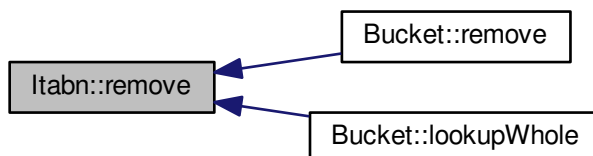
Implementowany w [tabn<T>](#).

5.18.3.10 `template<class T> virtual T ltabn< T >::remove () [pure virtual]`

Usuwa i zwraca element z końca tablicy.

Implementowany w [tabn< T >](#).

Oto graf wywoływań tej funkcji:



5.18.3.11 `template<class T> virtual T ltabn< T >::remove (int) [pure virtual]`

Usuwa i zwraca wybrany element z tablicy.

Parametry

<i>position</i>	indeks pola, z którego ma być usunięty element.
-----------------	---

Implementowany w [tabn< T >](#).

5.18.3.12 `template<class T> virtual bool ltabn< T >::search (T) [pure virtual]`

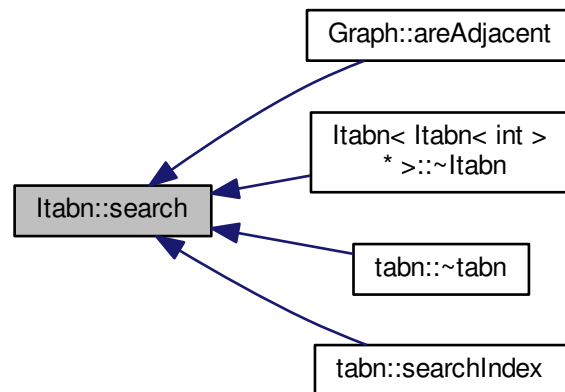
znajduje element w tablicy

Zwracane wartości

<i>true</i>	gdy element został znaleziony
-------------	-------------------------------

Implementowany w [tabn< T >](#).

Oto graf wywołań tej funkcji:



5.18.3.13 `template<class T> virtual int Itabn< T >::searchIndex (T) [pure virtual]`

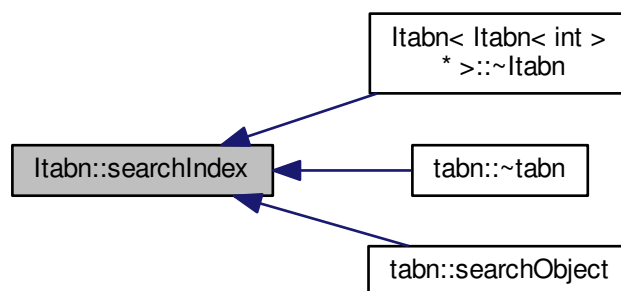
znajduje element w tablicy

Zwracane wartości

<i>indeks</i>	znalezionego elementu
---------------	-----------------------

Implementowany w `tabn< T >`.

Oto graf wywołań tej funkcji:



5.18.3.14 `template<class T> virtual T Itabn< T >::searchObject (T) [pure virtual]`

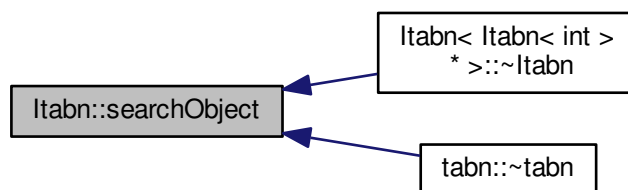
znajduje element w tablicy

Zwracane wartości

<i>indeks</i>	znalezonego elementu
---------------	----------------------

Implementowany w [tabn< T >](#).

Oto graf wywoływań tej funkcji:

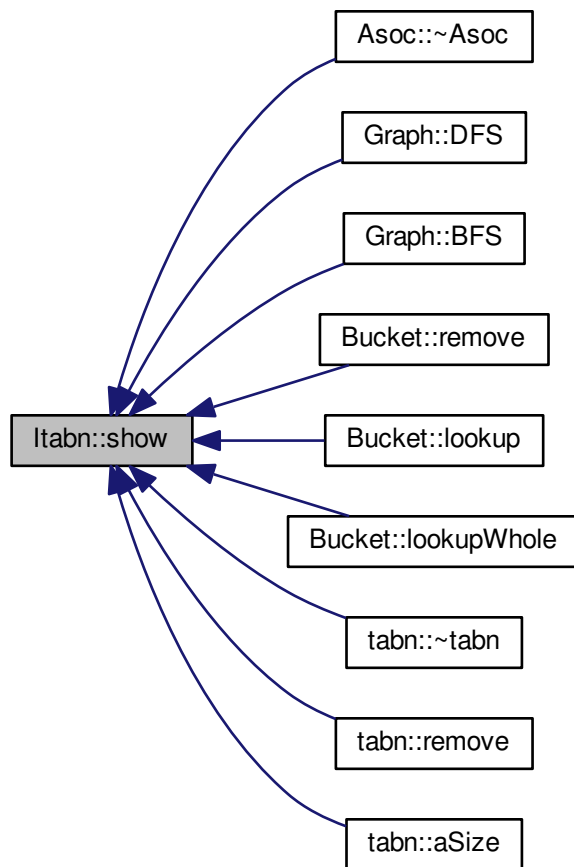


5.18.3.15 `template<class T> virtual T Itabn< T >::show (int) const [pure virtual]`

Zwraca żądany element, o ile istnieje.

Implementowany w [tabn< T >](#).

Oto graf wywoływań tej funkcji:

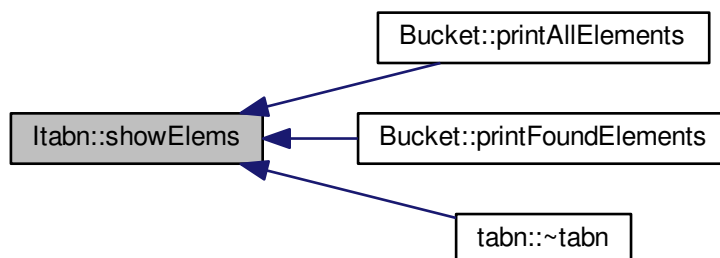


5.18.3.16 `template<class T> virtual void Itabn<T>::showElems (void) [pure virtual]`

Wyświetla elementy tablicy.

Implementowany w `tabn<T>`.

Oto graf wywoływań tej funkcji:



5.18.4 Dokumentacja przyjaciół i funkcji związanych

5.18.4.1 `template<class T> std::ostream& operator<< (std::ostream & output, Itabn< T > * to) [friend]`

Definicja w linii 119 pliku `tabl.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

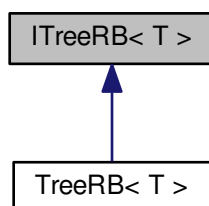
- [tabl.hh](#)

5.19 Dokumentacja szablonu klasy `ITreeRB< T >`

Interfejs klasy drzewa czerwono-czarnego.

```
#include <tree.hh>
```

Diagram dziedziczenia dla `ITreeRB< T >`



Metody publiczne

- virtual void `insert` (T)=0
- virtual void `insert` (T, `nodeRB`< T > *)=0
- virtual bool `search` (T)=0
- virtual `~ITreeRB` ()
- virtual void `leftRot` (`nodeRB`< T > *)=0
- virtual void `rightRot` (`nodeRB`< T > *)=0
- virtual `nodeRB`< T > * `retRoot` (void)=0

Przyjaciele

- std::ostream & `operator<<` (std::ostream &output, `ITreeRB` *to)

5.19.1 Opis szczegółowy

```
template<class T>
class ITreeRB< T >
```

Interfejs klasy drzewa czerwono-czarnego.

Definicja w linii 230 pliku tree.hh.

5.19.2 Dokumentacja konstruktora i destruktor

5.19.2.1 `template<class T> virtual ITreeRB< T >::~ITreeRB () [inline],[virtual]`

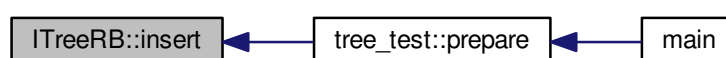
Definicja w linii 237 pliku tree.hh.

5.19.3 Dokumentacja funkcji składowych

5.19.3.1 `template<class T> virtual void ITreeRB< T >::insert (T) [pure virtual]`

Implementowany w `TreeRB< T >`.

Oto graf wywołań tej funkcji:



5.19.3.2 `template<class T> virtual void ITreeRB< T >::insert (T, nodeRB< T > *) [pure virtual]`

Implementowany w [TreeRB< T >](#).

5.19.3.3 `template<class T> virtual void ITreeRB< T >::leftRot (nodeRB< T > *) [pure virtual]`

Implementowany w [TreeRB< T >](#).

5.19.3.4 `template<class T> virtual nodeRB< T > * ITreeRB< T >::retRoot (void) [pure virtual]`

Implementowany w [TreeRB< T >](#).

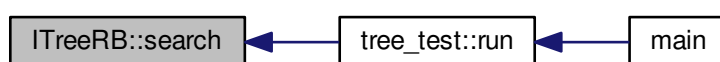
5.19.3.5 `template<class T> virtual void ITreeRB< T >::rightRot (nodeRB< T > *) [pure virtual]`

Implementowany w [TreeRB< T >](#).

5.19.3.6 `template<class T> virtual bool ITreeRB< T >::search (T) [pure virtual]`

Implementowany w [TreeRB< T >](#).

Oto graf wywołań tej funkcji:



5.19.4 Dokumentacja przyjaciół i funkcji związanych

5.19.4.1 `template<class T> std::ostream& operator<< (std::ostream & output, ITreeRB< T > * to) [friend]`

Definicja w linii 242 pliku `tree.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tree.hh](#)

5.20 Dokumentacja szablonu klasy Kolejka< T >

Klasa modeluje kolejkę

```
#include <kolejka.hh>
```

Diagram dziedziczenia dla Kolejka< T >

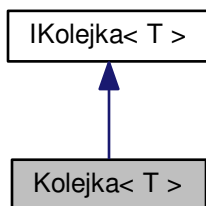
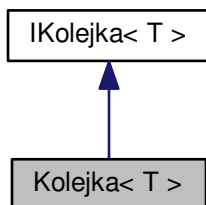


Diagram współpracy dla Kolejka< T >:



Metody publiczne

- `Kolejka ()`
Konstruktor tablicy obsługującej kolejkę
- virtual void `enqueue (T)`
Dodaje element na koniec kolejki.
- virtual T `dequeue (void)`
Usuwa i zwraca element z początku kolejki.
- virtual bool `isEmpty (void)`
Sprawdza, czy kolejka nie jest pusta.
- virtual T `get (void)`
Zwraca element z początku kolejki bez usuwania.
- virtual `~Kolejka ()`
Destruktor klasy Kolejka.

5.20.1 Opis szczegółowy

```
template<class T>  
class Kolejka< T >
```

Klasa modeluje kolejkę

Definicja w linii 70 pliku kolejka.hh.

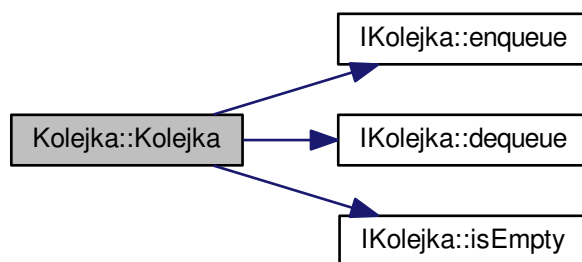
5.20.2 Dokumentacja konstruktora i destruktor

5.20.2.1 `template<class T > Kolejka< T >::Kolejka () [inline]`

Konstruktor tablicy obsługującej kolejkę

Definicja w linii 77 pliku kolejka.hh.

Oto graf wywołań dla tej funkcji:



5.20.2.2 `template<class T > virtual Kolejka< T >::~~Kolejka () [inline],[virtual]`

Destruktor klasy [Kolejka](#).

Definicja w linii 123 pliku kolejka.hh.

5.20.3 Dokumentacja funkcji składowych

5.20.3.1 `template<class T > T Kolejka< T >::dequeue (void) [virtual]`

Usuwa i zwraca element z początku kolejki.

Zwracane wartości

<i>T</i>	element z początku kolejki
----------	----------------------------

Wyjątki

<i>CriticalException</i>	re-throw z <code>tabn<T>::remove()</code>
<i>ContinueException</i>	re-throw z <code>tabn<T>::remove()</code>

Implementuje `IKolejka< T >`.

Definicja w linii 140 pliku kolejka.hh.

5.20.3.2 `template<class T> void Kolejka< T >::enqueue (T element) [virtual]`

Dodaje element na koniec kolejki.

Parametry

<i>element</i>	- element do umieszczenia w kolejce
----------------	-------------------------------------

Implementuje `IKolejka< T >`.

Definicja w linii 134 pliku kolejka.hh.

5.20.3.3 `template<class T> T Kolejka< T >::get (void) [virtual]`

Zwraca element z początku kolejki bez usuwania.

Zwracane wartości

<i>T</i>	element z początku kolejki
----------	----------------------------

Wyjątki

<i>CriticalException</i>	re-throw z <code>tab::show(int)</code>
--------------------------	--

Ostrzeżenie

Uwaga! Próba odczytania elementu z pustej kolejki spowoduje wyrzucenie wyjątku.

Przykład sprawdzenia:

```
//Przykład korzystania z get()
IKolejka<int> * kolejka = new Kolejka<int>;
if (kolejka->isEmpty() == false) {
    cout << kolejka->get() << endl;
}
else
    cerr << "Kolejka pusta" << endl;
```

Implementuje [IKolejka< T >](#).

Definicja w linii 157 pliku kolejka.hh.

5.20.3.4 `template<class T> bool Kolejka< T >::isEmpty (void) [virtual]`

Sprawdza, czy kolejka nie jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementuje [IKolejka< T >](#).

Definicja w linii 152 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [kolejka.hh](#)

5.21 Dokumentacja szablonu klasy Lista< T >

Klasa lista.

```
#include <lista.hh>
```

Diagram dziedziczenia dla Lista< T >

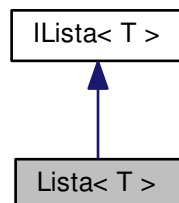
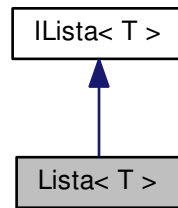


Diagram współpracy dla Lista< T >:



Metody publiczne

- [Lista](#) ()
Konstruktor tablicy obsługującej listę
- virtual void [add](#) (T, int)
Dodaje element do zadanego miejsca listy.
- virtual void [add](#) (T)
Dodaje element na koniec listy.
- virtual T [remove](#) (int position)
Usuwa element z zadanego miejsca listy.
- virtual T [remove](#) (void)
Usuwa element z końca listy.
- virtual bool [isEmpty](#) (void)
Sprawdza, czy lista jest pusta.
- virtual T [get](#) (int position)
Zwraca element z zadanego miejsca bez usunięcia.
- virtual int [size](#) (void)
Zwraca ilość elementów w liście.
- virtual void [qs](#) (int, int)
- virtual [~Lista](#) ()
Destruktor Listy.

5.21.1 Opis szczegółowy

```
template<class T>
class Lista< T >
```

Klasa lista.

Modeluje pojęcie listy

Definicja w linii 84 pliku lista.hh.

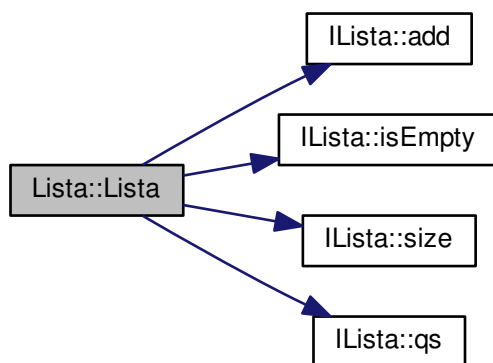
5.21.2 Dokumentacja konstruktora i destruktora

5.21.2.1 `template<class T> Lista< T >::Lista () [inline]`

Konstruktor tablicy obsługującej listę

Definicja w linii 91 pliku `lista.hh`.

Oto graf wywołań dla tej funkcji:



5.21.2.2 `template<class T> virtual Lista< T >::~~Lista () [inline],[virtual]`

Destruktor Listy.

Definicja w linii 183 pliku `lista.hh`.

5.21.3 Dokumentacja funkcji składowych

5.21.3.1 `template<class T> void Lista< T >::add (T element, int position) [virtual]`

Dodaje element do zadanego miejsca listy.

Jeśli następuje próba dodania elementu w miejscu istniejącego, następuje przesunięcie następujących po nim elementów na następne pozycje

Wyjątki

<i>ContinueException</i>	re-throw z <code>tabn<T>::add(T,int)</code>
--	---

Nota

Próba dodania elementu na miejsce dalsze niż pierwsze następujące po obecnie istniejącym spowoduje wyrzucenie wyjątku.

Implementuje [ILista< T >](#).

Definicja w linii 189 pliku lista.hh.

5.21.3.2 `template<class T> void Lista< T >::add (T element) [virtual]`

Dodaje element na koniec listy.

Implementuje [ILista< T >](#).

Definicja w linii 199 pliku lista.hh.

5.21.3.3 `template<class T> T Lista< T >::get (int position) [virtual]`

Zwraca element z zadanego miejsca bez usunięcia.

Zwracane wartości

<i>T</i>	element w zadanym miejscu
----------	---------------------------

Wyjątki

CriticalException	re-throw z <code>tab<T>::show(int)</code>
-----------------------------------	---

Ostrzeżenie

Próba podglądu elementu nieistniejącego spowoduje wyrzucenie wyjątku.
Przykład sprawdzenia:

```
//Przykład sprawdzenia poprawności podglądu
ILista<int> * list = new Lista<int>;
list->add(2); //skomentuj odpowiednio linię aby sprawdzić działanie obu przypadków
int positionToCheckAndShow = 0;
if(list->size()>positionToCheckAndShow) {
    cout << list->get(positionToCheckAndShow) << endl;
}
else
    cerr << "Element nie istnieje!" << endl;
```

Implementuje [ILista< T >](#).

Definicja w linii 226 pliku lista.hh.

5.21.3.4 `template<class T> bool Lista< T >::isEmpty (void) [virtual]`

Sprawdza, czy lista jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementuje [ILista< T >](#).

Definicja w linii 221 pliku lista.hh.

5.21.3.5 `template<class T> void Lista< T>::qs (int indexFront, int indexBack) [virtual]`

Implementuje [ILista< T >](#).

Definicja w linii 261 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



5.21.3.6 `template<class T> T Lista< T>::remove (int position) [virtual]`

Usuwa element z zadanego miejsca listy.

Jeśli usunięcie następuje w środku listy, następujące po usuwanym elementy są przesuwane o jedną pozycję wcześniej.

Zwracane wartości

<i>T</i>	Usunięty element
----------	------------------

Wyjątki

CriticalException	re-throw z tabn<T>::remove()
ContinueException	re-throw z tabn<T>::remove()

Ostrzeżenie

Próba usunięcia elementu nieistniejącego lub z pustej listy spowoduje wyrzucenie wyjątku.

Przykład sprawdzenia:

```
//Przykład sprawdzenia poprawności usuwania
```

```

ILista<int> * list = new Lista<int>;
list->add(2); //skomentuj odpowiednio linię aby sprawdzić działanie obu przypadków
int positionToCheckAndRemove = 0;
if(list->size()>positionToCheckAndRemove) {
    list->remove(positionToCheckAndRemove);
}
else
    cerr << "Element nie istnieje!" << endl;

```

Implementuje `ILista< T >`.

Definicja w linii 204 pliku lista.hh.

5.21.3.7 `template<class T> T Lista< T >::remove (void) [virtual]`

Usuwa element z końca listy.

Implementuje `ILista< T >`.

Definicja w linii 216 pliku lista.hh.

5.21.3.8 `template<class T> int Lista< T >::size (void) [virtual]`

Zwraca ilość elementów w liście.

Zwracane wartości

<code>int</code>	ilość elementów
------------------	-----------------

Implementuje `ILista< T >`.

Definicja w linii 238 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [lista.hh](#)

5.22 Dokumentacja klasy lista_test

Definiuje sposób testowania wypełniania listy.

```
#include <lista.hh>
```

Diagram dziedziczenia dla lista_test

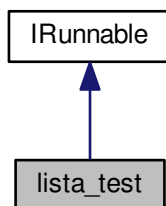
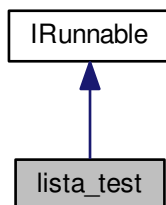


Diagram współpracy dla lista_test:



Metody publiczne

- `lista_test ()`
Konstruktor klasy testującej.
- `~lista_test ()`
Destruktor klasy testującej.
- `virtual bool prepare (int sizeOfTest)`
Przygotowuje rozmiar testu.
- `virtual bool run ()`
Wykonuje test.

5.22.1 Opis szczegółowy

Definiuje sposób testowania wypełniania listy.

Definicja w linii 303 pliku lista.hh.

5.22.2 Dokumentacja konstruktora i destruktora

5.22.2.1 lista_test::lista_test () [inline]

Konstruktor klasy testującej.

Definicja w linii 315 pliku lista.hh.

5.22.2.2 lista_test::~~lista_test () [inline]

Destruktor klasy testującej.

Definicja w linii 321 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



5.22.3 Dokumentacja funkcji składowych

5.22.3.1 virtual bool lista_test::prepare (int *sizeOfTest*) [inline],[virtual]

Przygotowuje rozmiar testu.

Parametry

<i>sizeOfTest</i>	- rozmiar testu
-------------------	-----------------

Zwracane wartości

<i>true</i>	gdy plik ze słownikiem został pomyślnie otwarty
<i>false</i>	gdy otwieranie pliku zakończyło się błędem

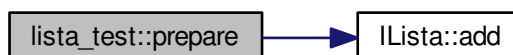
Wyjątki

<i>CriticalException</i>	gdy wystąpił błąd przy otwarciu pliku
--	---------------------------------------

Implementuje [IRunnable](#).

Definicja w linii 359 pliku `lista.hh`.

Oto graf wywołań dla tej funkcji:



5.22.3.2 `virtual bool lista_test::run(void) [inline],[virtual]`

Wykonuje test.

Pozwala na wykonanie testu.

Zwracane wartości

<i>true</i>	gdy test zakończył się sukcesem
<i>false</i>	gdy test zakończył się niepomyślnie

Wyjątki

<i>CriticalException</i>	re-throw z <code>lista_test::wordSearch(std::string)</code>
--	---

Implementuje [IRunnable](#).

Definicja w linii 390 pliku `lista.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [lista.hh](#)

5.23 Dokumentacja szablonu klasy `node< T >`

Węzeł kolejki.

```
#include <kolejka.hh>
```

Metody publiczne

- `node` (`T o`)
- `node` (`void`)
- `node< T > & operator=` (`const node< T > &read`)

Atrybuty publiczne

- `T value`
- `class node< T > * next`
- `class node< T > * previous`

Przyjaciele

- `bool operator<` (`node< T > one, node< T > two`)
- `bool operator>` (`node< T > one, node< T > two`)
- `bool operator<=` (`node< T > one, node< T > two`)
- `bool operator>=` (`node< T > one, node< T > two`)
- `bool operator==` (`node< T > one, node< T > two`)
- `std::ostream & operator<<` (`std::ostream &output, const node< T > &to`)

5.23.1 Opis szczegółowy

```
template<class T>
class node< T >
```

Węzeł kolejki.

Definicja w linii 175 pliku kolejka.hh.

5.23.2 Dokumentacja konstruktora i destruktor

5.23.2.1 `template<class T> node< T >::node (T o)` `[inline]`

Definicja w linii 181 pliku kolejka.hh.

5.23.2.2 `template<class T> node< T >::node (void)` `[inline]`

Definicja w linii 184 pliku kolejka.hh.

5.23.3 Dokumentacja funkcji składowych

5.23.3.1 `template<class T> node< T >& node< T >::operator= (const node< T > & read)` `[inline]`

Definicja w linii 188 pliku kolejka.hh.

5.23.4 Dokumentacja przyjaciół i funkcji związanych

5.23.4.1 `template<class T> bool operator< (node< T > one, node< T > two) [friend]`

Definicja w linii 195 pliku `kolejka.hh`.

5.23.4.2 `template<class T> std::ostream& operator<< (std::ostream & output, const node< T > & to) [friend]`

Definicja w linii 220 pliku `kolejka.hh`.

5.23.4.3 `template<class T> bool operator<= (node< T > one, node< T > two) [friend]`

Definicja w linii 205 pliku `kolejka.hh`.

5.23.4.4 `template<class T> bool operator== (node< T > one, node< T > two) [friend]`

Definicja w linii 215 pliku `kolejka.hh`.

5.23.4.5 `template<class T> bool operator> (node< T > one, node< T > two) [friend]`

Definicja w linii 200 pliku `kolejka.hh`.

5.23.4.6 `template<class T> bool operator>= (node< T > one, node< T > two) [friend]`

Definicja w linii 210 pliku `kolejka.hh`.

5.23.5 Dokumentacja atrybutów składowych

5.23.5.1 `template<class T> class node< T >* node< T >::next`

Definicja w linii 178 pliku `kolejka.hh`.

5.23.5.2 `template<class T> class node< T >* node< T >::previous`

Definicja w linii 179 pliku `kolejka.hh`.

5.23.5.3 `template<class T> T node< T >::value`

Definicja w linii 177 pliku `kolejka.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [kolejka.hh](#)

5.24 Dokumentacja szablonu klasy nodeRB< T >

```
#include <tree.hh>
```

Metody publiczne

- `nodeRB` (T addKey, Colour col=red, nodeRB< T > *addUp=NULL, nodeRB< T > *addLeft=NULL, nodeRB< T > *addRight=NULL)
Konstruktor węzła.
- T getKey (void)
Zwraca klucz węzła.
- Colour getColour (void)
Zwraca kolor węzła.
- nodeRB< T > * getLeft (void)
Zwraca wskaźnik na lewe dziecko.
- nodeRB< T > * getRight (void)
Zwraca wskaźnik na prawe dziecko.
- nodeRB< T > * getParent (void)
Zwraca wskaźnik na rodzica.
- T getLeftKey (void)
Zwraca klucz lewego dziecka.
- T getRightKey (void)
Zwraca klucz prawego dziecka.
- T getParentKey (void)
Zwraca klucz rodzica.
- void setKey (T keyToSet)
Ustawia klucz.
- void setColour (Colour colourToSet)
Ustawia kolor.
- void setLeft (nodeRB< T > *leftDescendant)
Ustawia wskaźnik na lewe dziecko.
- void setRight (nodeRB< T > *rightDescendant)
Ustawia wskaźnik na prawe dziecko.
- void setParent (nodeRB< T > *up)
Ustawia wskaźnik na rodzica.
- nodeRB< T > & operator= (const nodeRB< T > &read)
Przeciążenie operatora przypisania.

Atrybuty publiczne

- T key
- Colour colour
- class nodeRB< T > * left
- class nodeRB< T > * right
- class nodeRB< T > * up

Przyjaciele

- bool `operator<` (nodeRB< T > one, nodeRB< T > two)
Przeciążenie operatora porównania <.
- bool `operator>` (nodeRB< T > one, nodeRB< T > two)
Przeciążenie operatora porównania >.
- bool `operator<=` (nodeRB< T > one, nodeRB< T > two)
Przeciążenie operatora porównania <=.
- bool `operator>=` (nodeRB< T > one, nodeRB< T > two)
Przeciążenie operatora porównania >=.
- bool `operator==` (nodeRB< T > one, nodeRB< T > two)
Przeciążenie operatora porównania ==.
- std::ostream & `operator<<` (std::ostream &output, const nodeRB< T > *to)
Przeciążenie operatora strumienia wyjściowego.

5.24.1 Opis szczegółowy

```
template<class T>
class nodeRB< T >
```

Definicja w linii 23 pliku tree.hh.

5.24.2 Dokumentacja konstruktora i destruktor

5.24.2.1 `template<class T> nodeRB< T >::nodeRB (T addKey, Colour col = red, nodeRB< T > * addUp = NULL, nodeRB< T > * addLeft = NULL, nodeRB< T > * addRight = NULL) [inline]`

Konstruktor węzła.

Definicja w linii 37 pliku tree.hh.

5.24.3 Dokumentacja funkcji składowych

5.24.3.1 `template<class T> Colour nodeRB< T >::getColour (void) [inline]`

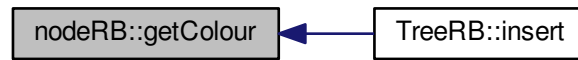
Zwraca kolor węzła.

Zwracane wartości

<code>black</code>	gdy liść
--------------------	----------

Definicja w linii 57 pliku tree.hh.

Oto graf wywoływań tej funkcji:

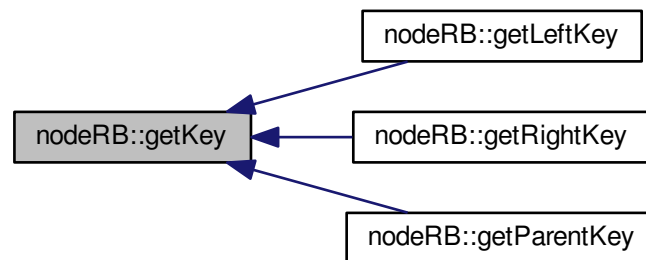


5.24.3.2 `template<class T> T nodeRB< T >::getKey (void) [inline]`

Zwraca klucz węzła.

Definicja w linii 49 pliku `tree.hh`.

Oto graf wywoływań tej funkcji:



5.24.3.3 `template<class T> nodeRB<T>* nodeRB< T >::getLeft (void) [inline]`

Zwraca wskaźnik na lewe dziecko.

Definicja w linii 67 pliku `tree.hh`.

5.24.3.4 `template<class T> T nodeRB< T >::getLeftKey (void) [inline]`

Zwraca klucz lewego dziecka.

Definicja w linii 88 pliku `tree.hh`.

Oto graf wywołań dla tej funkcji:



5.24.3.5 `template<class T> nodeRB<T>* nodeRB< T >::getParent (void)` `[inline]`

Zwraca wskaźnik na rodzica.

Definicja w linii 81 pliku `tree.hh`.

5.24.3.6 `template<class T> T nodeRB< T >::getParentKey (void)` `[inline]`

Zwraca klucz rodzica.

Definicja w linii 108 pliku `tree.hh`.

Oto graf wywołań dla tej funkcji:



5.24.3.7 `template<class T> nodeRB<T>* nodeRB< T >::getRight (void)` `[inline]`

Zwraca wskaźnik na prawe dziecko.

Definicja w linii 74 pliku `tree.hh`.

5.24.3.8 `template<class T> T nodeRB< T >::getRightKey (void) [inline]`

Zwraca klucz prawego dziecka.

Definicja w linii 98 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



5.24.3.9 `template<class T> nodeRB<T>& nodeRB< T >::operator= (const nodeRB< T > & read) [inline]`

Przeciążenie operatora przypisania.

Definicja w linii 154 pliku tree.hh.

5.24.3.10 `template<class T> void nodeRB< T >::setColour (Colour colourToSet) [inline]`

Ustawia kolor.

Definicja w linii 125 pliku tree.hh.

Oto graf wywołań tej funkcji:



5.24.3.11 `template<class T> void nodeRB< T >::setKey (T keyToSet) [inline]`

Ustawia klucz.

Definicja w linii 118 pliku tree.hh.

5.24.3.12 `template<class T> void nodeRB< T >::setLeft (nodeRB< T > * leftDescendant) [inline]`

Ustawia wskaźnik na lewe dziecko.

Definicja w linii 133 pliku `tree.hh`.

Oto graf wywoływań tej funkcji:



5.24.3.13 `template<class T> void nodeRB< T >::setParent (nodeRB< T > * up) [inline]`

Ustawia wskaźnik na rodzica.

Definicja w linii 147 pliku `tree.hh`.

5.24.3.14 `template<class T> void nodeRB< T >::setRight (nodeRB< T > * rightDescendant) [inline]`

Ustawia wskaźnik na prawe dziecko.

Definicja w linii 140 pliku `tree.hh`.

Oto graf wywoływań tej funkcji:



5.24.4 Dokumentacja przyjaciół i funkcji związanych

5.24.4.1 `template<class T> bool operator< (nodeRB< T > one, nodeRB< T > two) [friend]`

Przeciążenie operatora porównania `<`.

Definicja w linii 166 pliku `tree.hh`.

5.24.4.2 `template<class T> std::ostream& operator<< (std::ostream & output, const nodeRB< T > * to) [friend]`

Przeciążenie operatora strumienia wyjściowego.

Definicja w linii 207 pliku tree.hh.

5.24.4.3 `template<class T> bool operator<= (nodeRB< T > one, nodeRB< T > two) [friend]`

Przeciążenie operatora porównania <=.

Definicja w linii 182 pliku tree.hh.

5.24.4.4 `template<class T> bool operator== (nodeRB< T > one, nodeRB< T > two) [friend]`

Przeciążenie operatora porównania ==.

Definicja w linii 199 pliku tree.hh.

5.24.4.5 `template<class T> bool operator> (nodeRB< T > one, nodeRB< T > two) [friend]`

Przeciążenie operatora porównania >

Definicja w linii 174 pliku tree.hh.

5.24.4.6 `template<class T> bool operator>= (nodeRB< T > one, nodeRB< T > two) [friend]`

Przeciążenie operatora porównania >=.

Definicja w linii 191 pliku tree.hh.

5.24.5 Dokumentacja atrybutów składowych

5.24.5.1 `template<class T> Colour nodeRB< T >::colour`

Definicja w linii 26 pliku tree.hh.

5.24.5.2 `template<class T> T nodeRB< T >::key`

Definicja w linii 25 pliku tree.hh.

5.24.5.3 `template<class T> class nodeRB< T > * nodeRB< T >::left`

Definicja w linii 28 pliku tree.hh.

5.24.5.4 `template<class T> class nodeRB< T >* nodeRB< T >::right`

Definicja w linii 29 pliku tree.hh.

5.24.5.5 `template<class T> class nodeRB< T >* nodeRB< T >::up`

Definicja w linii 30 pliku tree.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tree.hh](#)

5.25 Dokumentacja szablonu klasy Queue< T >

[Kolejka](#) oparta na węzłach.

```
#include <kolejka.hh>
```

Diagram dziedziczenia dla Queue< T >

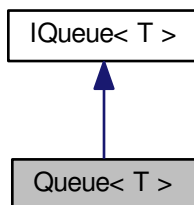
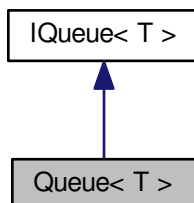


Diagram współpracy dla Queue< T >:



Metody publiczne

- `Queue` (void)
- virtual void `enqueue` (T element)
Dodaje element na koniec kolejki.
- virtual T `dequeue` (void)
Usuwa i zwraca element z początku kolejki.
- virtual bool `isEmpty` (void)
Sprawdza, czy kolejka nie jest pusta.
- virtual T `get` (void)
Zwraca element z początku kolejki bez usuwania.
- virtual `~Queue` ()
Destruktor.

5.25.1 Opis szczegółowy

```
template<class T>
class Queue< T >
```

`Kolejka` oparta na węzłach.

Definicja w linii 283 pliku `kolejka.hh`.

5.25.2 Dokumentacja konstruktora i destruktora

5.25.2.1 `template<class T > Queue< T >::Queue (void) [inline]`

Definicja w linii 289 pliku `kolejka.hh`.

5.25.2.2 `template<class T > virtual Queue< T >::~~Queue () [inline],[virtual]`

Destruktor.

Definicja w linii 363 pliku `kolejka.hh`.

5.25.3 Dokumentacja funkcji składowych

5.25.3.1 `template<class T > virtual T Queue< T >::dequeue (void) [inline],[virtual]`

Usuwa i zwraca element z początku kolejki.

Zwracane wartości

<code>T</code>	element z początku kolejki
----------------	----------------------------

Implementuje [IQueue< T >](#).

Definicja w linii 327 pliku kolejka.hh.

5.25.3.2 `template<class T> virtual void Queue< T >::enqueue (T element) [inline],[virtual]`

Dodaje element na koniec kolejki.

Parametry

<i>element</i>	- element do umieszczenia w kolejce
----------------	-------------------------------------

Implementuje [IQueue< T >](#).

Definicja w linii 310 pliku kolejka.hh.

5.25.3.3 `template<class T> virtual T Queue< T >::get (void) [inline],[virtual]`

Zwraca element z początku kolejki bez usuwania.

Implementuje [IQueue< T >](#).

Definicja w linii 355 pliku kolejka.hh.

5.25.3.4 `template<class T> virtual bool Queue< T >::isEmpty (void) [inline],[virtual]`

Sprawdza, czy kolejka nie jest pusta.

Zwracane wartości

0	gdy niepusta
1	gdy pusta

Implementuje [IQueue< T >](#).

Definicja w linii 347 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [kolejka.hh](#)

5.26 Dokumentacja klasy Stoper

Klasa stoper implementująca interfejs [IStoper](#).

```
#include <stoper.hh>
```

Diagram dziedziczenia dla Stoper

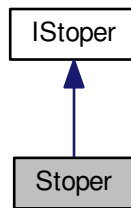
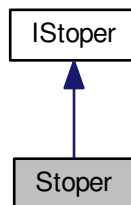


Diagram współpracy dla Stoper:



Metody publiczne

- virtual void **start** (void)
Uruchamia zegar.
- virtual void **stop** (void)
Zatrzymuje zegar.
- virtual long double **getElapsedTimeMs** (void)
Oblicza i zwraca czas pomiędzy uruchomieniem zegara a jego zatrzymaniem.

5.26.1 Opis szczegółowy

Klasa stoper implementująca interfejs **IStoper**.

Klasa symuluje działanie stopera - zapisuje początkowy i końcowy moment działania (użycie start i stop), oraz odejmuje obie te wartości od siebie, by uzyskać czas działania.

Definicja w linii 35 pliku stoper.hh.

5.26.2 Dokumentacja funkcji składowych

5.26.2.1 long double **Stoper::getElapsedTimeMs** (void) [virtual]

Oblicza i zwraca czas pomiędzy uruchomieniem zegara a jego zatrzymaniem.

Zwracane wartości

<code>long_double</code>	Czas pomiędzy startem a zatrzymaniem zegara
--------------------------	---

Implementuje [IStoper](#).

Definicja w linii 12 pliku stoper.cpp.

5.26.2.2 `void Stoper::start (void) [virtual]`

Uruchamia zegar.

Implementuje [IStoper](#).

Definicja w linii 4 pliku stoper.cpp.

5.26.2.3 `void Stoper::stop (void) [virtual]`

Zatrzymuje zegar.

Implementuje [IStoper](#).

Definicja w linii 8 pliku stoper.cpp.

Dokumentacja dla tej klasy została wygenerowana z plików:

- [stoper.hh](#)
- [stoper.cpp](#)

5.27 Dokumentacja szablonu klasy Stos< T >

Klasa [Stos](#).

```
#include <stos.hh>
```

Diagram dziedziczenia dla Stos< T >

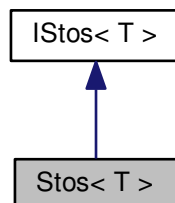
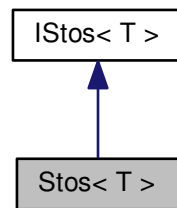


Diagram współpracy dla Stos< T >:



Metody publiczne

- [Stos](#) ()
Konstruktor tablicy obsługującej stos.
- virtual void [push](#) (T)
Umieszcza element na szczycie stosu.
- virtual T [pop](#) (void)
Zdejmuje element ze szczytu stosu.
- virtual bool [isEmpty](#) (void)
Sprawdza, czy stos jest pusty.
- virtual T [get](#) (void)
Zwraca element ze szczytu stosu bez jego usuwania.
- virtual [~Stos](#) ()
Destruktor stosu.

5.27.1 Opis szczegółowy

```
template<class T>
class Stos< T >
```

Klasa [Stos](#).

Modeluje pojęcie stosu

Definicja w linii 79 pliku stos.hh.

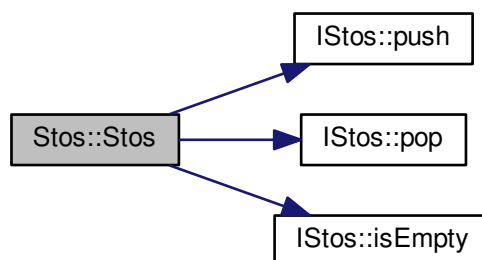
5.27.2 Dokumentacja konstruktora i destruktora

5.27.2.1 `template<class T> Stos< T>::Stos () [inline]`

Konstruktor tablicy obsługującej stos.

Definicja w linii 86 pliku stos.hh.

Oto graf wywołań dla tej funkcji:



5.27.2.2 `template<class T> virtual Stos< T>::~~Stos () [inline],[virtual]`

Destruktor stosu.

Definicja w linii 145 pliku stos.hh.

5.27.3 Dokumentacja funkcji składowych

5.27.3.1 `template<class T> T Stos< T>::get (void) [virtual]`

Zwraca element ze szczytu stosu bez jego usuwania.

Zwracane wartości

<i>T</i>	element ze szczytu stosu
----------	--------------------------

Wyjątki

<i>CriticalException</i>	re-throw z <code>tabn<T>::show(int)</code>
--------------------------	--

Ostrzeżenie

Uwaga! Próba odczytania elementu z pustego stosu spowoduje wyrzucenie wyjątku

Przykład sprawdzenia:

```
//Przykład korzystania z get()
IStos<int> * stos = new Stos<int>;
if (stos->isEmpty() == false) {
    cout << stos->get() << endl;
}
else
    cerr << "Stos pusty" << endl;
```

Implementuje `IStos< T >`.

Definicja w linii 179 pliku stos.hh.

5.27.3.2 `template<class T> bool Stos< T>::isEmpty (void) [virtual]`

Sprawdza, czy stos jest pusty.

Zwracane wartości

0	gdy niepusty
1	gdy pusty

Implementuje `IStos< T >`.

Definicja w linii 174 pliku stos.hh.

5.27.3.3 `template<class T> T Stos< T>::pop (void) [virtual]`

Zdejmuje element ze szczytu stosu.

Zwracane wartości

T	element ze szczytu stosu
---	--------------------------

Wyjątki

<i>CriticalException</i>	re-throw z <code>tabn<T>::remove()</code>
<i>ContinueException</i>	re-throw z <code>tabn<T>::remove()</code>

Ostrzeżenie

Uwaga! Próba zdjęcia elementu z pustego stosu spowoduje wyrzucenie wyjątku

Przykład sprawdzenia:

```
//Przykład korzystania z get()
IStos<int> * stos = new Stos<int>;
if (stos->isEmpty() == false) {
    cout << stos->pop() << endl;
}
else
    cerr << "Stos pusty" << endl;
```

Implementuje `IStos< T >`.

Definicja w linii 162 pliku `stos.hh`.

5.27.3.4 `template<class T> void Stos< T >::push (T element) [virtual]`

Umieszcza element na szczycie stosu.

Parametry

<i>element</i>	- element do umieszczenia na stosie
----------------	-------------------------------------

Implementuje `IStos< T >`.

Definicja w linii 157 pliku `stos.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [stos.hh](#)

5.28 Dokumentacja szablonu klasy `tabn< T >`

Modeluje tablicę dynamicznie rozszerzalną

```
#include <tabl.hh>
```

Diagram dziedziczenia dla `tabn< T >`

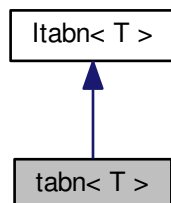
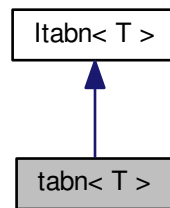


Diagram współpracy dla `tabn< T >`:



Metody publiczne

- `tabn ()`
Konstruktor klasy `tabn`.
- virtual `~tabn ()`
Destruktor klasy `tabn`.
- virtual bool `isEmpty ()`
Sprawdza, czy tablica jest pusta.
- virtual void `add (T)`
Dodaje element. Dodaje element do tablicy dynamicznej, odpowiednio ją rozszerzając.
- virtual void `add (T, int)`
Dodaje element w dane miejsce do tablicy, przesuwając wszystkie następne elementy o miejsce w prawo.
- virtual T `remove ()`
Usuwa i zwraca ostatni element z tablicy.
- virtual T `remove (int)`
Usuwa i zwraca wybrany element z tablicy, przesuwając wszystkie następne elementy o miejsce w lewo.
- virtual T `show (int) const`
Zwraca żądany element, o ile istnieje, bez jego usuwania.
- virtual void `showElems ()`
Wyświetla listę elementów.
- virtual int `noE ()`
zwraca liczbę elementów w tablicy
- virtual int `maxIndex ()`
Zwraca index ostatniego elementu.
- virtual int `aSize ()`
zwraca wielkość zaalokowanej przestrzeni dla tablicy
- virtual bool `search (T)`
znajduje element w tablicy
- virtual int `searchIndex (T)`
znajduje element w tablicy
- virtual T `searchObject (T)`
znajduje element w tablicy
- virtual T & `operator[] (int index)`
Umożliwia dostęp do dowolnego elementu tablicy bez sprawdzania zakresu (debug)
- virtual T `operator[] (int index) const`
Umożliwia odczyt dowolnego elementu tablicy bez sprawdzania zakresu (debug)
- virtual void `bubblesort ()`
Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.

5.28.1 Opis szczegółowy

```
template<class T>  
class tabn< T >
```

Modeluje tablicę dynamicznie rozszerzalną

Przechowuje elementy w rozszerzalnej tablicy o rozmiarze początkowym `SIZE`

Definicja w linii 136 pliku `tabl.hh`.

5.28.2 Dokumentacja konstruktora i destruktor

5.28.2.1 `template<class T> tabn< T >::tabn () [inline]`

Konstruktor klasy `tabn`.

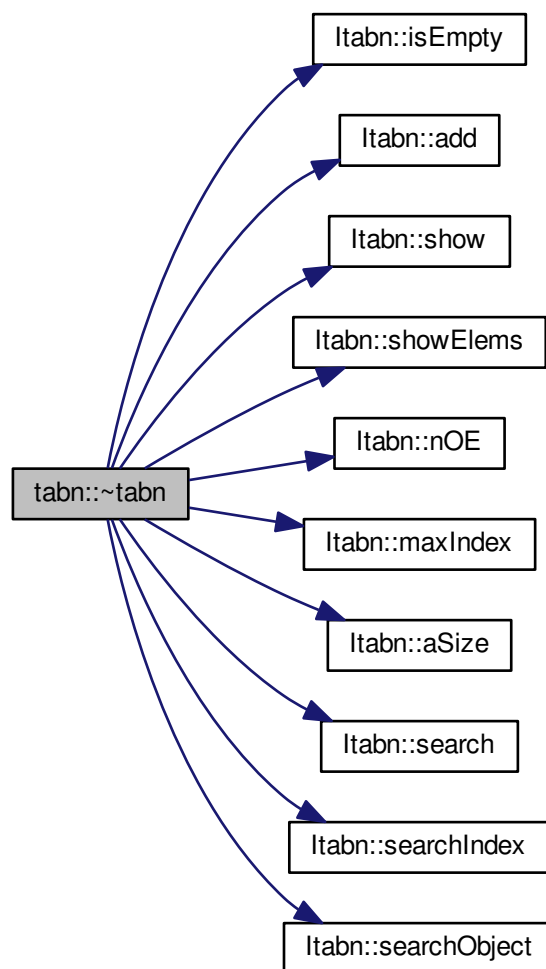
Definicja w linii 147 pliku `tabl.hh`.

5.28.2.2 `template<class T> virtual tabn< T >::~~tabn () [inline],[virtual]`

Destruktor klasy `tabn`.

Definicja w linii 156 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



5.28.3 Dokumentacja funkcji składowych

5.28.3.1 `template<class T> void tabn< T>::add (T element) [virtual]`

Dodaje element Dodaje element do tablicy dynamicznej, odpowiednio ją rozszerzając.

Parametry

<i>element</i>	- element do dodania
----------------	----------------------

Implementuje `Itabn< T >`.

Definicja w linii 383 pliku `tabl.hh`.

5.28.3.2 `template<class T> void tabn< T >::add (T element, int position) [virtual]`

Dodaje element w dane miejsce do tablicy, przesuwając wszystkie następne elementy o miejsce w prawo.

Parametry

<i>element</i>	- wstawiany element
<i>positionShifted</i>	- indeks pola, w które ma być wstawiony element.

Wyjątki

<i>ContinueException</i>	WrongPositionToShiftFromRightException przy próbie dodania elementu do niewłaściwego miejsca (re-throw z <code>tabn<T>::shiftRight(T,int)</code>).
--	---

Implementuje [`Itabn< T >`](#).

Definicja w linii 391 pliku `tabl.hh`.

5.28.3.3 `template<class T> int tabn< T >::aSize (void) [virtual]`

zwraca wielkość zaalokowanej przestrzeni dla tablicy

Zwracane wartości

<i>int</i>	Ilość zaalokowanych pól
------------	-------------------------

Implementuje [`Itabn< T >`](#).

Definicja w linii 563 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:

5.28.3.4 `template<class T> void tabn< T >::bubblesort (void) [virtual]`

Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.

Ostrzeżenie

Wymaga typu danych ze zdefiniowanym operatorem porównania "większe od"

Wyjątki

CriticalException	re-throw z <code>tabn<T>::swap(int,int)</code>
-----------------------------------	--

Implementuje [ltabn< T >](#).

Definicja w linii 584 pliku `tabl.hh`.

5.28.3.5 `template<class T> bool tabn<T>::isEmpty (void) [virtual]`

Sprawdza, czy tablica jest pusta.

Zwracane wartości

0	gdy tablica nie jest pusta
1	gdy tablica jest pusta

Implementuje [ltabn< T >](#).

Definicja w linii 517 pliku `tabl.hh`.

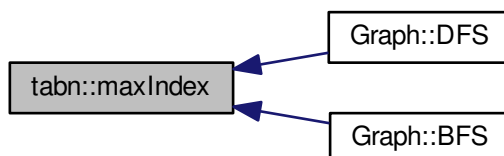
5.28.3.6 `template<class T> int tabn<T>::maxIndex (void) [virtual]`

Zwraca index ostatniego elementu.

Implementuje [ltabn< T >](#).

Definicja w linii 558 pliku `tabl.hh`.

Oto graf wywoływań tej funkcji:



5.28.3.7 `template<class T> int tabn<T>::nOE (void) [virtual]`

zwraca liczbę elementów w tablicy

Zwracane wartości

<i>int</i>	Liczba elementów w tablicy
------------	----------------------------

Implementuje `ltabn< T >`.

Definicja w linii 553 pliku `tabl.hh`.

5.28.3.8 `template<class T> virtual T& tabn< T >::operator[](int index) [inline],[virtual]`

Umożliwia dostęp do dowolnego elementu tablicy bez sprawdzania zakresu (debug)

Parametry

<i>index</i>	- numer elementu tablicy
--------------	--------------------------

Zwracane wartości

<i>T*</i>	Wskaźnik na wybrany element tablicy
-----------	-------------------------------------

Implementuje `ltabn< T >`.

Definicja w linii 312 pliku `tabl.hh`.

5.28.3.9 `template<class T> virtual T tabn< T >::operator[](int index) const [inline],[virtual]`

Umożliwia odczyt dowolnego elementu tablicy bez sprawdzania zakresu (debug)

Parametry

<i>index</i>	- numer elementu tablicy
--------------	--------------------------

Zwracane wartości

<i>T</i>	Element tablicy
----------	-----------------

Implementuje `ltabn< T >`.

Definicja w linii 324 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



5.28.3.10 `template<class T> T tabn<T>::remove (void) [virtual]`

Usuwa i zwraca ostatni element z tablicy.

Wyjątki

<i>CriticalException</i>	EmptyTableException przy próbie usunięcia z pustej tablicy (re-throw z <code>tabn<T>::isEmptyException()</code>).
<i>CriticalException</i>	WrongIndexException przy próbie usunięcia z pustej tablicy (re-throw z <code>tabn<T>::show(int)</code>).
<i>ContinueException</i>	re-throw z <code>tabn<T>::reduce2()</code> .

Implementuje `ltabn<T>`.

Definicja w linii 422 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



5.28.3.11 `template<class T> T tabn<T>::remove (int position) [virtual]`

Usuwa i zwraca wybrany element z tablicy, przesuwając wszystkie następne elementy o miejsce w lewo.

Parametry

<i>position</i>	indeks pola, z którego ma być usunięty element.
-----------------	---

Wyjątki

<i>CriticalException</i>	EmptyTableException przy próbie usunięcia z pustej tablicy lub nieistniejącego elementu (re-throw z <code>tabn<T>::isEmptyException()</code>).
<i>CriticalException</i>	WrongIndexException przy próbie usunięcia z pustej tablicy lub nieistniejącego elementu (re-throw z <code>tabn<T>::show(int)</code>).
<i>ContinueException</i>	re-throw z <code>tabn<T>::reduce2()</code> .

Implementuje `Itabn< T >`.

Definicja w linii 445 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



5.28.3.12 `template<class T> bool tabn< T >::search (T elem) [virtual]`

znajduje element w tablicy

Zwracane wartości

<i>true</i>	gdy element został znaleziony
-------------	-------------------------------

Implementuje `Itabn< T >`.

Definicja w linii 358 pliku `tabl.hh`.

5.28.3.13 `template<class T> int tabn< T >::searchIndex (T elem) [virtual]`

znajduje element w tablicy

Zwracane wartości

<i>index</i>	indeks znalezionego elementu
--------------	------------------------------

Implementuje `Itabn< T >`.

Definicja w linii 366 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



5.28.3.14 `template<class T> T tabn<T>::searchObject (T obj) [virtual]`

znajduje element w tablicy

Zwracane wartości

<i>znaleziony</i>	element
-------------------	---------

Implementuje `ltabn<T>`.

Definicja w linii 377 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



5.28.3.15 `template<class T> T tabn<T>::show (int position) const [virtual]`

Zwraca żądany element, o ile istnieje, bez jego usuwania.

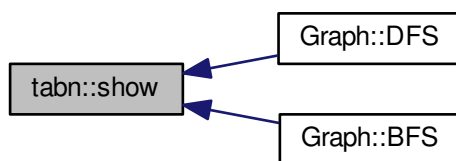
Wyjątki

<i>CriticalException</i>	WrongIndexException przy próbie odczytania z pustej tablicy lub dostępu do nieistniejącego elementu.
--------------------------	--

Implementuje `ltabn<T>`.

Definicja w linii 533 pliku `tabl.hh`.

Oto graf wywoływań tej funkcji:



5.28.3.16 `template<class T> void tabn<T>::showElems (void) [virtual]`

Wyświetla listę elementów.

Implementuje `Itabn<T>`.

Definicja w linii 543 pliku `tabl.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)

5.29 Dokumentacja klasy `tabn_test`

Definiuje sposób testowania wypełniania tablicy `tabn`.

```
#include <tabl.hh>
```

Diagram dziedziczenia dla `tabn_test`

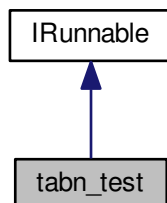
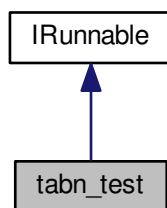


Diagram współpracy dla `tabn_test`:



Metody publiczne

- `tabn_test ()`
Konstruktor klasy `tabn_test`.
- `virtual ~tabn_test ()`
Destruktor klasy `tabn_test`.
- `virtual bool prepare (int sizeOfTest)`
Przygotowuje rozmiar testu.
- `virtual bool run ()`
Wykonuje test.

5.29.1 Opis szczegółowy

Definiuje sposób testowania wypełniania tablicy `tabn`.

Definicja w linii 618 pliku `tabl.hh`.

5.29.2 Dokumentacja konstruktora i destruktora

5.29.2.1 `tabn_test::tabn_test ()` [inline]

Konstruktor klasy `tabn_test`.

Definicja w linii 626 pliku `tabl.hh`.

5.29.2.2 `virtual tabn_test::~~tabn_test ()` [inline], [virtual]

Destruktor klasy `tabn_test`.

Definicja w linii 632 pliku `tabl.hh`.

5.29.3 Dokumentacja funkcji składowych

5.29.3.1 `virtual bool tabn_test::prepare (int sizeOfTest)` [inline], [virtual]

Przygotowuje rozmiar testu.

Parametry

<code>sizeofTest</code>	- rozmiar testu
-------------------------	-----------------

Zwracane wartości

<code>bool</code>	zawsze true
-------------------	-------------

Implementuje [IRunnable](#).

Definicja w linii 663 pliku `tabl.hh`.

5.29.3.2 `virtual bool tabn_test::run (void) [inline],[virtual]`

Wykonuje test.

Pozwala na wykonanie testu w pętli `for` iterującej `counter` razy. Zasila funkcję dodawania generując losowe cyfry w funkcji `generateRandomDgt()`

Zwracane wartości

<code>bool</code>	zawsze true
-------------------	-------------

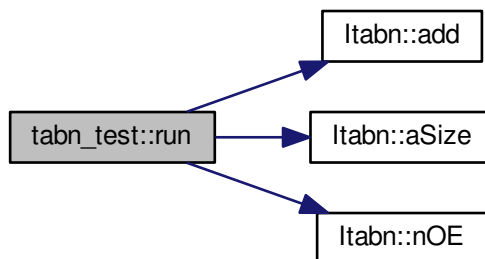
Wyjątki

ContinueException	re-throw <code>tabn<T>::add(int)</code>
-----------------------------------	---

Implementuje [IRunnable](#).

Definicja w linii 680 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)

5.30 Dokumentacja klasy test_graph_BFS

```
#include <graph.hh>
```

Diagram dziedziczenia dla test_graph_BFS

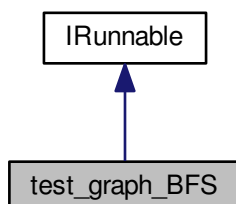
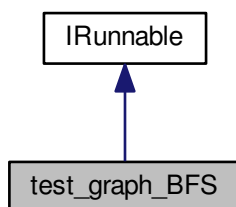


Diagram współpracy dla test_graph_BFS:



Metody publiczne

- `test_graph_BFS` ()
- `bool prepare` (int testSize)
Przygotowanie badań
- `bool run` (void)
Przeprowadzanie badań

5.30.1 Opis szczegółowy

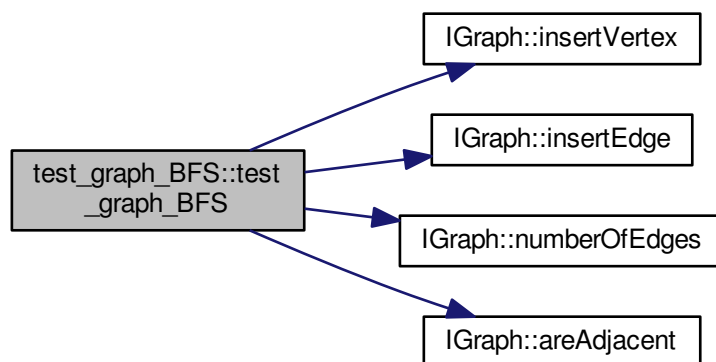
Definicja w linii 215 pliku graph.hh.

5.30.2 Dokumentacja konstruktora i destruktora

5.30.2.1 test_graph_BFS::test_graph_BFS () [inline]

Definicja w linii 232 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



5.30.3 Dokumentacja funkcji składowych

5.30.3.1 bool test_graph_BFS::prepare (int) [inline],[virtual]

Przygotowanie badań

Implementuje [IRunnable](#).

Definicja w linii 307 pliku graph.hh.

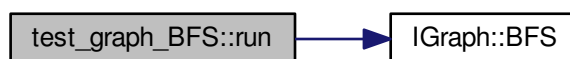
5.30.3.2 bool test_graph_BFS::run (void) [inline],[virtual]

Przeprowadzanie badań

Implementuje [IRunnable](#).

Definicja w linii 318 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [graph.hh](#)

5.31 Dokumentacja klasy test_graph_DFS

```
#include <graph.hh>
```

Diagram dziedziczenia dla test_graph_DFS

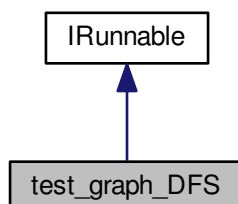
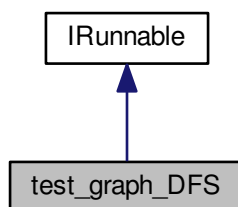


Diagram współpracy dla test_graph_DFS:



Metody publiczne

- [test_graph_DFS](#) ()
- [bool prepare](#) (int testSize)
Przygotowanie badań
- [bool run](#) (void)
Przeprowadzanie badań

5.31.1 Opis szczegółowy

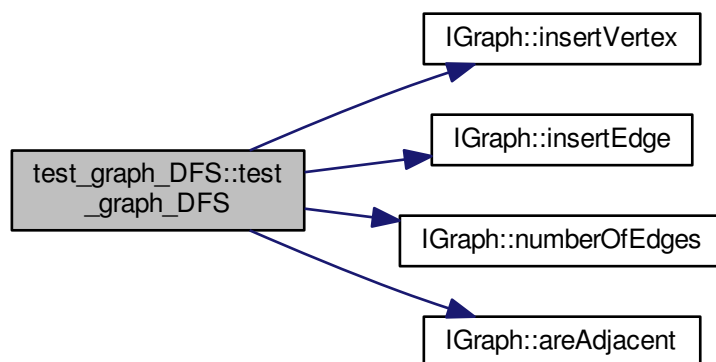
Definicja w linii 327 pliku graph.hh.

5.31.2 Dokumentacja konstruktora i destruktora

5.31.2.1 test_graph_DFS::test_graph_DFS () [inline]

Definicja w linii 344 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



5.31.3 Dokumentacja funkcji składowych

5.31.3.1 bool test_graph_DFS::prepare (int) [inline],[virtual]

Przygotowanie badań

Implementuje [IRunnable](#).

Definicja w linii 419 pliku graph.hh.

5.31.3.2 bool test_graph_DFS::run (void) [inline],[virtual]

Przeprowadzanie badań

Implementuje [IRunnable](#).

Definicja w linii 430 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [graph.hh](#)

5.32 Dokumentacja klasy tree_test

Klasa testująca drzewo czerwono-czarne.

```
#include <tree.hh>
```

Diagram dziedziczenia dla tree_test

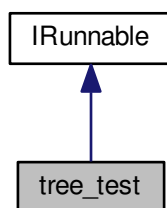
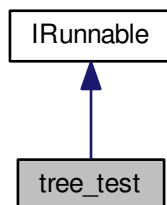


Diagram współpracy dla tree_test:



Metody publiczne

- `tree_test ()`
Konstruktor.
- `virtual ~tree_test ()`
Destruktor.
- `virtual bool prepare (int sizeOfTest)`
Przygotowuje test, wypełniając drzewo.
- `virtual bool run ()`
Wykonuje test, szukając w drzewie.

5.32.1 Opis szczegółowy

Klasa testująca drzewo czerwono-czarne.

Definicja w linii 503 pliku `tree.hh`.

5.32.2 Dokumentacja konstruktora i destruktor

5.32.2.1 `tree_test::tree_test ()` `[inline]`

Konstruktor.

Definicja w linii 529 pliku `tree.hh`.

5.32.2.2 `virtual tree_test::~~tree_test ()` `[inline]`, `[virtual]`

Destruktor.

Definicja w linii 536 pliku `tree.hh`.

5.32.3 Dokumentacja funkcji składowych

5.32.3.1 `virtual bool tree_test::prepare (int sizeOfTest)` `[inline]`, `[virtual]`

Przygotowuje test, wypełniając drzewo.

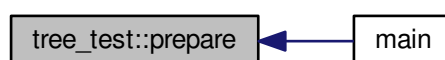
Implementuje [IRunnable](#).

Definicja w linii 543 pliku `tree.hh`.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



5.32.3.2 `virtual bool tree_test::run (void) [inline],[virtual]`

Wykonuje test, szukając w drzewie.

Implementuje [IRunnable](#).

Definicja w linii 560 pliku `tree.hh`.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tree.hh](#)

5.33 Dokumentacja szablonu klasy `TreeRB< T >`

Klasa implementująca interfejs drzewa czerwono-czarnego.

```
#include <tree.hh>
```

Diagram dziedziczenia dla `TreeRB< T >`

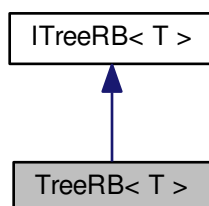
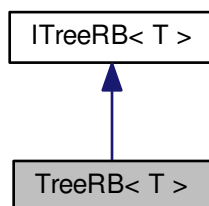


Diagram współpracy dla TreeRB< T >:



Metody publiczne

- virtual `nodeRB< T > * retRoot` (void)
Zwraca roota drzewa.
- virtual void `leftRot` (`nodeRB< T > *nd`)
Obraca wybrane poddrzewo w lewo.
- virtual void `rightRot` (`nodeRB< T > *nd`)
Obraca wybrane poddrzewo w prawo.
- `TreeRB` ()
Konstruktor.
- virtual `~TreeRB` ()
Destruktor.
- virtual void `insert` (T element)
Wstawia element do drzewa.
- virtual void `insert` (T element, `nodeRB< T > *node`)
Wstawia element jako potomka danego elementu.
- virtual bool `search` (T k)
Wyszukuje elementy w drzewie.

5.33.1 Opis szczegółowy

```
template<class T>
class TreeRB< T >
```

Klasa implementująca interfejs drzewa czerwono-czarnego.

Definicja w linii 253 pliku tree.hh.

5.33.2 Dokumentacja konstruktora i destruktora

5.33.2.1 `template<class T> TreeRB< T >::TreeRB ()` [inline]

Konstruktor.

Definicja w linii 377 pliku tree.hh.

5.33.2.2 `template<class T > virtual TreeRB< T >::~~TreeRB () [inline],[virtual]`

Destruktor.

Definicja w linii 383 pliku tree.hh.

5.33.3 Dokumentacja funkcji składowych

5.33.3.1 `template<class T > virtual void TreeRB< T >::insert (T element) [inline],[virtual]`

Wstawia element do drzewa.

Ostrzeżenie

Nowy element jest domyślnie zakolorowany na czerwono (patrz konstruktor [nodeRB\(\)](#))

Implementuje [ITreeRB< T >](#).

Definicja w linii 391 pliku tree.hh.

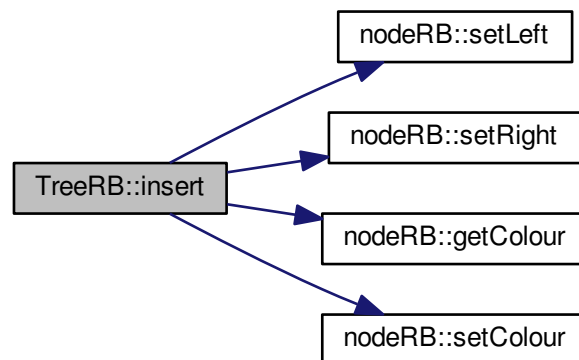
5.33.3.2 `template<class T > virtual void TreeRB< T >::insert (T element, nodeRB< T > * node) [inline],[virtual]`

Wstawia element jako potomka danego elementu.

Implementuje [ITreeRB< T >](#).

Definicja w linii 398 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



5.33.3.3 `template<class T > virtual void TreeRB< T >::leftRot (nodeRB< T > * nd) [inline],[virtual]`

Obraca wybrane poddrzewo w lewo.

Parametry

<i>in</i>	element najstarszy do obrotu
-----------	------------------------------

Implementuje [ITreeRB< T >](#).

Definicja w linii 302 pliku `tree.hh`.

5.33.3.4 `template<class T> virtual nodeRB<T>* TreeRB< T >::retRoot(void) [inline],[virtual]`

Zwraca roota drzewa.

Implementuje [ITreeRB< T >](#).

Definicja w linii 294 pliku `tree.hh`.

5.33.3.5 `template<class T> virtual void TreeRB< T >::rightRot(nodeRB< T >* nd) [inline],[virtual]`

Obraca wybrane poddrzewo w prawo.

Parametry

<i>in</i>	element najstarszy do obrotu
-----------	------------------------------

Implementuje [ITreeRB< T >](#).

Definicja w linii 339 pliku `tree.hh`.

5.33.3.6 `template<class T> virtual bool TreeRB< T >::search(T k) [inline],[virtual]`

Wyszukuje elementy w drzewie.

Implementuje [ITreeRB< T >](#).

Definicja w linii 478 pliku `tree.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tree.hh](#)

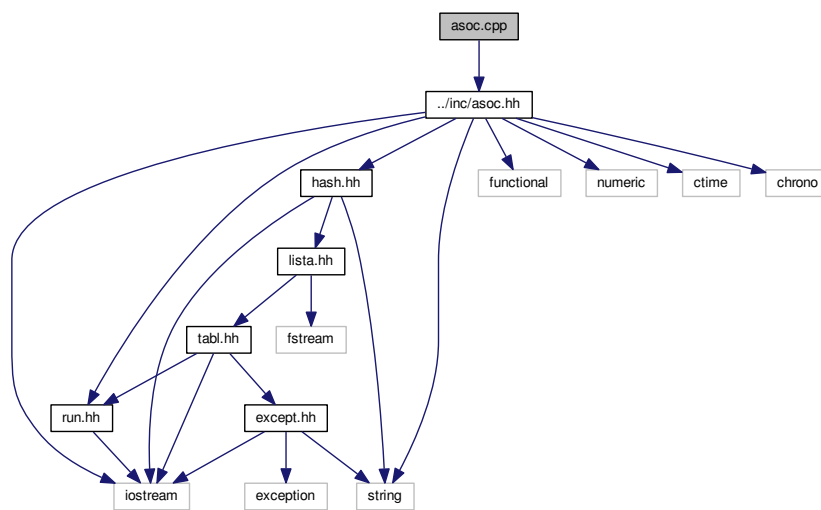
Rozdział 6

Dokumentacja plików

6.1 Dokumentacja pliku asoc.cpp

```
#include "../inc/asoc.hh"
```

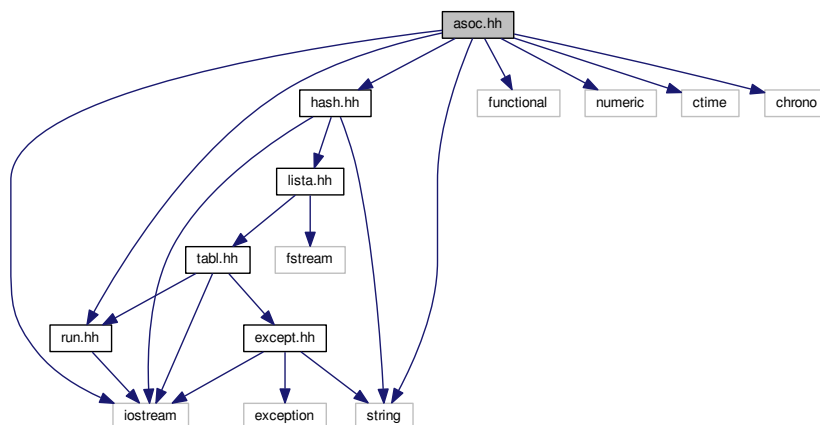
Wykres zależności załączania dla asoc.cpp:



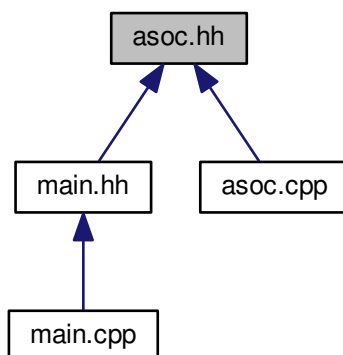
6.2 Dokumentacja pliku asoc.hh

```
#include <iostream>
#include <string>
#include <functional>
#include <numeric>
#include "hash.hh"
#include "run.hh"
#include <ctime>
#include <chrono>
```

Wykres zależności załączania dla asoc.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



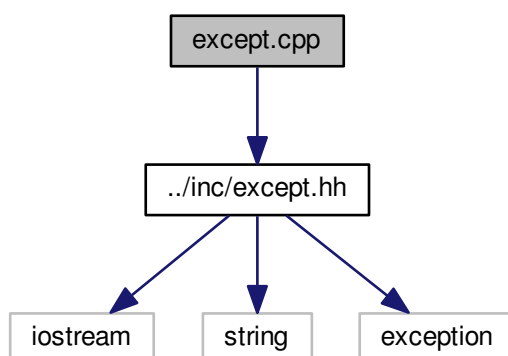
Komponenty

- class `lAsoc< T, T2 >`
- class `Asoc< T, T2 >`
- class `asoc_test`

6.3 Dokumentacja pliku except.cpp

```
#include "../inc/except.hh"
```


Wykres zależności załączania dla except.cpp:

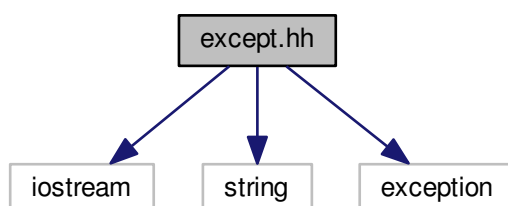


6.4 Dokumentacja pliku except.hh

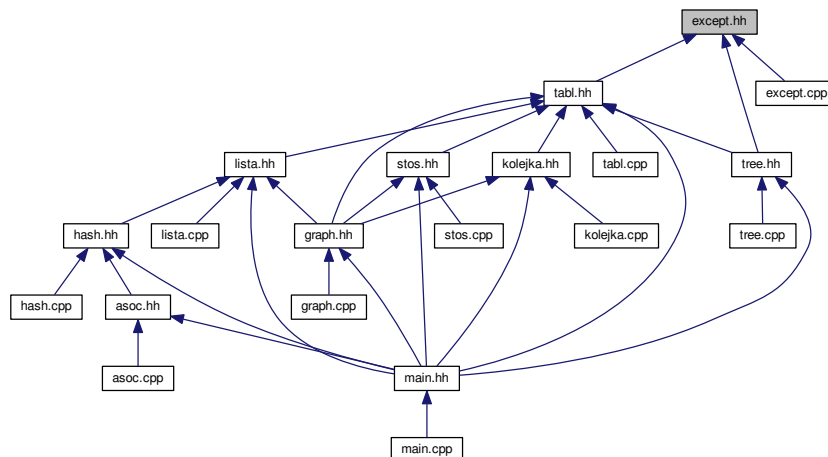
Plik zawiera definicje wyjątków.

```
#include <iostream>
#include <string>
#include <exception>
```

Wykres zależności załączania dla except.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class [ExceptionBase](#)
Ogólny wyjątek.
- class [CriticalException](#)
Wyjątek krytyczny, wymagający zamknięcia programu.
- class [ContinueException](#)
Wyjątek, który może spowodować nieprzewidziane działanie programu, ale program mógłby dalej działać.

Funkcje

- template<class ExceptT >
void [what](#) (ExceptT &except)

6.4.1 Opis szczegółowy

Plik zawiera definicje wyjątków.

6.4.2 Dokumentacja funkcji

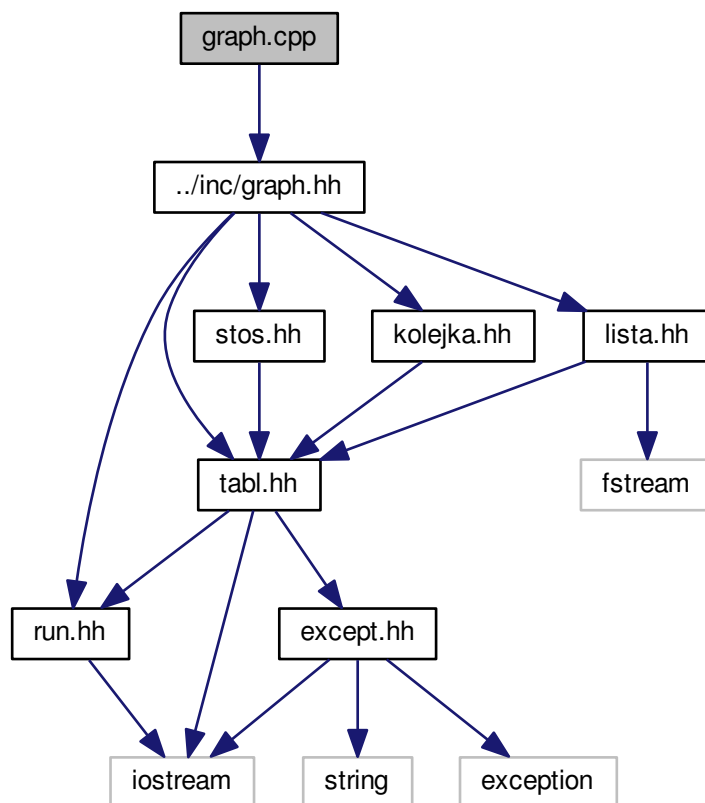
6.4.2.1 template<class ExceptT > void what (ExceptT & except)

Definicja w linii 72 pliku except.hh.

6.5 Dokumentacja pliku graph.cpp

```
#include "../inc/graph.hh"
```

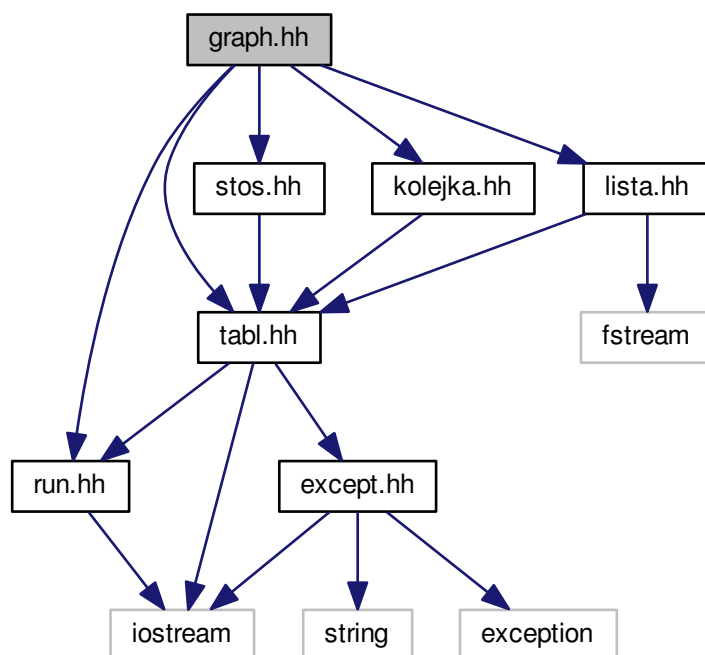
Wykres zależności załączania dla graph.cpp:



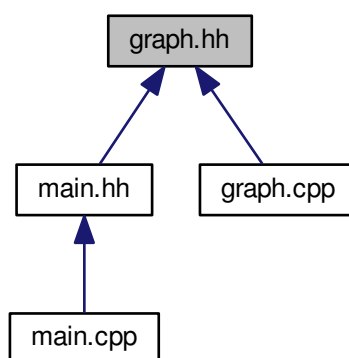
6.6 Dokumentacja pliku graph.hh

```
#include "tabl.hh"
#include "lista.hh"
#include "stos.hh"
#include "kolejka.hh"
#include "run.hh"
```

Wykres zależności załączania dla graph.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class [IGraph](#)

Interfejs grafu.

- class [Graph](#)

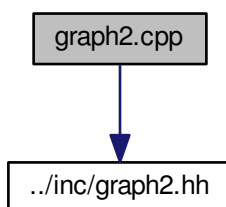
Klasa implementująca inerfejs grafu.

- class [test_graph_BFS](#)
- class [test_graph_DFS](#)

6.7 Dokumentacja pliku graph2.cpp

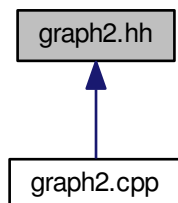
```
#include "../inc/graph2.hh"
```

Wykres zależności załączania dla graph2.cpp:



6.8 Dokumentacja pliku graph2.hh

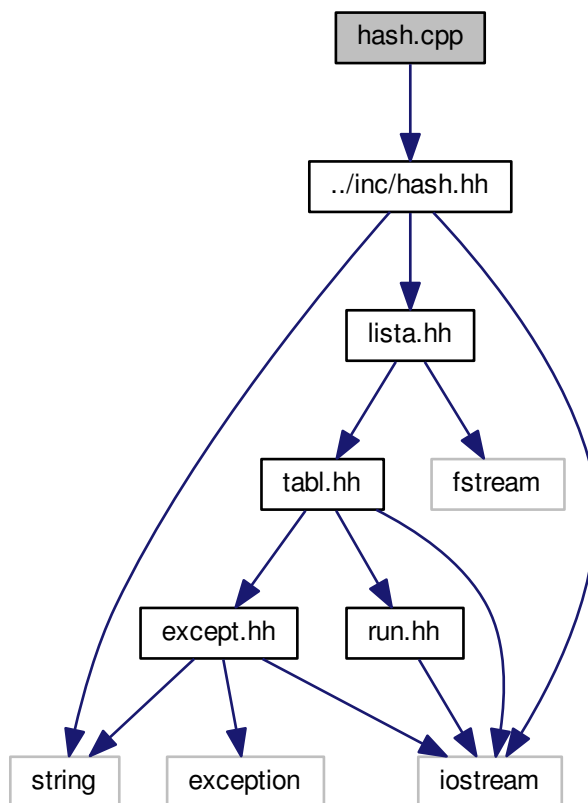
Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



6.9 Dokumentacja pliku hash.cpp

```
#include "../inc/hash.hh"
```

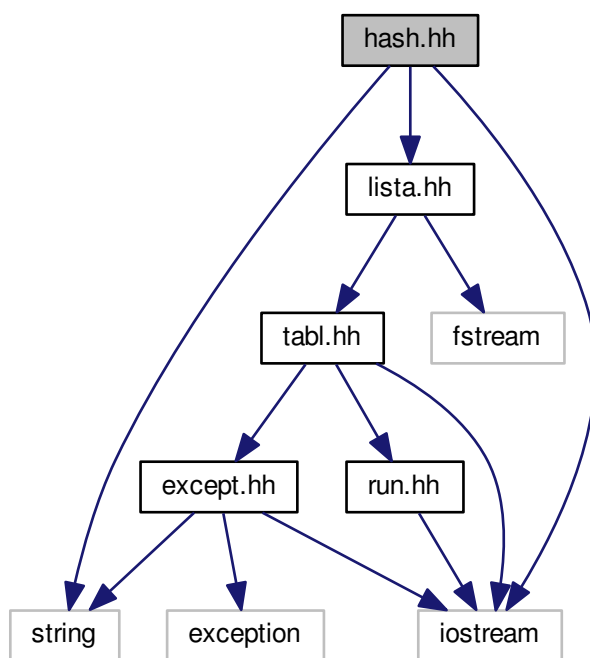
Wykres zależności załączania dla hash.cpp:



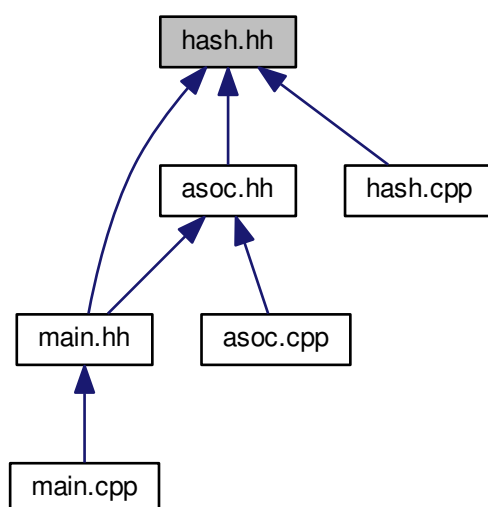
6.10 Dokumentacja pliku hash.hh

```
#include <iostream>
#include <string>
#include "lista.hh"
```

Wykres zależności załączania dla hash.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class `entry< T, T2 >`

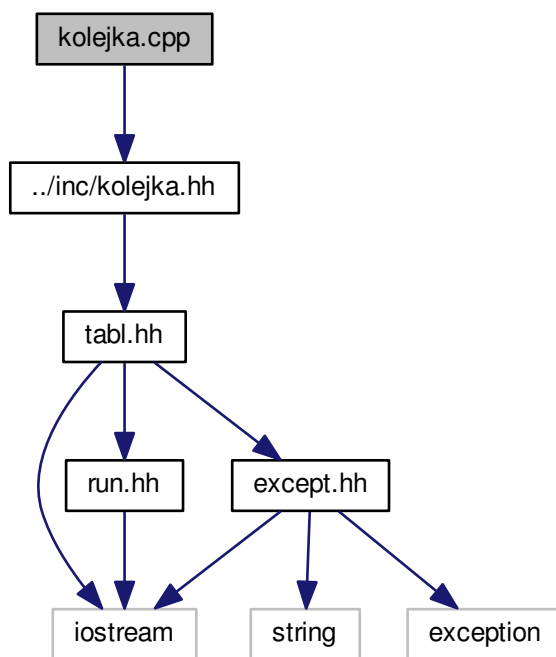
Klasa definiująca obiekt typu wpis.

- class `IBucket< T, T2 >`
- class `Bucket< T, T2 >`

6.11 Dokumentacja pliku kolejka.cpp

```
#include "../inc/kolejka.hh"
```

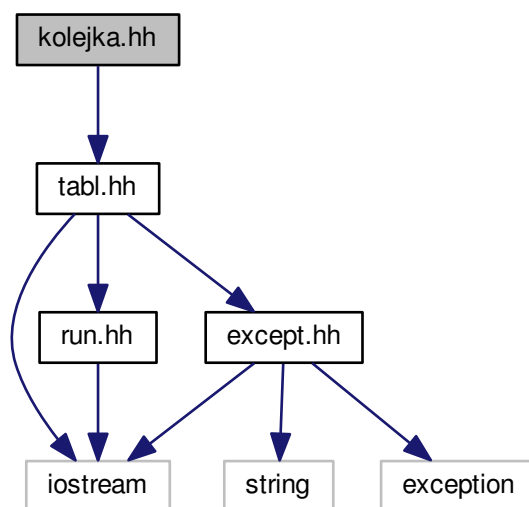
Wykres zależności załączania dla kolejka.cpp:



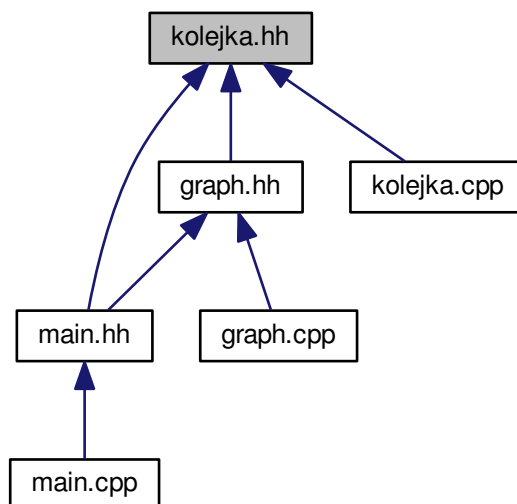
6.12 Dokumentacja pliku kolejka.hh

```
#include "tabl.hh"
```


Wykres zależności załączania dla kolejka.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class [IKolejka< T >](#)

Interfejs klasy *Kolejka* Definiuje operacje dostępne dla klasy *Kolejka*.

- class *Kolejka*< T >

Klasa modeluje kolejkę

- class *node*< T >

Węzeł kolejki.

- class *IQueue*< T >

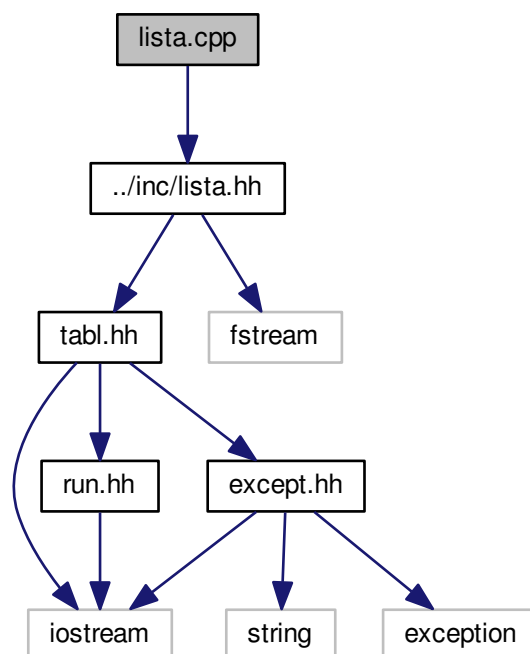
- class *Queue*< T >

Kolejka oparta na węzłach.

6.13 Dokumentacja pliku lista.cpp

```
#include "../inc/lista.hh"
```

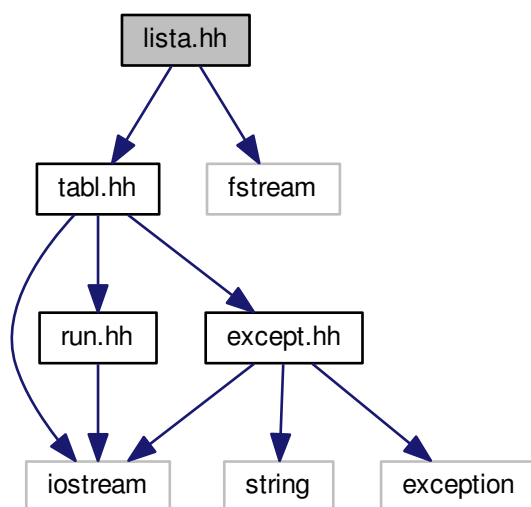
Wykres zależności załączania dla lista.cpp:



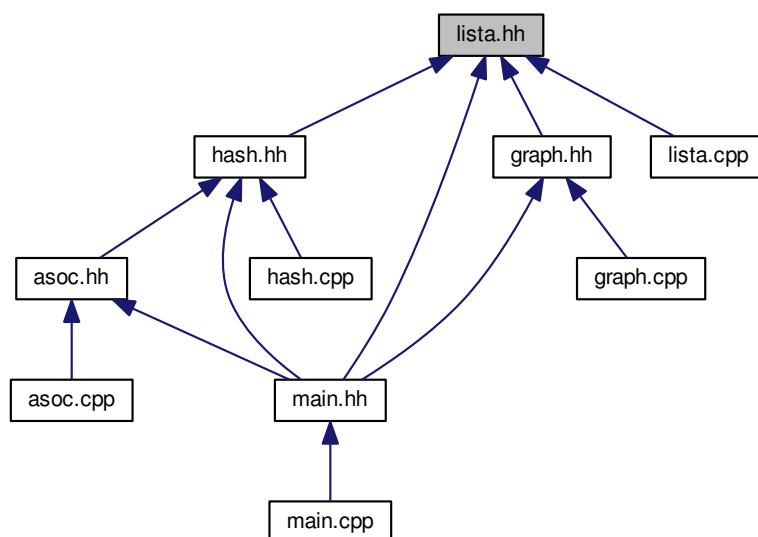
6.14 Dokumentacja pliku lista.hh

```
#include "tabl.hh"
#include <fstream>
```

Wykres zależności załączania dla lista.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class `ILista< T >`

Interfejs listy.

- class `Lista< T >`

Klasa lista.

- class `lista_test`

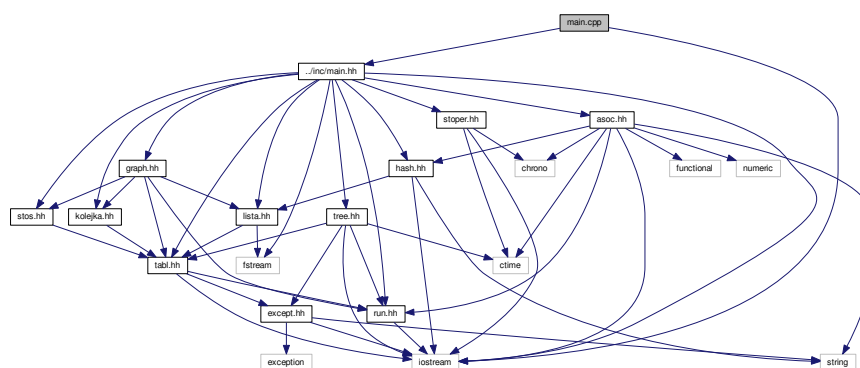
Definiuje sposób testowania wypełniania listy.

6.15 Dokumentacja pliku main.cpp

Główny plik programu.

```
#include <iostream>
#include "../inc/main.hh"
```

Wykres zależności załączania dla main.cpp:



Funkcje

- int `main` (void)
- void `dumpToFile` (string nameOfFile, unsigned int testsize, `IStoper` *stoper)
- void `printOnscreen` (unsigned int testsize, `IStoper` *stoper)

Wyświetla wynik na standardowym wyjściu.

6.15.1 Opis szczegółowy

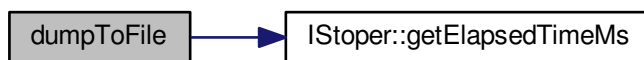
Główny plik programu.

6.15.2 Dokumentacja funkcji

6.15.2.1 void dumpToFile (string nameOfFile, unsigned int testsize, IStoper * stoper)

Definicja w linii 443 pliku main.cpp.

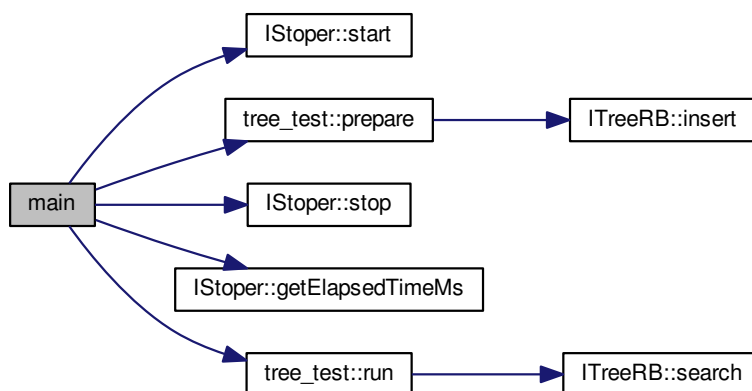
Oto graf wywołań dla tej funkcji:



6.15.2.2 `int main (void)`

Definicja w linii 99 pliku main.cpp.

Oto graf wywołań dla tej funkcji:



6.15.2.3 `void printOnscreen (unsigned int, IStoper *)`

Wyświetla wynik na standardowym wyjściu.

Parametry

<i>testSize</i>	rozmiar testu
<i>stoper</i>	klasa stopera

Definicja w linii 457 pliku main.cpp.

Oto graf wywołań dla tej funkcji:

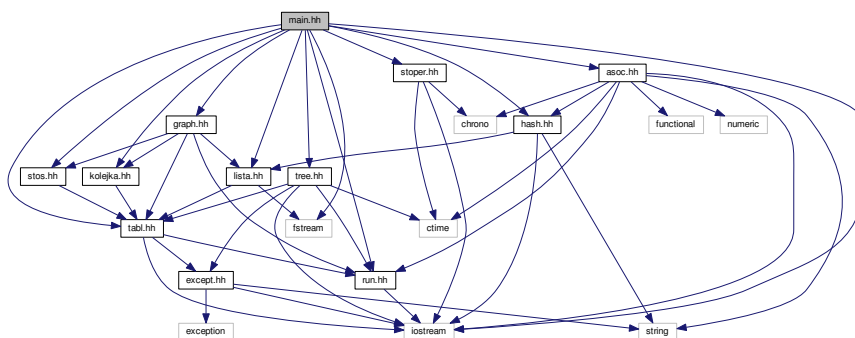


6.16 Dokumentacja pliku main.hh

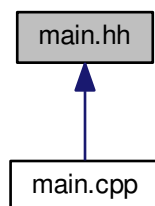
```

#include <iostream>
#include <fstream>
#include "stoper.hh"
#include "tabl.hh"
#include "run.hh"
#include "lista.hh"
#include "stos.hh"
#include "kolejka.hh"
#include "asoc.hh"
#include "hash.hh"
#include "tree.hh"
#include "graph.hh"
  
```

Wykres zależności załączania dla main.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Funkcje

- void `dumpToFile` (std::string, unsigned int, `IStoper *`)
Zrzuca dane wynikowe do pliku.
- void `printOnscreen` (unsigned int, `IStoper *`)
Wyświetla wynik na standardowym wyjściu.

6.16.1 Dokumentacja funkcji

6.16.1.1 void dumpToFile (std::string , unsigned int, IStoper *)

Zrzuca dane wynikowe do pliku.

Parametry

<i>nameOfFile</i>	nazwa pliku wynikowego
<i>testsize</i>	rozmiar testu
<i>stoper</i>	klasa stopera

6.16.1.2 void printOnscreen (unsigned int, IStoper *)

Wyświetla wynik na standardowym wyjściu.

Parametry

<i>testSize</i>	rozmiar testu
<i>stoper</i>	klasa stopera

Definicja w linii 457 pliku main.cpp.

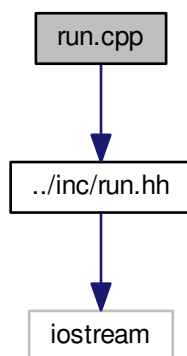
Oto graf wywołań dla tej funkcji:



6.17 Dokumentacja pliku run.cpp

```
#include "../inc/run.hh"
```

Wykres zależności załączania dla run.cpp:

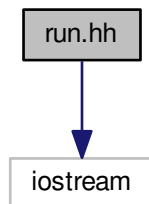


6.18 Dokumentacja pliku run.hh

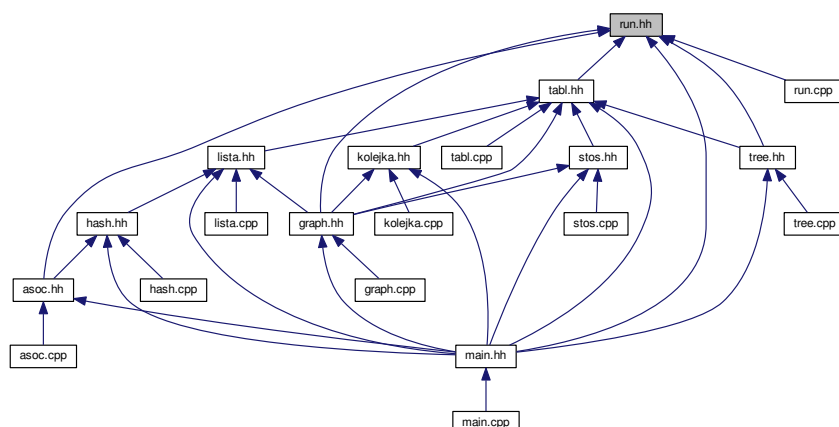
Plik definiuje interfejs [IRunnable](#), ujednolicający klasy umożliwiające badanie algorytmów.


```
#include <iostream>
```

Wykres zależności załączania dla run.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class [IRunnable](#)

Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.

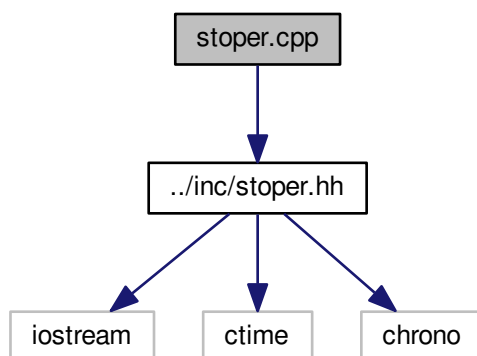
6.18.1 Opis szczegółowy

Plik definiuje interfejs [IRunnable](#), ujednolicający klasy umożliwiające badanie algorytmów.

6.19 Dokumentacja pliku stoper.cpp

```
#include "../inc/stoper.hh"
```

Wykres zależności załączania dla stoper.cpp:



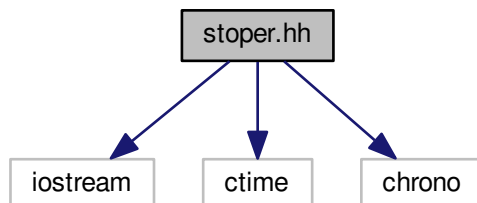
6.20 Dokumentacja pliku stoper.hh

```
#include <iostream>
```

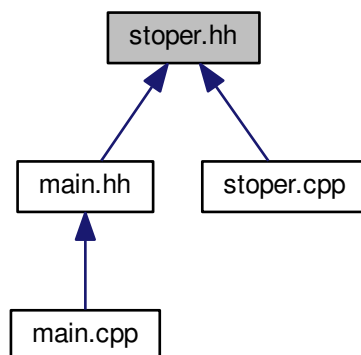
```
#include <ctime>
```

```
#include <chrono>
```

Wykres zależności załączania dla stoper.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class [IStoper](#)

Plik definiuje stoper, obliczający czas wykonywania badanych funkcji.

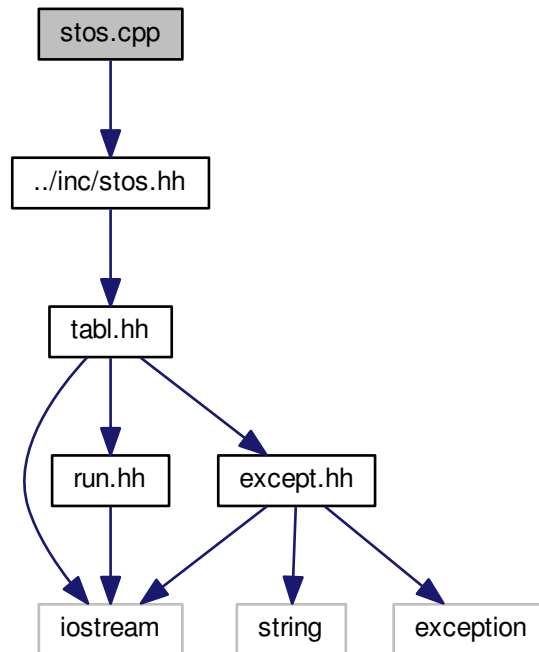
- class [Stoper](#)

Klasa stoper implementująca interfejs [IStoper](#).

6.21 Dokumentacja pliku stos.cpp

```
#include "../inc/stos.hh"
```

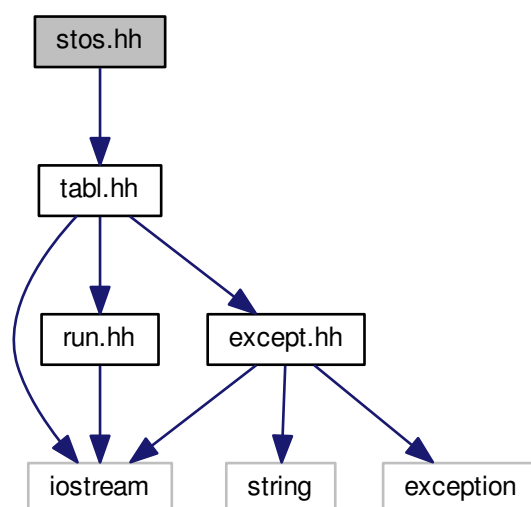
Wykres zależności załączania dla stos.cpp:



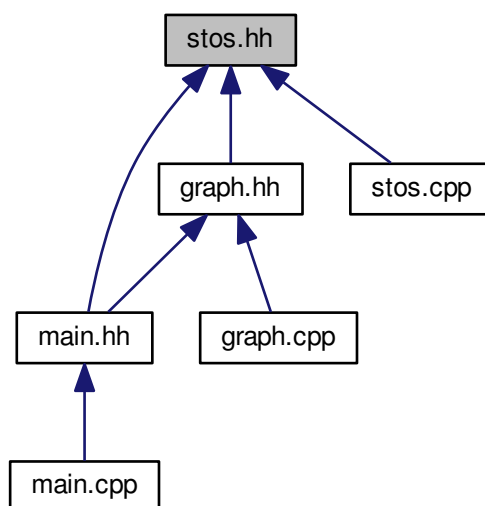
6.22 Dokumentacja pliku stos.hh

```
#include "tabl.hh"
```

Wykres zależności załączania dla stos.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class `IStos< T >`

Interfejs `stosu`.

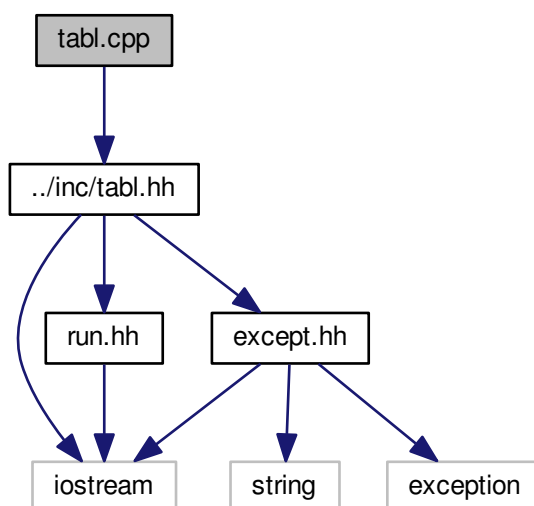
- class `Stos< T >`

Klasa `Stos`.

6.23 Dokumentacja pliku `tabl.cpp`

```
#include "../inc/tabl.hh"
```

Wykres zależności załączania dla `tabl.cpp`:

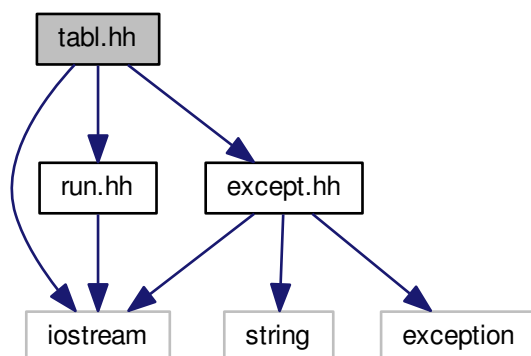


6.24 Dokumentacja pliku `tabl.hh`

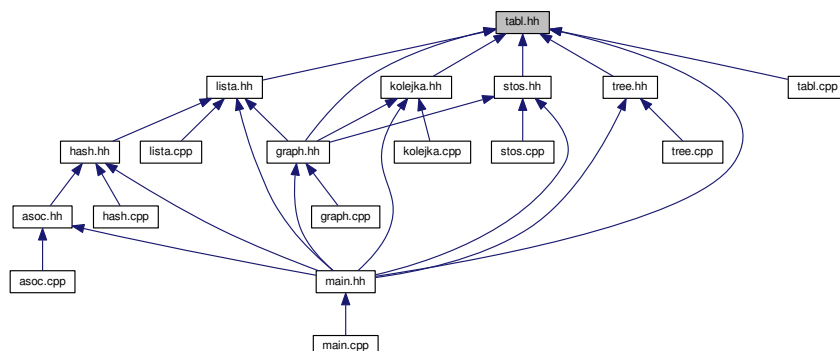
Definicja interfejsu `ltabn`, klasy `tabn` oraz klasy `tabn_test`.

```
#include <iostream>
#include "run.hh"
#include "except.hh"
```

Wykres zależności załączania dla tabl.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class `Itabn< T >`
Interfejs klasy tabn.
- class `tabn< T >`
Modeluje tablicę dynamicznie rozszerzalną
- class `tabn_test`
Definiuje sposób testowania wypełniania tablicy tabn.

Definicje

- #define **SIZE** 10

6.24.1 Opis szczegółowy

Definicja interfejsu [ltabn](#), klasy [tabn](#) oraz klasy [tabn_test](#).

6.24.2 Dokumentacja definicji

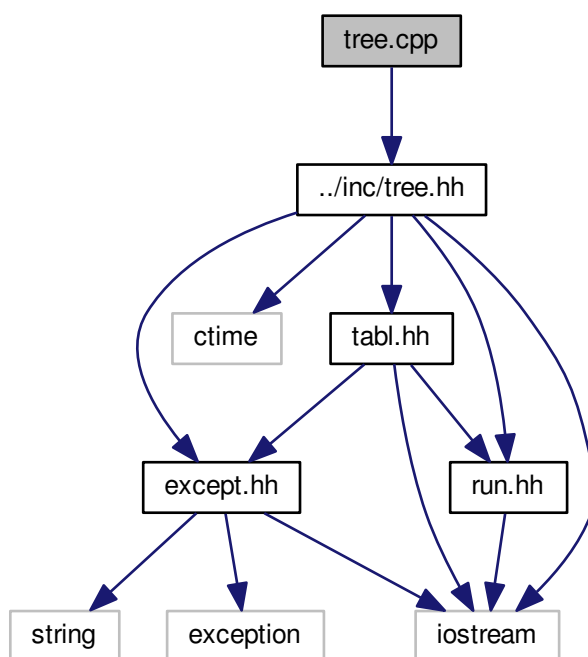
6.24.2.1 #define SIZE 10

Definicja w linii 12 pliku [tabl.hh](#).

6.25 Dokumentacja pliku [tree.cpp](#)

```
#include "../inc/tree.hh"
```

Wykres zależności załączania dla [tree.cpp](#):



Funkcje

- `std::ostream & operator<< (std::ostream &output, Colour col)`

Wyświetlanie koloru node'a.

6.25.1 Dokumentacja funkcji

6.25.1.1 `std::ostream& operator<< (std::ostream & output, Colour col)`

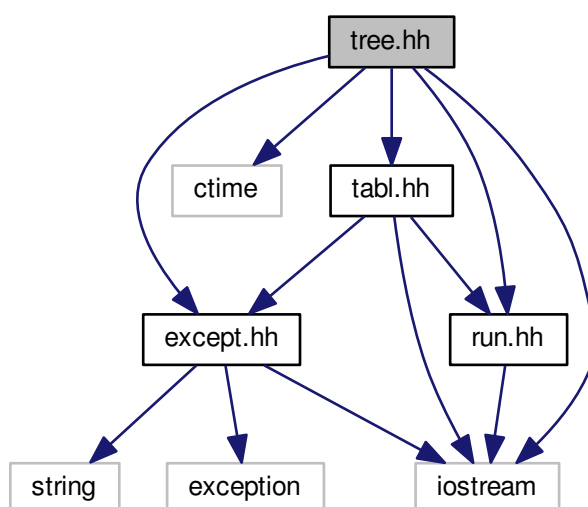
Wyświetlanie koloru node'a.

Definicja w linii 3 pliku tree.cpp.

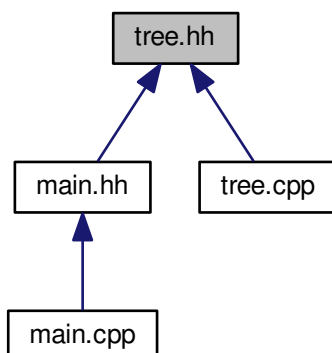
6.26 Dokumentacja pliku tree.hh

```
#include <iostream>
#include <ctime>
#include "except.hh"
#include "tabl.hh"
#include "run.hh"
```

Wykres zależności załączania dla tree.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class `nodeRB< T >`
- class `ITreeRB< T >`
Interfejs klasy drzewa czerwono-czarnego.
- class `TreeRB< T >`
Klasa implementująca interfejs drzewa czerwono-czarnego.
- class `tree_test`
Klasa testująca drzewo czerwono-czarne.

Wyliczenia

- enum `Colour { red, black }`
Kolory node'a drzewa czerwono-czarnego.

Funkcje

- `std::ostream & operator<< (std::ostream &, Colour)`
Wyświetlanie koloru node'a.

6.26.1 Dokumentacja typów wyliczanych

6.26.1.1 enum `Colour`

Kolory node'a drzewa czerwono-czarnego.

Wartości wyliczeń

red

black

Definicja w linii 12 pliku `tree.hh`.

6.26.2 Dokumentacja funkcji

6.26.2.1 `std::ostream& operator<< (std::ostream & , Colour)`

Wyświetlanie koloru node'a.

Definicja w linii 3 pliku tree.cpp.

Skorowidz

- ~Asoc
 - Asoc, [10](#)
- ~Bucket
 - Bucket, [15](#)
- ~Graph
 - Graph, [26](#)
- ~IAsoc
 - IAsoc, [32](#)
- ~IBucket
 - IBucket, [35](#)
- ~IGraph
 - IGraph, [37](#)
- ~IKolejka
 - IKolejka, [42](#)
- ~ILista
 - ILista, [46](#)
- ~IQueue
 - IQueue, [50](#)
- ~IRunnable
 - IRunnable, [53](#)
- ~IStoper
 - IStoper, [54](#)
- ~IStos
 - IStos, [56](#)
- ~ITreeRB
 - ITreeRB, [70](#)
- ~Itabn
 - Itabn, [60](#)
- ~Kolejka
 - Kolejka, [73](#)
- ~Lista
 - Lista, [77](#)
- ~Queue
 - Queue, [94](#)
- ~Stos
 - Stos, [99](#)
- ~TreeRB
 - TreeRB, [121](#)
- ~asoc_test
 - asoc_test, [12](#)
- ~lista_test
 - lista_test, [82](#)
- ~tabn
 - tabn, [103](#)
- ~tabn_test
 - tabn_test, [112](#)
- ~tree_test
 - tree_test, [119](#)
- aSize

- Itabn, [61](#)
- tabn, [105](#)
- add
 - Asoc, [11](#)
 - Bucket, [15](#)
 - IAsoc, [32](#)
 - IBucket, [35](#)
 - ILista, [46](#)
 - Itabn, [61](#)
 - Lista, [77](#), [78](#)
 - tabn, [104](#)
- areAdjacent
 - Graph, [27](#)
 - IGraph, [37](#)
- Asoc
 - ~Asoc, [10](#)
 - add, [11](#)
 - Asoc, [10](#)
 - find, [11](#)
 - findOne, [11](#)
- Asoc< T, T2 >, [9](#)
- asoc.cpp, [125](#)
- asoc.hh, [125](#)
- asoc_test, [11](#)
 - ~asoc_test, [12](#)
 - asoc_test, [12](#)
 - prepare, [13](#)
 - run, [13](#)
- BFS
 - Graph, [27](#)
 - IGraph, [38](#)
- black
 - tree.hh, [152](#)
- bubblesort
 - Itabn, [62](#)
 - tabn, [105](#)
- Bucket
 - ~Bucket, [15](#)
 - add, [15](#)
 - Bucket, [15](#)
 - getID, [15](#)
 - lookup, [16](#)
 - lookupWhole, [16](#)
 - printAllElements, [16](#)
 - printFoundElements, [17](#)
 - remove, [17](#)
 - temp, [18](#)
- Bucket< T, T2 >, [14](#)

- cause
 - ExceptionBase, 25
- Colour
 - tree.hh, 152
- colour
 - nodeRB, 92
- ContinueException, 18
 - ContinueException, 19
 - Throw, 19
- CriticalException, 20
 - CriticalException, 21
 - Throw, 21
- DFS
 - Graph, 28
 - IGraph, 38
- dequeue
 - IKolejka, 43
 - IQueue, 50
 - Kolejka, 73
 - Queue, 94
- dumpToFile
 - main.cpp, 138
 - main.hh, 141
- enqueue
 - IKolejka, 43
 - IQueue, 50
 - Kolejka, 74
 - Queue, 95
- entry
 - entry, 22
 - getKey, 22
 - getVal, 22
 - operator<, 23
 - operator<<, 23
 - operator<=, 23
 - operator>, 23
 - operator>>, 23
 - operator>=, 23
 - operator=, 22
 - operator==, 23
- entry< T, T2 >, 21
- except.cpp, 126
- except.hh, 127
 - what, 128
- ExceptionBase, 24
 - cause, 25
 - ExceptionBase, 24
 - operator<<, 25
 - Throw, 25
- find
 - Asoc, 11
 - IASoc, 32
- findOne
 - Asoc, 11
 - IASoc, 33
- get
 - IKolejka, 43
 - ILista, 46
 - IQueue, 51
 - IStos, 56
 - Kolejka, 74
 - Lista, 78
 - Queue, 95
 - Stos, 99
- getColour
 - nodeRB, 87
- getElapsedTimeMs
 - IStoper, 54
 - Stoper, 96
- getID
 - Bucket, 15
 - IBucket, 35
- getKey
 - entry, 22
 - nodeRB, 88
- getLeft
 - nodeRB, 88
- getLeftKey
 - nodeRB, 88
- getNeighbours
 - Graph, 29
 - IGraph, 39
- getParent
 - nodeRB, 89
- getParentKey
 - nodeRB, 89
- getRight
 - nodeRB, 89
- getRightKey
 - nodeRB, 89
- getVal
 - entry, 22
- Graph, 25
 - ~Graph, 26
 - areAdjacent, 27
 - BFS, 27
 - DFS, 28
 - getNeighbours, 29
 - Graph, 26
 - insertEdge, 29
 - insertVertex, 30
 - isEmpty, 30
 - numberOfEdges, 31
- graph.cpp, 129
- graph.hh, 129
- graph2.cpp, 131
- graph2.hh, 131
- hash.cpp, 132
- hash.hh, 132
- IASoc
 - ~IASoc, 32
 - add, 32

- find, 32
- findOne, 33
- operator<<, 33
- IAsoc< T, T2 >, 31
- IBucket
 - ~IBucket, 35
 - add, 35
 - getID, 35
 - lookup, 35
 - lookupWhole, 35
 - printAllElements, 35
 - printFoundElements, 35
 - remove, 35
- IBucket< T, T2 >, 34
- IGraph, 36
 - ~IGraph, 37
 - areAdjacent, 37
 - BFS, 38
 - DFS, 38
 - getNeighbours, 39
 - insertEdge, 39
 - insertVertex, 40
 - isEmpty, 40
 - numberOfEdges, 41
- IKolejka
 - ~IKolejka, 42
 - dequeue, 43
 - enqueue, 43
 - get, 43
 - isEmpty, 44
 - operator<<, 44
- IKolejka< T >, 41
- ILista
 - ~ILista, 46
 - add, 46
 - get, 46
 - isEmpty, 47
 - qs, 47
 - remove, 48
 - size, 48
- ILista< T >, 45
- IQueue
 - ~IQueue, 50
 - dequeue, 50
 - enqueue, 50
 - get, 51
 - isEmpty, 51
 - operator<<, 52
- IQueue< T >, 49
- IRunnable, 52
 - ~IRunnable, 53
 - prepare, 53
 - run, 53
- IStoper, 53
 - ~IStoper, 54
 - getElapsedTimeMs, 54
 - start, 54
 - stop, 55
- IStos
 - ~IStos, 56
 - get, 56
 - isEmpty, 57
 - operator<<, 59
 - pop, 57
 - push, 58
- IStos< T >, 55
- ITreeRB< T >, 69
- ITreeRB
 - ~ITreeRB, 70
 - insert, 70
 - leftRot, 71
 - operator<<, 71
 - retRoot, 71
 - rightRot, 71
 - search, 71
- insert
 - ITreeRB, 70
 - TreeRB, 122
- insertEdge
 - Graph, 29
 - IGraph, 39
- insertVertex
 - Graph, 30
 - IGraph, 40
- isEmpty
 - Graph, 30
 - IGraph, 40
 - IKolejka, 44
 - ILista, 47
 - IQueue, 51
 - IStos, 57
 - Itabn, 62
 - Kolejka, 75
 - Lista, 78
 - Queue, 95
 - Stos, 100
 - tabn, 106
- Itabn
 - ~Itabn, 60
 - aSize, 61
 - add, 61
 - bubblesort, 62
 - isEmpty, 62
 - maxIndex, 63
 - nOE, 63
 - operator<<, 69
 - operator[], 64
 - remove, 64, 65
 - search, 65
 - searchIndex, 66
 - searchObject, 66
 - show, 67
 - showElems, 68
- Itabn< T >, 59
- key
 - nodeRB, 92

- Kolejka
 - ~Kolejka, [73](#)
 - dequeue, [73](#)
 - enqueue, [74](#)
 - get, [74](#)
 - isEmpty, [75](#)
 - Kolejka, [73](#)
- Kolejka< T >, [72](#)
- kolejka.cpp, [134](#)
- kolejka.hh, [134](#)
- left
 - nodeRB, [92](#)
- leftRot
 - ITreeRB, [71](#)
 - TreeRB, [122](#)
- Lista
 - ~Lista, [77](#)
 - add, [77](#), [78](#)
 - get, [78](#)
 - isEmpty, [78](#)
 - Lista, [77](#)
 - qs, [79](#)
 - remove, [79](#), [80](#)
 - size, [80](#)
- Lista< T >, [75](#)
- lista.cpp, [136](#)
- lista.hh, [136](#)
- lista_test, [81](#)
 - ~lista_test, [82](#)
 - lista_test, [82](#)
 - prepare, [82](#)
 - run, [83](#)
- lookup
 - Bucket, [16](#)
 - IBucket, [35](#)
- lookupWhole
 - Bucket, [16](#)
 - IBucket, [35](#)
- main
 - main.cpp, [139](#)
- main.cpp, [138](#)
 - dumpToFile, [138](#)
 - main, [139](#)
 - printOnscreen, [139](#)
- main.hh, [140](#)
 - dumpToFile, [141](#)
 - printOnscreen, [141](#)
- maxIndex
 - Itabn, [63](#)
 - tabn, [106](#)
- nOE
 - Itabn, [63](#)
 - tabn, [106](#)
- next
 - node, [85](#)
- node
 - next, [85](#)
 - node, [84](#)
 - operator<, [85](#)
 - operator<<, [85](#)
 - operator<=, [85](#)
 - operator>, [85](#)
 - operator>=, [85](#)
 - operator=, [84](#)
 - operator==, [85](#)
 - previous, [85](#)
 - value, [85](#)
- node< T >, [83](#)
- nodeRB< T >, [86](#)
- nodeRB
 - colour, [92](#)
 - getColour, [87](#)
 - getKey, [88](#)
 - getLeft, [88](#)
 - getLeftKey, [88](#)
 - getParent, [89](#)
 - getParentKey, [89](#)
 - getRight, [89](#)
 - getRightKey, [89](#)
 - key, [92](#)
 - left, [92](#)
 - nodeRB, [87](#)
 - operator<, [91](#)
 - operator<<, [91](#)
 - operator<=, [92](#)
 - operator>, [92](#)
 - operator>=, [92](#)
 - operator=, [90](#)
 - operator==, [92](#)
 - right, [92](#)
 - setColour, [90](#)
 - setKey, [90](#)
 - setLeft, [90](#)
 - setParent, [91](#)
 - setRight, [91](#)
 - up, [93](#)
- numberOfEdges
 - Graph, [31](#)
 - IGraph, [41](#)
- operator<
 - entry, [23](#)
 - node, [85](#)
 - nodeRB, [91](#)
- operator<<
 - entry, [23](#)
 - ExceptionBase, [25](#)
 - IAsoc, [33](#)
 - IKolejka, [44](#)
 - IQueue, [52](#)
 - IStos, [59](#)
 - ITreeRB, [71](#)
 - Itabn, [69](#)
 - node, [85](#)
 - nodeRB, [91](#)

- tree.cpp, [151](#)
- tree.hh, [153](#)
- operator<=
 - entry, [23](#)
 - node, [85](#)
 - nodeRB, [92](#)
- operator>
 - entry, [23](#)
 - node, [85](#)
 - nodeRB, [92](#)
- operator>>
 - entry, [23](#)
- operator>=
 - entry, [23](#)
 - node, [85](#)
 - nodeRB, [92](#)
- operator=
 - entry, [22](#)
 - node, [84](#)
 - nodeRB, [90](#)
- operator==
 - entry, [23](#)
 - node, [85](#)
 - nodeRB, [92](#)
- operator[]
 - ltabn, [64](#)
 - tabn, [107](#)
- pop
 - IStos, [57](#)
 - Stos, [100](#)
- prepare
 - asoc_test, [13](#)
 - IRunnable, [53](#)
 - lista_test, [82](#)
 - tabn_test, [112](#)
 - test_graph_BFS, [115](#)
 - test_graph_DFS, [117](#)
 - tree_test, [119](#)
- previous
 - node, [85](#)
- printAllElements
 - Bucket, [16](#)
 - IBucket, [35](#)
- printFoundElements
 - Bucket, [17](#)
 - IBucket, [35](#)
- printOnscreen
 - main.cpp, [139](#)
 - main.hh, [141](#)
- push
 - IStos, [58](#)
 - Stos, [101](#)
- qs
 - ILista, [47](#)
 - Lista, [79](#)
- Queue
 - ~Queue, [94](#)
- dequeue, [94](#)
- enqueue, [95](#)
- get, [95](#)
- isEmpty, [95](#)
- Queue, [94](#)
- Queue< T >, [93](#)
- red
 - tree.hh, [152](#)
- remove
 - Bucket, [17](#)
 - IBucket, [35](#)
 - ILista, [48](#)
 - ltabn, [64](#), [65](#)
 - Lista, [79](#), [80](#)
 - tabn, [108](#)
- retRoot
 - ITreeRB, [71](#)
 - TreeRB, [123](#)
- right
 - nodeRB, [92](#)
- rightRot
 - ITreeRB, [71](#)
 - TreeRB, [123](#)
- run
 - asoc_test, [13](#)
 - IRunnable, [53](#)
 - lista_test, [83](#)
 - tabn_test, [113](#)
 - test_graph_BFS, [115](#)
 - test_graph_DFS, [117](#)
 - tree_test, [119](#)
- run.cpp, [142](#)
- run.hh, [142](#)
- SIZE
 - tabl.hh, [150](#)
- search
 - ITreeRB, [71](#)
 - ltabn, [65](#)
 - tabn, [109](#)
 - TreeRB, [123](#)
- searchIndex
 - ltabn, [66](#)
 - tabn, [109](#)
- searchObject
 - ltabn, [66](#)
 - tabn, [110](#)
- setColour
 - nodeRB, [90](#)
- setKey
 - nodeRB, [90](#)
- setLeft
 - nodeRB, [90](#)
- setParent
 - nodeRB, [91](#)
- setRight
 - nodeRB, [91](#)
- show

- ltabn, 67
- tabn, 110
- showElems
 - ltabn, 68
 - tabn, 111
- size
 - lLista, 48
 - Lista, 80
- start
 - IStoper, 54
 - Stoper, 97
- stop
 - IStoper, 55
 - Stoper, 97
- Stoper, 95
 - getElapsedTimeMs, 96
 - start, 97
 - stop, 97
- stoper.cpp, 144
- stoper.hh, 144
- Stos
 - ~Stos, 99
 - get, 99
 - isEmpty, 100
 - pop, 100
 - push, 101
 - Stos, 99
- Stos< T >, 97
- stos.cpp, 145
- stos.hh, 146
- tabl.cpp, 148
- tabl.hh, 148
 - SIZE, 150
- tabn
 - ~tabn, 103
 - aSize, 105
 - add, 104
 - bubblesort, 105
 - isEmpty, 106
 - maxIndex, 106
 - nOE, 106
 - operator[], 107
 - remove, 108
 - search, 109
 - searchIndex, 109
 - searchObject, 110
 - show, 110
 - showElems, 111
 - tabn, 103
- tabn< T >, 101
- tabn_test, 111
 - ~tabn_test, 112
 - prepare, 112
 - run, 113
 - tabn_test, 112
- temp
 - Bucket, 18
- test_graph_BFS, 114
 - prepare, 115
 - run, 115
 - test_graph_BFS, 115
- test_graph_DFS, 116
 - prepare, 117
 - run, 117
 - test_graph_DFS, 117
- Throw
 - ContinueException, 19
 - CriticalException, 21
 - ExceptionBase, 25
- tree.cpp, 150
 - operator<<, 151
- tree.hh, 151
 - black, 152
 - Colour, 152
 - operator<<, 153
 - red, 152
- tree_test, 118
 - ~tree_test, 119
 - prepare, 119
 - run, 119
 - tree_test, 119
- TreeRB< T >, 120
- TreeRB
 - ~TreeRB, 121
 - insert, 122
 - leftRot, 122
 - retRoot, 123
 - rightRot, 123
 - search, 123
 - TreeRB, 121
- up
 - nodeRB, 93
- value
 - node, 85
- what
 - except.hh, 128