

pamsi

0.5

Wygenerowano przez Doxygen 1.8.11



# Spis treści

<b>1</b>	<b>Strona główna</b>	<b>1</b>
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
<b>2</b>	<b>Indeks hierarchiczny</b>	<b>3</b>
2.1	Hierarchia klas . . . . .	3
<b>3</b>	<b>Indeks klas</b>	<b>5</b>
3.1	Lista klas . . . . .	5
<b>4</b>	<b>Indeks plików</b>	<b>7</b>
4.1	Lista plików . . . . .	7
<b>5</b>	<b>Dokumentacja klas</b>	<b>9</b>
5.1	Dokumentacja szablonu klasy <code>Asoc&lt; T, T2 &gt;</code> . . . . .	9
5.1.1	Opis szczegółowy . . . . .	10
5.1.2	Dokumentacja konstruktora i destruktora . . . . .	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	<code>ContinueException()</code>	19
5.4.2.2	<code>ContinueException(std::string description)</code>	19
5.4.3	Dokumentacja funkcji składowych	19
5.4.3.1	<code>Throw()</code>	19
5.5	Dokumentacja klasy <code>CriticalException</code>	20
5.5.1	Opis szczegółowy	20
5.5.2	Dokumentacja konstruktora i destruktora	21
5.5.2.1	<code>CriticalException()</code>	21
5.5.2.2	<code>CriticalException(std::string description)</code>	21
5.5.3	Dokumentacja funkcji składowych	21
5.5.3.1	<code>Throw()</code>	21
5.6	Dokumentacja szablonu klasy <code>entry&lt; T, T2 &gt;</code>	21
5.6.1	Opis szczegółowy	22
5.6.2	Dokumentacja konstruktora i destruktora	22
5.6.2.1	<code>entry()</code>	22
5.6.2.2	<code>entry(T entryKey, T2 entryData)</code>	22
5.6.3	Dokumentacja funkcji składowych	22
5.6.3.1	<code>getKey(void)</code>	22
5.6.3.2	<code>getVal(void)</code>	22
5.6.3.3	<code>operator=(const entry&lt; T, T2 &gt; &amp;read)</code>	23
5.6.4	Dokumentacja przyjaciół i funkcji związanych	23
5.6.4.1	<code>operator&lt;</code>	23
5.6.4.2	<code>operator&lt;&lt;</code>	23
5.6.4.3	<code>operator&lt;=</code>	23
5.6.4.4	<code>operator==</code>	23
5.6.4.5	<code>operator&gt;</code>	23
5.6.4.6	<code>operator&gt;=</code>	23
5.6.4.7	<code>operator&gt;&gt;</code>	23
5.7	Dokumentacja klasy <code>ExceptionBase</code>	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() . . . . .	26
5.8.3	Dokumentacja funkcji składowych . . . . .	27
5.8.3.1	areAdjacent(int index1, int index2) . . . . .	27
5.8.3.2	getNeighbours(int index) . . . . .	27
5.8.3.3	insertEdge(int index1, int index2) . . . . .	27
5.8.3.4	insertVertex(int elem) . . . . .	28
5.9	Dokumentacja szablonu klasy IAsoc< T, T2 > . . . . .	28
5.9.1	Opis szczegółowy . . . . .	29
5.9.2	Dokumentacja konstruktora i destruktora . . . . .	29
5.9.2.1	~IAsoc() . . . . .	29
5.9.3	Dokumentacja funkcji składowych . . . . .	29
5.9.3.1	add(T, T2)=0 . . . . .	29
5.9.3.2	find(T)=0 . . . . .	29
5.9.3.3	findOne(T)=0 . . . . .	30
5.9.4	Dokumentacja przyjaciół i funkcji związanych . . . . .	30
5.9.4.1	operator<< . . . . .	30
5.10	Dokumentacja szablonu klasy IBucket< T, T2 > . . . . .	30

5.10.1	Opis szczegółowy . . . . .	31
5.10.2	Dokumentacja konstruktora i destruktora . . . . .	31
5.10.2.1	~IBucket() . . . . .	31
5.10.3	Dokumentacja funkcji składowych . . . . .	31
5.10.3.1	add(entry< T, T2 >)=0 . . . . .	31
5.10.3.2	getID(void)=0 . . . . .	32
5.10.3.3	lookup(T)=0 . . . . .	32
5.10.3.4	lookupWhole(T)=0 . . . . .	32
5.10.3.5	printAllElements()=0 . . . . .	32
5.10.3.6	printFoundElements(void)=0 . . . . .	32
5.10.3.7	remove(T)=0 . . . . .	32
5.11	Dokumentacja klasy IGraph . . . . .	32
5.11.1	Opis szczegółowy . . . . .	33
5.11.2	Dokumentacja konstruktora i destruktora . . . . .	33
5.11.2.1	~IGraph() . . . . .	33
5.11.3	Dokumentacja funkcji składowych . . . . .	33
5.11.3.1	areAdjacent(int, int)=0 . . . . .	33
5.11.3.2	getNeighbours(int)=0 . . . . .	34
5.11.3.3	insertEdge(int, int)=0 . . . . .	34
5.11.3.4	insertVertex(int)=0 . . . . .	35
5.12	Dokumentacja szablonu klasy IKolejka< T > . . . . .	35
5.12.1	Opis szczegółowy . . . . .	36
5.12.2	Dokumentacja konstruktora i destruktora . . . . .	36
5.12.2.1	~IKolejka() . . . . .	36
5.12.3	Dokumentacja funkcji składowych . . . . .	36
5.12.3.1	dequeue(void)=0 . . . . .	36
5.12.3.2	enqueue(T)=0 . . . . .	36
5.12.3.3	get(void)=0 . . . . .	37
5.12.3.4	isEmpty(void)=0 . . . . .	37
5.13	Dokumentacja szablonu klasy ILista< T > . . . . .	38

5.13.1	Opis szczegółowy . . . . .	39
5.13.2	Dokumentacja konstruktora i destruktora . . . . .	39
5.13.2.1	~ILista() . . . . .	39
5.13.3	Dokumentacja funkcji składowych . . . . .	39
5.13.3.1	add(T, int)=0 . . . . .	39
5.13.3.2	add(T)=0 . . . . .	40
5.13.3.3	get(int)=0 . . . . .	40
5.13.3.4	isEmpty(void)=0 . . . . .	41
5.13.3.5	qs(int, int)=0 . . . . .	41
5.13.3.6	remove(int)=0 . . . . .	42
5.13.3.7	remove(void)=0 . . . . .	42
5.13.3.8	size(void)=0 . . . . .	42
5.14	Dokumentacja klasy IRunnable . . . . .	43
5.14.1	Opis szczegółowy . . . . .	43
5.14.2	Dokumentacja konstruktora i destruktora . . . . .	43
5.14.2.1	~IRunnable() . . . . .	43
5.14.3	Dokumentacja funkcji składowych . . . . .	43
5.14.3.1	prepare(int)=0 . . . . .	43
5.14.3.2	run()=0 . . . . .	44
5.15	Dokumentacja klasy IStoper . . . . .	44
5.15.1	Opis szczegółowy . . . . .	44
5.15.2	Dokumentacja konstruktora i destruktora . . . . .	45
5.15.2.1	~IStoper() . . . . .	45
5.15.3	Dokumentacja funkcji składowych . . . . .	45
5.15.3.1	getElapsedTimeMs(void)=0 . . . . .	45
5.15.3.2	start(void)=0 . . . . .	45
5.15.3.3	stop(void)=0 . . . . .	45
5.16	Dokumentacja szablonu klasy IStos< T > . . . . .	46
5.16.1	Opis szczegółowy . . . . .	46
5.16.2	Dokumentacja konstruktora i destruktora . . . . .	46



5.16.2.1	~IStos()	46
5.16.3	Dokumentacja funkcji składowych	47
5.16.3.1	get(void)=0	47
5.16.3.2	isEmpty(void)=0	47
5.16.3.3	pop(void)=0	47
5.16.3.4	push(T)=0	48
5.17	Dokumentacja szablonu klasy Itabn< T >	48
5.17.1	Opis szczegółowy	50
5.17.2	Dokumentacja konstruktora i destruktor	50
5.17.2.1	~Itabn()	50
5.17.3	Dokumentacja funkcji składowych	50
5.17.3.1	add(T)=0	50
5.17.3.2	add(T, int)=0	50
5.17.3.3	aSize(void)=0	51
5.17.3.4	bubblesort()=0	51
5.17.3.5	isEmpty(void)=0	52
5.17.3.6	nOE(void)=0	52
5.17.3.7	operator[](int)=0	53
5.17.3.8	operator[](int) const =0	53
5.17.3.9	remove()=0	53
5.17.3.10	remove(int)=0	53
5.17.3.11	search(T)=0	53
5.17.3.12	show(int) const =0	54
5.17.3.13	showElems(void)=0	55
5.17.4	Dokumentacja przyjaciół i funkcji związanych	56
5.17.4.1	operator<<	56
5.18	Dokumentacja szablonu klasy ITreeRB< T >	56
5.18.1	Opis szczegółowy	56
5.18.2	Dokumentacja konstruktora i destruktor	57
5.18.2.1	~ITreeRB()	57

5.18.3 Dokumentacja funkcji składowych	57
5.18.3.1 insert(T)=0	57
5.18.3.2 insert(T, nodeRB< T > *)=0	57
5.18.3.3 search(T)=0	57
5.18.4 Dokumentacja przyjaciół i funkcji związanych	58
5.18.4.1 operator<<	58
5.19 Dokumentacja szablonu klasy Kolejka< T >	58
5.19.1 Opis szczegółowy	59
5.19.2 Dokumentacja konstruktora i destruktor	59
5.19.2.1 Kolejka()	59
5.19.2.2 ~Kolejka()	60
5.19.3 Dokumentacja funkcji składowych	60
5.19.3.1 dequeue(void)	60
5.19.3.2 enqueue(T)	60
5.19.3.3 get(void)	60
5.19.3.4 isEmpty(void)	61
5.20 Dokumentacja szablonu klasy Lista< T >	61
5.20.1 Opis szczegółowy	63
5.20.2 Dokumentacja konstruktora i destruktor	63
5.20.2.1 Lista()	63
5.20.2.2 ~Lista()	63
5.20.3 Dokumentacja funkcji składowych	63
5.20.3.1 add(T, int)	63
5.20.3.2 add(T)	64
5.20.3.3 get(int position)	64
5.20.3.4 isEmpty(void)	65
5.20.3.5 qs(int, int)	65
5.20.3.6 remove(int position)	65
5.20.3.7 remove(void)	66
5.20.3.8 size(void)	66

5.21 Dokumentacja klasy lista_test . . . . .	67
5.21.1 Opis szczegółowy . . . . .	68
5.21.2 Dokumentacja konstruktora i destruktor . . . . .	68
5.21.2.1 lista_test() . . . . .	68
5.21.2.2 ~lista_test() . . . . .	68
5.21.3 Dokumentacja funkcji składowych . . . . .	68
5.21.3.1 prepare(int sizeOfTest) . . . . .	68
5.21.3.2 run() . . . . .	69
5.22 Dokumentacja szablonu klasy nodeRB< T > . . . . .	69
5.22.1 Opis szczegółowy . . . . .	70
5.22.2 Dokumentacja konstruktora i destruktor . . . . .	70
5.22.2.1 nodeRB(T addKey, Colour col=red, nodeRB< T > *addUp=NULL, nodeRB< T > *addLeft=NULL, nodeRB< T > *addRight=NULL) . . . . .	70
5.22.3 Dokumentacja funkcji składowych . . . . .	71
5.22.3.1 getColour(void) . . . . .	71
5.22.3.2 getKey(void) . . . . .	71
5.22.3.3 getLeft(void) . . . . .	71
5.22.3.4 getLeftKey(void) . . . . .	71
5.22.3.5 getParent(void) . . . . .	72
5.22.3.6 getParentKey(void) . . . . .	72
5.22.3.7 getRight(void) . . . . .	72
5.22.3.8 getRightKey(void) . . . . .	72
5.22.3.9 operator=(const nodeRB< T > &read) . . . . .	72
5.22.3.10 setColour(Colour colourToSet) . . . . .	72
5.22.3.11 setKey(T keyToSet) . . . . .	73
5.22.3.12 setLeft(nodeRB< T > *leftDescendant) . . . . .	73
5.22.3.13 setParent(nodeRB< T > *parent) . . . . .	73
5.22.3.14 setRight(nodeRB< T > *rightDescendant) . . . . .	73
5.22.4 Dokumentacja przyjaciół i funkcji związanych . . . . .	73
5.22.4.1 operator< . . . . .	73
5.22.4.2 operator<< . . . . .	74

5.22.4.3	operator<=	74
5.22.4.4	operator==	74
5.22.4.5	operator>	74
5.22.4.6	operator>=	74
5.22.4.7	operator>>	74
5.22.5	Dokumentacja atrybutów składowych	74
5.22.5.1	colour	74
5.22.5.2	key	74
5.22.5.3	left	74
5.22.5.4	right	74
5.22.5.5	up	75
5.23	Dokumentacja klasy Stoper	75
5.23.1	Opis szczegółowy	76
5.23.2	Dokumentacja funkcji składowych	76
5.23.2.1	getElapsedTimeMs(void)	76
5.23.2.2	start(void)	76
5.23.2.3	stop(void)	76
5.24	Dokumentacja szablonu klasy Stos< T >	77
5.24.1	Opis szczegółowy	78
5.24.2	Dokumentacja konstruktora i destruktora	78
5.24.2.1	Stos()	78
5.24.2.2	~Stos()	78
5.24.3	Dokumentacja funkcji składowych	78
5.24.3.1	get(void)	78
5.24.3.2	isEmpty(void)	79
5.24.3.3	pop(void)	79
5.24.3.4	push(T)	80
5.25	Dokumentacja szablonu klasy tabn< T >	80
5.25.1	Opis szczegółowy	82
5.25.2	Dokumentacja konstruktora i destruktora	82

5.25.2.1	tabn()	82
5.25.2.2	~tabn()	82
5.25.3	Dokumentacja funkcji składowych	83
5.25.3.1	add(T)	83
5.25.3.2	add(T, int)	83
5.25.3.3	aSize(void)	83
5.25.3.4	bubblesort(void)	84
5.25.3.5	isEmpty(void)	84
5.25.3.6	nOE(void)	84
5.25.3.7	operator[](int index)	85
5.25.3.8	operator[](int index) const	85
5.25.3.9	remove()	86
5.25.3.10	remove(int)	86
5.25.3.11	search(T)	87
5.25.3.12	show(int) const	87
5.25.3.13	showElems(void)	88
5.26	Dokumentacja klasy tabn_test	88
5.26.1	Opis szczegółowy	89
5.26.2	Dokumentacja konstruktora i destruktora	89
5.26.2.1	tabn_test()	89
5.26.2.2	~tabn_test()	89
5.26.3	Dokumentacja funkcji składowych	89
5.26.3.1	prepare(int sizeOfTest)	89
5.26.3.2	run()	90
5.27	Dokumentacja klasy tree_test	90
5.27.1	Opis szczegółowy	91
5.27.2	Dokumentacja konstruktora i destruktora	91
5.27.2.1	tree_test()	91
5.27.2.2	~tree_test()	92
5.27.3	Dokumentacja funkcji składowych	92
5.27.3.1	prepare(int sizeOfTest)	92
5.27.3.2	run()	92
5.28	Dokumentacja szablonu klasy TreeRB< T >	93
5.28.1	Opis szczegółowy	94
5.28.2	Dokumentacja konstruktora i destruktora	94
5.28.2.1	TreeRB()	94
5.28.2.2	~TreeRB()	94
5.28.3	Dokumentacja funkcji składowych	94
5.28.3.1	insert(T element)	94
5.28.3.2	insert(T element, nodeRB< T > *node)	94
5.28.3.3	search(T k)	95

<b>6 Dokumentacja plików</b>	<b>97</b>
6.1 Dokumentacja pliku asoc.cpp	97
6.2 Dokumentacja pliku asoc.hh	97
6.3 Dokumentacja pliku except.cpp	98
6.4 Dokumentacja pliku except.hh	99
6.4.1 Opis szczegółowy	100
6.4.2 Dokumentacja funkcji	100
6.4.2.1 what(ExceptT &except)	100
6.5 Dokumentacja pliku graph.cpp	101
6.6 Dokumentacja pliku graph.hh	102
6.7 Dokumentacja pliku hash.cpp	103
6.8 Dokumentacja pliku hash.hh	103
6.9 Dokumentacja pliku kolejka.cpp	105
6.10 Dokumentacja pliku kolejka.hh	105
6.11 Dokumentacja pliku lista.cpp	107
6.12 Dokumentacja pliku lista.hh	107
6.13 Dokumentacja pliku main.cpp	109
6.13.1 Opis szczegółowy	109
6.13.2 Dokumentacja funkcji	110
6.13.2.1 dumpToFile(string nameOfFile, unsigned int testsize, IStoper *stoper)	110
6.13.2.2 main(void)	110
6.13.2.3 printOnscreen(unsigned int testsize, IStoper *stoper)	110
6.14 Dokumentacja pliku main.hh	111
6.14.1 Dokumentacja funkcji	112
6.14.1.1 dumpToFile(std::string, unsigned int, IStoper *)	112
6.14.1.2 printOnscreen(unsigned int, IStoper *)	112
6.15 Dokumentacja pliku run.cpp	113
6.16 Dokumentacja pliku run.hh	113
6.16.1 Opis szczegółowy	114
6.17 Dokumentacja pliku stoper.cpp	115

6.18	Dokumentacja pliku stoper.hh . . . . .	115
6.19	Dokumentacja pliku stos.cpp . . . . .	116
6.20	Dokumentacja pliku stos.hh . . . . .	117
6.21	Dokumentacja pliku tabl.cpp . . . . .	119
6.22	Dokumentacja pliku tabl.hh . . . . .	119
6.22.1	Opis szczegółowy . . . . .	120
6.22.2	Dokumentacja definicji . . . . .	120
6.22.2.1	SIZE . . . . .	120
6.23	Dokumentacja pliku tree.cpp . . . . .	121
6.23.1	Dokumentacja funkcji . . . . .	121
6.23.1.1	operator<<(std::ostream &output, Colour col) . . . . .	121
6.24	Dokumentacja pliku tree.hh . . . . .	122
6.24.1	Dokumentacja typów wyliczanych . . . . .	123
6.24.1.1	Colour . . . . .	123
6.24.2	Dokumentacja funkcji . . . . .	123
6.24.2.1	operator<<(std::ostream &, Colour) . . . . .	123
<b>Indeks</b>		<b>125</b>





# 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 > . . . . .	30
Bucket< T, T2 > . . . . .	14
ExceptionBase . . . . .	24
ContinueException . . . . .	18
CriticalException . . . . .	20
IAsoc< T, T2 > . . . . .	28
Asoc< T, T2 > . . . . .	9
IAsoc< std::string, int > . . . . .	28
IGraph . . . . .	32
Graph . . . . .	25
IKolejka< T > . . . . .	35
Kolejka< T > . . . . .	58
ILista< T > . . . . .	38
Lista< T > . . . . .	61
ILista< std::string > . . . . .	38
IRunnable . . . . .	43
asoc_test . . . . .	11
lista_test . . . . .	67
tabn_test . . . . .	88
tree_test . . . . .	90
IStoper . . . . .	44
Stoper . . . . .	75
IStos< T > . . . . .	46
Stos< T > . . . . .	77
Itabn< T > . . . . .	48
tabn< T > . . . . .	80
Itabn< Bucket< T, T2 > > . . . . .	48
Itabn< entry< T, T2 > > . . . . .	48
Itabn< int > . . . . .	48
Itabn< Itabn< int > * > . . . . .	48
Itabn< T2 > . . . . .	48
ITreeRB< T > . . . . .	56
TreeRB< T > . . . . .	93
ITreeRB< int > . . . . .	56
nodeRB< T > . . . . .	69



## Rozdział 3

# Indeks klas

### 3.1 Lista klas

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

<a href="#">Asoc&lt; T, T2 &gt;</a>	9
<a href="#">asoc_test</a>	11
<a href="#">Bucket&lt; T, T2 &gt;</a>	14
<a href="#">ContinueException</a>	
Wyjątek, który może spowodować nieprzewidziane działanie programu, ale program mógłby dalej działać	18
<a href="#">CriticalException</a>	
Wyjątek krytyczny, wymagający zamknięcia programu	20
<a href="#">entry&lt; T, T2 &gt;</a>	
Klasa definiująca obiekt typu wpis	21
<a href="#">ExceptionBase</a>	
Ogólny wyjątek	24
<a href="#">Graph</a>	
Klasa implementująca interfejs grafu	25
<a href="#">IAsoc&lt; T, T2 &gt;</a>	28
<a href="#">IBucket&lt; T, T2 &gt;</a>	30
<a href="#">IGraph</a>	
Interfejs grafu	32
<a href="#">IKolejka&lt; T &gt;</a>	
Interfejs klasy <a href="#">Kolejka</a> Definiuje operacje dostępne dla klasy <a href="#">Kolejka</a>	35
<a href="#">ILista&lt; T &gt;</a>	
Interfejs listy	38
<a href="#">IRunnable</a>	
Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm	43
<a href="#">IStoper</a>	
Plik definiuje stoper, obliczający czas wykonywania badanych funkcji	44
<a href="#">IStos&lt; T &gt;</a>	
Interfejs stosu	46
<a href="#">Itabn&lt; T &gt;</a>	
Interfejs klasy tabn	48
<a href="#">ITreeRB&lt; T &gt;</a>	
Interfejs klasy drzewa czerwono-czarnego	56
<a href="#">Kolejka&lt; T &gt;</a>	
Klasa modeluje kolejkę	58
<a href="#">Lista&lt; T &gt;</a>	
Klasa lista	61

<a href="#">lista_test</a>		
	Definiuje sposób testowania wypełniania listy . . . . .	67
<a href="#">nodeRB&lt; T &gt;</a>	. . . . .	69
<a href="#">Stoper</a>		
	Klasa stoper implementująca interfejs <a href="#">IStoper</a> . . . . .	75
<a href="#">Stos&lt; T &gt;</a>		
	Klasa <a href="#">Stos</a> . . . . .	77
<a href="#">tabn&lt; T &gt;</a>		
	Modeluje tablicę dynamicznie rozszerzalną . . . . .	80
<a href="#">tabn_test</a>		
	Definiuje sposób testowania wypełniania tablicy tabn . . . . .	88
<a href="#">tree_test</a>	. . . . .	90
<a href="#">TreeRB&lt; T &gt;</a>		
	Klasa implementująca interfejs drzewa czerwono-czarnego . . . . .	93

## Rozdział 4

# Indeks plików

### 4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

<a href="#">asoc.cpp</a>	97
<a href="#">asoc.hh</a>	97
<a href="#">except.cpp</a>	98
<a href="#">except.hh</a>	
Plik zawiera definicje wyjątków	99
<a href="#">graph.cpp</a>	101
<a href="#">graph.hh</a>	102
<a href="#">hash.cpp</a>	103
<a href="#">hash.hh</a>	103
<a href="#">kolejka.cpp</a>	105
<a href="#">kolejka.hh</a>	105
<a href="#">lista.cpp</a>	107
<a href="#">lista.hh</a>	107
<a href="#">main.cpp</a>	
Główny plik programu	109
<a href="#">main.hh</a>	111
<a href="#">run.cpp</a>	113
<a href="#">run.hh</a>	
Plik definiuje interfejs <a href="#">IRunnable</a> , ujednolicający klasy umożliwiające badanie algorytmów	113
<a href="#">stoper.cpp</a>	115
<a href="#">stoper.hh</a>	115
<a href="#">stos.cpp</a>	116
<a href="#">stos.hh</a>	117
<a href="#">tabl.cpp</a>	119
<a href="#">tabl.hh</a>	
Definicja interfejsu <a href="#">Itabn</a> , klasy <a href="#">tabn</a> oraz klasy <a href="#">tabn_test</a>	119
<a href="#">tree.cpp</a>	121
<a href="#">tree.hh</a>	122





## Rozdział 5

# Dokumentacja klas

### 5.1 Dokumentacja szablonu klasy Asoc< T, T2 >

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

Diagram dziedziczenia dla Asoc< T, T2 >

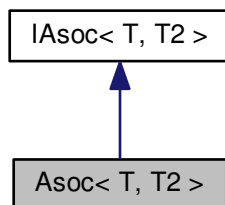
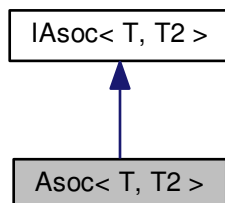


Diagram współpracy dla Asoc< T, T2 >:



## 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

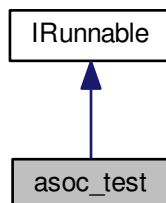
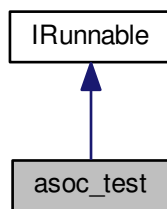


Diagram współpracy dla `asoc_test`:



## 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 destruktor

#### 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 ( ) [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 >`

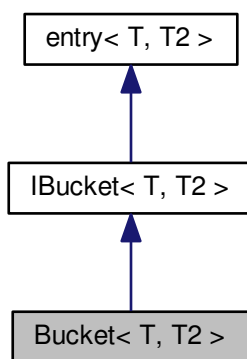
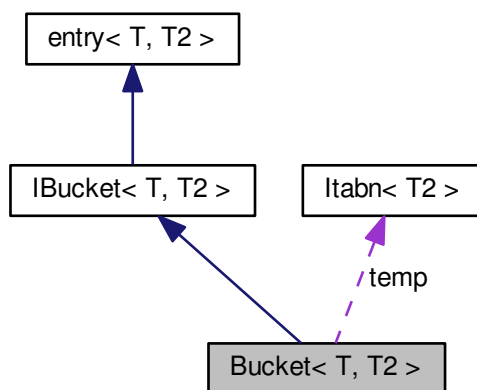


Diagram współpracy dla `Bucket< T, T2 >`:



#### 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 >`.

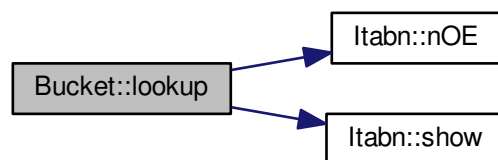
Definicja w linii 133 pliku hash.hh.

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:

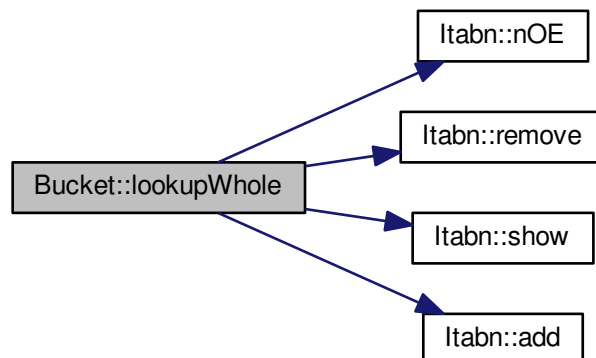


5.3.3.4 `template<class T, class T2> virtual Itabn<T2>* Bucket< T, T2 >::lookupWhole ( T position ) [inline],[virtual]`

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:

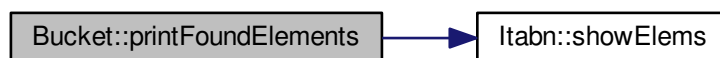


5.3.3.6 `template<class T, class T2> virtual void Bucket< T, T2 >::printFoundElements ( void ) [inline], [virtual]`

Implementuje `IBucket< T, T2 >`.

Definicja w linii 141 pliku `hash.hh`.

Oto graf wywołań dla tej funkcji:

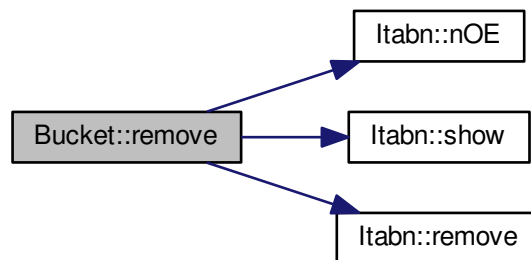


5.3.3.7 `template<class T, class T2> virtual T2 Bucket< T, T2 >::remove ( T position ) [inline], [virtual]`

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`

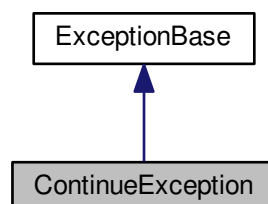
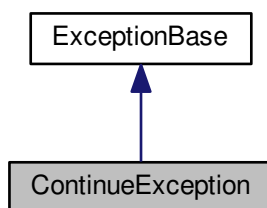


Diagram współpracy dla `ContinueException`:



### 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 destruktora

##### 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`

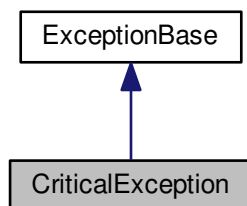
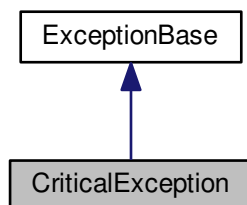


Diagram współpracy dla `CriticalException`:



### 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:

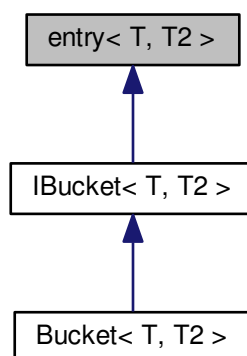
- [except.hh](#)

## 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 destruktora

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:

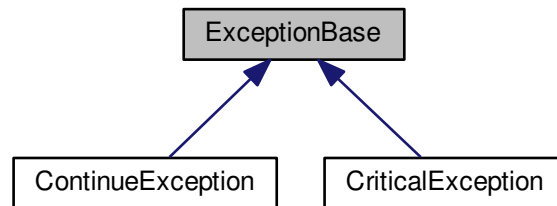
- [hash.hh](#)

## 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

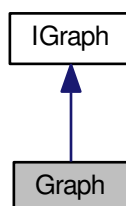
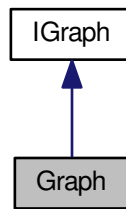


Diagram współpracy dla Graph:



## Metody publiczne

- [Graph](#) ()  
*Konstruktor klasy [Graph](#).*
- virtual [~Graph](#) ()  
*Destruktor klasy [Graph](#).*
- virtual void [insertVertex](#) (int elem)  
*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)

### 5.8.1 Opis szczegółowy

Klasa implementująca interfejs grafu.

Definicja w linii 28 pliku graph.hh.

### 5.8.2 Dokumentacja konstruktora i destruktora

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

Konstruktor klasy [Graph](#).

Definicja w linii 41 pliku graph.hh.

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

Destruktor klasy [Graph](#).

Definicja w linii 50 pliku graph.hh.

### 5.8.3 Dokumentacja funkcji składowych

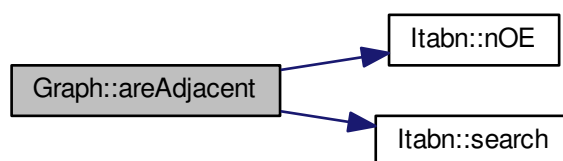
5.8.3.1 `virtual bool Graph::areAdjacent ( int index1, int index2 ) [inline],[virtual]`

Sprawdza, czy dwa elementy mają krawędź między sobą

Implementuje [IGraph](#).

Definicja w linii 84 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



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

Implementuje [IGraph](#).

Definicja w linii 97 pliku graph.hh.

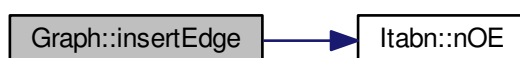
5.8.3.3 `virtual void Graph::insertEdge ( int index1, int index2 ) [inline],[virtual]`

Dodaje powiązanie między dwoma elementami.

Implementuje [IGraph](#).

Definicja w linii 70 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



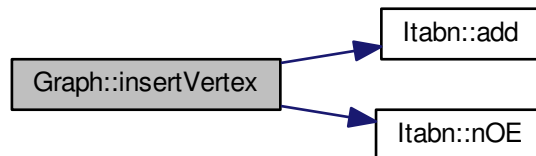
#### 5.8.3.4 virtual void Graph::insertVertex ( int *elem* ) [inline],[virtual]

Dodaje element do grafu.

Implementuje [IGraph](#).

Definicja w linii 58 pliku graph.hh.

Oto graf wywołań dla tej funkcji:



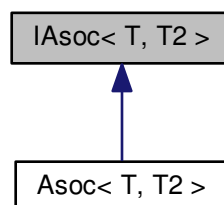
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [graph.hh](#)

## 5.9 Dokumentacja szablonu klasy `IAsoc< T, T2 >`

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

Diagram dziedziczenia dla `IAsoc< T, T2 >`



### Metody publiczne

- virtual void `add` (T, T2)=0
- virtual `~IAsoC` ()
- virtual `Itabn< 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 IAso< 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 IAso< T, T2 >::~IAso( ) [inline],[virtual]`

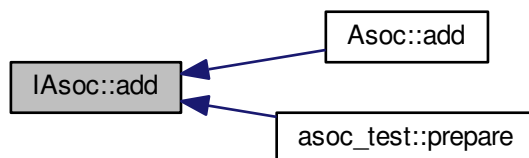
Definicja w linii 24 pliku `asoc.hh`.

### 5.9.3 Dokumentacja funkcji składowych

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

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

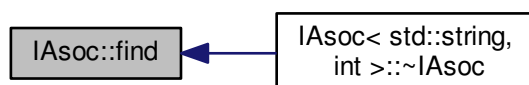
Oto graf wywołań tej funkcji:



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

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

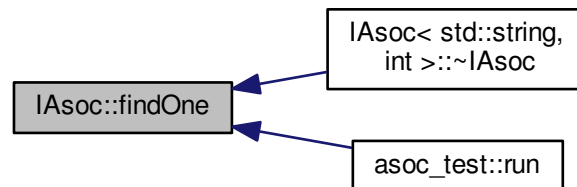
Oto graf wywołań tej funkcji:



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 >`

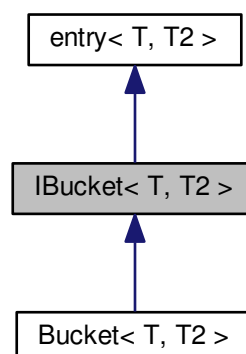
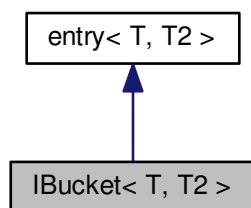


Diagram współpracy dla IBucket< T, T2 >:



### Metody publiczne

- virtual void [add](#) ([entry](#)< T, T2 >)=0
- virtual T2 [remove](#) (T)=0
- virtual T2 [lookup](#) (T)=0
- virtual [ltabn](#)< 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:

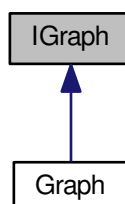
- [hash.hh](#)

## 5.11 Dokumentacja klasy IGraph

Interfejs grafu.

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

Diagram dziedziczenia dla IGraph





## Metody publiczne

- virtual [~IGraph](#) ()
- virtual void [insertVertex](#) (int)=0
- virtual void [insertEdge](#) (int, int)=0
- virtual [ltabn](#)< int > \* [getNeighbours](#) (int)=0
- virtual bool [areAdjacent](#) (int, int)=0

### 5.11.1 Opis szczegółowy

Interfejs grafu.

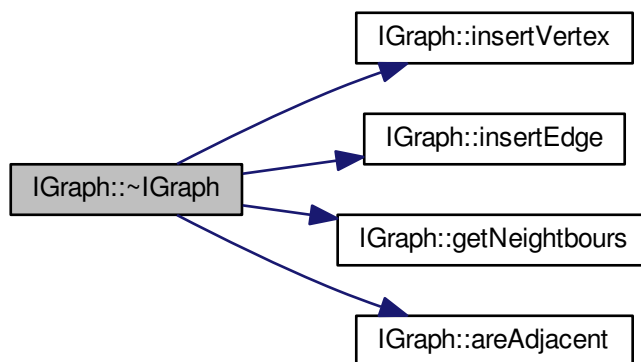
Definicja w linii 9 pliku graph.hh.

### 5.11.2 Dokumentacja konstruktora i destruktor

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

Definicja w linii 12 pliku graph.hh.

Oto graf wywołań dla tej funkcji:

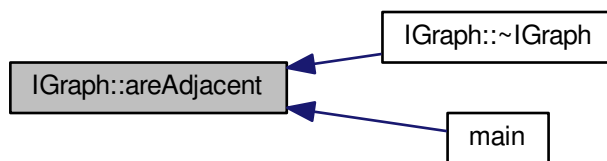


### 5.11.3 Dokumentacja funkcji składowych

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

Implementowany w [Graph](#).

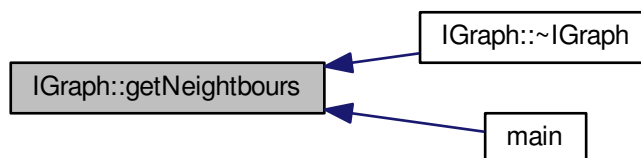
Oto graf wywoływań tej funkcji:



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

Implementowany w [Graph](#).

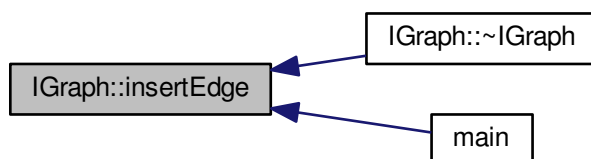
Oto graf wywoływań tej funkcji:



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

Implementowany w [Graph](#).

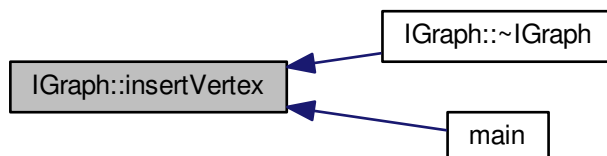
Oto graf wywoływań tej funkcji:



## 5.11.3.4 virtual void IGraph::insertVertex ( int ) [pure virtual]

Implementowany w [Graph](#).

Oto graf wywołań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

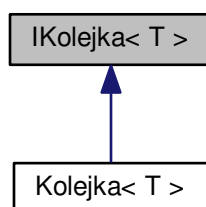
- [graph.hh](#)

## 5.12 Dokumentacja szablonu klasy IKolejka&lt; T &gt;

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.*

### 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:



Dokumentacja dla tej klasy została wygenerowana z pliku:

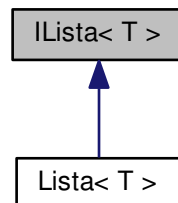
- [kolejka.hh](#)

### 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.

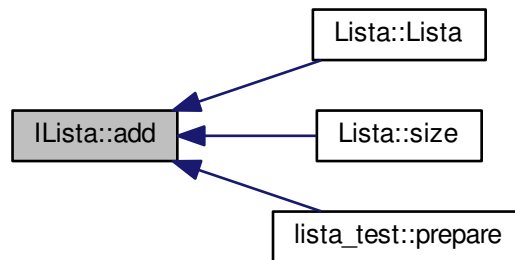
Jeśli następuje próba dodania elementu w miejscu istniejącego, następuje przesunięcie następujących po nim elementów na następne pozycje

**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**

<code>T</code>	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 >](#).

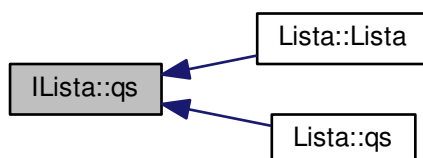
Oto graf wywoływań tej funkcji:



#### 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

<i>T</i>	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

<i>int</i>	ilość elementów
------------	-----------------

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

Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

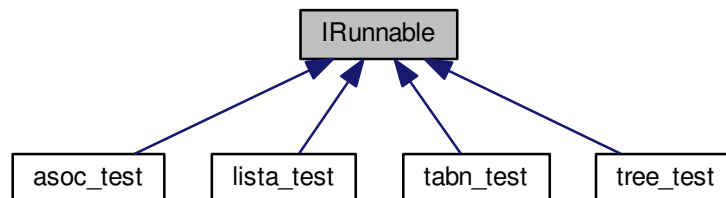
- [lista.hh](#)

## 5.14 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`.*

#### 5.14.1 Opis szczegółowy

Interfejs ujednolicający sposób uruchamiania klasy badającej algorytm.

Definicja w linii 16 pliku `run.hh`.

#### 5.14.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny `IRunnable`.

Definicja w linii 31 pliku `run.hh`.

#### 5.14.3 Dokumentacja funkcji składowych

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

Przygotowanie badań

Implementowany w `tabn_test`, `tree_test`, `lista_test` i `asoc_test`.

#### 5.14.3.2 virtual bool IRunnable::run ( ) [pure virtual]

Przeprowadzanie badań

Implementowany w [tabn\\_test](#), [tree\\_test](#), [lista\\_test](#) i [asoc\\_test](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

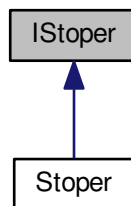
- [run.hh](#)

## 5.15 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](#) ()

#### 5.15.1 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`.

### 5.15.2 Dokumentacja konstruktora i destruktora

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

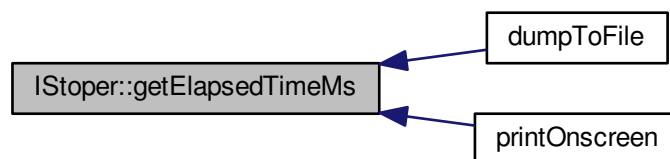
Definicja w linii 25 pliku stoper.hh.

### 5.15.3 Dokumentacja funkcji składowych

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

Implementowany w [Stoper](#).

Oto graf wywołań tej funkcji:



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

Implementowany w [Stoper](#).

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

Implementowany w [Stoper](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

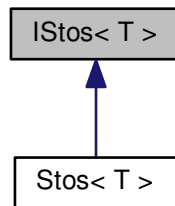
- [stoper.hh](#)

## 5.16 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`.*

### 5.16.1 Opis szczegółowy

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

Interfejs stosu.

Definiuje dostępne operacje na klasie `Stos`

Definicja w linii 16 pliku `stos.hh`.

### 5.16.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny `IStos`.

Definicja w linii 53 pliku `stos.hh`.

### 5.16.3 Dokumentacja funkcji składowych

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

Zwraca element ze szczytu stosu bez jego usuwania.

Zwracane wartości

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

#### 5.16.3.2 `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:



#### 5.16.3.3 `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:

**5.16.3.4** `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:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [stos.hh](#)

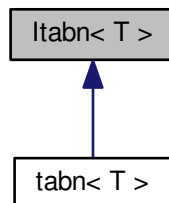
**5.17 Dokumentacja szablonu klasy Itabn< T >**

Interfejs klasy tabn.



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

Diagram dziedziczenia dla `ltabn< 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 `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 `~ltabn` ()  
*Destruktor wirtualny interfejsu.*
- virtual void `bubblesort` ()=0  
*Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.*
- virtual bool `search` (T)=0  
*znajduje element w tablicy*

### Przyjaciele

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

### 5.17.1 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.

### 5.17.2 Dokumentacja konstruktora i destruktora

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

Destruktor wirtualny interfejsu.

Definicja w linii 83 pliku tabl.hh.

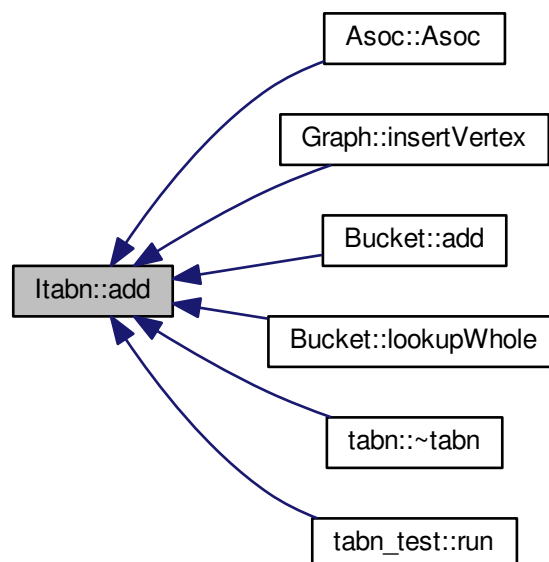
### 5.17.3 Dokumentacja funkcji składowych

5.17.3.1 `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:



5.17.3.2 `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.

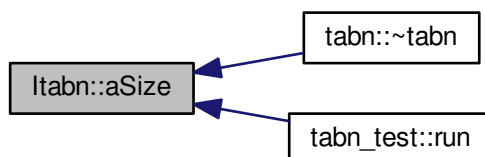
Implementowany w `tabn< T >`.

5.17.3.3 `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:

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

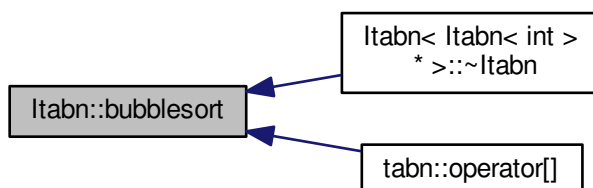
Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.

## Ostrzeżenie

Wymaga typu danych ze zdefiniowanym operatorem porównania "większe od"

Implementowany w `tabn< T >`.

Oto graf wywoływań tej funkcji:



5.17.3.5 `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:

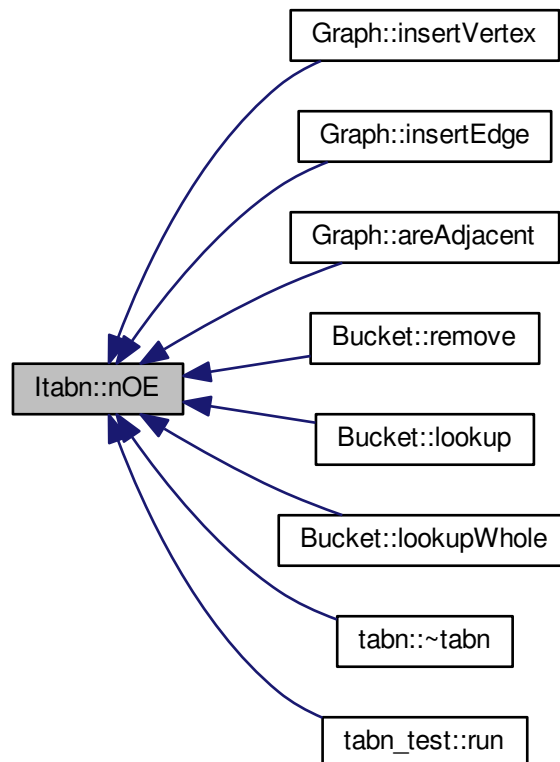


5.17.3.6 `template<class T> virtual int Itabn< T >::nOE ( void ) [pure virtual]`

Zwraca liczbę elementów w tablicy.

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

Oto graf wywołań tej funkcji:



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

Pozwala na dostęp do dowolnego elementu.

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

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

Pozwala na dostęp do dowolnego elementu.

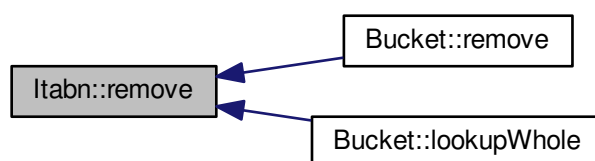
Implementowany w [tabn< T >](#).

5.17.3.9 `template<class T> virtual T Itabn< T >::remove ( ) [pure virtual]`

Usuwa i zwraca element z końca tablicy.

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

Oto graf wywoływań tej funkcji:



5.17.3.10 `template<class T> virtual T Itabn< 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 >](#).

5.17.3.11 `template<class T> virtual bool Itabn< T >::search ( T ) [pure virtual]`

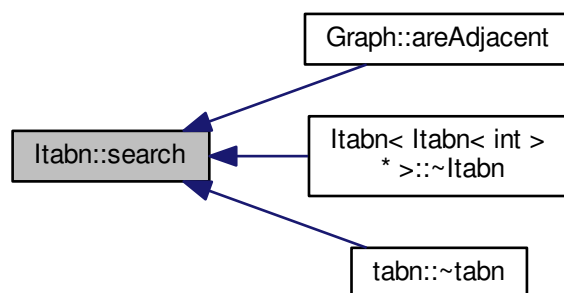
znajduje element w tablicy

## Zwracane wartości

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

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

Oto graf wywoływań tej funkcji:

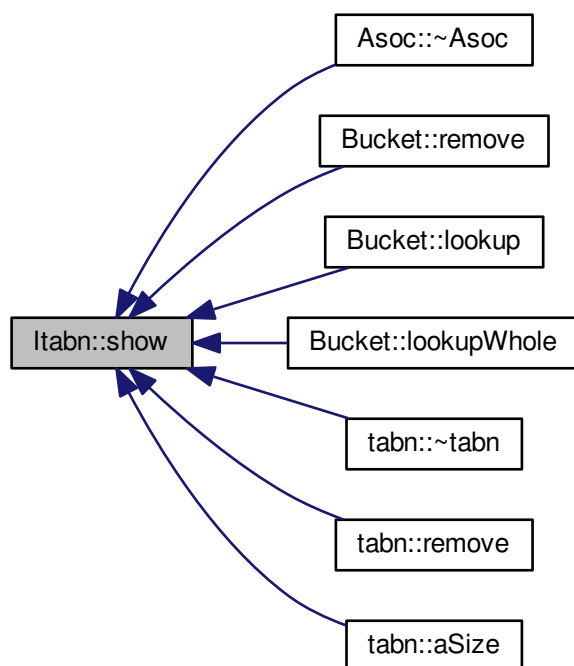


**5.17.3.12** `template<class T> virtual T Itabn< T >::show ( int ) const` `[pure virtual]`

Zwraca żądany element, o ile istnieje.

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

Oto graf wywołań tej funkcji:

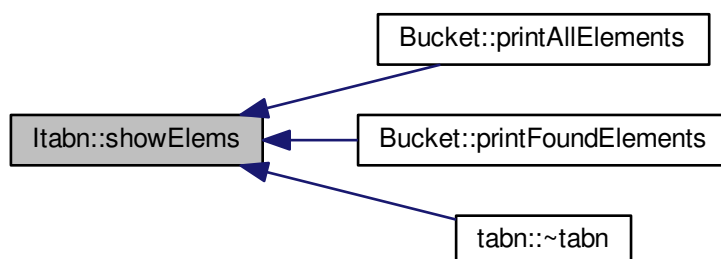


**5.17.3.13** `template<class T> virtual void Itabn< T >::showElems ( void ) [pure virtual]`

Wyświetla elementy tablicy.

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

Oto graf wywołań tej funkcji:



## 5.17.4 Dokumentacja przyjaciół i funkcji związanych

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

Definicja w linii 99 pliku `tabl.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

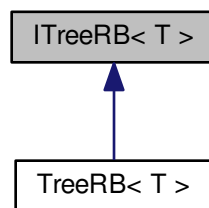
- [tabl.hh](#)

## 5.18 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` ()

### Przyjaciele

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

### 5.18.1 Opis szczegółowy

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

Interfejs klasy drzewa czerwono-czarnego.

Definicja w linii 170 pliku `tree.hh`.



## 5.18.2 Dokumentacja konstruktora i destruktor

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

Definicja w linii 177 pliku tree.hh.

## 5.18.3 Dokumentacja funkcji składowych

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

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

Oto graf wywołań tej funkcji:



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

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

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

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

Oto graf wywołań tej funkcji:



### 5.18.4 Dokumentacja przyjaciół i funkcji związanych

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

Definicja w linii 179 pliku `tree.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tree.hh](#)

## 5.19 Dokumentacja szablonu klasy Kolejka< T >

Klasa modeluje kolejkę

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

Diagram dziedziczenia dla Kolejka< T >

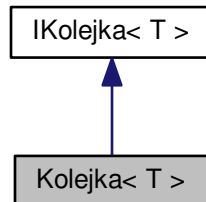
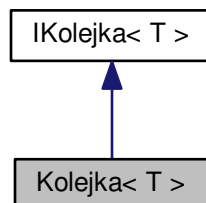


Diagram współpracy dla Kolejka< T >:



## 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.*

### 5.19.1 Opis szczegółowy

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

Klasa modeluje kolejkę

Definicja w linii 54 pliku kolejka.hh.

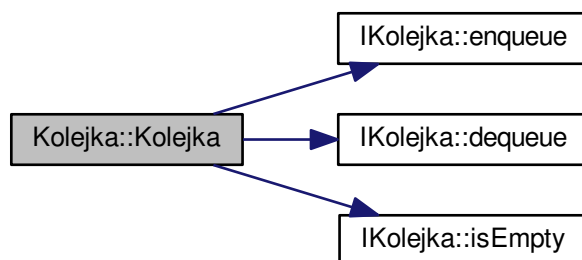
### 5.19.2 Dokumentacja konstruktora i destruktora

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

Konstruktor tablicy obsługującej kolejkę

Definicja w linii 61 pliku kolejka.hh.

Oto graf wywołań dla tej funkcji:



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

Destruktor klasy [Kolejka](#).

Definicja w linii 107 pliku kolejka.hh.

### 5.19.3 Dokumentacja funkcji składowych

5.19.3.1 `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

<a href="#">CriticalException</a>	re-throw z <a href="#">tabn&lt;T&gt;::remove()</a>
<a href="#">ContinueException</a>	re-throw z <a href="#">tabn&lt;T&gt;::remove()</a>

Implementuje [IKolejka<T>](#).

Definicja w linii 119 pliku kolejka.hh.

5.19.3.2 `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 113 pliku kolejka.hh.

5.19.3.3 `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

<a href="#">CriticalException</a>	re-throw z tab::show(int)
-----------------------------------	---------------------------

## 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 136 pliku kolejka.hh.

5.19.3.4 `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 131 pliku kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [kolejka.hh](#)

## 5.20 Dokumentacja szablonu klasy Lista&lt; T &gt;

Klasa lista.

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

Diagram dziedziczenia dla Lista< T >

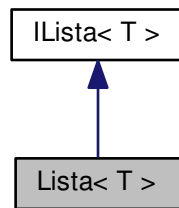
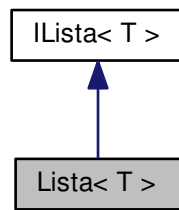


Diagram współpracy dla Lista< T >:



## 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.*

### 5.20.1 Opis szczegółowy

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

Klasa lista.

Modeluje pojęcie listy

Definicja w linii 84 pliku lista.hh.

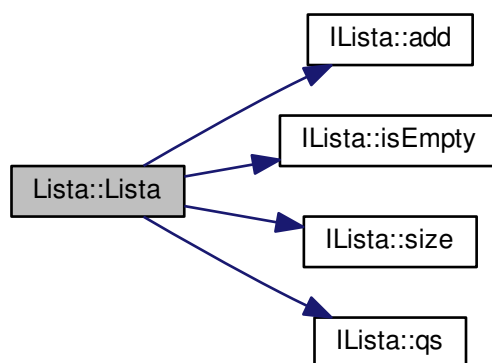
### 5.20.2 Dokumentacja konstruktora i destruktor

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

Konstruktor tablicy obsługującej listę

Definicja w linii 91 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



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

Destruktor Listy.

Definicja w linii 183 pliku lista.hh.

### 5.20.3 Dokumentacja funkcji składowych

5.20.3.1 `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

<a href="#"><i>ContinueException</i></a>	re-throw z <a href="#">tabn&lt;T&gt;::add(T,int)</a>
--	--

## 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.

5.20.3.2 `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.

5.20.3.3 `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

<a href="#"><i>CriticalException</i></a>	re-throw z <a href="#">tab&lt;T&gt;::show(int)</a>
--	--

## 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.



5.20.3.4 `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.

5.20.3.5 `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:



5.20.3.6 `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

<a href="#">CriticalException</a>	re-throw z <a href="#">tabn&lt;T&gt;::remove()</a>
<a href="#">ContinueException</a>	re-throw z <a href="#">tabn&lt;T&gt;::remove()</a>

**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 linie 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.

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

Usuwa element z końca listy.

Implementuje [ILista< T >](#).

Definicja w linii 216 pliku lista.hh.

**5.20.3.8** `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](#)

## 5.21 Dokumentacja klasy lista\_test

Definiuje sposób testowania wypełniania listy.

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

Diagram dziedziczenia dla lista\_test

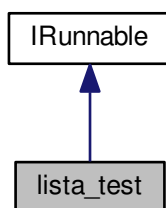
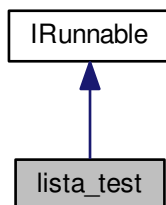


Diagram współpracy dla lista\_test:



### 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.*

### 5.21.1 Opis szczegółowy

Definiuje sposób testowania wypełniania listy.

Definicja w linii 303 pliku lista.hh.

### 5.21.2 Dokumentacja konstruktora i destruktora

#### 5.21.2.1 lista\_test::lista\_test ( ) [inline]

Konstruktor klasy testującej.

Definicja w linii 315 pliku lista.hh.

#### 5.21.2.2 lista\_test::~~lista\_test ( ) [inline]

Destruktor klasy testującej.

Definicja w linii 321 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



### 5.21.3 Dokumentacja funkcji składowych

#### 5.21.3.1 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

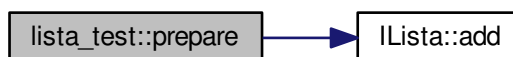
## Wyjątki

<a href="#"><i>CriticalException</i></a>	gdy wystąpił błąd przy otwarciu pliku
--	---------------------------------------

Implementuje [IRunnable](#).

Definicja w linii 359 pliku lista.hh.

Oto graf wywołań dla tej funkcji:



## 5.21.3.2 virtual bool lista\_test::run ( ) [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

<a href="#"><i>CriticalException</i></a>	re-throw z lista_test::wordSearch(std::string)
--	--

Implementuje [IRunnable](#).

Definicja w linii 390 pliku lista.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [lista.hh](#)

## 5.22 Dokumentacja szablonu klasy nodeRB&lt; T &gt;

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

## Metody publiczne

- `nodeRB` (T addKey, Colour col=red, `nodeRB`< T > \*addUp=NULL, `nodeRB`< T > \*addLeft=NULL, `nodeRB`< T > \*addRight=NULL)
- *brief\_desc*
- T `getKey` (void)
- Colour `getColour` (void)
- `nodeRB`< T > \* `getLeft` (void)
- `nodeRB`< T > \* `getRight` (void)
- `nodeRB`< T > \* `getParent` (void)
- T `getLeftKey` (void)
- T `getRightKey` (void)
- T `getParentKey` (void)
- void `setKey` (T keyToSet)
- void `setColour` (Colour colourToSet)
- void `setLeft` (`nodeRB`< T > \*leftDescendant)
- void `setRight` (`nodeRB`< T > \*rightDescendant)
- void `setParent` (`nodeRB`< T > \*parent)
- `nodeRB`< T > & `operator=` (const `nodeRB`< T > &read)

## 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)
- bool `operator>` (`nodeRB`< T > one, `nodeRB`< T > two)
- bool `operator<=` (`nodeRB`< T > one, `nodeRB`< T > two)
- bool `operator>=` (`nodeRB`< T > one, `nodeRB`< T > two)
- bool `operator==` (`nodeRB`< T > one, `nodeRB`< T > two)
- std::ostream & `operator<<` (std::ostream &output, const `nodeRB`< T > \*to)
- std::istream & `operator>>` (std::istream &input, const `nodeRB`< T > \*to)

### 5.22.1 Opis szczegółowy

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

Definicja w linii 23 pliku tree.hh.

### 5.22.2 Dokumentacja konstruktora i destruktora

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

*brief\_desc*

Definicja w linii 36 pliku tree.hh.

### 5.22.3 Dokumentacja funkcji składowych

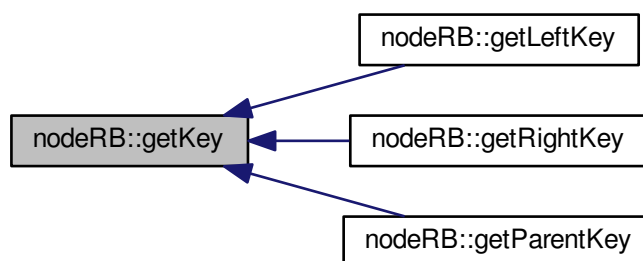
5.22.3.1 `template<class T> Colour nodeRB< T >::getColour ( void ) [inline]`

Definicja w linii 48 pliku `tree.hh`.

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

Definicja w linii 44 pliku `tree.hh`.

Oto graf wywołań tej funkcji:



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

Definicja w linii 53 pliku `tree.hh`.

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

Definicja w linii 65 pliku `tree.hh`.

Oto graf wywołań dla tej funkcji:



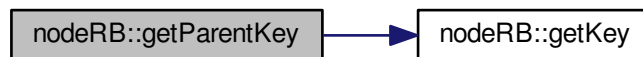
5.22.3.5 `template<class T> nodeRB<T>* nodeRB< T >::getParent ( void ) [inline]`

Definicja w linii 61 pliku tree.hh.

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

Definicja w linii 79 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



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

Definicja w linii 57 pliku tree.hh.

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

Definicja w linii 72 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



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

Definicja w linii 106 pliku tree.hh.

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

Definicja w linii 90 pliku tree.hh.



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

Definicja w linii 86 pliku `tree.hh`.

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

Definicja w linii 94 pliku `tree.hh`.

Oto graf wywołań tej funkcji:



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

Definicja w linii 102 pliku `tree.hh`.

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

Definicja w linii 98 pliku `tree.hh`.

Oto graf wywołań tej funkcji:



## 5.22.4 Dokumentacja przyjaciół i funkcji związanych

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

Definicja w linii 115 pliku `tree.hh`.

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

Definicja w linii 141 pliku tree.hh.

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

Definicja w linii 125 pliku tree.hh.

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

Definicja w linii 136 pliku tree.hh.

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

Definicja w linii 120 pliku tree.hh.

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

Definicja w linii 131 pliku tree.hh.

5.22.4.7 `template<class T> std::istream& operator>> ( std::istream & input, const nodeRB< T > * to ) [friend]`

Definicja w linii 152 pliku tree.hh.

## 5.22.5 Dokumentacja atrybutów składowych

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

Definicja w linii 26 pliku tree.hh.

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

Definicja w linii 25 pliku tree.hh.

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

Definicja w linii 27 pliku tree.hh.

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

Definicja w linii 28 pliku tree.hh.

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

Definicja w linii 29 pliku tree.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tree.hh](#)

## 5.23 Dokumentacja klasy Stoper

Klasa stoper implementująca interfejs [IStoper](#).

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

Diagram dziedziczenia dla Stoper

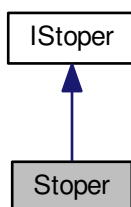
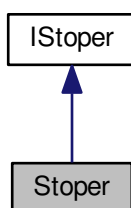


Diagram współpracy dla Stoper:



### 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.*

### 5.23.1 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.

### 5.23.2 Dokumentacja funkcji składowych

#### 5.23.2.1 `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.

#### 5.23.2.2 `void Stoper::start ( void ) [virtual]`

Uruchamia zegar.

Implementuje [IStoper](#).

Definicja w linii 4 pliku stoper.cpp.

#### 5.23.2.3 `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](#)

## 5.24 Dokumentacja szablonu klasy Stos< T >

Klasa [Stos](#).

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

Diagram dziedziczenia dla Stos< T >

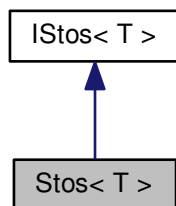
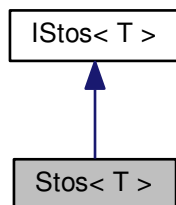


Diagram współpracy dla Stos< T >:



### 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.*

### 5.24.1 Opis szczegółowy

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

Klasa [Stos](#).

Modeluje pojęcie stosu

Definicja w linii 63 pliku stos.hh.

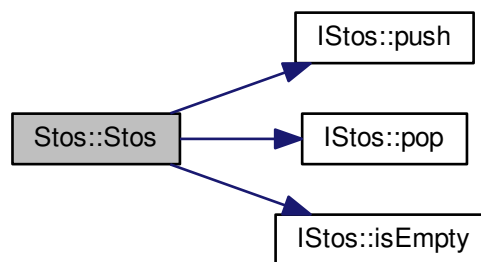
### 5.24.2 Dokumentacja konstruktora i destruktor

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

Konstruktor tablicy obsługującej stos.

Definicja w linii 70 pliku stos.hh.

Oto graf wywołań dla tej funkcji:



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

Destruktor stosu.

Definicja w linii 129 pliku stos.hh.

### 5.24.3 Dokumentacja funkcji składowych

5.24.3.1 `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&lt;T&gt;::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 157 pliku `stos.hh`.

**5.24.3.2** `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 152 pliku `stos.hh`.

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

Zdejmuje element ze szczytu stosu.

## Zwracane wartości

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

## Wyjątki

<i>CriticalException</i>	re-throw z <code>tabn&lt;T&gt;::remove()</code>
<i>ContinueException</i>	re-throw z <code>tabn&lt;T&gt;::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 140 pliku `stos.hh`.

#### 5.24.3.4 `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 135 pliku `stos.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [stos.hh](#)

## 5.25 Dokumentacja szablonu klasy `tabn< T >`

Modeluje tablicę dynamicznie rozszerzalną

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

Diagram dziedziczenia dla `tabn< T >`

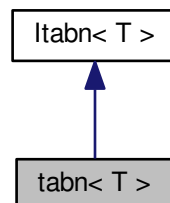
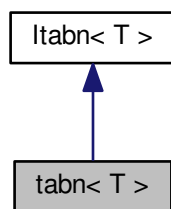




Diagram współpracy dla `tabn< T >`:



## Metody publiczne

- `tabn ()`  
*Konstruktor klasy `tabn`.*
- `virtual ~tabn ()`  
*Destruktor klasy `tabn`.*
- `virtual bool isEmpty (void)`  
*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 (void)`  
*Wyświetla listę elementów.*
- `virtual int nOE (void)`  
*zwraca liczbę elementów w tablicy*
- `virtual int aSize (void)`  
*zwraca wielkość zaalokowanej przestrzeni dla tablicy*
- `virtual bool search (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 (void)`  
*Sortowanie elementów tablicy algorytmem sortowania bąbelkowego.*

### 5.25.1 Opis szczegółowy

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

Modeluje tablicę dynamicznie rozszerzalną

Przechowuje elementy w rozszerzalnej tablicy o rozmiarze początkowym SIZE

Definicja w linii 116 pliku tabl.hh.

### 5.25.2 Dokumentacja konstruktora i destruktor

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

Konstruktor klasy tabn.

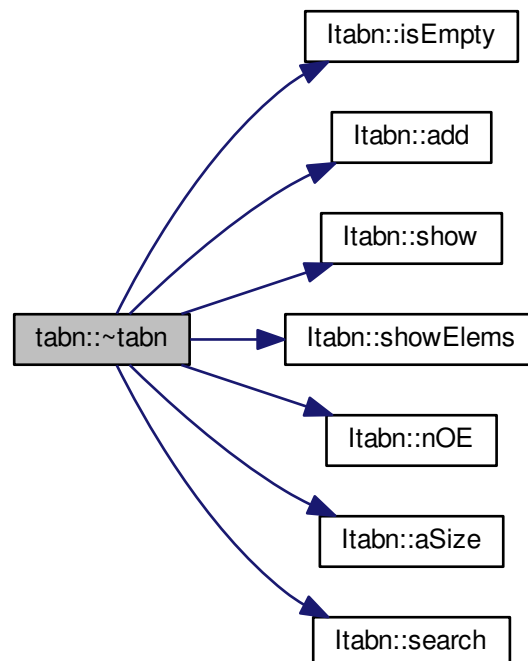
Definicja w linii 127 pliku tabl.hh.

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

Destruktor klasy tabn.

Definicja w linii 136 pliku tabl.hh.

Oto graf wywołań dla tej funkcji:



### 5.25.3 Dokumentacja funkcji składowych

#### 5.25.3.1 `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 327 pliku `tabl.hh`.

#### 5.25.3.2 `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

<a href="#"><i>ContinueException</i></a>	<code>WrongPositionToShiftFromRightException</code> przy próbie dodania elementu do niewłaściwego miejsca (re-throw z <code>tabn&lt;T&gt;::shiftRight(T,int)</code> ).
--	--

Implementuje `Itabn< T >`.

Definicja w linii 335 pliku `tabl.hh`.

#### 5.25.3.3 `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 502 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



**5.25.3.4** `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

<i>CriticalException</i>	re-throw z <code>tabn&lt;T&gt;::swap(int,int)</code>
--------------------------	--

Implementuje `ltabn<T>`.

Definicja w linii 523 pliku `tabl.hh`.

**5.25.3.5** `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 461 pliku `tabl.hh`.

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

zwraca liczbę elementów w tablicy

## Zwracane wartości

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

Implementuje `ltabn< T >`.

Definicja w linii 497 pliku `tabl.hh`.

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

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

## Parametry

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

## Zwracane wartości

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

Implementuje `ltabn< T >`.

Definicja w linii 273 pliku `tabl.hh`.

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

Umożliwia odczyt dowolnego elementu tablicy bez sprawdzania zakresu (debug)

## Parametry

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

## Zwracane wartości

<code>T</code>	Element tablicy
----------------	-----------------

Implementuje `ltabn< T >`.

Definicja w linii 285 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



#### 5.25.3.9 `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&lt;T&gt;::isEmptyException()</code> ).
<i>CriticalException</i>	WrongIndexException przy próbie usunięcia z pustej tablicy (re-throw z <code>tabn&lt;T&gt;::show(int)</code> ).
<i>ContinueException</i>	re-throw z <code>tabn&lt;T&gt;::reduce2()</code> .

Implementuje `ltabn<T>`.

Definicja w linii 366 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



#### 5.25.3.10 `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&lt;T&gt;::isEmptyException()</code> ).
<i>CriticalException</i>	WrongIndexException przy próbie usunięcia z pustej tablicy lub nieistniejącego elementu (re-throw z <code>tabn&lt;T&gt;::show(int)</code> ).
<i>ContinueException</i>	re-throw z <code>tabn&lt;T&gt;::reduce2()</code> .

Implementuje `ltabn< T >`.

Definicja w linii 389 pliku `tabl.hh`.

Oto graf wywołań dla tej funkcji:



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

znajduje element w tablicy

Zwracane wartości

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

Implementuje `ltabn< T >`.

Definicja w linii 319 pliku `tabl.hh`.

5.25.3.12 `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 477 pliku `tabl.hh`.

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

Wyświetla listę elementów.

Implementuje `Itabn<T>`.

Definicja w linii 487 pliku `tabl.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)

## 5.26 Dokumentacja klasy `tabn_test`

Definiuje sposób testowania wypełniania tablicy `tabn`.

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

Diagram dziedziczenia dla `tabn_test`

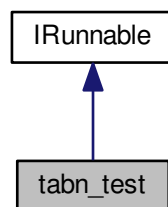
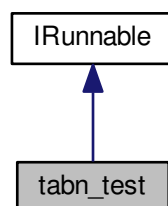


Diagram współpracy dla `tabn_test`:





## 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.*

### 5.26.1 Opis szczegółowy

Definiuje sposób testowania wypełniania tablicy `tabn`.

Definicja w linii 557 pliku `tabl.hh`.

### 5.26.2 Dokumentacja konstruktora i destruktora

#### 5.26.2.1 `tabn_test::tabn_test ( )` `[inline]`

Konstruktor klasy `tabn_test`.

Definicja w linii 565 pliku `tabl.hh`.

#### 5.26.2.2 `virtual tabn_test::~~tabn_test ( )` `[inline]`, `[virtual]`

Destruktor klasy `tabn_test`.

Definicja w linii 571 pliku `tabl.hh`.

### 5.26.3 Dokumentacja funkcji składowych

#### 5.26.3.1 `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 602 pliku tabl.hh.

**5.26.3.2** `virtual bool tabn_test::run ( ) [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
-------------------	-------------

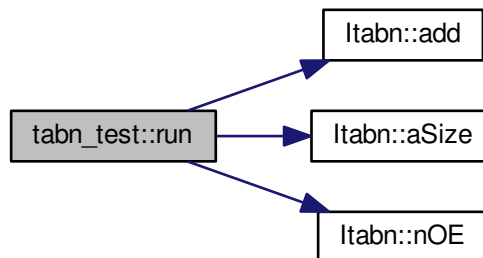
Wyjątki

<a href="#">ContinueException</a>	re-throw <code>tabn&lt;T&gt;::add(int)</code>
-----------------------------------	---

Implementuje [IRunnable](#).

Definicja w linii 619 pliku tabl.hh.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tabl.hh](#)

## 5.27 Dokumentacja klasy `tree_test`

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

Diagram dziedziczenia dla `tree_test`

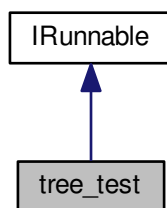
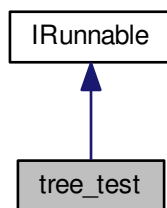


Diagram współpracy dla `tree_test`:



### Metody publiczne

- `tree_test()`
- `virtual ~tree_test()`
- `virtual bool prepare(int sizeofTest)`  
*Przygotowanie badań*
- `virtual bool run()`  
*Przeprowadzanie badań*

#### 5.27.1 Opis szczegółowy

Definicja w linii 364 pliku `tree.hh`.

#### 5.27.2 Dokumentacja konstruktora i destruktora

##### 5.27.2.1 `tree_test::tree_test()` [inline]

Definicja w linii 387 pliku `tree.hh`.

#### 5.27.2.2 `virtual tree_test::~~tree_test ( ) [inline],[virtual]`

Definicja w linii 391 pliku tree.hh.

### 5.27.3 Dokumentacja funkcji składowych

#### 5.27.3.1 `virtual bool tree_test::prepare ( int ) [inline],[virtual]`

Przygotowanie badań

Implementuje [IRunnable](#).

Definicja w linii 395 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



#### 5.27.3.2 `virtual bool tree_test::run ( ) [inline],[virtual]`

Przeprowadzanie badań

Implementuje [IRunnable](#).

Definicja w linii 409 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tree.hh](#)

## 5.28 Dokumentacja szablonu klasy TreeRB< T >

Klasa implementująca interfejs drzewa czerwono-czarnego.

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

Diagram dziedziczenia dla TreeRB< T >

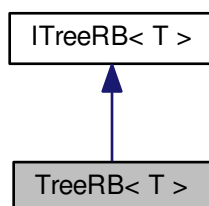
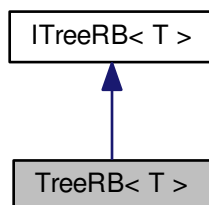


Diagram współpracy dla TreeRB< T >:



### Metody publiczne

- `TreeRB()`  
*Konstruktor.*
- `virtual ~TreeRB()`  
*Destruktor.*
- `virtual void insert(T element)`  
*Wstawia element do drzewa.*
- `virtual void insert(T element, nodeRB< T > *node)`
- `virtual bool search(T k)`

### 5.28.1 Opis szczegółowy

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

Klasa implementująca interfejs drzewa czerwono-czarnego.

Definicja w linii 190 pliku tree.hh.

### 5.28.2 Dokumentacja konstruktora i destruktora

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

Konstruktor.

Definicja w linii 298 pliku tree.hh.

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

Destruktor.

Definicja w linii 304 pliku tree.hh.

### 5.28.3 Dokumentacja funkcji składowych

5.28.3.1 `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 >](#).

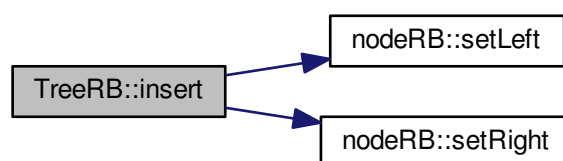
Definicja w linii 312 pliku tree.hh.

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

Implementuje [ITreeRB< T >](#).

Definicja w linii 317 pliku tree.hh.

Oto graf wywołań dla tej funkcji:



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

Implementuje `ITreeRB< T >`.

Definicja w linii 341 pliku `tree.hh`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [tree.hh](#)





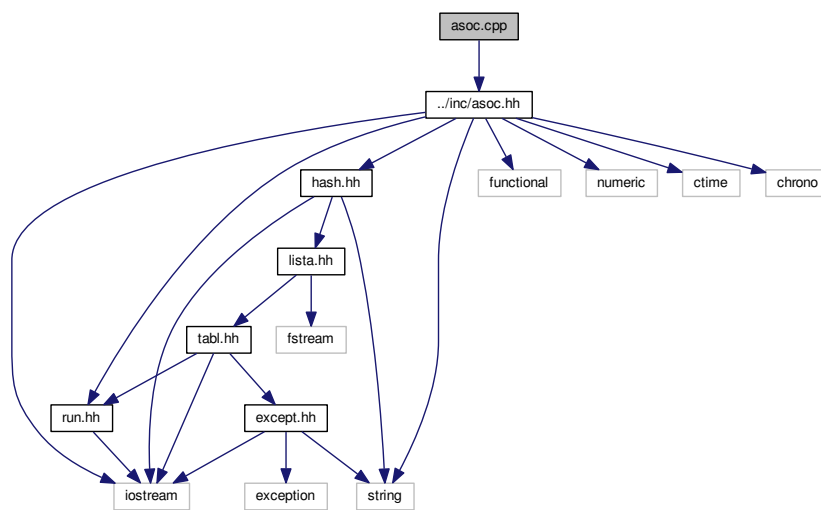
## Rozdział 6

# Dokumentacja plików

### 6.1 Dokumentacja pliku asoc.cpp

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

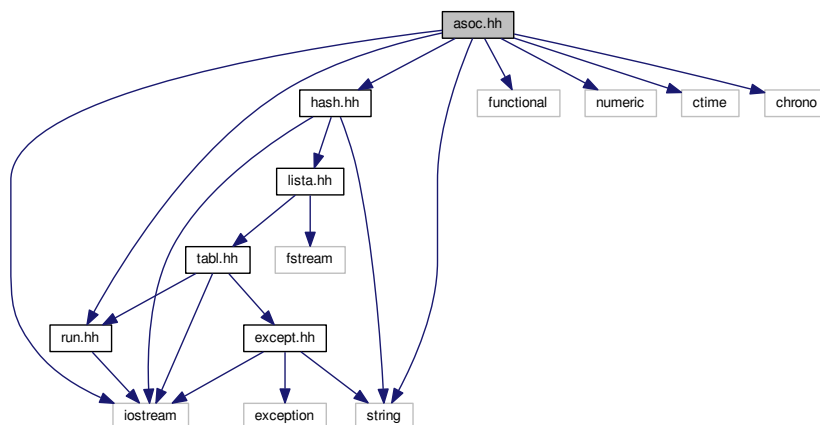
Wykres zależności załączania dla asoc.cpp:



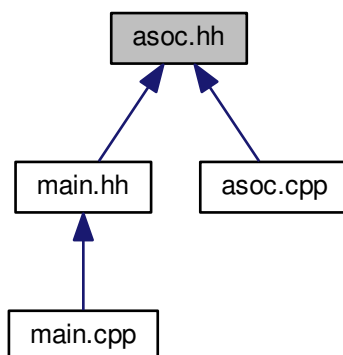
### 6.2 Dokumentacja pliku asoc.hh

```
#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:



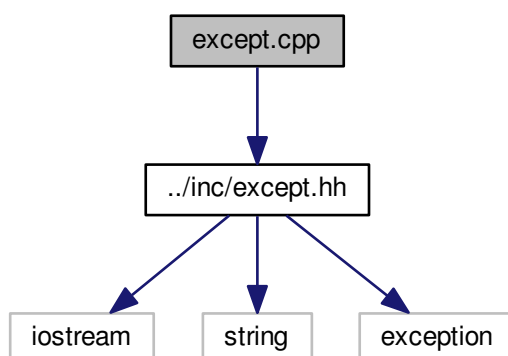
## Komponenty

- 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:

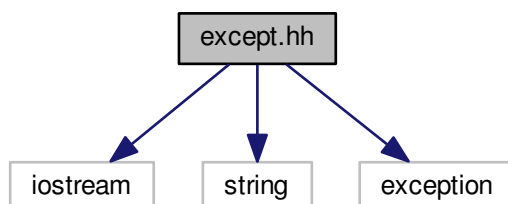


## 6.4 Dokumentacja pliku except.hh

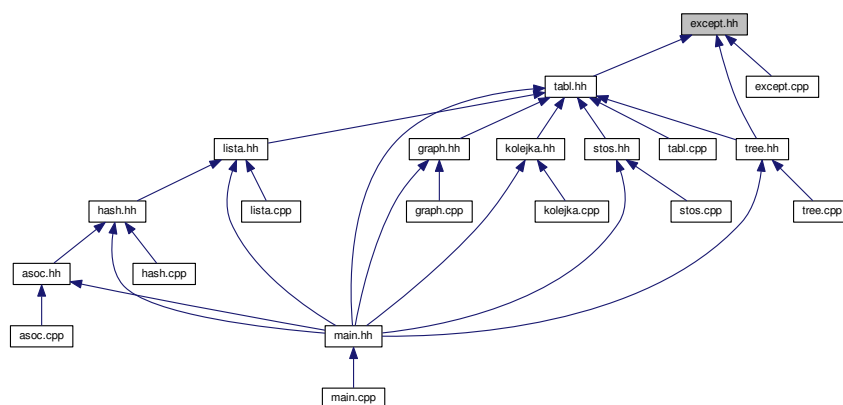
Plik zawiera definicje wyjątków.

```
#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

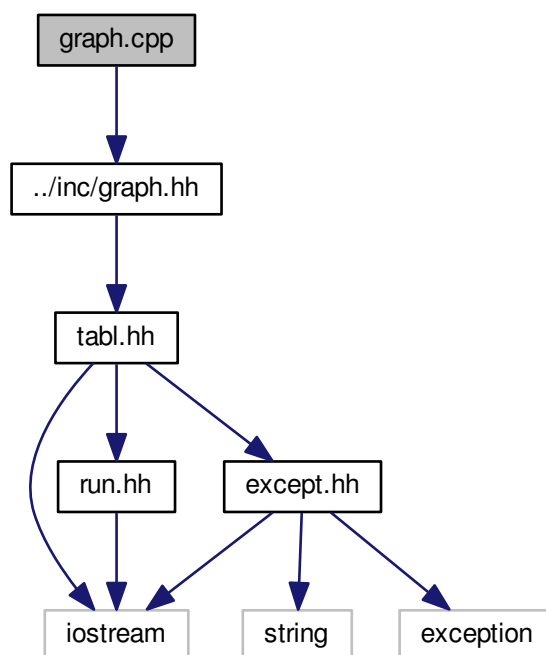
#### 6.4.2.1 `template<class ExceptT > void what ( ExceptT & except )`

Definicja w linii 72 pliku `except.hh`.

## 6.5 Dokumentacja pliku graph.cpp

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

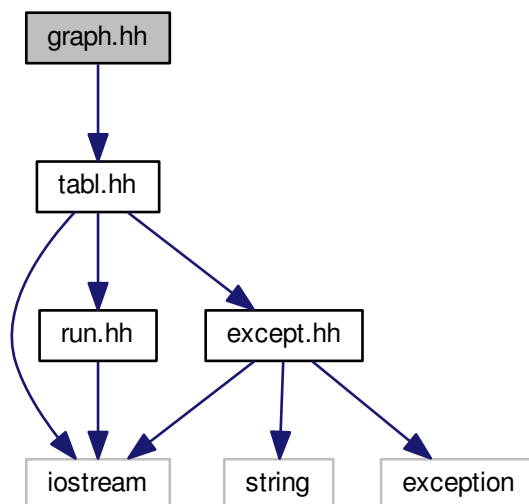
Wykres zależności załączania dla graph.cpp:



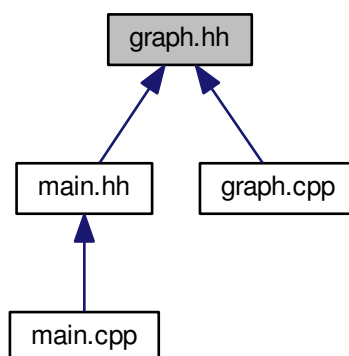
## 6.6 Dokumentacja pliku graph.hh

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

Wykres zależności załączania dla graph.hh:



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



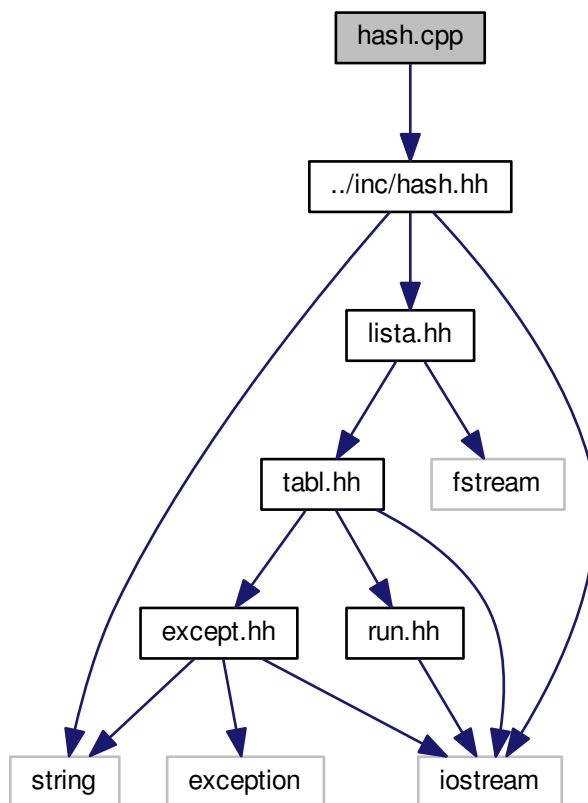
### Komponenty

- class [IGraph](#)  
*Interfejs grafu.*
- class [Graph](#)  
*Klasa implementująca interfejs grafu.*

## 6.7 Dokumentacja pliku hash.cpp

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

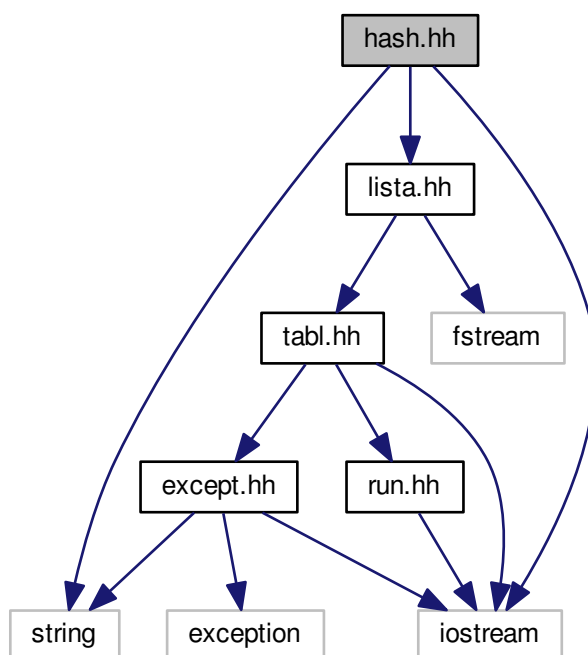
Wykres zależności załączania dla hash.cpp:



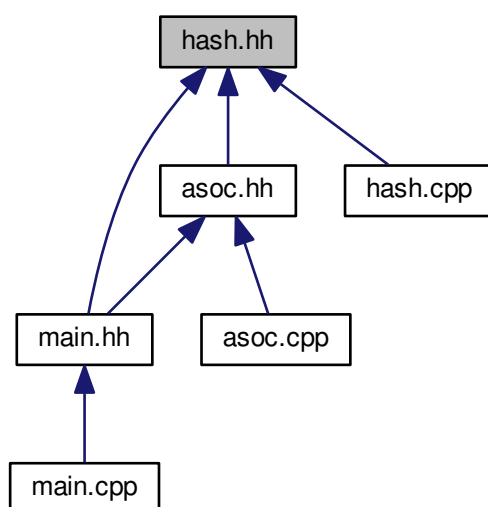
## 6.8 Dokumentacja pliku hash.hh

```
#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 >`

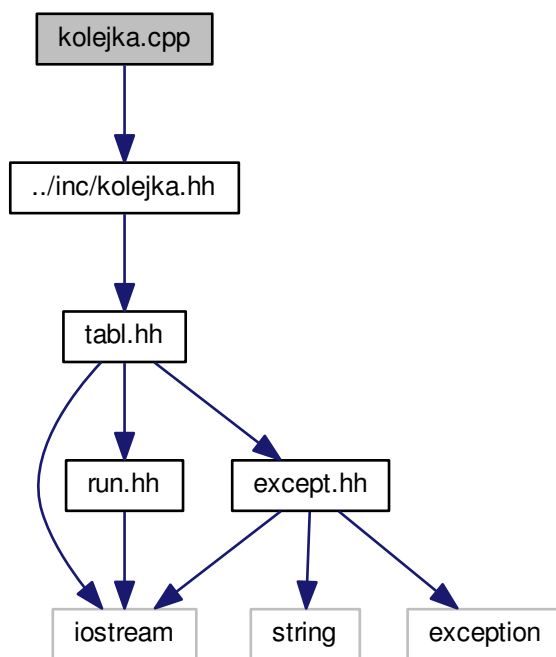
*Klasa definiująca obiekt typu wpis.*

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

## 6.9 Dokumentacja pliku kolejka.cpp

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

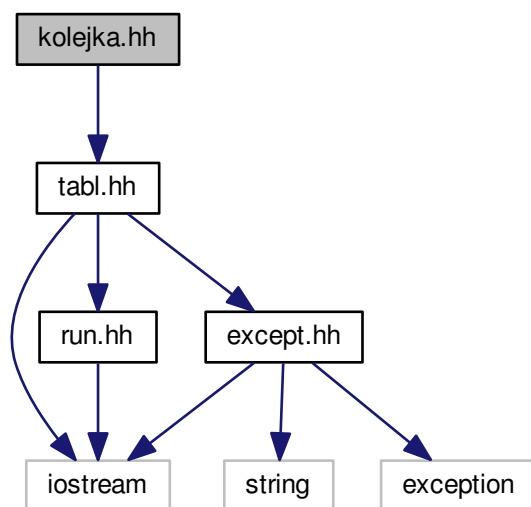
Wykres zależności załączania dla kolejka.cpp:



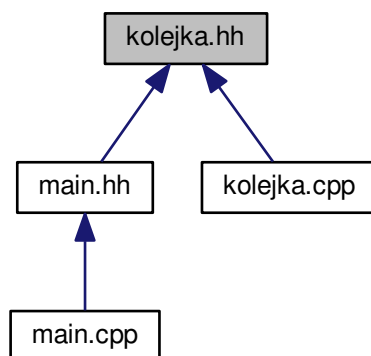
## 6.10 Dokumentacja pliku kolejka.hh

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

Wykres zależności załączania dla kolejka.hh:



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



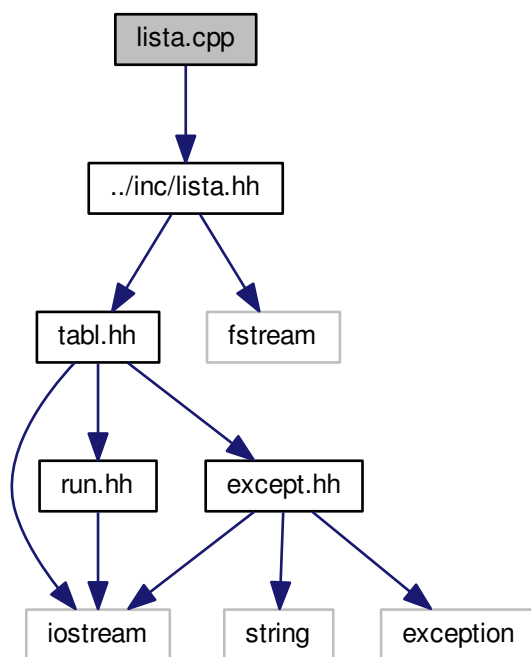
## Komponenty

- class `IKolejka< T >`  
Interfejs klasy `Kolejka` Definiuje operacje dostępne dla klasy `Kolejka`.
- class `Kolejka< T >`  
Klasa modeluje kolejkę

## 6.11 Dokumentacja pliku lista.cpp

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

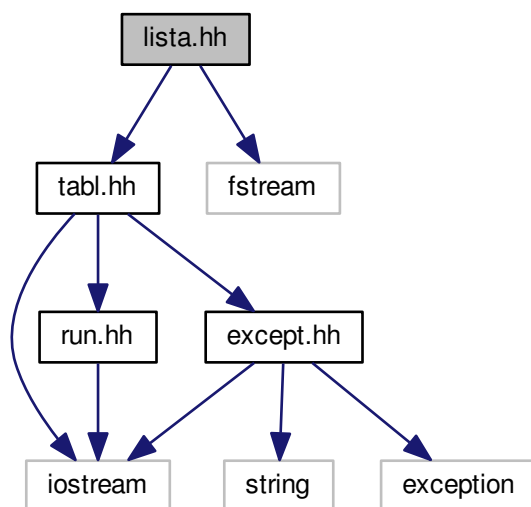
Wykres zależności załączania dla lista.cpp:



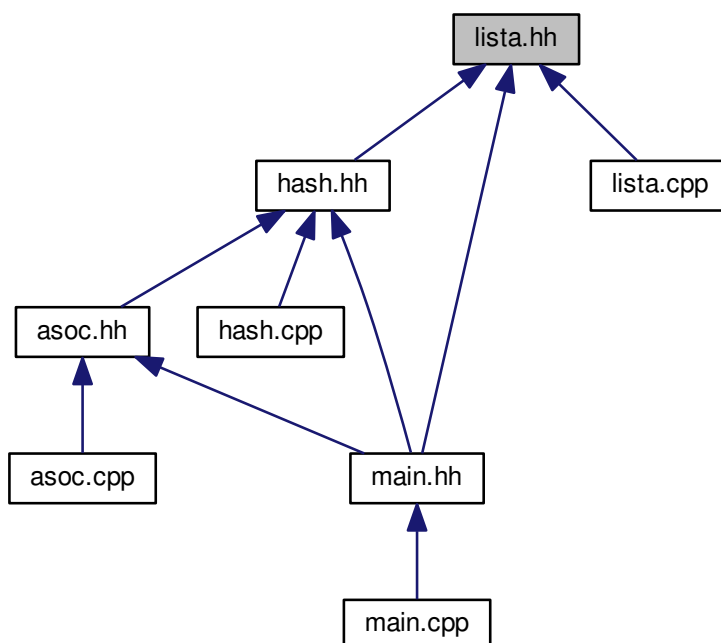
## 6.12 Dokumentacja pliku lista.hh

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

Wykres zależności załączania dla lista.hh:



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



## Komponenty

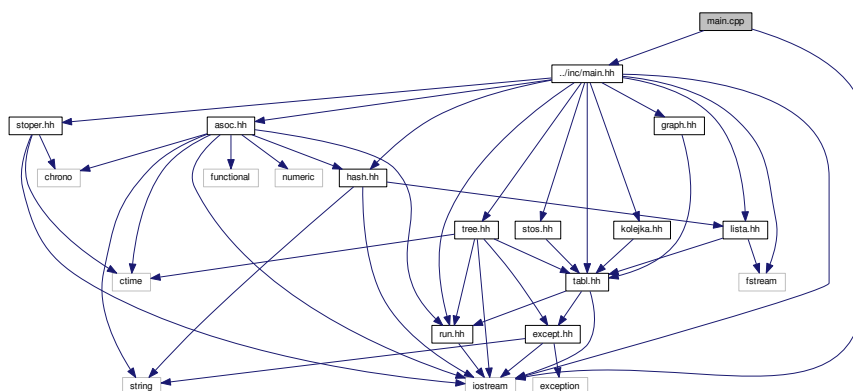
- class `ILista< T >`  
*Interfejs listy.*
- class `Lista< T >`  
*Klasa lista.*
- class `lista_test`  
*Definiuje sposób testowania wypełniania listy.*

### 6.13 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.

### 6.13.1 Opis szczegółowy

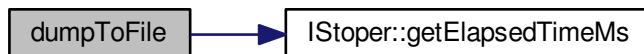
Główny plik programu.

### 6.13.2 Dokumentacja funkcji

#### 6.13.2.1 void dumpToFile ( string *nameOfFile*, unsigned int *testsize*, IStoper \* *stoper* )

Definicja w linii 371 pliku main.cpp.

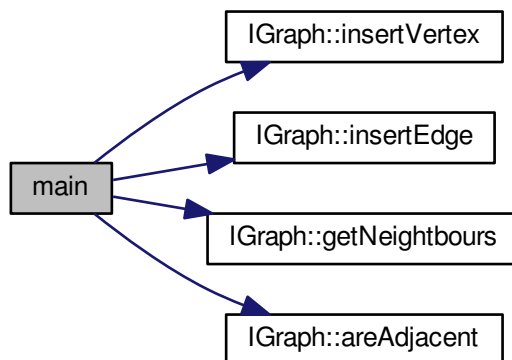
Oto graf wywołań dla tej funkcji:



#### 6.13.2.2 int main ( void )

Definicja w linii 12 pliku main.cpp.

Oto graf wywołań dla tej funkcji:



#### 6.13.2.3 void printOnscreen ( unsigned int, IStoper \* )

Wyświetla wynik na standardowym wyjściu.

Parametry

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

Definicja w linii 385 pliku main.cpp.

Oto graf wywołań dla tej funkcji:

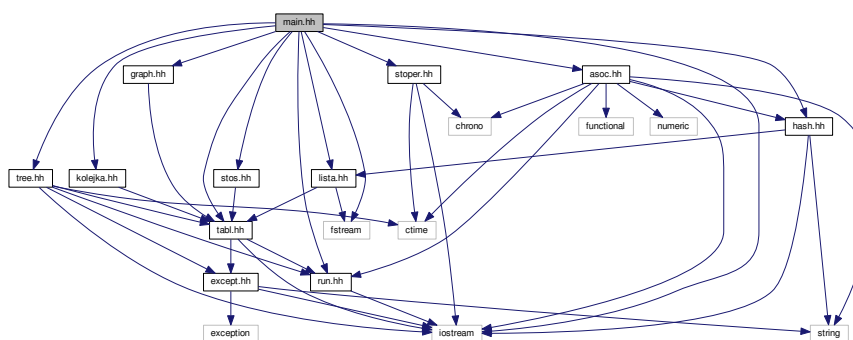


## 6.14 Dokumentacja pliku main.hh

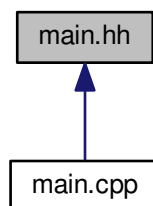
```

#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.*

### 6.14.1 Dokumentacja funkcji

#### 6.14.1.1 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

#### 6.14.1.2 void printOnscreen ( unsigned int, IStoper \* )

Wyświetla wynik na standardowym wyjściu.

##### Parametry

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

Definicja w linii 385 pliku main.cpp.



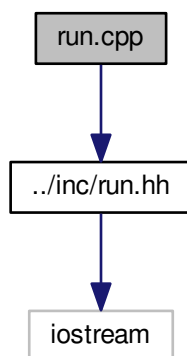
Oto graf wywołań dla tej funkcji:



## 6.15 Dokumentacja pliku run.cpp

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

Wykres zależności załączania dla run.cpp:

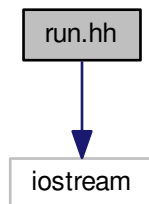


## 6.16 Dokumentacja pliku run.hh

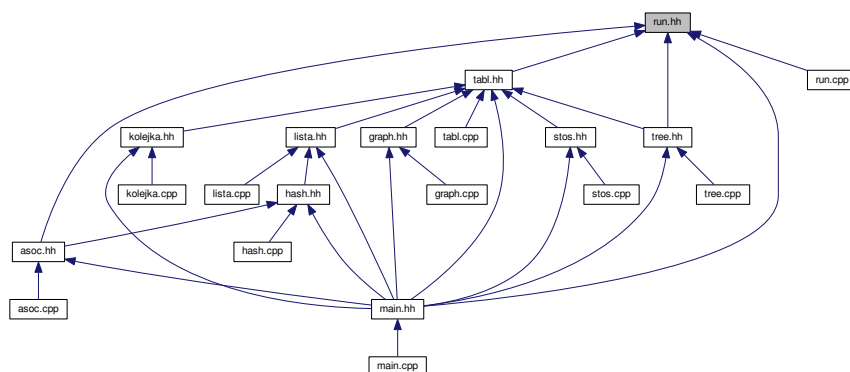
Plik definiuje interfejs [IRunnable](#), ujednolicający klasy umożliwiające badanie algorytmów.

```
#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.*

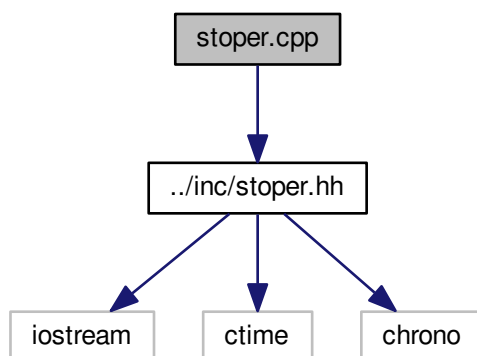
### 6.16.1 Opis szczegółowy

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

## 6.17 Dokumentacja pliku stoper.cpp

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

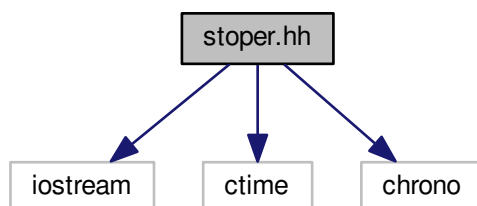
Wykres zależności załączania dla stoper.cpp:



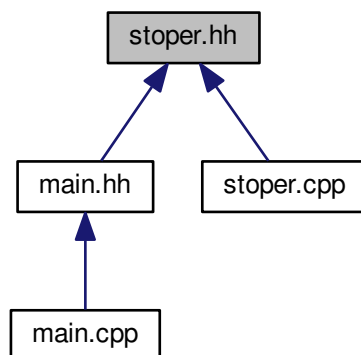
## 6.18 Dokumentacja pliku stoper.hh

```
#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.*

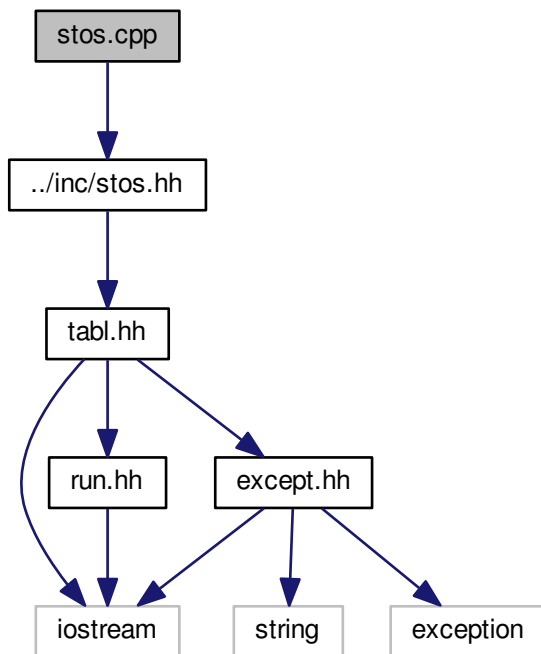
- class `Stoper`

*Klasa `stoper` implementująca interfejs `IStoper`.*

## 6.19 Dokumentacja pliku `stos.cpp`

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

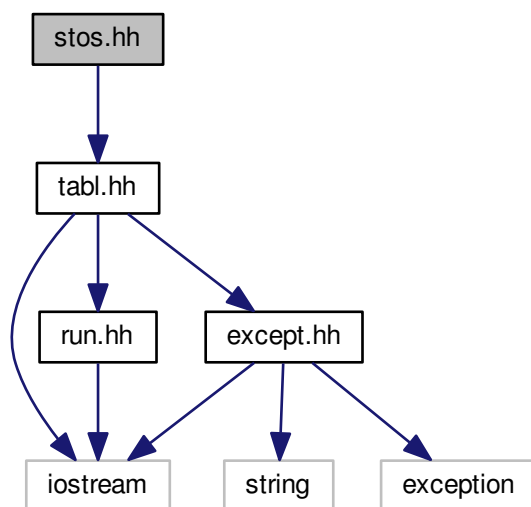
Wykres zależności załączania dla stos.cpp:



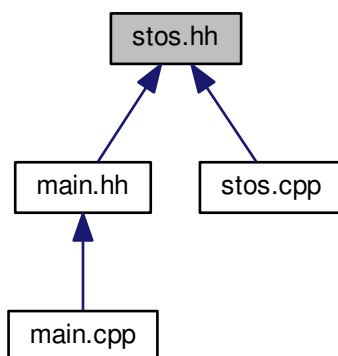
## 6.20 Dokumentacja pliku stos.hh

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

Wykres zależności załączania dla stos.hh:



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



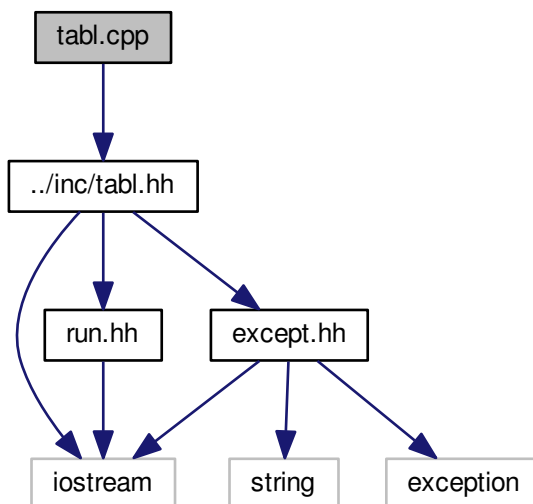
## Komponenty

- class `IStos< T >`  
*Interfejs stosu.*
- class `Stos< T >`  
*Klasa `Stos`.*

## 6.21 Dokumentacja pliku tabl.cpp

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

Wykres zależności załączania dla tabl.cpp:

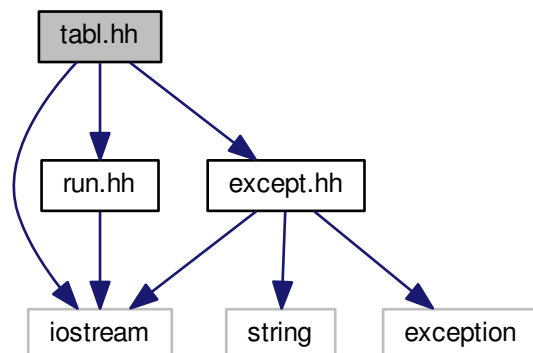


## 6.22 Dokumentacja pliku tabl.hh

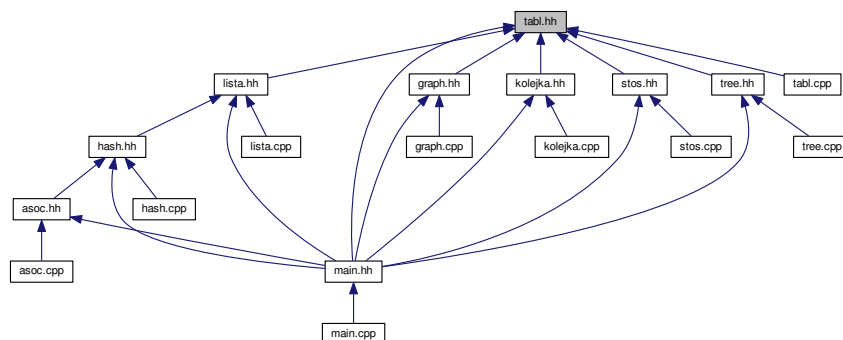
Definicja interfejsu `ltabn`, klasy `tabn` oraz klasy `tabn_test`.

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

## Definicje

- `#define SIZE 10`

### 6.22.1 Opis szczegółowy

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

### 6.22.2 Dokumentacja definicji

#### 6.22.2.1 `#define SIZE 10`

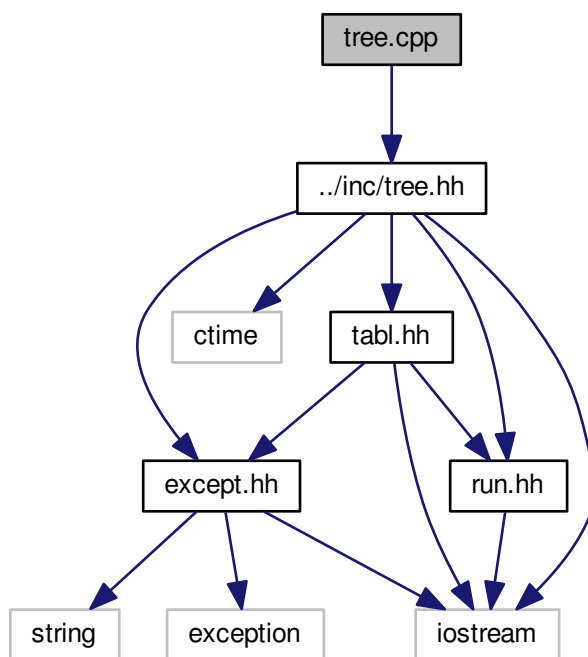
Definicja w linii 12 pliku `tabl.hh`.



## 6.23 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.*

### 6.23.1 Dokumentacja funkcji

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

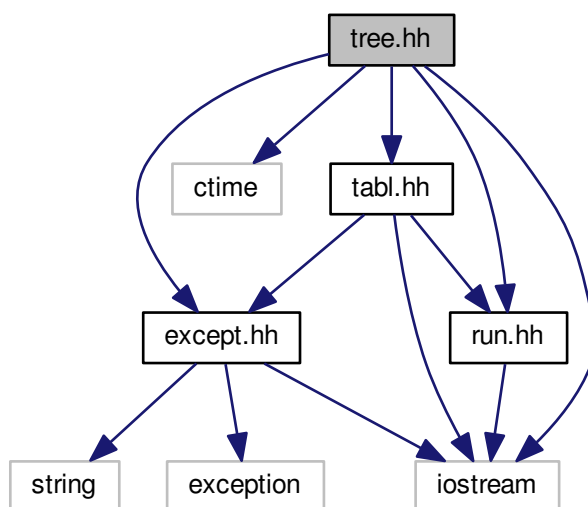
Wyświetlanie koloru node'a.

Definicja w linii 3 pliku tree.cpp.

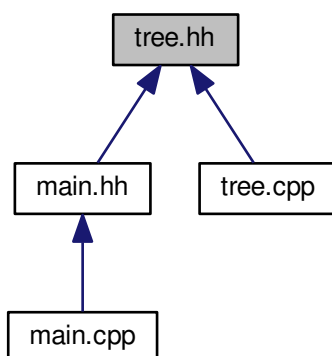
## 6.24 Dokumentacja pliku tree.hh

```
#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`

## Wyliczenia

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

## Funkcje

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

### 6.24.1 Dokumentacja typów wyliczanych

#### 6.24.1.1 enum Colour

Kolory node'a drzewa czerwono-czarnego.

Wartości wyliczeń

***red***

***black***

Definicja w linii 12 pliku tree.hh.

### 6.24.2 Dokumentacja funkcji

#### 6.24.2.1 `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, [29](#)
- ~IBucket
  - IBucket, [31](#)
- ~IGraph
  - IGraph, [33](#)
- ~IKolejka
  - IKolejka, [36](#)
- ~ILista
  - ILista, [39](#)
- ~IRunnable
  - IRunnable, [43](#)
- ~IStoper
  - IStoper, [45](#)
- ~IStos
  - IStos, [46](#)
- ~ITreeRB
  - ITreeRB, [57](#)
- ~Itabn
  - Itabn, [50](#)
- ~Kolejka
  - Kolejka, [59](#)
- ~Lista
  - Lista, [63](#)
- ~Stos
  - Stos, [78](#)
- ~TreeRB
  - TreeRB, [94](#)
- ~asoc\_test
  - asoc\_test, [12](#)
- ~lista\_test
  - lista\_test, [68](#)
- ~tabn
  - tabn, [82](#)
- ~tabn\_test
  - tabn\_test, [89](#)
- ~tree\_test
  - tree\_test, [91](#)
- aSize
  - Itabn, [51](#)
  - tabn, [83](#)
- add
  - Asoc, [11](#)

- Bucket, [15](#)
- IAsoc, [29](#)
- IBucket, [31](#)
- ILista, [39](#), [40](#)
- Itabn, [50](#)
- Lista, [63](#), [64](#)
- tabn, [83](#)
- areAdjacent
  - Graph, [27](#)
  - IGraph, [33](#)
- Asoc
  - ~Asoc, [10](#)
  - add, [11](#)
  - Asoc, [10](#)
  - find, [11](#)
  - findOne, [11](#)
- Asoc< T, T2 >, [9](#)
- asoc.cpp, [97](#)
- asoc.hh, [97](#)
- asoc\_test, [11](#)
  - ~asoc\_test, [12](#)
  - asoc\_test, [12](#)
  - prepare, [13](#)
  - run, [13](#)
- black
  - tree.hh, [123](#)
- bubblesort
  - Itabn, [51](#)
  - tabn, [84](#)
- 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, [123](#)
- colour
  - nodeRB, [74](#)
- ContinueException, [18](#)

- ContinueException, [19](#)
- Throw, [19](#)
- CriticalException, [20](#)
  - CriticalException, [21](#)
  - Throw, [21](#)
- dequeue
  - IKolejka, [36](#)
  - Kolejka, [60](#)
- dumpToFile
  - main.cpp, [110](#)
  - main.hh, [112](#)
- enqueue
  - IKolejka, [36](#)
  - Kolejka, [60](#)
- 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, [98](#)
- except.hh, [99](#)
  - what, [100](#)
- ExceptionBase, [24](#)
  - cause, [25](#)
  - ExceptionBase, [24](#)
  - operator<<, [25](#)
  - Throw, [25](#)
- find
  - Asoc, [11](#)
  - IAsoc, [29](#)
- findOne
  - Asoc, [11](#)
  - IAsoc, [29](#)
- get
  - IKolejka, [37](#)
  - ILista, [40](#)
  - IStos, [47](#)
  - Kolejka, [60](#)
  - Lista, [64](#)
  - Stos, [78](#)
- getColour
  - nodeRB, [71](#)
- getElapsedTimeMs
  - IStoper, [45](#)
  - Stoper, [76](#)
- getID
  - Bucket, [15](#)
  - IBucket, [31](#)
  - getKey
    - entry, [22](#)
    - nodeRB, [71](#)
  - getLeft
    - nodeRB, [71](#)
  - getLeftKey
    - nodeRB, [71](#)
  - getNeighbours
    - Graph, [27](#)
    - IGraph, [34](#)
  - getParent
    - nodeRB, [71](#)
  - getParentKey
    - nodeRB, [72](#)
  - getRight
    - nodeRB, [72](#)
  - getRightKey
    - nodeRB, [72](#)
  - getVal
    - entry, [22](#)
  - Graph, [25](#)
    - ~Graph, [26](#)
    - areAdjacent, [27](#)
    - getNeighbours, [27](#)
    - Graph, [26](#)
    - insertEdge, [27](#)
    - insertVertex, [27](#)
  - graph.cpp, [101](#)
  - graph.hh, [102](#)
  - hash.cpp, [103](#)
  - hash.hh, [103](#)
  - IAsoc
    - ~IAsoc, [29](#)
    - add, [29](#)
    - find, [29](#)
    - findOne, [29](#)
    - operator<<, [30](#)
  - IAsoc< T, T2 >, [28](#)
  - IBucket
    - ~IBucket, [31](#)
    - add, [31](#)
    - getID, [31](#)
    - lookup, [32](#)
    - lookupWhole, [32](#)
    - printAllElements, [32](#)
    - printFoundElements, [32](#)
    - remove, [32](#)
  - IBucket< T, T2 >, [30](#)
  - IGraph, [32](#)
    - ~IGraph, [33](#)
    - areAdjacent, [33](#)
    - getNeighbours, [34](#)
    - insertEdge, [34](#)
    - insertVertex, [34](#)
  - IKolejka
    - ~IKolejka, [36](#)

- dequeue, 36
- enqueue, 36
- get, 37
- isEmpty, 37
- IKolejka< T >, 35
- ILista
  - ~ILista, 39
  - add, 39, 40
  - get, 40
  - isEmpty, 41
  - qs, 41
  - remove, 41, 42
  - size, 42
- ILista< T >, 38
- IRunnable, 43
  - ~IRunnable, 43
  - prepare, 43
  - run, 43
- IStoper, 44
  - ~IStoper, 45
  - getElapsedTimeMs, 45
  - start, 45
  - stop, 45
- IStos
  - ~IStos, 46
  - get, 47
  - isEmpty, 47
  - pop, 47
  - push, 48
- IStos< T >, 46
- ITreeRB< T >, 56
- ITreeRB
  - ~ITreeRB, 57
  - insert, 57
  - operator<<, 58
  - search, 57
- insert
  - ITreeRB, 57
  - TreeRB, 94
- insertEdge
  - Graph, 27
  - IGraph, 34
- insertVertex
  - Graph, 27
  - IGraph, 34
- isEmpty
  - IKolejka, 37
  - ILista, 41
  - IStos, 47
  - Itabn, 51
  - Kolejka, 61
  - Lista, 64
  - Stos, 79
  - tabn, 84
- Itabn
  - ~Itabn, 50
  - aSize, 51
  - add, 50
  - bubblesort, 51
  - isEmpty, 51
  - nOE, 52
  - operator<<, 56
  - operator[], 52, 53
  - remove, 53
  - search, 53
  - show, 54
  - showElems, 55
- Itabn< T >, 48
- key
  - nodeRB, 74
- Kolejka
  - ~Kolejka, 59
  - dequeue, 60
  - enqueue, 60
  - get, 60
  - isEmpty, 61
  - Kolejka, 59
- Kolejka< T >, 58
- kolejka.cpp, 105
- kolejka.hh, 105
- left
  - nodeRB, 74
- Lista
  - ~Lista, 63
  - add, 63, 64
  - get, 64
  - isEmpty, 64
  - Lista, 63
  - qs, 65
  - remove, 65, 66
  - size, 66
- Lista< T >, 61
- lista.cpp, 107
- lista.hh, 107
- lista\_test, 67
  - ~lista\_test, 68
  - lista\_test, 68
  - prepare, 68
  - run, 69
- lookup
  - Bucket, 16
  - IBucket, 32
- lookupWhole
  - Bucket, 16
  - IBucket, 32
- main
  - main.cpp, 110
- main.cpp, 109
  - dumpToFile, 110
  - main, 110
  - printOnscreen, 110
- main.hh, 111
  - dumpToFile, 112
  - printOnscreen, 112

- nOE
  - ltabn, 52
  - tabn, 84
- nodeRB< T >, 69
- nodeRB
  - colour, 74
  - getColour, 71
  - getKey, 71
  - getLeft, 71
  - getLeftKey, 71
  - getParent, 71
  - getParentKey, 72
  - getRight, 72
  - getRightKey, 72
  - key, 74
  - left, 74
  - nodeRB, 70
  - operator<, 73
  - operator<<, 73
  - operator<=, 74
  - operator>, 74
  - operator>>, 74
  - operator>=, 74
  - operator=, 72
  - operator==, 74
  - right, 74
  - setColour, 72
  - setKey, 72
  - setLeft, 73
  - setParent, 73
  - setRight, 73
  - up, 74
- operator<
  - entry, 23
  - nodeRB, 73
- operator<<
  - entry, 23
  - ExceptionBase, 25
  - IAsoc, 30
  - ITreeRB, 58
  - ltabn, 56
  - nodeRB, 73
  - tree.cpp, 121
  - tree.hh, 123
- operator<=
  - entry, 23
  - nodeRB, 74
- operator>
  - entry, 23
  - nodeRB, 74
- operator>>
  - entry, 23
  - nodeRB, 74
- operator>=
  - entry, 23
  - nodeRB, 74
- operator=
  - entry, 22
- nodeRB, 72
- operator==
  - entry, 23
  - nodeRB, 74
- operator[]
  - ltabn, 52, 53
  - tabn, 85
- pop
  - IStos, 47
  - Stos, 79
- prepare
  - asoc\_test, 13
  - IRunnable, 43
  - lista\_test, 68
  - tabn\_test, 89
  - tree\_test, 92
- printAllElements
  - Bucket, 16
  - IBucket, 32
- printFoundElements
  - Bucket, 17
  - IBucket, 32
- printOnscreen
  - main.cpp, 110
  - main.hh, 112
- push
  - IStos, 48
  - Stos, 80
- qs
  - ILista, 41
  - Lista, 65
- red
  - tree.hh, 123
- remove
  - Bucket, 17
  - IBucket, 32
  - ILista, 41, 42
  - ltabn, 53
  - Lista, 65, 66
  - tabn, 86
- right
  - nodeRB, 74
- run
  - asoc\_test, 13
  - IRunnable, 43
  - lista\_test, 69
  - tabn\_test, 90
  - tree\_test, 92
- run.cpp, 113
- run.hh, 113
- SIZE
  - tabl.hh, 120
- search
  - ITreeRB, 57
  - ltabn, 53



- tabn, [87](#)
- TreeRB, [94](#)
- setColour
  - nodeRB, [72](#)
- setKey
  - nodeRB, [72](#)
- setLeft
  - nodeRB, [73](#)
- setParent
  - nodeRB, [73](#)
- setRight
  - nodeRB, [73](#)
- show
  - ltabn, [54](#)
  - tabn, [87](#)
- showElems
  - ltabn, [55](#)
  - tabn, [87](#)
- size
  - lLista, [42](#)
  - Lista, [66](#)
- start
  - IStoper, [45](#)
  - Stoper, [76](#)
- stop
  - IStoper, [45](#)
  - Stoper, [76](#)
- Stoper, [75](#)
  - getElapsedTimeMs, [76](#)
  - start, [76](#)
  - stop, [76](#)
- stoper.cpp, [115](#)
- stoper.hh, [115](#)
- Stos
  - ~Stos, [78](#)
  - get, [78](#)
  - isEmpty, [79](#)
  - pop, [79](#)
  - push, [80](#)
  - Stos, [78](#)
- Stos< T >, [77](#)
- stos.cpp, [116](#)
- stos.hh, [117](#)
- tabl.cpp, [119](#)
- tabl.hh, [119](#)
  - SIZE, [120](#)
- tabn
  - ~tabn, [82](#)
  - aSize, [83](#)
  - add, [83](#)
  - bubblesort, [84](#)
  - isEmpty, [84](#)
  - nOE, [84](#)
  - operator[], [85](#)
  - remove, [86](#)
  - search, [87](#)
  - show, [87](#)
  - showElems, [87](#)
  - tabn, [82](#)
  - tabn< T >, [80](#)
  - tabn\_test, [88](#)
    - ~tabn\_test, [89](#)
    - prepare, [89](#)
    - run, [90](#)
    - tabn\_test, [89](#)
  - temp
    - Bucket, [18](#)
  - Throw
    - ContinueException, [19](#)
    - CriticalException, [21](#)
    - ExceptionBase, [25](#)
  - tree.cpp, [121](#)
    - operator<<, [121](#)
  - tree.hh, [122](#)
    - black, [123](#)
    - Colour, [123](#)
    - operator<<, [123](#)
    - red, [123](#)
  - tree\_test, [90](#)
    - ~tree\_test, [91](#)
    - prepare, [92](#)
    - run, [92](#)
    - tree\_test, [91](#)
  - TreeRB< T >, [93](#)
  - TreeRB
    - ~TreeRB, [94](#)
    - insert, [94](#)
    - search, [94](#)
    - TreeRB, [94](#)
  - up
    - nodeRB, [74](#)
  - what
    - except.hh, [100](#)