**Zadania – spis treści**

1) **Napisz szablon funkcji, która wyznacza sumę elementów tablicy elementów T (paramatr szablonu). Napisz przykład zastosowania po parametryzacji typem std::string.**

2) **Mapa (graf) zawiera miasta i drogi. Odcinek łączący (krawędź) można opisać jako (m1,m2,d) gdzie mx to nr miast, a d to droga między nimi (waga kraw.). Wykorzystując STL napisz class Mapa + dodajOdcinek dodajMiasto czy trasa dlugosc wypisz**

3) **Zad z grafem, analogiczne do miast, z tym że metody: dodajWierzcholek, dodajKrawedz, usunKrawedz, usunKrawedzieWierzch, usunWierzcholek, dlug – długość trasy R1 by forum**

4) **Klasy B i C dziedzicza po A{public: virtual~A..}; Klasa ArrayOfA przechwuje w tablicy wskaźniki typu A* do obiektów A,B,C dla których pamieć przydzielana jest na starcie. Zaimplementuj ArrayOfA**
   a) **konstruktor (ustala wielkość tablicy), destruktor, operator przypisania, operator kopiujący, operator + (łączenie tablic)**

5) **Bez STL;**
   **Zdefiniuj szablon będący implementacją IQueue<T> (kolejka FIFO)**
   **template <class T> class IQueue**
   **{**
   **public:**
   **virtual bool put (const T&t) = 0;**
   **virtual bool get (T &t) = 0;**
   **};**
   **Napisz konstruktor ustalający rozmiar kolejki oraz metody put (na koniec) get( zwraca pierwszy el, przesówa pozostałe)**

6) **Użyj STL. Klasy B, C i D dziedzicza po class A**
   **Klasa Alist dziedziczy po list<A*> i może przechowywać obiekty klas A,B,C i D. Napisz klasę AList:**
   a) **deklaracja, konstruktor, destruktor, operator przypisania, konstruktor kopijący, oprator +**

7) **Użyj STL. Klasa Directory zawiera dowolną liczbę obiektów klasy File lub Directory. Obie klasy dziedxziczą po DirElement. File i Directory mają swoją nazwe. Do przechowywania obiektów wybierz dowoly kontener, w którym umieszczone są wskażźniki typu DirElement***
   a) **do Director dodaj bool add(DirElement*) - dodaj elementy unikalnych nazw - bool delete(const char*name)**

8) **Klasa MapTextInt przechowuje wskaźnik do tekstów, dla których pamięć jest przechowywana na stercie oraz ich liczbę, czyli paty (char*, int). Zaimplementuj w postaci listy jednokierunkowej**
   **Deklaracja struktur danych, konstruktor, funkcja void add(const char*text), destruktor, konstruktor kopiujacy, operator przypisania**

**9) [ Zadanie 1 25pkt]**

**Wykorzystujac kontener STL vector napisz klase VectorOfIntArray**
**Przechowywujaca tablice elementow typu int o zmiennej dlugosci.(Potraktujmy je jak wiersze**
**Dwuwymiarowej tablicy)Klasa zarzadza pamiecia wierszy,JAWNIE ALOKUJE JA I ZWALNIA.Dla wierszy**
**//nie jest stosowany kontener STL!**
**//Metody:**
**// a) void add(int*tab,int size)-dodaje wiersz,przydziela pamiec,kopiuje**

**elementy**
**// b) int getSize(int rowidx)-zwraca liczbe elementow wiersza o indeksie rowidx**
**// c) int*get (int rowidx)-zwraca wskaznik do elementow wiersza rowidx**
**// d) destruktor**
**// e) konstruktor kopiujący**

**10) [ Zadanie 2 25pkt]**

**Bez STL. Napisz szablon template <class T>class Array<T>{ ...}klasy przechowywujacych elementy typu T w tablicy, ktora moze dynamicznie przyrastac.**
**// a)zadeklaruj klase oraz podaj konstruktor**
**// b)zaimplementuj destruktor**
**// c)funkcja pushBack(const &t)**
**// d)funkcje bool reserve(int size), ktora bedzie przedluzala tablice do zadanego**
**// rozmiaru**
**// e)operator[]**

**11) [ Zadanie 3 20pkt]**
**Wykorzystaj kontener z zad.2 UnaryPredicate to klasa reprezentujaca jednoargumentowa funkcje zwracajaca bool.**
**//a)napisz szablon zewnetrznej funkcji copyIf()parametryzowanej typami T i UnaryPredicate**
**// template <classT, class UnaryPredicate> void copyIf(..**
**//umozliwiajacej skopiowanie z Array<T> src do Array<T> target wszystkich elementow**
**//spelniajacych predykat przekazany,jako obiekt funkcyjny klasy UnaryPredicate.**
**//b)Napisz klase dla obiektu funkcyjnego umozliwiajaca testowanie, czy liczba calkowita jest wieksza od 0 i mniejsza od 10**
**//c)Wywolaj dla tablic parametryzowanych typem int i utworzonego obiektu funkcyjnego**

**12) [ Zadanie 4 ]**
**Klasa Iterator ma:**
**// a)konstruktor**
**// b)metode get() ,ktora zwraca biezacy element**
**// c)metode void next() , ktora przesuwa iterator na nastepny element,**
**// d)metode bool good(), ktora zwraca true, jesli stan iteratora pozwala na**
**// poprawny odczyt elementu.**
**Napisz iterator, ktory bedzie w konstruktorze otrzymywal nazwe pliku tekstowego, otwieral go,a nastepnie wydzielal z niego niepuste linie i zwracal je w postaci obiektow klasy string.**

**13) [ Zadanie 5 15pkt]**

**Dla klasy struct Point(double tab[3]); zadeklaruj i zaimplementuj**
**a)operator+**
**b)operator==**
**c)operator<< umozliwiajacy zapis do strumienia ostream.**

**14) [ Zadanie 1 25pkt]**
**Wykorzystując kontener STL list lub vector zaimplementuj klasę String2String reprezentującą/relację string x string, czyli przechowującą unikalne pary tekstów.**
**Metody:**
**//a) void add(const char*a, const char *b) – dodaje parę (a,b)**
**//b) void remove(const char*a, const char*b) – usuwa parę (a,b)**
**//c) int find(const char* a, int start) wyszukuje parę, której pierwszym elementem jest a, //rozpoczyna wyszukiwanie od indeksu start. Zwraca -1, jeśli takiego elementu brak**
**//d) void get(const char*a, String2String&target) – dodaje do target wszystkie pary, których //pierwszym elementem jest a**
**//e) oprator +=**

**15) [ Zadanie 2 25pkt]**

**Bez STL. Napisz szablon template <class T >class List<T>{ …} klasy przechowującej element typu T na liście jednokierunkowej.**
**//a) Zadeklaruj klasę i pomocnicze struktury danych oraz zaimplementuj konstruktor**
**//b) zaimplementuj destruktor**

**//c) funkcję pushFront(const T&t)**
**//d) funkcję bool getLast(T&t), która wartość elementu umieszcza w t**
**//e) operator przypisania**

**16) [ Zadanie 3 15pkt]**

**Szablon klasy zapewniającej standardową funkcję do porównywania jest zdefiniowany jako:**
**template <class T> class DefaultComparator**
**//{**
**//public:**
**//static bool compare(const T&a, const T&b){return a == b;}**
**//};**
**Wykorzystując funkcjonalność listy z zad.2 napisz szablon zbioru:**
**template <class T, class Comparator = DefaultComparator<T> >**
**class Set …**
**Z metodami (a)insert(const T&t) dodaje unikalny element do zbioru (b) metodę**
**Bool has(const T&) sprawdzającą czy element należy do zbioru(c) operator <**
**sprawdzający inkluzję zbiorów.**

**17) [ Zadanie 4 ]**

**Klasa Iterator ma (a) konstruktor (b) metodę get(), która zwraca bieżący element, (c) metodę void next(), która przesuwa iterator na następny element, (d) metodę bool good(), która**
**zwraca true, jeśli stan iteratora pozwala na poprawny odczyt elementu. Napisz iterator, który będzie w konstruktorze otrzymywał nazwę pliku tekstowego, otwierał go, a następnie wydzielał z niego wyłącznie litery i cyfry, zastępując wszystkie pozostałe znaki pojedynczym znakiem spacji.**

**18) [ Zadanie 5 ]**

**Dla klasy class Ulamek(int licznik; int mianownik); zadeklaruj i zaimplementuj:**
**a) Ulamek& operator += (const Ulamek&)**
**b) operator ==**
**//c) operator >> umożliwiający odczyt ze strumienia istream**

**19) [ Zadanie 1 20pkt]**
**Napisz szablon template <class T>class List<T>{...} klasy przechowywujacych elementy typu T na liscie dwukierunkowej**
**// a)Zadeklaruj klase i pomocnicze struktury danych**
**// b)zaimplementuj konstruktor**
**// c)destruktor**
**// d)funkcje pushBack(const T&t)**

**20) ----------[ Zadanie 2 25pkt]----------**
**Zaimplementuj klase Iterator pozwalajaca na wydzieleniemz tekstu slow(ciagow znakow oddzielonych bialymi znakami) z metodami:**
**// a) Iterator(const char*) konstruktor, jego parametrem**
**// jest analizowany tekst**
**// b) bool haveMoreWords() - czy jest jeszcze jakies slowo?**
**// c) void nextWord() -wyszukuje nastepne slowo i przesuwa**
**// iterator**
**// d) const char*get() -zwraca biezace slowo**
**// e) podaj przyklad wywolania**

**21) ----------[ Zadanie 4 15pkt]----------**
**Dla klasy class Point{double x,y;}zadeklaruj i zaimplementuj**
**// a) operator+**

**// b) operator-=**
**// c) operator==**

**22) //----------[ Zadanie 6 10pkt]----------**
**//Co zostanie wypisane i dlaczego?**

**23) a) Zadeklaruj szablon template <class T> class Array<T> {} klasy bedącej tablica elementów T**
**b) Zaimplementuj konstruktor Array<T> (int size) ustalający rozmiar tab i alokujacy pamięć**
**c) destruktor zwalniajacy pamięć oraz operator Array<T>&operator+=(const Array<T>&other) rozszerzający tablicę o elementy argumentu other**

**24)Klasa FileIterator pozwala odczytać znaki z pliku traktowanego jak kontener. Zadeklaruj i zaimplementuj:**
**a) konstruktor FileIterator(const char*name)**
**b) destruktor klasy FileIterator**
**c) andEnd() - czy iterator sosiągnął koniec kontenera?**
**d) next() - przesuwanie iteratora na anstępny element (znak)**
**e) int get() - zwraca aktualny element**

**25) Korzystając z kontenera STL list zaimplementuj klasę PersonSet przechopwującą unikalne ze względu na wartości atrybuty PESEL instancje obiektów klasy class Person(....) Zaimplementuj:**
**a) void insert(..), bool remove(...) bool exist (..) [sprawdza czy osoba jest w kontenerze]**
**void select [dodaje do target wszystkie obiekty zawierające łańcuch txtx w jednym z atrybutów**
**operator + dodający dwa zbiory do siebbie**

**26) Graf nieskierowany zawiera pewna liczbę ponuerowanych wierzvchołków oraz pwną liczbę krawedzi.**
**Przechowywane są one w kontenerach STL. Krawedzie są reprezentowane przez trójki (start, end, dlug) gdzie start,end to numery wierzchołków. Parametr dlug jest waga krawedz. Z wierzchołka moze wychodzic wiecej niz jedna krawedz, dwa wierzcholki moga byc polaczone tylko jedna krawedzia. Def krawedzi (start, end, w) - kolejnosc nie ma znaczenia. Droga w grafie jest sekwencja wierzcholkow, z których kazde dwa kolejne są połaczone krawedziami.**

**a) Zadeklaruj klaszę Graf + konstruktor.**
**metody: bool dodajWIerzcholek(int n), dodajKrawdz, usunKrawedz,usunKrawWierzch, usunWierzcholek, dlug - oblicza długosc drogi, zwraca false jesli d nie jest droga**

**27) Graf skierowany zawiera n wierzchołków numerowanych od 0 do n-1 oraz pwną liczbę krawdęedzi przechowywanych w kontenerze STL jako pary (start, end).**
**Dwa wierzchołki mogą być tylko jedną skierowana krawędzia.**

**a) konstruktor Graf(int n) - ustala liczbe wierzchołków**
**b) metode dodajKrawedz, operator < - sprawdza czy jeden graf jest podgrafem drugiego, bool jestDroga - sprawdza czy jest droga**

**28) Napisz szablon funkcji parametryzowanej typem T, która do kontenera vector<T>& result przepisuje wszystkie elementy kontenera const vector<T>& source mniejsze niż const T& key. Podaj przykład wywołania.**

**29) Klasa Person zdefiniowana jako : class Person {string surname; string name; char pesel[12]} Napisz klasę PersonSet przechowującą pojedyncze instancje obiektów klasy Person z metodami:**
**void insert(Person) - dodaje element do zbioru**
**bool has(Person) - sprawdza czy element należy do zbioru**
**operator == - do porównywania czy zbiory są równe**

**operator < do sprawdzania czy jeden zbiór zawiera się w drugim**

**Jako kontenera użyj dowolnego szablonu z biblioteki standardowej C++ za wyjątkiem set. Możesz dodać metody do klasy Person**

**30) Zadeklaruj i zaimplementuj klasę Dictionary, która słowom (string) przypisuje jedno lub więcej znaczeń (string). Słownik powinien przechowywać unikalne pary (słowo, znaczenie). Jako kontenera użyj dowolnego szablonu biblioteki standardowej STL**

*** void add(const char* w, const char* m) - dodanie pary do słownika**
*** void find(ocnst char* w,list<string>& target) - poszukuje wszystkich znaczeń słowa w i umieszcza na liście target**
*** void deleteWord(const char* w) - usuwanie wszystkich par dla słowa w**
*** void join(const Dictionary& other) - dołącza zawartość drugiego słownika**

**31)** zad.1 25pkt
**Wykorzystując kontener STL *list* lub *vector* zaimplementuj klasę *StringToInt* reprezentującej odwzorowanie tekst->liczba nieujemna. Na przykład klasa może być używana do zliczania wystąpień słow w dokumencie.**
**Napisz metody**
**a) *int get(const char*key)* zwraca liczbę przypisaną *key***
**b) *void add(const char*key)* zwiększa liczbę przypisaną *key* o 1**
**c) *void remove(const char*key)* zmniejsza o 1 liczbę przypisaną *key.* (jeśli jest ona większa niż**
**d) operator + dla klasy *StringToInt***
**e) oprator <= dla klasy *StringToInt:* sprawdza czy: dla każdego (text, val1) należącego do A istnieje (text, val2) należące do B: $val_1 \leq val_2$**

**32)**zad.2 25pkt
**Bez STL. Napisz szablon *template <class T >class List<T>{ …}* klasy przechowującej elementy**
**typu *T* na liście dwukierunkowej.**
**a) Zadeklaruj klasę i pomocnicze struktury danych oraz zaimplementuj konstruktor**
**b) zaimplementuj destruktor**
**c) funkcję pushBack(const T&t)**
**d) funkcję void deleteFront()**
**e) konstruktor kopiujący**

**33)** zad.3 15pkt
**Wykorzystaj kontener z zad.2. *UnaryFunction* to klasa reprezentująca jednoargumentową funkcję typu *void*.**
**a) Napisz szablon funkcji *forEach()* parametryzowanej typami *T* i *UnaryFunction***
***template <class T, class UnaryFunction> void forEach(...***
**umożliwiającej przeprowadzenie na każdym elemencie kontenera *List<T>* operacji przekazanej**
**jako obiekt funkcyjny**
**b) Napisz klasę dla obiektu funkcyjnego umożliwiającą wydruk elementow całkowitych**
**c) Wywołaj funkcję dla listy parametryzowanej typem int i utworzonego obiektu funkcyjnego**

**34)** zad.4 20pkt
**Klasa *Iterator* ma (a) konstruktor (b) metodę *get(),* ktora zwraca bieżący element, (c) metodę**
***void next()*, ktora przesuwa iterator na następny element, (d) metodę *bool good()*, ktora zwraca true, jeśli stan iteratora pozwala na poprawny odczyt elementu.**
**Napisz iterator, ktory będzie w konstruktorze otrzymywał wskaźnik typu *const char** wskazujący tekst, a następnie wydzielał z niego słowa (ciągi znakow oddzielone białym**

**znakami) i zwracał je w postaci obiektow klasy string.**

**35)** Zad.5 15pkt
**Dla klasy class Time(int hour, min, sec), zadeklaruj i zaimplementuj:**
**a) operator ++ (zwiększa czas o jedną sekundę)**
**b) operator <**
**c) Time&operator +=(const Time&)**

**1) Napisz szablon funkcji, która wyznacza sumę elementów tablicy elementów T (paramatr szablonu). Napisz przykład zastosowania po parametryzacji typem std::string.**

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

template<typename T>
T suma(T tab[], int size)
{
   T sum = T(); // zerowanie jakiegokolwiek typu!
   for(int i = 0; i < size; i++) sum += tab[i];
   return sum;
}

int main()
{
   string tab[] = {"Tak ", "to ", "wlasnie ", "dziala "};  // Wszystko w jednym lub po kolei:
         /*string tab[4];
         tab[0] = 'x';
         tab[1] = 'm';
         tab[2] = 'x';
         tab[3] = 'y';*/
   string wyn = suma(tab, 4);
   cout << wyn << endl;

   return 0;
}
```

**2) Mapa (graf) zawiera miasta i drogi. Odcinek łączący (krawędź) można opisać jako (m1,m2,d) gdzie mx to nr miast, a d to droga między nimi (waga kraw.). Wykorzystując STL napisz class Mapa + dodajOdcinek dodajMiasto czy trasa dlugosc wypisz**

```
#include <iostream>
#include <string>
#include <vector>
#include <map>

using namespace std;

class P
{
  public:
  int m1,m2;
  double dlugosc;
  P(const int em1, const int em2,const double d): m1(em1), m2(em2), dlugosc(d) {}

};

class Mapa
{

  public:
  vector<string> miasta;
  vector<P> drogi;
  Mapa(){}
  int dodajMiasto(const char* name)
  {
    for(unsigned int i=0;i<miasta.size();++i)
      if(miasta[i]==name) return -1;

    miasta.push_back(string(name));
    return miasta.size()-1;
  }
  bool dodajOdcinek(int m1, int m2, double d)
  {
    if(miasta.size()<m1 || miasta.size()<m2 || m1<0 || m2<0) return 0;

    for(unsigned int i=0;i<drogi.size();++i)
      if( (drogi[i].m1==m1 && drogi[i].m2==m2) || (drogi[i].m1==m2 && drogi[i].m2==m1) )
        return 0;
    drogi.push_back(P(m1,m2,d));
    return 1;
  }
  bool czyTrasa(const vector<int>& tr)
  {
    bool check=0;
    for(unsigned int i=0;i<tr.size()-1;++i)
    {
       check=0;
       for(unsigned int j=0;j<drogi.size();++j)
       {
         if( (drogi[j].m1==tr[i] && drogi[j].m2==tr[i+1]) || (drogi[j].m2==tr[i] && drogi[j].m1==tr[i+1]) )
           check=1;
       }

       if(!check) return 0;
    }
    return 1;
  }
  double dlugosc(const vector<int>& tr)
  {
    if(!czyTrasa(tr)) return -1;

    double dl=0;
    for(unsigned int i=0;i<tr.size()-1;++i)
    {
       for(unsigned int j=0;j<drogi.size();++j)
       {
         if( (drogi[j].m1==tr[i] && drogi[j].m2==tr[i+1]) || (drogi[j].m2==tr[i] && drogi[j].m1==tr[i+1]) )
           dl+=drogi[j].dlugosc;
       }
    }

    return dl;
  }
  bool wypisz(const vector<int>& tr)
```

```
 {
  if(!czyTrasa(tr)) return -1;

  for(unsigned int i=0;i<tr.size();++i)
 {
  cout<<miasta[tr[i]]<<endl;
 }
return 1;  }};
```

**3) Zad z grafem, analogiczne do miast, z tym że metody: dodajWierzcholek, dodajKrawedz, usunKrawedz, usunKrawedzieWierzch, usunWierzcholek, dlug – długość trasy R1 by forum**

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <queue>
#include <stack>
#include <cstdio>
#include <set>
#include <map>
#include <string>
#include <queue>
#include <stack>
using namespace std;

class Edge
{
   public:
      int start, end;
      double dlug;

      Edge(int u, int v, double d = 0)
      {
         if(u == v || u <= 0 || v <= 0); //
polecenie nie mowi co wtedy zrobic
         if(u > v) swap(u, v);
         start = u;
         end = v;
         dlug = d;
      }
};

bool operator<(const Edge &a, const Edge
&b) // przyda sie do seta
{
   if(a.start == b.start) return a.end <
b.end;
   return a.start < b.start;
}

class Graph
{
   private:
      int v_count; // ile wierzcholkow
      set<int> vertex; // istniejace
wierzcholki
      set<Edge> edges; // istniejace
krawedzie

   public:
      Graph() : v_count(0) {}
      bool dodajWierzcholek(int n);
      bool dodajKrawedz(int s, int e, double
dlug);
      bool usunKrawedz(int s, int e);
      bool usunKrawedzWierzch(int n);
      bool usunWierzcholek(int n);
      bool dlug(double& wynik, const
vector<int>& d);

      // do testowania
      void wierzch() const;
      void kraw() const;
};

bool Graph::dodajWierzcholek(int n)
{
   if(binary_search(vertex.begin(),
vertex.end(), n) || n < 0) return 0;
   vertex.insert(n);
   v_count++;
   return 1;
}

bool Graph::dodajKrawedz(int s, int e,
double dlug)
{
   if(s == e || s < 0 || e < 0) return 0;
   if(!binary_search(vertex.begin(),
vertex.end(), s)) return 0;
   if(!binary_search(vertex.begin(),
vertex.end(), e)) return 0;
   Edge temp = Edge(s, e, dlug);
   if(binary_search(edges.begin(),
edges.end(), temp)) return 0;
   edges.insert(temp);
   return 1;
}

bool Graph::usunKrawedz(int s, int e)
{
   Edge temp(s, e);
   set<Edge>::iterator it =
edges.find(temp);
   if(it == edges.end()) return 0;
   edges.erase(it);
   return 1;
}

bool Graph::usunKrawedzWierzch(int n)
{
   if(!binary_search(vertex.begin(),
vertex.end(), n)) return 0;
   for(set<Edge>::iterator it =
edges.begin(); it != edges.end(); it++)
   {
      if(it->start == n || it->end == n)
      {
         edges.erase(it);
      }
   }
   return 1;
}

bool Graph::usunWierzcholek(int n)
{
   bool ok = usunKrawedzWierzch(n);
   if(!ok) return 0;
   vertex.erase(n);
   v_count--;
   return 1;
}

bool Graph::dlug(double& wynik, const
vector<int>& d)
{
   wynik = 0;
   if(d.size() < 2) return 0;
   for(int i = 1; i < d.size(); i++)
   {
      Edge temp(d[i-1], d[i]);
      set<Edge>::iterator it =
edges.find(temp);
      if(it == edges.end()) return 0;
      wynik += it->dlug;
   }
   return 1;
}

// do testow
void Graph::wierzch() const
{
   cout << "Wierzcholki: " << endl;
   for(set<int>::iterator it =
vertex.begin(); it != vertex.end(); it++)
      cout << *it << ' ';
   cout << endl;
}

void Graph::kraw() const
{
   cout << "Krawedzie: " << endl;
   for(set<Edge>::iterator it =
edges.begin(); it != edges.end(); it++)
      cout << it->start << " -> " << it-
>end << " : " << it->dlug << endl;
   cout << "------" << endl;
}

int main()
{
   ios_base:: sync_with_stdio(0);
   int a, b, c, x;
   double d;
   Graph *G = new Graph;
   while(cin >> x)
   {
      switch(x)
      {
         case 1: // dodaj wierzcholek
            cin >> a;
            cout << G-
>dodajWierzcholek(a) << endl;
            break;

         case 2: // dodaj krawedz
            cin >> a >> b >> d;
            cout << G->dodajKrawedz(a, b,
d) << endl;
            break;

         case 3: // usun krawedz
            cin >> a >> b;
            cout << G->usunKrawedz(a, b)
<< endl;
            break;

         case 4: // usun kraw. do wierzch.
            cin >> a;
            cout << G-
>usunKrawedzWierzch(a)<< endl;
            break;

         case 5: // usun wierzch.
            cin >> a;
            cout << G-
>usunWierzcholek(a)<< endl;
            break;

         case 6: // droga
            {
               cin >> a;
               vector<int> w;
               for(int i = 0; i < a; i++)
               {
                  cin >> b;
                  w.push_back(b);
               }
               cout << G->dlug(d, w)<< "
droga: " << d << endl;
               break;
            }

         case 7: // wypisz wierzch
            G->wierzch();
            break;

         case 8: //wypisz kraw
            G->kraw();
            break;
      }
   }
   delete G;
   system("pause");
   return 0;
}
```

**R2 do grafu** - moje (dziwnie!)

```
#include <iostream>
#include <vector>
#include <string>
#include <cstdio>
#include <deque>

using namespace std;

class Krawedz
{

   public:
   int start,end;
   double dlugosc;

   Krawedz(int _start, int _end, double _dlugosc){
      start=_start;
      end=_end;
      dlugosc=_dlugosc;
   }

}

class Graf {

   int *wierzcholki;
   int i;
   vector<Krawedz> polaczenia;

   public:
   Graf(){i=0;};

   bool dodajWierzch(int n){
      int *tmp;
      tmp=new int[i];
      for (int k=0; k<i;k++)
         tmp[k]=wierzcholki[k];
      delete[]wierzcholki;
      wierzcholki=new int [i+1];
      for (int k=0; k<i;k++)
         wierzcholki[k]=tmp[k];
      delete[]tmp;
      wierzcholki[i+1]=n;
      i++;
      return 1;
      }

   bool dodajKrawedz(int s, int e, double dlug){
   if(s==e || dlug<0) return 0;
   int ok=0;
   for(int g=0;g<i;g++){
      if(wierzcholki[i]==s || wierzcholki[i]==e)
      ok++;}
   if (ok<2) return 0;
   Krawedz tmp(s,e,dlug);
   polaczenia.push_back(tmp);
   return 1;
   }

    bool usunKrawedz(int s, int e){
    for(unsigned d=0;d<polaczenia.size();d++){
    if(( polaczenia[d].start==s &&
polaczenia[d].end==e ) || (polaczenia[d].start==e
&& polaczenia[d].end==s){

      vector<int>::iterator it;
      it=polaczenia.begin();
      polaczenia.erase(it+d);

      return 1;
    }

    }

   }

};
```

4) **Klasy B i C dziedzicza po A{public: virtual~A..}; Klasa ArrayOfA przechwuje w tablicy wskaźniki typu A* do obiektów A,B,C dla których pamieć przydzielana jest na starcie. Zaimplementuj ArrayOfA**
**a) konstruktor (ustala wielkość tablicy), destruktor, operator przypisania, operator kopiujący, operator + (łączenie tablic)**

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

class A
{
   public:
      int x;
      A(int _x = 0) : x(_x) {};
      virtual ~A() {}
};


class B : public A
{

};

class C : public A
{

};

class ArrayOfA
{
   public: // wszystko publicznie - zeby szybciej
napisac...
      int n; // rozmiar tablicy;
      A **tab; // tablica

      ArrayOfA(int _n) : n(_n) // zakladamy n > 0
      {
         tab = new A*[n];
      }
      ~ArrayOfA()
      {
         delete [] tab;
      }
      ArrayOfA(const ArrayOfA& s);
      ArrayOfA operator=(const ArrayOfA& s);
      ArrayOfA operator+(const ArrayOfA& s);
};

ArrayOfA::ArrayOfA(const ArrayOfA& s)
{
   n = s.n;
   tab = new A*[n];
   for(int i = 0; i < n; i++) tab[i] = new
A(*(s.tab[i]));
}

ArrayOfA ArrayOfA::operator=(const ArrayOfA&
s)
{
   if(this == &s) return *this;
   delete [] tab;
   n = s.n;
   tab = new A*[n];
   for(int i = 0; i < n; i++) tab[i] = new
A(*(s.tab[i]));
   return *this;
}

ArrayOfA ArrayOfA::operator+(const ArrayOfA&
s)
{
   A **temp = new A*[n+s.n];
   for(int i = 0; i < n; i++) temp[i] = tab[i];
   for(int i = 0; i < n; i++) temp[i+n] = s.tab[i];
   n += s.n;
   delete [] tab;
   tab = new A*[n];
   for(int i = 0; i < n; i++) tab[i] = temp[i];
   delete [] temp;
   return *this;
}

int main()
{
   A *a1 = new A();
   A *a2 = new A(2);
   ArrayOfA t1(2);
   t1.tab[0] = a1; t1.tab[1] = a2;
   cout << t1.tab[0]->x << endl;
   cout << t1.tab[1]->x << endl;
   cout << "---" << endl;
   ArrayOfA t2(100);
   t2 = t1;
   t2.tab[0]->x = 5;
   t2.tab[1]->x = 111;
   cout << t2.tab[0]->x << endl;
   cout << t2.tab[1]->x << endl;
   cout << t1.tab[0]->x << endl;
   cout << t1.tab[1]->x << endl;
   ArrayOfA t3(1);
   t3 = t1+t2;
   cout << "---" << endl;
   for(int i = 0; i < 4; i++) cout << t3.tab[i]->x
<< endl;
   system("pause");
   return 0;
}
```

**5) Bez STL;**
**Zdefiniuj szablon będący implementacją IQueue<T> (kolejka FIFO)**

```
template <class T> class IQueue
{
   public:
   virtual bool put (const T&t) = 0;
   virtual bool get (T &t) = 0;
};
```

**Napisz konstruktor ustalający rozmiar kolejki oraz metody put (na koniec) get( zwraca pierwszy el, przesówa pozostałe)**

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

template <class T>
class IQueue
{
   public:
   virtual bool put (const T&t) = 0;
   virtual bool get (T &t) = 0;
};

template <class T>
class Fifo : public IQueue<T>
{
   public:
   //private:
   T* queue;
   int last;
   int size;
   public:
   Fifo(int size) {
      queue = new T [size];
      this->size = size;
      last = 0;
   }

   bool put(const T& t) {
      if (last < size) {
         queue[last] = t;
         last++;
         return true;
      } else {
         return false;
      }
   }

   bool get(T& t) {
      if (!last) {
         return false;
      } else {
         t = queue[0];
         for (int i = 0; i < size - 1; i++) {
            queue[i] = queue[i + 1];
         }
         queue[last] = NULL;
         last--;
         return true;
      }
   }
};
#endif // KLASA_H_INCLUDED
```

```
/// main

#include <iostream>
#include "klasa.h"

using namespace std;

int main()
{
   Fifo<int> q(4);

   q.put(3);
   q.put(4);
   q.put(5);
   q.put(3);
   q.put(3);
   q.put(3);

   //cout << q.last << endl;

   for (int i = 0; i < 6; i++) {
      cout << q.queue[i] << endl;
   }

   int x;
   q.get(x);

   cout << x << endl;
   return 0;
}
```

**6) Użyj STL. Klasy B, C i D dziedzicza po class A**
**Klasa Alist dziedziczy po  list<A*> i może przechowywać obiekty klas A,B,C i D. Napisz klasę AList:**
**a) deklaracja, konstruktor, destruktor, operator przypisania, konstruktor kopijący, oprator +**

```
#include <iostream>
#include <list>
#include <typeinfo>

using namespace std;

class A {
public:
  virtual ~A() {}
};

class B : public A {
public:
  B() {}
  ~B() {}
  void wypisz() {
    cout << "B" << endl;
  }
};

class C : public A {
public:
  C() {}
  ~C() {}
  void wypisz() {
    cout << "C" << endl;
  }
};

class D : public A {
public:
  D() {}
  ~D() {}
  void wypisz() {
    cout << "D" << endl;
  }
};

class AList : public list<A*> {
public:
  AList() {}
  ~AList() {
    this->clear();
  }
  AList(const AList& input) {
    *this = input;
  }
  AList& operator=(const AList& input) {
    if (this != &input) {
      this->~AList();
      *this = input;
    }
    return *this;
  }
  AList operator+(const AList& input) {
    AList::const_iterator it;
    AList tmp(*this);

    for (it = input.begin(); it != input.end(); ++it) {
      tmp.push_back(*it);
    }

    return tmp;}};

int main()
{
    AList a;

    a.push_back(new B);
    a.push_back(new C);

    list<A*>::const_iterator it;

    for (it = a.begin(); it != a.end(); ++it) {
      if (typeid(**it) == typeid(B)) {
        (dynamic_cast<B*>(*it))->wypisz();
      }
      if (typeid(**it) == typeid(C)) {
        (dynamic_cast<C*>(*it))->wypisz();
      }
    }
    return 0;
}
```

**7)Użyj STL. Klasa Directory zawiera dowolną liczbę obiektów klasy File lub Directory. Obie klasy dziedxziczą po DirElement. File i Directory mają swoją nazwe. Do przechowywania obiektów wybierz dowoly kontener, w którym umieszczone są wskażźniki typu DirElement* a) do Director dodaj bool add(DirElement*) - dodaj elementy unikalnych nazw - bool delete(const char*name)**

```
#include <iostream>
#include <list>
#include <vector>
#include <cstdio>

struct DirElem {
   typedef std::vector<DirElem*>::const_iterator VCI;
   virtual unsigned getType() const =0;
   std::string getName() const { return _name; }
   void setName(std::string name) { _name = name; }
   virtual ~DirElem() {}
   private:
      std::string _name;

};

struct File : public DirElem {
   File() {}
   unsigned getType() const { return 1; }
};

struct Directory : public DirElem {
   unsigned getType() const { return 2; }
   Directory() {}
   Directory(const Directory& dir);
   void add(const DirElem& e);
   void find(const char* key, std::list<std::string>& target)
const;
   void dump() const;
   private:
      std::vector<DirElem*> _dir;
};

using namespace std;

Directory::Directory(const Directory& rhs) {
   for(VCI i = rhs._dir.begin(); i != rhs._dir.end(); ++i)
      add(**i);
}

void Directory::add(const DirElem& e) {
   DirElem& el = const_cast<DirElem&>(e);
   if(e.getType() == 1) {
      _dir.push_back(new File(dynamic_cast<File&>(el)));
   } else if(e.getType() == 2) {
      Directory* d = new
Directory(dynamic_cast<Directory&>(el));
      d->setName(e.getName());
      _dir.push_back(d);
   }
}

void Directory::find(const char* key, list<string>& target)
const {
   for(VCI i = _dir.begin(); i != _dir.end(); ++i) {
      string name = (*i)->getName();
      if(name == key)
         target.push_back(name);
      if((*i)->getType() == 2)
         (dynamic_cast<Directory*>(*i))->find(key, target);
   }
}

void Directory::dump() const {
   for(VCI i = _dir.begin(); i != _dir.end(); ++i) {
      cout << (*i)->getName();
      if((*i)->getType() == 2) {
         cout << "<dir>: [ ";
         (dynamic_cast<Directory*>(*i))->dump();
         cout << " ]";
      }
      cout << ", ";
   }
}
```

**8)Klasa MapTextInt przechowuje wskaźnik do tekstów, dla których pamięć jest przechowywana na stercie oraz ich liczbę, czyli paty (char*, int). Zaimplementuj w postaci listy jednokierunkowej**
**Deklaracja struktur danych, konstruktor, funkcja void add(const char*text), destruktor, konstruktor kopiujacy, operator przypisania**

```
#include <iostream>
#include <list>
#include <typeinfo>

using namespace std;

class Txt{

public:

 char *txt;
 int i;
 Txt *next;

 Txt(char *_txt){
 int g=sizeof(_txt)/sizeof(char);
 txt=new char[g];
 for(int m=0; m<g;m++){
   txt[m]=_txt[m];
 }
 i=1;
 next=NULL;
 }

};

class MapTxtInt {

   private:
   Txt*head;

   public:
   MapTxtInt(){head=NULL;}
   void add(const char*text){
   if(head==NULL){
      head=&Txt(text);

   }else{
      Txt *tmp;
      tmp=head;
      while(tmp->next!=NULL || tmp->txt==text){
      tmp=tmp->next;
      }
      if(tmp->txt==text)
         tmp->i++;
      else
         tmp->next=&Txt(text);
   }

   }

   ~MapTxtInt()
   {
      Txt *tmp1, *tmp2;
      tmp1=head;
      while(tmp1!=NULL)
      {
         tmp2=tmp1;
         tmp1=tmp1->next;
         delete [] tmp2;

      }

   }
};
```

**9) [ Zadanie 1 25pkt]**

**//Wykorzystujac kontener STL vector napisz klase VectorOfIntArray przechowywujaca**
**//tablice elementow typu int o zmiennej dlugosci.(Potraktujmy je jak wiersze dwuwymiarowej**
**//tablicy)Klasa zarzadza pamiecia wierszy,JAWNIE ALOKUJE JA I ZWALNIA.Dla wierszy**
**//nie jest stosowany kontener STL!**
**//Metody:**
**// a) void add(int*tab,int size)-dodaje wiersz,przydziela pamiec,kopiuje elementy**
**// b) int getSize(int rowidx)-zwraca liczbe elementow wiersza o indeksie rowidx**
**// c) int*get (int rowidx)-zwraca wskaznik do elementow wiersza rowidx**
**// d) destruktor**
**// e) konstruktor kopiujacy**
**//----------**

```
class VectorOfIntArray
{
private:
vector<pair<int *, int> > tabl;
public:
VectorOfIntArray(void) {}
void add(int *, int);
int getSize(int) const;
int * get(int) const;
~VectorOfIntArray(void);
VectorOfIntArray(VectorOfIntArray const
&);
};
void VectorOfIntArray::add(int * tab, int
size)
{
pair<int *, int> nowy;
nowy.first = new int[size];
for (int i = 0 ; i < size ; ++i)
nowy.first[i] = tab[i];
nowy.second = size;
tabl.push_back(nowy);
}
int VectorOfIntArray::getSize(int rowidx)
const
{
return tabl[rowidx].second;
}
int * VectorOfIntArray::get(int rowidx)
const
{
return tabl[rowidx].first;
}
VectorOfIntArray::~VectorOfIntArray(void
)
{
for (size_t i = 0 ; i < tabl.size() ; ++i)
delete [] tabl[i].first;
}
VectorOfIntArray::VectorOfIntArray(Vecto
rOfIntArray const & drugi)
{
for (size_t i = 0 ; i < drugi.tabl.size() ; +
+i)
add(drugi.tabl[i].first,
drugi.tabl[i].second);
}
```

**10) [ Zadanie 2 25pkt]**

**Bez STL. Napisz szablon template <class T>class Array<T>{ ...}klasy przechowywujacych elementy typu T w tablicy, ktora moze dynamicznie przyrastac.**
**// a)zadeklaruj klase oraz podaj konstruktor**
**// b)zaimplementuj destruktor**
**// c)funkcja pushBack(const &t)**
**// d)funkcje bool reserve(int size), ktora bedzie przedluzala tablice do zadanego**
**// rozmiaru**
**// e)operator[]**

```
template <class T>
class Array
{
private:
T * tab;
int rozm;
public:
Array(void) : tab(NULL), rozm(0) {}
~Array(void);
void pushBack(T const &);
bool reserve(int);
T & operator[](unsigned) const;
int Rozm(void) const; // do 3. zadania
};
template <class T>
Array<T>::~Array(void)
{
delete [] tab;
}
template <class T>
void Array<T>::pushBack(T const & t)
{
reserve(rozm + 1);
tab[rozm - 1] = t;
}
template <class T>
bool Array<T>::reserve(int size)
{
if (size > rozm)
{
T * pom = new T[size];
for (int i = 0 ; i < rozm ; ++i)
pom[i] = tab[i];
delete [] tab;
tab = pom;
rozm = size;
return true;
}
else
return false;
}
template <class T>
T & Array<T>::operator[](unsigned i)
const
{
return tab[i];
}
template <class T>
int Array<T>::Rozm(void) const
{
return rozm;
}
```

**11) [ Zadanie 3 20pkt]**
**Wykorzystaj kontener z zad.2 UnaryPredicate to klasa reprezentujaca jednoargumentowa funkcje zwracajaca bool.**
**//a)napisz szablon zewnetrznej funkcji copyIf()parametryzowanej typami T i UnaryPredicate**
**// template <classT, class UnaryPredicate> void copyIf(..**
**//umozliwiajacej skopiowanie z Array<T> src do Array<T> target wszystkich elementow**
**//spelniajacych predykat przekazany,jako obiekt funkcyjny klasy UnaryPredicate.**
**//b)Napisz klase dla obiektu funkcyjnego umozliwiajaca testowanie, czy liczba calkowita jest wieksza od 0 i mniejsza od 10**
**//c)Wywolaj dla tablic parametryzowanych typem int i utworzonego obiektu funkcyjnego**

```
class Predykat
{
public:
bool operator()(int) const;
};
bool Predykat::operator()(int liczba)
const
{
return liczba > 0 && liczba < 10;
}
```

```
template <class T, class
UnaryPredicate>
void copyIf(Array<T> const & src,
Array<T> & target, UnaryPredicate p)
{
for (int i = 0 ; i < src.Rozm() ; ++i)
if (p(src[i]))
target.pushBack(src[i]);
}
```

**12) [ Zadanie 4 ]**

**Klasa Iterator ma:**
**// a)konstruktor**
**// b)metode get() ,ktora zwraca biezacy element**
**// c)metode void next() , ktora przesuwa iterator na nastepny element,**
**// d)metode bool good(), ktora zwraca true, jesli stan iteratora pozwala na**
**// poprawny odczyt elementu.**
**Napisz iterator, ktory bedzie w konstruktorze otrzymywal nazwe pliku tekstowego, otwieral go,a nastepnie wydzielal z niego niepuste linie i zwracal je w postaci obiektow klasy string.**

```
class Iterator
{
private:
ifstream plik;
public:
Iterator(string);
string get(void);
void next(void);
bool good(void);
};
Iterator::Iterator(string nazwa)
{
plik.open(nazwa.c_str());
}
string Iterator::get(void)
{
string linia;
streampos pozycja;
do
{
pozycja = plik.tellg();
getline(plik, linia);
}
while (linia == "");
plik.seekg(pozycja);
return linia;
}
void Iterator::next(void)
{
string slowo;
getline(plik, slowo);
}
bool Iterator::good(void)
{
return plik && plik.peek() != EOF;
}
```

**13) [ Zadanie 5 15pkt]**

**Dla klasy struct Point(double tab[3]); zadeklaruj i zaimplementuj**
**a)operator+**
**b)operator==**
**c)operator<< umozliwiajacy zapis do strumienia ostream.**

```
struct Point
{
double tab[3];
Point operator+(Point const &);
bool operator==(Point const &);
};
Point Point::operator+(Point const &
drugi)
{
Point p;
p.tab[0] = tab[0] + drugi.tab[0];
p.tab[1] = tab[1] + drugi.tab[1];
p.tab[2] = tab[2] + drugi.tab[2];
return p;
}
bool Point::operator==(Point const &
drugi)
{
return tab[0] == drugi.tab[0]
&& tab[1] == drugi.tab[1]
&& tab[2] == drugi.tab[2];
}
ostream & operator<<(ostream & s,
Point const & p)
{
return s << "(" << p.tab[0] << ", " <<
p.tab[1]
<< ", " << p.tab[2] << ")";
}
// ------------------------------ [ Koniec ]
---------------------------------
int main(void)
{
// zadanie 3.
Array<int> a, b;
Predykat p;
copyIf<int>(a, b, p);
// koniec zadania 3.
return 0; // O tym nie wolno
zapomnieć. :]
```

}



```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <locale>
using namespace std;
```

**14) [ Zadanie 1 25pkt]**
**Wykorzystując kontener STL list lub vector zaimplementuj klasę String2String reprezentującą/relację string x string, czyli przechowującą unikalne pary tekstów.**
**Metody:**
**//a) void add(const char*a, const char *b) – dodaje parę (a,b)**
**//b) void remove(const char*a, const char*b) – usuwa parę (a,b)**
**//c) int find(const char* a, int start) wyszukuje parę, której pierwszym elementem jest a,**
**//rozpoczyna wyszukiwanie od indeksu start. Zwraca -1, jeśli takiego elementu brak**
**//d) void get(const char*a, String2String&target) – dodaje do target wszystkie pary, których**
**//pierwszym elementem jest a**
**//e) oprator +=**

```
class String2String
{
private:
vector<pair<string, string> > tab;
public:
void add(const char *, const char *);
void remove(const char *, const char *);
int find(const char *, int) const;
void get(const char *, String2String &)
const;
String2String &
operator+=(String2String const &);
};
void String2String::add(const char * a,
const char * b)
{
bool jest = false;
for (size_t i = 0 ; i < tab.size() ; ++i)
if (a == tab[i].first && b ==
tab[i].second)
{
jest = true;
i = tab.size();
}
if (!jest)
tab.push_back(pair<string, string>(a,
b));
}
void String2String::remove(const char *
a, const char * b)
{
for (size_t i = 0 ; i < tab.size() ; ++i)
if (a == tab[i].first && b ==
tab[i].second)
{
tab.erase(tab.begin() + i);
return;
}
}
int String2String::find(const char * a, int
start) const
{
for (size_t i = start ; i < tab.size() ; ++i)
if (a == tab[i].first)
return i;
return -1;
}
void String2String::get(const char * a,
String2String & target) const
{
for (size_t i = 0 ; i < tab.size() ; ++i)
if (a == tab[i].first)
target.add(tab[i].first.c_str(),
tab[i].second.c_str());
}
String2String &
String2String::operator+=(String2String
const & drugi)
{
for (size_t i = 0 ; i <drugi.tab.size() ; +
+i)
add(drugi.tab[i].first.c_str(),
drugi.tab[i].second.c_str());
return *this;
}
```

**15) [ Zadanie 2 25pkt]**

**Bez STL. Napisz szablon template <class T >class List<T>{ …} klasy przechowującej element typu T na liście jednokierunkowej.**
**//a) Zadeklaruj klasę i pomocnicze struktury danych oraz zaimplementuj konstruktor**
**//b) zaimplementuj destruktor**
**//c) funkcję pushFront(const T&t)**
**//d) funkcję bool getLast(T&t), która wartość elementu umieszcza w t**
**//e) operator przypisania**

```
template <class T>
class List
{
protected:
struct wezel
{
wezel * nast;
T element;
};
wezel * poczatek;
void czysc(void);
public:
List(void) : poczatek(NULL) {}
~List(void);
void pushFront(T const &);
bool getLast(T &) const;
List & operator=(List const &);
};
template <class T>
void List<T>::czysc(void)
{
wezel * pom;
while (poczatek)
{
pom = poczatek;
poczatek = poczatek->nast;
delete pom;
}
}
template <class T>
List<T>::~List(void)
{
czysc();
}
template <class T>
void List<T>::pushFront(T const & t)
{
wezel * nowy = new wezel;
nowy->element = t;
nowy->nast = poczatek;
poczatek = nowy;
}
template <class T>
bool List<T>::getLast(T & t) const
{
if (!poczatek)
return false;
wezel * pom = poczatek;
while (pom->nast)
pom = pom->nast;
t = pom->element;
return true;
}
template <class T>
List<T> & List<T>::operator=(List const
& drugi)
{
if (poczatek == drugi.poczatek)
return *this;
czysc();
wezel * dr;
wezel * pom;
if (drugi.poczatek)
{
poczatek = new wezel;
poczatek->element = drugi.poczatek-
>element;
dr = drugi.poczatek->nast;
pom = poczatek;
}
while (dr)
{
pom->nast = new wezel;
pom = pom->nast;
pom->element = dr->element;
dr = dr->nast;
}
pom->nast = NULL;
return *this;
}
```

**16) [ Zadanie 3 15pkt]**

**Szablon klasy zapewniającej standardową funkcję do porównywania jest zdefiniowany jako:**
**template <class T> class DefaultComparator**
**//{**
**//public:**
**//static bool compare(const T&a, const T&b){return a == b;}**
**//};**
**Wykorzystując funkcjonalność listy z zad.2 napisz szablon zbioru:**
**template <class T, class Comparator = DefaultComparator<T> >**
**class Set …**
**Z metodami (a)insert(const T&t) dodaje unikalny element do zbioru (b) metodę Bool has(const T&) sprawdzającą czy element należy do zbioru(c) operator < sprawdzający inkluzję zbiorów.**

```
template <class T>
class DefaultComparator
{
public:
static bool compare(T const & a, T const
& b)
{
return a == b;
}
};
template <class T, class Comparator =
DefaultComparator<T> >
class Set : public List<T>
{
public:
void insert(T const &);
bool has(T const &) const;
bool operator<(Set<T> const &) const;
};
template <class T, class Comparator>
void Set<T, Comparator>::insert(const T
& t)
{
if (!has(t))
pushFront(t);
}
template <class T, class Comparator>
bool Set<T, Comparator>::has(T const &
t) const
{
bool jest = false;
typename List<T>::wezel * pom =
List<T>::poczatek;
while (pom && !jest)
{
if (Comparator::compare(t, pom-
>element))
jest = true;
pom = pom->nast;
}
return jest;
}
template <class T, class Comparator>
bool Set<T,
Comparator>::operator<(Set<T> const
& drugi) const
{
bool zawiera = true;
typename List<T>::wezel * pom =
List<T>::poczatek;
while (pom && zawiera)
{
if (!drugi.has(pom->element))
zawiera = false;
pom = pom->nast;
}
return zawiera;
}
```

**17) [ Zadanie 4 ]**

**Klasa Iterator ma (a) konstruktor (b) metodę get(), która zwraca bieżący element, (c) metodę void next(), która przesuwa iterator na następny element, (d) metodę bool good(), która zwraca true, jeśli stan iteratora pozwala na poprawny odczyt elementu. Napisz iterator, który będzie w konstruktorze otrzymywał nazwę pliku tekstowego, otwierał go, a następnie wydzielał z niego wyłącznie litery i cyfry, zastępując wszystkie pozostałe znaki pojedynczym znakiem spacji.**

```
class Iterator
{
private:
ifstream plik;
public:
Iterator(string);
char get(void);
void next(void);
bool good(void);
};
Iterator::Iterator(string nazwa)
{
plik.open(nazwa.c_str());
}
char Iterator::get(void)
{
char znak = plik.peek();
if (!((znak >= '0' && znak <= '9')
|| (znak >= 'A' && znak <= 'Z')
|| (znak >= 'a' && znak <= 'z')))
znak = ' ';
return znak;
}
void Iterator::next(void)
{
plik.ignore();
}
bool Iterator::good(void)
{
return plik && plik.peek() != EOF;
}
```

**18) [ Zadanie 5 ]**

**Dla klasy class Ulamek(int licznik; int mianownik); zadeklaruj i zaimplementuj:**
**a) Ulamek& operator += (const Ulamek&)**
**b) operator ==**
**//c) operator >> umożliwiający odczyt ze strumienia istream**

```
class Ulamek
{
private:
int licznik;
int mianownik;
public:
Ulamek & operator+=(Ulamek const &);
bool operator==(Ulamek const &) const;
friend istream & operator>>(istream &,
Ulamek &);
};
Ulamek & Ulamek::operator+=(Ulamek
const & drugi)
{
int mniejszy;
licznik = licznik * drugi.mianownik +
drugi.licznik * mianownik;
mianownik = mianownik *
drugi.mianownik;
if (licznik < mianownik)
mniejszy = licznik;
else
mniejszy = mianownik;
for (int i = mniejszy ; i > 0 ; --i)
if (licznik % i == 0 && mianownik % i ==
0)
{
licznik /= i;
mianownik /= i;
}
return *this;
}
bool Ulamek::operator==(Ulamek const
& drugi) const
{
return licznik == drugi.licznik &&
mianownik == drugi.mianownik;
}
istream & operator>>(istream & s,
Ulamek & u)
{
return s >> u.licznik >> u.mianownik;
}
// ------------------------------ [ Koniec ]
---------------------------------
int main(void)
{
// Pusto. :P
return 0; // Ale o tym nie wolno
zapomnieć. :]
}
```

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
using namespace std;
```

**19) [ Zadanie 1 20pkt]**
**Napisz szablon template <class T>class List<T>{...} klasy przechowywujacych elementy typu T na liscie dwukierunkowej**
**// a)Zadeklaruj klase i pomocnicze struktury danych**
**// b)zaimplementuj konstruktor**
**// c)destruktor**
**// d)funkcje pushBack(const T&t)**

```
template<class T>
class List
{
public:
struct wezel
{ wezel* nast;
wezel*poprz;
T element;
};
private:
wezel* poczatek;
wezel* koniec;
public:
// b)zaimplementuj konstruktor
List(void):poczatek(NULL),koniec(NULL)
{};
~List(void);
wezel* Poczatek(void) const;
void pushBack(T const &);
};
template <class T>
typename List<T>::wezel *
List<T>::Poczatek(void) const
{
return poczatek;
}
// c)destruktor
template <class T>
List<T>::~List(void)
{
wezel * pom;
while (poczatek)
{
pom = poczatek;
poczatek = poczatek->nast;
delete pom;
}
}
// d)funkcje pushBack(const T&t)
template <class T>
void List<T>::pushBack(T const & t)
{
if (!poczatek)
{
poczatek = new wezel;
poczatek->element = t;
poczatek->poprz = NULL;
poczatek->nast = NULL;
koniec = poczatek;
}
else
{
koniec->nast = new wezel;
koniec->nast->element = t;
koniec->nast->poprz = NULL;
koniec->nast->nast = NULL;
koniec = koniec->nast;
}
}
```

**20) ----------[ Zadanie 2 25pkt]----------**
**Zaimplementuj klase Iterator pozwalajaca na wydzieleniemz tekstu slow(ciagow znakow oddzielonych bialymi znakami) z metodami:**
**// a) Iterator(const char*) konstruktor, jego parametrem**
**// jest analizowany tekst**
**// b) bool haveMoreWords() - czy jest jeszcze jakies slowo?**
**// c) void nextWord() -wyszukuje nastepne slowo i przesuwa**
**// iterator**
**// d) const char*get() -zwraca biezace slowo**
**// e) podaj przyklad wywolania**

```
class Iterator
{
private:
stringstream strumien;
public:
Iterator(const char *);
string get(void);
void next(void);
bool haveMoreWords(void);
};
// a) Iterator(const char*) konstruktor,
jego parametrem
// jest analizowany tekst
Iterator::Iterator(const char * t)
{
strumien << t;
}
// d) const char*get() -zwraca biezace
slowo
string Iterator::get(void)
{
string slowo;
streampos pozycja;
pozycja = strumien.tellg();
strumien >> slowo;
strumien.seekg(pozycja);
return slowo;
}
// c) void nextWord() -wyszukuje
nastepne slowo i przesuwa
// iterator
void Iterator::next(void)
{
string slowo;
strumien >> slowo;
}
// b) bool haveMoreWords() - czy jest
jeszcze jakies slowo?
bool Iterator::haveMoreWords(void)
{
return strumien.peek() != EOF;
}
```

----------[ Zadanie 3 25pkt]----------
Korzystajac z kontenera STL vector zaimplementuj klase StringVector przechowywujaca wskazniki
typu char* .Pamiec dla tekstow jest przydzielana na stercie. Zaimplementuj:
// a)konstruktor
// b)konstruktor kopiujacy
// c)destruktor
// d)operator przypisania
// e)funkcje void add(const char*txt)

**-21) ----------[ Zadanie 4 15pkt]----------**
**Dla klasy class Point{double x,y;}zadeklaruj i zaimplementuj**
**// a) operator+**
**// b) operator-=**
**// c) operator==**

```
class Point
{
private:
double x,y;
public:
double getx(){return x;};
double gety(){return y;};
void set(double,double);
Point operator+(Point);
Point operator-=(Point);
bool operator==(Point);
};
// a) operator+
Point Point::operator+(Point drugi)
{
Point suma;
suma.set((*this).getx()+drugi.getx(),
(*this).gety()+drugi.gety());
return suma;
}
// b) operator-=
Point Point::operator-=(Point odjemna)
{
(*this).set((*this).getx()-odjemna.getx(),
(*this).gety()-odjemna.gety());
return *this;
}
// c) operator==
bool Point::operator==(Point drugi)
{
if ( ((*this).getx()==drugi.getx() ) &&
((*this).gety()==drugi.gety()) )
return true;
else
return false;
}
```

**22) //----------[ Zadanie 6 10pkt]----------**
**//Co zostanie wypisane i dlaczego?**

Wypisze A.fA.fB.fM w punkcie 1) wypisze sie A.f potem w 2) konstruktor B() wypisze A.fB.f i ostatecznie w 3) wypisze sie M

```
class A
{
public:
virtual void f()
{cout <<"A.f";}
A(){f();}
};
```

```
class B
{
A a;
public:
void f()
{cout <<"B.f";}
B(){f();}
};
A a;
```

```
int main()
{
B b; //2)
cout <<"M"; //3)
//estetycznie dodam
pusta linie i pauze
cout <<"\n";
system("pause");
return 0;
}
```

**23) a) Zadeklaruj szablon template <class T> class Array<T> {} klasy bedącej tablica elementów T**
**b) Zaimplementuj konstruktor Array<T> (int size) ustalający rozmiar tab i alokujacy pamięć**
**c) destruktor zwalniajacy pamięć oraz operator Array<T>&operator+=(const Array<T>&other) rozszerzający tablicę o elementy argumentu other**

```
#include <cstdlib>
#include <iostream>
using namespace std;

template<class T>
class Array{

    public:
      T   *tab;
      int size;

      Array():size(0), tab(0){};
      Array(int s){
                size=s;
                tab = new T[size];

      };

      ~Array(){
                if(tab)
                delete []tab;
                size=0;
                }


Array & operator+=(Array& other){

      if(tab)
      {

              Array<T>
tablica(size+other.size);


              for(int i=0;i<size; i++)
              {

 tablica.tab[i]=tab[i];
              }

               for(int i=0;i<other.size; i+
+)
              {

tablica.tab[size+i]=other.tab[i];
              }

              size=tablica.size;
              tab = new T(size);

              for(int i=0;i<size; i++)
              {

tab[i]=tablica.tab[i];
              }

              return *this;
}  }

};

int main(int argc, char *argv[])
{

    Array<int> tablica1(5);

    for(int i=0;i<5; i++)
              {
                       tablica1.tab[i]=i;

              cout <<
tablica1.tab[i]<<endl;

                       }
                        cout << "size:"<<
tablica1.size<<endl;


      Array<int> tablica2(3);

              for(int i=0;i<3; i++)
              {
                       tablica2.tab[i]=i;
                        cout <<
tablica2.tab[i]<<endl;

              }
         cout << "size:"<<
tablica2.size<<endl;


       tablica1+=tablica2;

        cout << "po scaleniu.."<<endl;
        cout << "size:"<<
tablica1.size<<endl;

        for(int i=0;i<tablica1.size; i++)
              {


              cout <<
tablica1.tab[i]<<endl;
              }

    return EXIT_SUCCESS;
}
```

**24)Klasa FileIterator pozwala odczytać znaki z pliku traktowanego jak kontener. Zadeklaruj i zaimplementuj:**
**a) konstruktor FileIterator(const char*name)**
**b) destruktor klasy FileIterator**
**c) andEnd() - czy iterator sosiągnął koniec kontenera?**
**d) next() - przesuwanie iteratora na anstępny element (znak)**
**e) int get() - zwraca aktualny element**

```
#include <cstdlib>
#include <iostream>
#include <fstream>
using namespace std;

class FileIterator{

public:
      ifstream f;
      char* name;
      int pos;

      FileIterator(char* name):name(name)
      {

       f.open(name,ios::in);
       pos=1;
      }

      ~FileIterator()
      {
            f.close();
            delete name;
            pos=0;
      }


bool  atEnd(){
                  return f.eof();
         }
```

```
void next(){

     ++pos;
     f.seekg( pos );
     }

int get(){

      return f.get();


      }
};
int main(int argc, char *argv[])
{


    FileIterator file("test.txt");

    while(!file.atEnd())
    {

           cout << file.get()<< " = " ;
           cout << (char)file.f.get()<<
endl;
           file.next();

    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

**25) Korzystając z kontenera STL list zaimplementuj klasę PersonSet przechopwującą unikalne ze względu na wartości atrybuty PESEL instancje obiektów klasy class Person(....)**
**Zaimplementuj:**
**a) void insert(..), bool remove(...) bool exist (..) [sprawdza czy osoba jest w kontenerze] void select [dodaje do target wszystkie obiekty zawierające łańcuch txtx w jednym z atrybutów**
**operator + dodający dwa zbiory do siebbie**

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


class Person {

   public:
      string name;
      string surname;
      string pesel;

       Person(char* name, const char* surname,
const  char* pesel)
                  :name(name),
surname(surname), pesel(pesel){};

      };

class PersonSet {

public:
      vector<Person> data;
```

```
void     insert(char* name, const char* surname,
const char* pesel){

         Person pr(name, surname, pesel);

         vector<Person>::iterator
it=data.begin();
         for( ; it!=data.end(); it++)
         {
              if((*it).pesel==pr.pesel)
              {
               (*it).name=pr.name;
               (*it).surname=pr.surname;
               }
         }
         data.push_back(pr);
       }

void     insert(Person pr){

         vector<Person>::iterator
it=data.begin();
         for( ; it!=data.end(); it++)
         {
              if((*it).pesel==pr.pesel)
```

```
              {
               (*it).name=pr.name;
               (*it).surname=pr.surname;
              return;
               }

         }
         data.push_back(pr);
       }

bool exist(const char* PESEL){

     vector<Person>::iterator it=data.begin();
         for( ; it!=data.end(); it++)
         {
              if((*it).pesel==PESEL)
              return true;
         }
         return false;
}

void select(PersonSet& target, const char* text){


     vector<Person>::iterator it=data.begin();
         for( ; it!=data.end(); it++)
         {
              if (  ( (*it).pesel==text ) or
( (*it).surname==text ) or ( (*it).name==text) )
              target.data.push_back((*it));

         }

     }

PersonSet& operator+(PersonSet& other){

vector<Person>::iterator it=other.data.begin();
         for( ; it!=other.data.end(); it++)
         {
                insert((*it));
         }
         return *this;
}

};


int main(int argc, char *argv[])
{
    PersonSet katalog;


katalog.insert("jasiu","kowalski","2211134");
    katalog.insert("tomek","ciupaga","1111111");
    katalog.insert("wacek","waga","5151018");
    katalog.insert("igor","kowalski","33333333");

    if(katalog.exist("2211134")) cout << "2211134
jest w bazie" << endl;
    if(katalog.exist("0000000")) cout << "2211134
jest w bazie" << endl;

    PersonSet  maly_kat;
    katalog.select(maly_kat , "kowalski");

     cout << "maly katalog:" ;

     vector<Person>::iterator
it=maly_kat.data.begin();
         for( ; it!=maly_kat.data.end(); it++)
         {
              cout << (*it).name << ", " ;
         }

    maly_kat.insert("kasia","mrozek","6669999");


    PersonSet SCALENIE=katalog + maly_kat;

    cout <<endl<< "SCALENIE:" <<endl;

    it=SCALENIE.data.begin();
         for( ; it!=SCALENIE.data.end(); it++)
         {
              cout << (*it).name << ", " ;
         }


    system("PAUSE");
    return EXIT_SUCCESS;
}
```

**26) Graf nieskierowany zawiera pewna liczbę ponuerowanych wierzvchołków oraz pwną liczbę krawedzi.**
**Przechowywane są one w kontenerach STL. Krawedzie są reprezentowane przez trójki (start, end, dlug) gdzie start,end to numery wierzchołków. Parametr dlug jest waga krawedz. Z wierzchołka moze wychodzic wiecej niz jedna krawedz, dwa wierzcholki moga byc polaczone tylko jedna krawedzia. Def krawedzi (start, end, w) - kolejnosc nie ma znaczenia. Droga w grafie jest sekwencja wierzcholkow, z których kazde dwa kolejne są połaczone krawedziami.**

**a) Zadeklaruj klaszę Graf + konstruktor.**
**metody: bool dodajWIerzcholek(int n), dodajKrawdz, usunKrawedz,usunKrawWierzch, usunWierzcholek, dlug - oblicza długosc drogi, zwraca false jesli d nie jest droga**

```
class Krawedz
{
    public:
            int start,end;
            double dlugosc;
            bool operator==(Krawedz kr)
            {
                if(this->start==kr.start &&
this->end==kr.end){return true;}
                else if(this->end==kr.start
&& this->start==kr.end){return true;}
                else {return false;}
            }
            Krawedz(int s=0,int e=0, double
dl=0):start(s),end(e),dlugosc(dl){}
};

class Graf
{
    public:
            int ilosc_w;
            std::vector<int> wierzcholki;
            std::vector<Krawedz> krawedzie;

            Graf():ilosc_w(0){}

            bool dodajWierzcholek(int n)
            {

if(wierzcholki.end()==find(wierzcholki.begi
n(),wierzcholki.end(),n))
                {

wierzcholki.push_back(n);
                }
                else{return false;}
                return true;
            }

            bool dodajKrawedz(int s, int e,
int dl)
            {
                Krawedz kr(s,e,dl);

if(krawedzie.end()==find(krawedzie.begin(),
krawedzie.end(),kr))
                {

krawedzie.push_back(kr);
                }
                else{return false;}
                return true;
            }

            bool usunKrawedz(int s,int e)
            {
                Krawedz kr(s,e,0);

std::vector<Krawedz>::iterator it =
find(krawedzie.begin(),krawedzie.end(),kr);
                if(it==krawedzie.end())
{return false;}
                else{krawedzie.erase(it);}
                return true;
            }

            bool usunWierzcholek(int n)
            {
                std::vector<int>::iterator
p=find(wierzcholki.begin(),wierzcholki.end(
),n);
                if(p!=wierzcholki.end())
                {
                    wierzcholki.erase(p);

std::vector<Krawedz>::iterator it;

for(it=krawedzie.begin();it!
=krawedzie.end(); it++)
                    {
                        if(it!
=krawedzie.end() && ((*it).start==n ||
(*it).end==n)){krawedzie.erase(it);}
                    }
                    return true;
                }
                else {return false;}
            }

            bool droga(double &wynik, const
std::vector<int> &d)
            {
                wynik=0;

std::vector<Krawedz>::iterator it;

                for(unsigned int i=0;
i<d.size()-1;i++)
                {
                    Krawedz
kr(d[i],d[i+1],0);


it=find(krawedzie.begin(),krawedzie.end(),k
r);

                    if(it!=krawedzie.end()
&& *it==kr){wynik+=it->dlugosc;}
                    else {return false;}
                }
                return true;
            }


};
```

**27) Graf skierowany zawiera n wierzchołków numerowanych od 0 do n-1 oraz pwną liczbę krawdęedzi przechowywanych w kontenerze STL jako pary (start, end).**
**Dwa wierzchołki mogą być tylko jedną skierowana krawędzia.**

**a) konstruktor Graf(int n) - ustala liczbe wierzchołków**
**b) metode dodajKrawedz, operator < - sprawdza czy jeden graf jest podgrafem drugiego, bool jestDroga - sprawdza czy jest droga**

```
class Graf{
      typedef multimap<int,int>::iterator
__itr;
      typedef
multimap<int,int>::const_iterator
__itr_const;
      public:
             Graf(int _n) { wierzcholki=_n;
}
      private:
              multimap<int,int> krawedz;
              int wierzcholki;
      public:
      bool dodaj(int _s,int _e)
             {
                  pair<__itr,__itr> p;

p=krawedz.equal_range(_s);


                  if( _s>=wierzcholki ||
_s<0 || _e<0) { return false; }

                  for(__itr i=p.first;i!
=p.second;++i)          //sprawdzanie czy
                  {
                     //pary sie nie
                           if (i-
>second==_e) return false;   //powtarzaja

                  }


                  __itr
it=krawedz.begin();
                  while(it!=krawedz.end())
                  {
                   if(it->first==_e)
                     {
                      if(it->second==_s)
{ return false;}
                     }
                     it++;
                   }
krawedz.insert(make_pair(_s,_e));
                  return true;
             }

      bool jestDroga(const vector<int> &d)
             {
                  __itr it=krawedz.begin();

vector<int>::const_iterator vit=d.begin();

                  while(it!=krawedz.end())
                  {
                    if((*it).first==(*vit))
                     {
                       vit++;
                        if((*it).second!
=*vit) { vit--; }

                     }
                     if(vit==d.end()-1)
return true;
                     it++;
                  }
                  return false;
             }

friend bool operator<(const Graf &_k,const
Graf &_g)
{
      if(_g.wierzcholki<_k.wierzcholki)
return false;
       __itr_const k=_k.krawedz.begin();
       __itr_const g=_g.krawedz.begin();
       while( g!=_g.krawedz.end() && k!
=_k.krawedz.end() )
       {
              if(*g==*k) {  g++;   k++;  }
              else { g++;}
       }
       if(g==_g.krawedz.end() && k!
=_k.krawedz.end() ) {return false;}
       return true;
}
};
```

## 28) **Napisz szablon funkcji parametryzowanej typem T, która do kontenera vector<T>& result przepisuje wszystkie elementy kontenera const vector<T>& source mniejsze niż const T& key. Podaj przykład wywołania.**

**Kod:**
```
template<class T>
void costamol( vector<T>&  result , const vector<T>& source, const T& key)
{
   for(vector<T>::const_iterator i=source.begin();i!=source.end();i++)
   {
      if((*i)<key)
         result.push_back(*i);
   }
}
```
Podaj przykład wywołania

**Kod:**
```
vector<int> a;
   vector<int> b;
   a.push_back(1);
   a.push_back(6);
   a.push_back(8);
   costamol<int>(b,a,4);
```

## 29) Klasa Person zdefiniowana jako : class Person {string surname; string name; char pesel[12]} Napisz klasę PersonSet przechowującą pojedyncze instancje obiektów klasy Person z metodami:
## void insert(Person) - dodaje element do zbioru
## bool has(Person) - sprawdza czy element należy do zbioru
## operator == - do porównywania czy zbiory są równe
## operator < do sprawdzania czy jeden zbiór zawiera się w drugim

## Jako kontenera użyj dowolnego szablonu z biblioteki standardowej C++ za wyjątkiem set. Możesz dodać metody do klasy Person

**Kod:**
```
#include<string>
#include<vector>
using namespace std;
class Person
{
public:
   string surname;
   string name;
   char pesel[12];
   Person(string a, string b, char c[12]):
surname(a), name(b) {for(int i=0;i<12;i++)
{pesel[i]=c[i];}};
   bool operator==(Person& b);
};
bool Person::operator==(Person& b)
{
   for(int i=0;i<12;i++)
      if(b.pesel[i]!=pesel[i])
         return 0;
   if(surname!=b.surname)
      return 0;
   if(name!=b.name)
      return 0;
   return 1;
}

class PersonSet
{
public:
   vector<Person> data;
   void Insert(Person a);
   bool Has(Person a);
   bool operator==(PersonSet& b);
   bool operator<=(PersonSet& b);
};
void PersonSet::Insert(Person a)
{
   data.push_back(a);
}
bool PersonSet::Has(Person a)
{
   for(vector<Person>::iterator i=data.begin();i!
=data.end();i++)
      if((*i)==a)
         return 1;
   return 0;
}
bool PersonSet::operator==(PersonSet& b)
{
   if(b.data.size()!=data.size())
      return 0;
   else
      for(vector<Person>::iterator
i=data.begin();i!=data.end();i++)
         if(!b.Has(*i))
            return 0;
      for(vector<Person>::iterator
i=b.data.begin();i!=b.data.end();i++)
         if(Has(*i))
            return 0;
   return 1;
}
bool PersonSet::operator<=(PersonSet& b)
{
   for(vector<Person>::iterator i=data.begin();i!
=data.end();i++)
      if(!b.Has(*i))
         return 0;
   return 1;
}
```

**30) Zadeklaruj i zaimplementuj klasę Dictionary, która słowom (string) przypisuje jedno lub więcej znaczeń (string). Słownik powinien przechowywać unikalne pary (słowo, znaczenie). Jako kontenera użyj dowolnego szablonu biblioteki standardowej STL**

*** void add(const char* w, const char* m) - dodanie pary do słownika**
*** void find(ocnst char* w,list<string>& target) - poszukuje wszystkich znaczeń słowa w i umieszcza na liście target**
*** void deleteWord(const char* w) - usuwanie wszystkich par dla słowa w**
*** void join(const Dictionary& other) - dołącza zawartość drugiego słownika**

**Kod:**

```
#include <iostream>
#include <map>

using namespace std;

class Dictionary{
   public:
      map<string, string> slow;
      void add(const char* w, const char* m)
{slow[w]=m;}
      void find(const char* w, string &target){
           target=slow[w];
      }
      void deleteWord(const char* w)
{slow.erase(w);};
      void join(Dictionary& other){
           map<string,string>::iterator iter;
           iter=other.slow.begin();
           for(;iter!=other.slow.end();iter++)
               slow[iter->first]=iter->second;
      }

};



int main(){
    string cos;
    Dictionary a, b;

    a.add("plik", "plum");
    a.add("marek", "2");
    a.add("ala", "1");
    cout << "plik: " << a.slow["plik"] << endl;
    a.deleteWord("plik");
    a.find("marek", cos);
    cout << "cos: "<< cos << endl;
    b.add("fas", "da");
    b.add("234fas", "325da");
    b.add("23142fas", "daaf");
    a.join(b);
    cout << endl << "po dodaniu" << endl << endl;
    map<string,string>::iterator iter;
    iter=a.slow.begin();
    for(;iter!=a.slow.end();iter++)
         cout << iter->first << ": " << iter-
>second << endl;

    system("pause");
    return 0;
}
```

**31)** zad.1 25pkt

**Wykorzystując kontener STL *list* lub *vector* zaimplementuj klasę *StringToInt* reprezentującej odwzorowanie tekst->liczba nieujemna. Na przykład klasa może być używana do zliczania wystąpień słow w dokumencie.**

**Napisz metody**

**a) *int get(const char*key)* zwraca liczbę przypisaną *key***

**b) *void add(const char*key)* zwiększa liczbę przypisaną *key* o 1**

**c) *void remove(const char*key)* zmniejsza o 1 liczbę przypisaną *key.* (jeśli jest ona większa niż**

**d) operator + dla klasy *StringToInt***

**e) oprator <= dla klasy *StringToInt:* sprawdza czy: dla każdego (text, val1) należącego do A istnieje (text, val2) należące do B: $val_1 \leq val_2$**

```
class StringToInt
{
	private:
	vector<pair<string, unsigned> > tab;

	public:
	int get(const char *) const;
	void add(const char *);
	void remove(const char *);
	StringToInt operator+(StringToInt const &);
	bool operator<=(StringToInt const &);
};

int StringToInt::get(const char * key) const
{
	for (size_t i = 0 ; i < tab.size() ; ++i)
		if (tab[i].first == key)
			return tab[i].second;
	return 0;
}

void StringToInt::add(const char * key)
{
	for (size_t i = 0 ; i < tab.size() ; ++i)
		if (tab[i].first == key)
		{
			++tab[i].second;
			return;
		}
}

void StringToInt::remove(const char * key)
{
	for (size_t i = 0 ; i < tab.size() ; ++i)
		if (tab[i].first == key && tab[i].second >
0)
		{
			--tab[i].second;
			return;
		}
}

StringToInt StringToInt::operator+(StringToInt const &
drugi)
{
	StringToInt s;
	bool ok;
	for (size_t i = 0 ; i < tab.size() ; ++i)
		s.tab.push_back(tab[i]);
	for (size_t i = 0 ; i < drugi.tab.size() ; ++i)
	{
		ok = true;
		for (size_t j = 0 ; j < tab.size() ; ++j)
			if (tab[j].first ==
drugi.tab[i].first)
				ok = false;
		if (ok)
			s.tab.push_back(drugi.tab[i]);
	}
	return s;
}

bool StringToInt::operator<=(StringToInt const & drugi)
{
	if (tab.size() > drugi.tab.size())
		return false;
	bool ok;
	for (size_t i = 0 ; i < tab.size() ; ++i)
	{
		ok = false;
		for (size_t j = 0 ; j < drugi.tab.size() ; +
+j)
			if (tab[i].first ==
drugi.tab[j].first)
			{
				ok = true;
				if (tab[i].second >
drugi.tab[j].second)
					return false;
				j = drugi.tab.size();
			}
		if (!ok)
			return false;
	}
	return true;
}
```

**32)**zad.2 25pkt
**Bez STL. Napisz szablon *template <class T >class List<T>{ …}* klasy przechowującej elementy**
**typu *T* na liście dwukierunkowej.**
**a) Zadeklaruj klasę i pomocnicze struktury danych oraz zaimplementuj konstruktor**
**b) zaimplementuj destruktor**
**c) funkcję pushBack(const T&t)**
**d) funkcję void deleteFront()**
**e) konstruktor kopiujący**

```
template <class T>
class List
{
        public:
        struct wezel
        {
                wezel * nast;
                wezel * poprz;
                T element;
        };

   private:
        wezel * poczatek;
        wezel * koniec;

        public:
        List(void) : poczatek(NULL), koniec(NULL) {}
        ~List(void);
        wezel * Poczatek(void) const;
        void pushBack(T const &);
        void deleteFront(void);
        List(List const &);
};

template <class T>
typename List<T>::wezel * List<T>::Poczatek(void)
const
{
   return poczatek;
}

template <class T>
List<T>::~List(void)
{
        wezel * pom;
        while (poczatek)
        {
                pom = poczatek;
                poczatek = poczatek->nast;
                delete pom;
        }
}

template <class T>
void List<T>::pushBack(T const & t)
{
        if (!poczatek)
        {
                poczatek = new wezel;
                poczatek->element = t;
                poczatek->poprz = NULL;
                poczatek->nast = NULL;
                koniec = poczatek;
        }
        else
        {
                koniec->nast = new wezel;
                koniec->nast->element = t;
                koniec->nast->poprz = NULL;
                koniec->nast->nast = NULL;
                koniec = koniec->nast;
        }
}

template <class T>
void List<T>::deleteFront(void)
{
        if (poczatek)
        {
                wezel * pom = poczatek;
                poczatek = poczatek->nast;
                delete pom;
                if (!poczatek)
                        koniec = NULL;
        }
}

template <class T>
List<T>::List(List const & drugi)
{
        wezel * dr = NULL;
        wezel * pom;
        if (drugi.poczatek)
        {
                poczatek = new wezel;
                *poczatek = *drugi.poczatek;
                dr = drugi.poczatek->nast;
                pom = poczatek;
        }
        while (dr)
        {
                pom->nast = new wezel;
                pom = pom->nast;
                *pom = *dr;
                dr = dr->nast;
        }
        pom->nast = NULL;
}
```

**33)** zad.3 15pkt

**Wykorzystaj kontener z zad.2. *UnaryFunction* to klasa reprezentująca jednoargumentową funkcję typu *void*.**
**a) Napisz szablon funkcji *forEach()* parametryzowanej typami *T* i *UnaryFunction***
***template <class T, class UnaryFunction> void forEach(...***
**umożliwiającej przeprowadzenie na każdym elemencie kontenera *List<T>* operacji przekazanej**
**jako obiekt funkcyjny**
**b) Napisz klasę dla obiektu funkcyjnego umożliwiającą wydruk elementow całkowitych**
**c) Wywołaj funkcję dla listy parametryzowanej typem int i utworzonego obiektu funkcyjnego**

```
class Wypisz
{
        public:
        void operator()(int) const;
};

void Wypisz::operator()(int liczba) const
{
        cout << liczba << "\n";
}
```

```
template <class T, class UnaryFunction>
void forEach(List<T> const & l, UnaryFunction f)
{
        typename List<T>::wezel * pom = l.Poczatek();
        while (pom)
        {
                f(pom->element);
                pom = pom->nast;
        }
}
```

**34)** zad.4 20pkt

**Klasa *Iterator* ma (a) konstruktor (b) metodę *get(),* ktora zwraca bieżący element, (c) metodę**
***void next()*, ktora przesuwa iterator na następny element, (d) metodę *bool good()*, ktora zwraca true, jeśli stan iteratora pozwala na poprawny odczyt elementu.**
**Napisz iterator, ktory będzie w konstruktorze otrzymywał wskaźnik typu *const char**
**wskazujący tekst, a następnie wydzielał z niego słowa (ciągi znakow oddzielone białym znakami) i zwracał je w postaci obiektow klasy string.**

```
class Iterator
{
        private:
        stringstream strumien;

        public:
        Iterator(const char *);
        string get(void);
        void next(void);
        bool good(void);
};

Iterator::Iterator(const char * t)
{
        strumien << t;
}

string Iterator::get(void)
{
        string slowo;
        streampos pozycja;
        pozycja = strumien.tellg();
        strumien >> slowo;
        strumien.seekg(pozycja);
        return slowo;
}

void Iterator::next(void)
{
        string slowo;
        strumien >> slowo;
}

bool Iterator::good(void)
{
        return strumien.peek() != EOF;
}
```

**35)** Zad.5 15pkt
**Dla klasy class Time(int hour, min, sec), zadeklaruj i zaimplementuj:**
**a) operator ++ (zwiększa czas o jedną sekundę)**
**b) operator <**
**c) Time&operator +=(const Time&)**

```
class Time
{
	private:
	int hour;
	int min;
	int sec;

	public:
	Time & operator++(void);
	bool operator<(Time const &);
	Time & operator+=(Time const &);
};

Time & Time::operator++(void)
{
	++sec;
	if (sec > 59)
	{
		sec %= 60;
		++min;
		if (min > 59)
		{
			min %= 60;
			++hour;
			hour %= 24;
		}
	}
	return *this;
}

bool Time::operator<(Time const & drugi)
{
	return (hour == drugi.hour ?
		(min == drugi.min ? sec < drugi.sec :
min < drugi.min)
		: hour < drugi.hour);
}

Time & Time::operator+=(Time const & drugi)
{
	hour += drugi.hour;
	min += drugi.min;
	sec += drugi.sec;
	if (sec > 59)
	{
		sec %= 60;
		++min;
	}
	if (min > 59)
	{
		min %= 60;
		++hour;
	}
	hour %= 24;
	return *this;
}
```