Języki Programowania

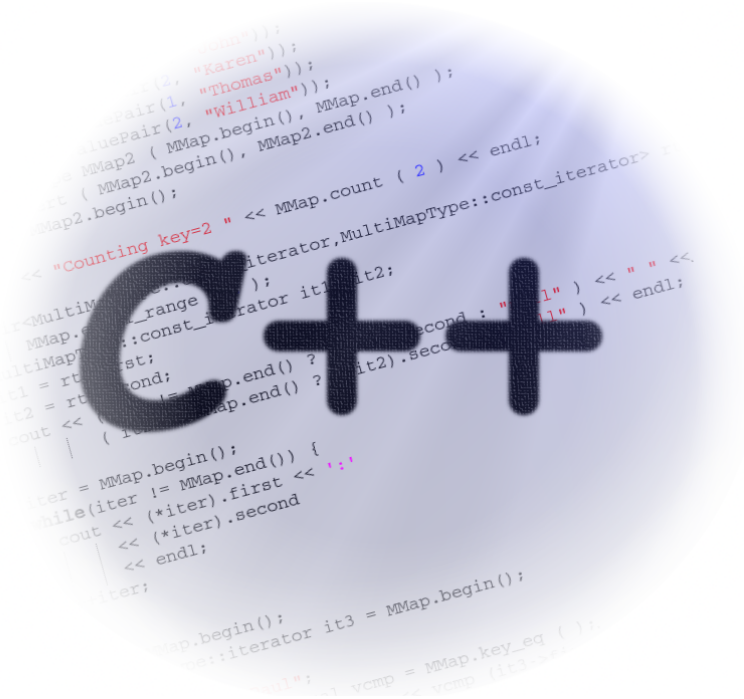
Prowadząca:
dr inż. Hanna Zbroszczyk

e-mail: *hanna.zbroszczyk@pw.edu.pl*
tel: +48 22 234 58 51

Konsultacje:
piątek: 14.00 – 15.30

www: http://www.if.pw.edu.pl/~gos/students/jp

Politechnika Warszawska
Wydział Fizyki
Pok. 117b (wejście przez 115)



STANDARDOWA BIBLIOTEKA SZABLONÓW

Standardowa biblioteka szablonów

Standard Template Library – zbiór szablonowych kontenerów, algorytmów I iteratorów.

Kontener (pojemnik) - struktura danych tego samego typu do przechowywania ich w sposób zorganizowany. Możliwe operacje: dostęp do danych, dodawanie I kasowanie elementów, ew. wyszukiwanie elementu; różnią się szybkością wykonywania poszczególnych operacji.

Złożoność obliczeniowa (O) – ilość podstawowych operacji do wykonania danego algorytmu przy zadanej liczbie danych wejściowych:

- O(K*1) – stała złożoność, szybkość wykonywania jest stała, niezależna od ilości danych
- O(log(K*n)) – złożoność logarytmiczna, przy n danych wejściowych szybkość wykonywania określana logarytmicznie
- O(K*n) - złożoność liniowa, szybkość wykonywania jest wprost proporcjonalna do ilości danych na wejściu
- O(K*n^X) – złożoność wielomianowa, szybkość wykonywania jest proporcjonalna do ilości danych wejściowych podniesionych do potęgi X (X>=2)
- O(K*X^n) – złożoność wykładnicza, szybkość wykonywania jest wprost proporcjonalna do X (X>=2) podniesionego do potęgi równej ilości danych wejściowych

K= const (najczęściej K=1, 2)

Standardowa Biblioteka Szablonów – kontenery - 1

Lista kontenerów dostępnych w STL:

•Lista (list)

•Tablica (vector)

•Tablica podwójnie kończona (deque)

•Tablica bitowa (bitset)

•Drzewo (zbiór) poszukiwań (set)

•Wielokrotne drzewo (zbiór) poszukiwań (multiset)

•Mapa poszukiwań (map)

•Wielokrotna mapa poszukiwań (multimap)

Adapter – wzorzec projektowy, przekształca interfejs klas, dając metody do pracy z danymi w określony sposób, umożliwia podmianę klasy zarządzającej danymi bez zmiany funkcjonowania aplikacji.

Lista adapterów dostępna w STL:

•Stos (stack)

•Kolejka (queue)

•Kolejka priorytetowa (priority_queue)

Iterator – obiekt pozwalający na sekwencyjny dostęp do danych w kontenerze, umożliwia łatwe poruszanie się po całym kontenerze (są uogólnieniem wskaźników). Nie istnieją dla adapterów.

Standardowa Biblioteka Szablonów – kontenery - 2

Wspólne metody dla kontenerów I adapterów:

- konstruktor
- konstruktor kopiujący
- destruktor
- operator=
- empty()
- max_size()
- size()
- swap()
- operatory relacyjne (>, <) - za wyjątkiem kolejki priorytetowej

Wspólne metody dla kontenerów:

- begin()
- end()
- rbegin()
- rend()
- erase()
- clear()

Standardowa Biblioteka Szablonów – kontenery - 3

			Sequence containers			Associative containers				
Headers			<vector>	<deque>	<list>	<set>		<map>		<bitset>
Members		complex	vector	deque	list	set	multiset	map	multimap	bitset
	constructor	*	constructor	constructor	constructor	constructor	constructor	constructor	constructor	constructor
	destructor	O(n)	destructor	destructor	destructor	destructor	destructor	destructor	destructor	
	operator=	O(n)	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operators
iterators	begin	O(1)	begin	begin	begin	begin	begin	begin	begin	
	end	O(1)	end	end	end	end	end	end	end	
	rbegin	O(1)	rbegin	rbegin	rbegin	rbegin	rbegin	rbegin	rbegin	
	rend	O(1)	rend	rend	rend	rend	rend	rend	rend	
capacity	size	*	size	size	size	size	size	size	size	size
	max_size	*	max_size	max_size	max_size	max_size	max_size	max_size	max_size	
	empty	O(1)	empty	empty	empty	empty	empty	empty	empty	
	resize	O(n)	resize	resize	resize					
element access	front	O(1)	front	front	front					
	back	O(1)	back	back	back					
	operator[]	*	operator[]	operator[]				operator[]		operator[]
	at	O(1)	at	at						
modifiers	assign	O(n)	assign	assign	assign					
	insert	*	insert	insert	insert	insert	insert	insert	insert	
	erase	*	erase	erase	erase	erase	erase	erase	erase	
	swap	O(1)	swap	swap	swap	swap	swap	swap	swap	
	clear	O(n)	clear	clear	clear	clear	clear	clear	clear	
	push_front	O(1)		push_front	push_front					
	pop_front	O(1)		pop_front	pop_front					
	push_back	O(1)	push_back	push_back	push_back					
	pop_back	O(1)	pop_back	pop_back	pop_back					
observers	key_comp	O(1)				key_comp	key_comp	key_comp	key_comp	
	value_comp	O(1)				value_comp	value_comp	value_comp	value_comp	
operations	find	O(log n)				find	find	find	find	
	count	O(log n)				count	count	count	count	count
	lower_bound	O(log n)				lower_bound	lower_bound	lower_bound	lower_bound	
	upper_bound	O(log n)				upper_bound	upper_bound	upper_bound	upper_bound	
	equal_range	O(log n)				equal_range	equal_range	equal_range	equal_range	
unique members			capacity reserve		splice remove remove_if unique merge sort reverse					set reset flip to_ulong to_string test any none

http://www.cplusplus.com/reference/

Standardowa Biblioteka Szablonów – algorytmy -1

Algorytmy: STL zawiera ponad 70 funkcji, jakie mogą być stosowane do kontenerów STL oraz tablic; Implementują operacje często używane w programach: znalezienie elementu w kontenerze, wstawienie Elementu w ciąg elementów, usunięcie danego elementu, modyfikacja, porównywanie elementów, Sortowanie, … Działają przy pomocy iteratorów; wymagają włączenia pliku nagłówkowego *algorithm*

Non-modifying sequence operations:

for_each	Apply function to range (template function)
find	Find value in range (function template)
find_if	Find element in range (function template)
find_end	Find last subsequence in range (function template)
find_first_of	Find element from set in range (function template)
adjacent_find	Find equal adjacent elements in range (function template)
count	Count appearances of value in range (function template)
count_if	Return number of elements in range satisfying condition (function template)
mismatch	Return first position where two ranges differ (function template)
equal	Test whether the elements in two ranges are equal (function template)
search	Find subsequence in range (function template)
search_n	Find succession of equal values in range (function template)

http://www.cplusplus.com/reference/

Standardowa Biblioteka Szablonów – algorytmy - 2

http://www.cplusplus.com/reference/

Modifying sequence operations:

copy	Copy range of elements (function template)
copy_backward	Copy range of elements backwards (function template)
swap	Exchange values of two objects (function template)
swap_ranges	Exchange values of two ranges (function template)
iter_swap	Exchange values of objects pointed by two iterators (function template)
transform	Apply function to range (function template)
replace	Replace value in range (function template)
replace_if	Replace values in range (function template)
replace_copy	Copy range replacing value (function template)
replace_copy_if	Copy range replacing value (function template)
fill	Fill range with value (function template)
fill_n	Fill sequence with value (function template)
generate	Generate values for range with function (function template)
generate_n	Generate values for sequence with function (function template)
remove	Remove value from range (function template)
remove_if	Remove elements from range (function template)
remove_copy	Copy range removing value (function template)
remove_copy_if	Copy range removing values (function template)
unique	Remove consecutive duplicates in range (function template)
unique_copy	Copy range removing duplicates (function template)
reverse	Reverse range (function template)
reverse_copy	Copy range reversed (function template)
rotate	Rotate elements in range (function template)
rotate_copy	Copy rotated range (function template)
random_shuffle	Rearrange elements in range randomly (function template)
partition	Partition range in two (function template)
stable_partition	Partition range in two - stable ordering (function template)

Standardowa Biblioteka Szablonów – algorytmy - 3

http://www.cplusplus.com/reference/

Sorting:

sort	Sort elements in range (function template)
stable_sort	Sort elements preserving order of equivalents (function template)
partial_sort	Partially Sort elements in range (function template)
partial_sort_copy	Copy and partially sort range (function template)
nth_element	Sort element in range (function template)

Binary search (operating on sorted ranges):

lower_bound	Return iterator to lower bound (function template)
upper_bound	Return iterator to upper bound (function template)
equal_range	Get subrange of equal elements (function template)
binary_search	Test if value exists in sorted array (function template)

Merge (operating on sorted ranges):

merge	Merge sorted ranges (function template)
inplace_merge	Merge consecutive sorted ranges (function template)
includes	Test whether sorted range includes another sorted range (function template)
set_union	Union of two sorted ranges (function template)
set_intersection	Intersection of two sorted ranges (function template)
set_difference	Difference of two sorted ranges (function template)
set_symmetric_difference	Symmetric difference of two sorted ranges (function template)

Heap:

push_heap	Push element into heap range (function template)
pop_heap	Pop element from heap range (function template)
make_heap	Make heap from range (function template)
sort_heap	Sort elements of heap (function template)

Min/max:

min	Return the lesser of two arguments (function template)
max	Return the greater of two arguments (function template)
min_element	Return smallest element in range (function template)
max_element	Return largest element in range (function template)
lexicographical_compare	Lexicographical less-than comparison (function template)
next_permutation	Transform range to next permutation (function template)
prev_permutation	Transform range to previous permutation (function template)

Stos - 1

Stos – struktura danych, w której możliwy jest dostęp jedynie do jednego elementu (wierzchołka).

LILO – Last In First Out, należy włączyć plik nagłówkowy: *stack*

```
template <typename T, typename kontenerDanych = deque <T> >
class stack {
        ..
        void push(const T& val) – dodanie elementu
        void pop() – usunięcie elementu
        bool empty() const– informacja, czy stos jest pusty
        size_type size() const – ilość elementów
        T& top() – wartość ostatniego elementu
};
```

Stos – przykład - 1

```
#include <iostream>
#include <string>
#include <iomanip>
#include <stack>
using namespace std;
class data {
        string name;
        unsigned int age;
        public:
        data(string nn="", unsigned int aa= 0): name(nn), age(aa) {}
        ~data() {}
        friend ostream& operator<<(ostream& out, data& dd) {
                out<<setw(25)<<dd.name<<setw(5)<<dd.age; return out;}
};
int main() {
        stack <int> st1; // stack <int, deque<int> > st1;
        stack <data> st2;
        for (int i=0; i<3; i++) st1.push(5*(i+1));
        st2.push(data("Piotr Kowalski", 30));
```

Stos – przykład - 2

```
        cout<<"Ostatnie na stosie: "<<endl;
        cout<<"st1: "<<st1.top()<<endl;
        cout<<"st2: "<<st2.top()<<endl;
        st1.top() *= 3;
        cout<<"st1: "<<st1.top()<<endl;
        cout<<"Liczba elementów na stosie: "<<endl;
        cout<<"st1: "<<st1.size()<<endl;
        cout<<"st2: "<<st2.size()<<endl;
        cout<<"Stos pusty?"<<endl;
        cout<<"st1: " <<boolalpha<<st1.empty()<<endl;
        cout<<"st2: "<<boolalpha<<st2.empty()<<endl;
        while(!st1.empty()) {      cout<<st1.top()<<endl; st1.pop(); }
        if (st1.empty()) cout<<"Stos st1 pusty"<<endl;
        while(!st2.empty()) { cout<<st2.top()<<endl; st2.pop(); }
        if (st2.empty()) cout<<"Stos st2 pusty"<<endl;
        return 1;
}
```

Kolejka - 1

Kolejka – struktura danych, w której możliwy jest dostęp do pierwszego o ostatniego elementu.

FIFO – First In First Out, należy włączyć plik nagłówkowy: *queue*

```
template <typename T, typename kontenerDanych = deque <T> >
class queue {
        ..
        void push(const T& val) – dodanie elementu do końca kolejki
        void pop() – usunięcie elementu z początku kolejki
        bool empty() const– informacja, czy kolejka jest pusty
        size_type size() const – ilość elementów
        T& front() – wartość pierwszego elementu
        T& back() – wartość ostatniego elementu
};
```

Kolejka – przykład - 1

```
#include <iostream>
#include <string>
#include <iomanip>
#include <queue>
using namespace std;
class data {
        string name;
        unsigned int age;
        public:
        data(string nn="", unsigned int aa= 0): name(nn), age(aa) {}
        ~data() {}
        friend ostream& operator<<(ostream& out, data& dd) {
                out<<setw(25)<<dd.name<<setw(5)<<dd.age; return out;}
};
int main() {
        queue <int> st1;
        queue <data> st2;
        for (int i=0; i<3; i++) st1.push(3*(i+1));
        st2.push(data("Anna Nowak", 25));
        st2.push(data("Piotr Kowalski", 30));
```

Kolejka – przykład - 2

```
cout<<"Ostatnie w kolejce: "<<endl;
cout<<"st1: "<<st1.back()<<endl;
cout<<"st2: "<<st2.back()<<endl;
cout<<"Pierwsze w kolejce: "<<endl;
cout<<"st1: "<<st1.front()<<endl;
cout<<"st2: "<<st2.front()<<endl;

st1.back() *= 3;
cout<<"st1: "<<st1.back()<<endl;
st2.front() = data("Tomasz Iksinski",20);
cout<<"st2: "<<st2.front()<<endl;

cout<<"Liczba elementów w kolejce: "<<endl;
cout<<"st1: "<<st1.size()<<endl;
cout<<"st2: "<<st2.size()<<endl;

cout<<"Kolejka pusta?"<<endl;
cout<<"st1: " <<boolalpha<<st1.empty()<<endl;
cout<<"st2: "<<boolalpha<<st2.empty()<<endl;
```

Kolejka – przykład - 3

```
    while(!st1.empty()) {
        cout<<st1.front()<<endl;
        st1.pop();
    }
    if (st1.empty()) cout<<"Kolejka st1 pusta"<<endl;
    while(!st2.empty()) {
        cout<<st2.front()<<endl;
        st2.pop();
    }
    if (st2.empty()) cout<<"Kolejka st2 pusta"<<endl;
    return 1;
}
```

Wektor - 1

Wektor – pojemnik na dane (dynamiczna tablica) z dostępem do każdego elementu;

należy włączyć plik nagłówkowy: *queue*

```
template <typename T>
class vector {
    ..
    void push_back(const T& val) – dodanie elementu do końca tablicy
    void insert(const size_type& where, const T& val) – dodanie elementu
        do tablicy w podanym miejscu
    void pop_back() - usuwa ostatni element
    void erase(const size_type& where) – usunięcie konkretnego elementu
    void clear() - usunięcie całej tablicy
    size_type size() const – ilość elementów
    resize(size_type ss)
    T& begin() – pierwszy element
    T& end() – ostatni element
};
```

Wektor – przykład - 1

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
	vector <int> v1;
	v1.push_back(0);
	//for (int i=0; i<5; i++) v1.push_back(10*i-10);
	for (int i=0; i<5; i++) v1.insert(v1.begin()+i, 10*i-10);

	cout<<"v1: "<<endl;
	for (int i=0; i<v1.size(); i++) cout<<v1[i]<<endl;

	sort(v1.begin(), v1.end());
	cout<<"Po posortowaniu v1: "<<endl;
	for (int i=0; i<v1.size(); i++) cout<<v1[i]<<endl;
```

Wektor – przykład - 2

```
    v1.pop_back();
    cout<<"Po usunięciu ostatniego elementu v1: "<<endl;
    for (int i=0; i<v1.size(); i++) cout<<v1[i]<<endl;

    v1.resize(3);
    cout<<"Po zmianie rozmiaru v1 na 3 elementy: "<<endl;
    for (int i=0; i<v1.size(); i++) cout<<v1[i]<<endl;

    v1.erase(v1.begin()+2);
    cout<<"Po usunięciu trzeciego elementu v1: "<<endl;
    for (int i=0; i<v1.size(); i++) cout<<v1[i]<<endl;

    v1.clear();
    cout<<"Po usunięciu tablicy v1: "<<endl;
    for (int i=0; i<v1.size(); i++) cout<<v1[i]<<endl;

    return 1;
}
```

Lista - 1

Lista – pojemnik na dane z dostępem do każdego elementu, gdzie każdy element wskazuje na kolejny (lista jednokierunkowa) lub poprzedni I kolejny (lista dwukierunkowa); należy włączyć plik nagłówkowy: *list*

```
template <typename T>
class list {
        ..
        void push_back(const T& val) – dodanie elementu do końca listy
        void push_front(const T& val) – dodanie elementu do początku listy
        void insert(iterator i, const int howMany, const T& val) – dodanie elementu
                do listy w podanym miejscu w podanej liczbie kopii
        void insert(iterator i, const T& first, const T& last) – wstawia elementy
                pomiędzy first i last  w miejscu wskazanym przez iterator
        void pop_back() - usuwa ostatni element
        void pop_front() - usuwa pierwszy element
        void remove(const T& el) – usuwa z listy wszystkie węzły z elementem el
        void erase(iterator first, iterator last) – usunięcie elementu pomiędzy first a last
        void clear() - usunięcie całej listy
}
```

Lista - 2

void splice (iterator ii, list<T,Allocator>& x) - przeniesienie elementów tablicy x

void splice (iterator ii, list<T,Allocator>& x, iterator i) - przeniesienie elementu i z listy x

void splice (iterator ii, list<T,Allocator>& x, iterator first, iterator last) - przeniesienie elementów z zadanego zakresu z listy x

void merge (list<T,Allocator>& x) - połączenie z listą x

size_type size() const – ilość elementów

resize(size_type ss) – zmiana rozmiaru listy

T& begin() – pierwszy element

T& end() – ostatni element

};

Lista – przykład

```
#include <iostream>
#include <list>
#include <algorithm>
#include <iomanip>
using namespace std;
int main() {
        list<int> lst1;
        list<int> lst2(3, 5); //5 5 5
        list<int>::iterator i1 = lst1.begin();
        list<int>::iterator i2;
        for (int i=0; i<10; i++) lst1.push_back(i);
        lst1.sort();
        list<int> lst3(lst1);
        lst3.splice(++i1, lst2);
        for (i2 = lst3.begin(); i2!=lst3.end(); i2++) cout<<setw(5)<<*i2; cout<<endl;
        for (i2 = lst1.begin(); i2!=lst1.end(); i2++) cout<<setw(5)<<*i2; cout<<endl;
        lst1.swap(lst3);
        for (i2 = lst3.begin(); i2!=lst3.end(); i2++) cout<<setw(5)<<*i2; cout<<endl;
        for (i2 = lst1.begin(); i2!=lst1.end(); i2++) cout<<setw(5)<<*i2; cout<<endl;
        return 1;
}
```

Tablica (kolejka) podwójnie kończona

Tablica(kolejka) podwójnie kończona (Double Ended QUE - deque) – pojemnik na dane;

należy włączyć plik nagłówkowy: *deque*

```
template <typename T>
class deque {
        ..
        void push_back(const T& val) – dodanie elementu do końca kolejki
        void push_front(const T& val) – dodanie elementu do początku kolejki
        iterator insert(iterator pos, const T& x) – wstawia element x w miejscu pos
        void insert(iterator pos, size_type n, const T& x) – wstawia element x w miejscu pos
                w ilości n
        void insert(iterator position, iterator first, iterator last) – wstawia elementy
                z zakresu [first, last) w miejscu pos
        void pop_back() - usuwa ostatni element
        void pop_front() - usuwa pierwszy element
        void erase(iterator first, iterator last) – usunięcie elementów z zakresu
        void erase(iterator i) – usunięcie elementi i-tego
        void clear() - usunięcie całej kolejki
        void resize(size_type s) – zmiana rozmiaru
        size_type size() - zwraca rozmiar (ilość elementów) w kolejce ...
};
```

Tablica (kolejka) podwójnie kończona – przykład

```
#include <iostream>
#include <deque>
#include <vector>
#include <iomanip>
using namespace std;
int main ()
{
        deque<int> md;
        deque<int>::iterator it;
        for (int i=1; i<6; i++) md.push_back(i); // 1 2 3 4 5
        it = md.begin();
        ++it;
        it = md.insert (it,10);                  // 1 10 2 3 4 5
        md.insert (it,2,20);                     // 1 20 20 10 2 3 4 5
        it = md.begin()+2;
        vector<int> myvector (2,30);
        md.insert(it,myvector.begin(),myvector.end());//1 20 30 30 20 10 2 3 4 5
        for (it=md.begin(); it<md.end(); it++) cout <<setw(5)<< *it;
        cout << endl;
        return 1;
}
```

Zbiór - 1

Zbiór (set) – struktura danych do przechowywania posortowanych, uporządkowanych elementów;

wymaga włączenia pliku nagłówkowego: *set*

```
template <typename T>
class set {
    ..
    void insert(const T& val) – dodanie elementu
    iterator insert(iterator position, const T& value) – dodanie elementu
    void erase (iterator position) – usuwanie elementu z pozycji
    size_type erase(const key_type& x ) - usuwanie elementów z kluczem
    void erase (iterator first, iterator last ) - usuwanie elementów z zakresu
    bool empty() - sprawdzenie, czy zbiór jest pusty
    size_type size() const – ilość elementów
    void resize(size_type ss) – zmiana rozmiaru zbioru
    T& begin() – pierwszy element
    T& end() – ostatni element
};
```

Zbiór – przykład - 1

```
#include <iostream>
#include <iomanip>
#include <set>
using namespace std;

void print(set<int> ss){
        set<int>::iterator ii;
        for (ii=ss.begin(); ii!=ss.end(); ii++) cout <<setw(5)<< *ii;
        cout << endl;
}

int main() {
        set<int> s1;
        set<int>::iterator it;
        if (s1.empty()) for (int i=1; i<=5; i++) s1.insert(i*10);    //10 20 30 40 50
        it=s1.end(); //it=s1.begin();
        s1.insert (it,1);
        s1.insert (it,2);
        int tab[]= {2,4,6,8,10};
```

Zbiór – przykład - 2

```
    s1.insert (tab, tab+5);
        print(s1); //1 2 4 6 8 10 20 30 40 50
        it=s1.begin();
        it++;
        s1.erase (it);
        s1.erase(10);
        it=s1.find(20);
        s1.erase (it, s1.end());
        print(s1); //1 4 6 8
        s1.clear();
        print(s1); //
        return 1;
}
```

Mapa - 1

Mapa (map) – struktura danych analogiczna do tablicy, jaką można indeksować danymi dowolnego typu. Indeksy mapy to klucze (*keys*), które są unikatowe; wymaga włączenia pliku nagłówkowego: *map*

```
template <typename T>
class map {
        ..
        void insert ( const T& val) – dodanie elementu
        iterator insert ( iterator position, const T& x) – dodanie elementu
        void insert (iterator first, iterator last ) - dodanie elementu
        void erase (iterator position) – usuwanie elementu z pozycji
        size_type erase(const key_type& x ) - usuwanie elementów z kluczem
        void erase (iterator first, iterator last ) - usuwanie elementów z zakresu
        void clear() - usunięcie mapy
        iterator find (const T& val)- znalezienie elementu
        size_type count ( const key_type& x) const – policzenie ilości elementów z kluczem
        size_type size() const – ilość elementów
        T& begin() – pierwszy element
        T& end() – ostatni element
};
```

Mapa – przykład - 1

```
#include <iostream>
#include <iomanip>
#include <map>
using namespace std;
int main() {
        map<char,int> m1;
        map<char,int>::iterator it;
        m1.insert(pair<char,int>('a',100) );
        m1.insert(pair<char,int>('z',200) );
        m1.insert(pair<char,int>('z',500) );
        it=m1.begin();
        m1.insert(it, pair<char,int>('b',300));
        m1.insert(it, pair<char,int>('c',400));
        map<char,int> m2;
        m2.insert(m1.begin(),m1.find('c'));
        cout << "m1"<<endl;
        for (it=m1.begin(); it!=m1.end(); it++) cout <<setw(5)<< it->first<<
                setw(5)<< (*it).second;
        cout << endl;
```

Mapa – przykład - 2

```
    cout << "m2"<<endl;
    for (it=m2.begin(); it!=m2.end(); it++) cout <<setw(5)<< it->first<<
        setw(5)<< (*it).second;
        map<char,string> m3;
    m3['a']="element";
    m3['b']="kolejny element";
    m3['c']=m3['b'];
    cout << "m3['a'] is " << m3['a'] << endl;
    cout << "m3['b'] is " << m3['b'] << endl;
    cout << "m3['c'] is " << m3['c'] << endl;
    cout << "m3['d'] is " << m3['d'] << endl;
    cout << "m3 zawiera " << (int) m3.size() << " elementow" << endl; //4
    m3.erase(m3.find('b'));
    cout << "m3 zawiera " <<(int)m3.size() << " elementow" << endl;   //3
    m3.clear();
    cout << "m3 zawiera " <<(int)m3.size() << " elementow" << endl;   //0
    return 1;
}
```

Para – dygresja

Szablon par o różnych lub identycznych typach.

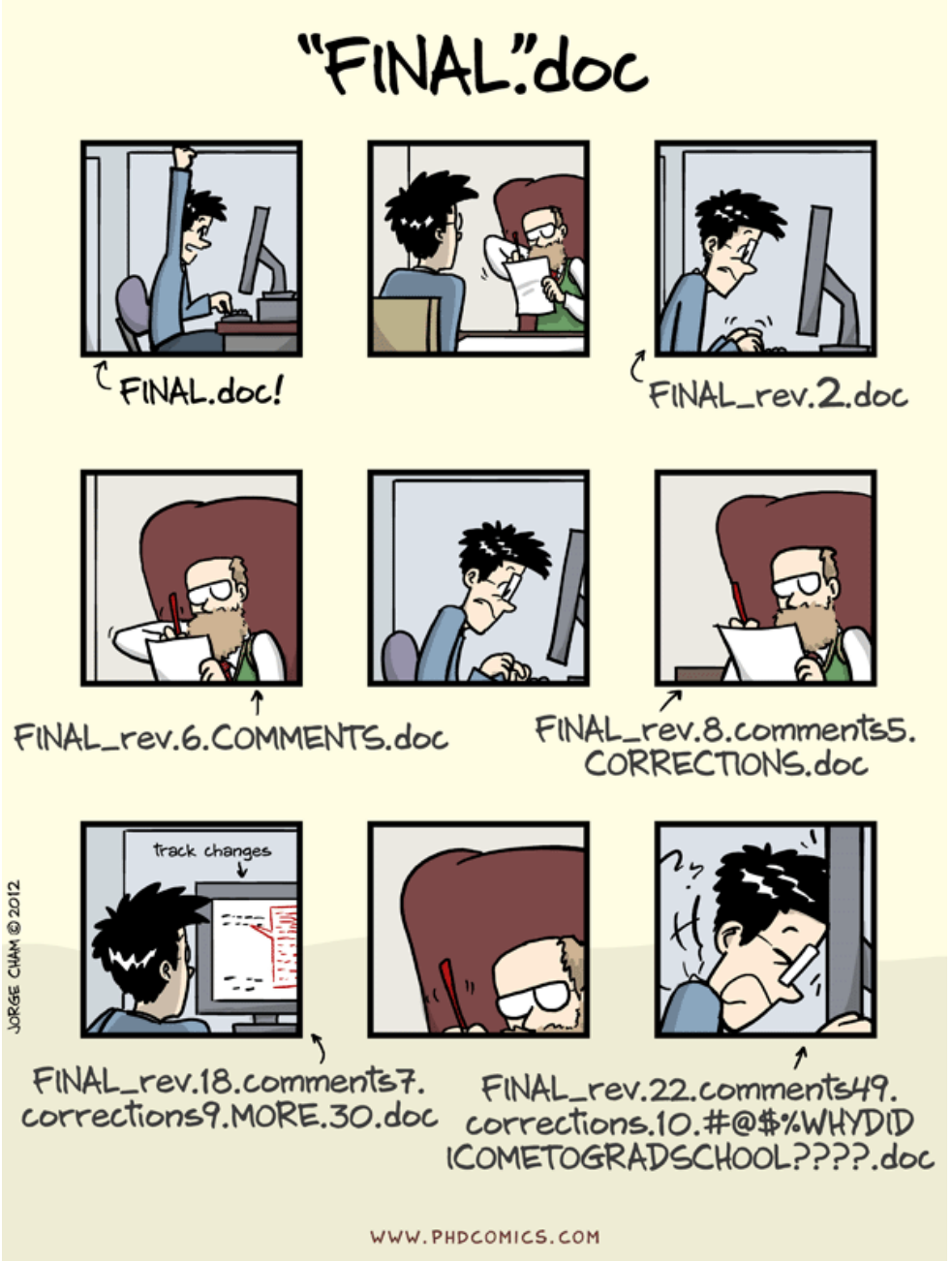
```
template <class T1, class T2> struct pair {
  T1 first;
  T2 second;
  pair() : first(T1()), second(T2()) {}
  pair(const T1& x, const T2& y) : first(x), second(y) {}
  template <class U, class V>
  pair (const pair<U,V> &p) : first(p.first), second(p.second) { }
};
#include <iostream>
#include <string>
using namespace std;
int main () {
  pair <string,double> product1("chleb",2);
  pair <string,double> product2, product3;
  product2.first = "mleko"; product2.second = 1.50;
  product3 = make_pair("ser", 20.0);
  cout<<"Cena "<<product1.first<<" is zl "<<product1.second<<endl;
  cout<<"Cena "<<product2.first<<" is zl "<<product2.second<<endl;
  cout<<"Cena "<<product3.first<<" is zl "<<product3.second<<endl;
  return 0;
}
```

Multimapa, multizbiór, zbiór bitowy..

Zbiór bitowy (bitset) – struktura danych do przechowywania bitów (0, 1; true, false); wymaga włączenia pliku nagłówkowego: *bitset*

Multizbiór (multiset) – struktura danych, analogiczna do zbioru z możliwością przechowywania wielu kluczy z tą samą wartością; wymaga włączenia pliku nagłówkowego: *multiset*

Multimapa (multimap) – struktura danych, analogiczna do mapy z możliwością przechowywania wielu kluczy z tą samą wartością mapy; wymaga włączenia pliku nagłówkowego: *multimap*



KONIEC WYKŁADU 10