Przykład 1. Biblioteka <priority_queue>.

```
double sum(std::vector<double>& v)
{
    double sum=0.0;
    sort(v.begin(), v.end());
    std::vector<double>::const_iterator it;
    it=find_if(v.begin(), v.end(), IsBTZ);
    std::priority_queue<double, std::vector<double>, std::greater<double> > pqp;
    std::priority_queue<double, std::vector<double>, std::less<double> > pqm;
    for(std::vector<double>::const_iterator i=it; i!=v.end(); i++)
    {
        pqp.push(*i);
        //std::cout<<pqp.top() <<"====\n";
    }
    for(std::vector<double>::const_iterator i=v.begin(); i!=it; i++)
    {
        pqm.push(*i);
        //std::cout<<pqm.top() <<"\n";
    }

    while(pqp.size() > 1)
    {
        double x = pqp.top(); pqp.pop();
        double y = pqp.top(); pqp.pop();
        //std::cout<<"["<<x<<"]["<<y<<"]plus\n";
        pqp.push(x+y);
    }

    while(pqm.size() > 1)
    {
        double x = pqm.top(); pqm.pop();
        double y = pqm.top(); pqm.pop();
        //std::cout<<"["<<x<<"]["<<y<<"]mines\n";
        pqm.push(x+y);
    }

    if(pqp.size() == 1)
        sum+=pqp.top();

    if(pqm.size() == 1)
        sum+=pqm.top();

    return sum;
}
```

Przykład 2. Kontener "list"[1].

```
#include <iostream>
#include <iterator>  // ostream_iterator
#include <string>
//#include <cctype>
#include <list>
#include <algorithm>
```

[1] "Accelerated C++: Practical Programming by Example"- Andrew Koenig, Barbara E. Moo, chapter 8, page 152.

```
// SPLIT
// generic function
// version of split function might return a vector<string>
// or list<string> or another kind of container

using std::cin;
using std::cout;
using std::ostream_iterator;

using std::string;
//using std::isspace;

using std::list;
//using std::find_if;

// 'true' if the argument is whitespace, 'false' otherwise
bool space(char c)
{
    return isspace(c);
}

// 'false' if the argument is whitespace, 'true' otherwise
bool not_space(char c)
{
    return !isspace(c);
}

// old version: ch07_p126
template <class Out>
void split(const string& str, Out os)
{
    typedef string::const_iterator iter;

    iter i = str.begin();
    while (i != str.end())
    {
        // ignore leading blanks
        i = find_if(i, str.end(), not_space);

        // find end of next word
        iter j = find_if(i, str.end(), space);

        // copy the characters in [i, j)
        if (i != str.end())
            *os++ = string(i, j); // changed: ret.push_back(string(i, j));

        i = j;
    }
}

int main()
{
    list<string> word_list;
    string s;

    cout<<"Example nr 1:\nEnter next line of words:\n";
    while (getline(cin, s)&&s!="END")
    {
        split(s, back_inserter(word_list));
        typedef list<string>::const_iterator iter; //iter aka
      //list<string>::const_iterator
```

```
        iter i = word_list.begin();
        iter e = word_list.end();
        while(i!=e)
        {
            cout<<*i<<"\n";
            ++i;
        }
        word_list.clear();
        cout<<"Enter next line of words (or type 'END'):\n";
    }

    cout<<"\nExample nr 2:\n";
    while (getline(cin, s))
    {
        cout<<"Enter next line of words (or type 'Ctrl+Z'):\n";
        split(s, ostream_iterator<string>(cout, "\n"));
    }

    return 0;
}
```

Przykład 3. Kontener "map"[2].

```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <cctype>   // isspace(), isalpha(), isdigit(), ...
#include <algorithm>// find_if()
#include <cstdlib>  // rand()
#include <stdexcept>// domain_error(), logic_error, ...

using std::istream;
using std::cin;
using std::cout;
using std::endl;

using std::vector;
using std::string;
using std::map;

using std::isspace;
using std::find_if;

using std::rand;
//using std::RAND_MAX;

using std::domain_error;
using std::logic_error;

typedef vector<string> Rule;
typedef vector<Rule> Rule_collection;
typedef map<string, Rule_collection> Grammar;

// 'true' if the argument is whitespace, 'false' otherwise
bool space(char c)
```

[2] "Accelerated C++: Practical Programming by Example"- Andrew Koenig, Barbara E. Moo, chapter 7, page 131.

```
{
    return isspace(c);
}

// 'false' if the argument is whitespace, 'true' otherwise
bool not_space(char c)
{
    return !isspace(c);
}

vector<string> split(const string& str)
{
    typedef string::const_iterator iter;
    vector<string> ret;

    iter i = str.begin();
    while (i != str.end())
    {
        // ignore leading blanks
        i = find_if(i, str.end(), not_space);

        // find end of next word
        iter j = find_if(i, str.end(), space);

        // copy the characters in [i, j)
        if (i != str.end())
            ret.push_back(string(i, j));
        i = j;
    }
    return ret;
}

// read a grammar from a given input stream
Grammar read_grammar(istream& in)
{
    Grammar ret;
    string line;

    // read the input
    while (getline(in, line))
    {

        // 'split' the input into words
        vector<string> entry = split(line);

        if (!entry.empty())
            // use the category to store the associated rule
            // " 'map' from 'string' to 'Rule_collection' "
            ret[entry[0]].push_back(Rule(entry.begin()+1, entry.end()));
    }
    return ret;
}

// predicate to checking whether the word is bracketed
bool bracketed(const string& s)
{
    return s.size() > 1 && s[0] == '<' && s[s.size() -1] == '>';
}

// return a random integer in the range [0, n)
int nrand(int n)
```

```
{
    if ( n <= 0 || n > RAND_MAX)
        throw domain_error("Argument to nrand is out of range");

    const int bucket_size = RAND_MAX / n; //bucket - komory
    int r;

    do r = rand() / bucket_size;
    while (r >= n);

    return r;
}

// auxiliary function
void gen_aux(const Grammar& g, const string& word, vector<string>& ret)
{
    if (!bracketed(word))
    {
        ret.push_back(word);
    }
    else{
        // locate the rule that corresponds to 'word'
        Grammar::const_iterator it = g.find(word);
        if (it == g.end())
            throw logic_error("empty rule");

        // fetch the set of possible rules | fetch - pobrac
        const Rule_collection& c = it->second;

        // from which we select one at random
        const Rule& r = c[nrand(c.size())];

        // recursively expand the second rule
        for (Rule::const_iterator i = r.begin(); i != r.end(); ++i)
            gen_aux(g, *i, ret);
    }
}

// sentence generator
vector<string> gen_sentence(const Grammar& g)
{
    vector<string> ret;
    gen_aux(g, "<sentence>", ret);
    return ret;
}

int main()
{
    // generate the sentence
    vector<string> sentence = gen_sentence(read_grammar(cin));

    // write the first word, if any
    vector<string>::const_iterator it = sentence.begin();
    if (!sentence.empty())
    {
        cout << *it;
        ++it;
    }

    // write the rest of the words, each preceded by a space
    while (it != sentence.end())
```

```
    {
        cout << " " << *it;
        ++it;
    }

    cout << endl;
    system("pause");

    return 0;
}
```

Przykład 4. Kontener "map"[3].

```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <cctype>
#include <fstream>
#include <algorithm> // Algorithms act on container elements - they do not act on
                     // containers.

using std::cin;
using std::cout;
using std::endl;

using std::vector;
using std::string;
using std::map;

using std::istream;
using std::ifstream;
using std::ofstream;

// SPLIT
// 'true' if the argument is whitespace, 'false' otherwise
bool space(char c)
{
    return isspace(c);
}

// 'false' if the argument is whitespace, 'true' otherwise
bool not_space(char c)
{
    return !isspace(c);
}

vector<string> split(const string& str)
{
    typedef string::const_iterator iter;
    vector<string> ret;

    iter i = str.begin();
    while (i != str.end())
    {
```

[3] "Accelerated C++: Practical Programming by Example"- Andrew Koenig, Barbara E. Moo, chapter 7, page 126.

```
        // ignore leading blanks
        i = find_if(i, str.end(), not_space);

        // find end of next word
        iter j = find_if(i, str.end(), space);

        // copy the characters in [i, j)
        if (i != str.end())
            ret.push_back(string(i, j));

        i = j;
    }
    return ret;
}
//END OF SPLIT

//FIND_URLS
bool not_url_char(char c)
{
    // characters, in addition to alphanumerics, that can appear in a URL
    static const string url_ch = "~;/?:@=&$-_.+!*'(),";

    //see whether "c" can appear in a URL and return the negative
    return !(isalnum(c) || find(url_ch.begin(), url_ch.end(), c) != url_ch.end());
    //isalnum -> <cctype>
}

string::const_iterator url_beg(string::const_iterator b, string::const_iterator e)
{
    static const string sep = "://";

    typedef string::const_iterator iter;

    // i marks where the separator was found
    iter i = b;

    while ((i = search(i, e, sep.begin(), sep.end())) != e)
    {
        // make sure the separator isn't at the beginning or end of the line
        if ( i != b && i +sep.size() != e)
        {
            // "beg" marks the beginning ot the protocol-name
            iter beg = i;
            while ( beg != b && isalpha(beg[-1]))
                --beg;

            // is there at least one appropriate character before and after the
            // seperator?
            if ( beg != i && !not_url_char(i[sep.size()]))
                return beg;
        }

        // the separator we found wasn't part of a URL; advance "i" past this
        // separator
        i += sep.size();
    }
    return e;
}

string::const_iterator url_end(string::const_iterator b, string::const_iterator e)
{
```

```
    return find_if(b, e, not_url_char);
}

vector<string> find_urls(const string& s)
{
     vector<string> ret;
     typedef string::const_iterator iter;
     iter b = s.begin(), e = s.end();


     //look throught the entire input
     while (b != e)
     {
         // look for one or more letters followed by ://
         b = url_beg(b, e);

         // if we found it
         if ( b != e)
         {
             //get the rest of URL
             iter after = url_end(b, e);

             // remember the URL
             ret.push_back(string(b, after));

             // advance b and check for more URLs on this line
             b = after;
         }
     }
     return ret;
}
//END OF FIND_URLS

// CROSS-REFERENCE TABLE
// find all the lines that refer to each word in the input
// you can use ifstream
map<string,  vector<int>  >  xref(istream&  in,  vector<string>  find_words(const
string&) = split)
{
    string line;
    int line_number = 0;
    map<string, vector<int> > ret;

    // read thenext line
    while(getline(in, line))
    {
        ++line_number;

        // break the input line into words
        vector<string> words = find_words(line);

        // remember that each word occurs on the current line
        for(vector<string>::const_iterator it = words.begin(); it != words.end();
     ++it)
            ret[*it].push_back(line_number);
    }
    return ret;
}
// END OF CROSS-REFERENCE TABLE

int main()
```

```
{
    string filein;
    map<string, vector<int> > ret;
    cout<<"Enter input filename:"<<endl;
    cin>>filein;

    filein+=".txt";

    ifstream in(filein.c_str());
    if(!in)
    {
        cout<<"Can't open "<<filein<<endl;
    }
    else
    {
        // call 'xref' using 'split' by default
        //ret = xref(in);
        ret = xref(in, find_urls);
        in.close();
    }

    // call 'xref' using 'split' by default
    //map<string, vector<int> > ret = xref(cin);

    // write the results
    // it - iterator (zmienna; definicja typu zmiennej wewnatrz klasy 'map')
    for  (map<string,  vector<int>  >::const_iterator  it  =  ret.begin();  it  !=
      ret.end(); ++it)
    {
        // write the word [occurs: eke:(r) wystepowac]
        cout << "[" << it->first << "] occurs on line(s): ";

        // followed by one or more line numbers
        vector<int>::const_iterator line_it = it->second.begin();
        cout << *line_it;   // write the first line number

        ++line_it;// write the rest of the line numbers, if any
        while (line_it != it->second.end())
        {
            cout << ", " << *line_it;
            ++line_it;
        }
        // write a new line to separate each word from the next
        cout << endl;
    }

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