Algoritmi i strukture podataka
1. Čas — Uvod u programski jezik C++

Sana Stojanović Đurđević

October 8, 2018

Materijal koji se pred vama nalazi u sebi sadrži kratak pregled osnovnih pojmova programskog jezika C++. Obrađeni su osnovni koncepti koji su relevantni za kasniju implementaciju algoritama koji će biti objašnjeni u ostatku kursa.

Većina primera koji su prikazani je preuzeta sa sajta `www.cplusplus.com`. Tu se može naći detaljan spisak klasa, funkcija i primera. Na sajtu `www.geeksforgeeks.org`, osim dodatnih primera, postoji i IDE (*Integrated Development Enviroment — Integrisano razvojno okruženje*) tako da se navedeni primeri mogu odmah i testirati.

Napomena: Tekući materijal je još uvek u izradi tako da su mogući određeni propusti. Ako naiđete na neku grešku ili nedoslednost, molim vas da mi to javite mail-om.

1 Pisanje prvog programa

Ime programa ima ekstenziju `.cpp`. Na primer: `program1.cpp`.
Prevođenje programa: `g++ program1.cpp`
(koristimo standard iz 2011. godine, što prilikom prevođenja naglašavamo: `g++ program1.cpp -std=c++11`).
Nakon uspešnog prevođenja dobija se izvršni program sa imenom `a.out`.
Možemo navesti novo ime izvršnog fajla: `g++ -o program1.exe program1.cpp`, nakon čega dobijamo izvršni program sa imenom `program1.exe`.

2 Biblioteke

- `<iostream>` - standardna ulazno-izlazna biblioteka koja omogućava ispis na ekranu
  `#include <iostream>` - pretprocesorska direktiva koja uključuje biblioteku u program
- `<cstdio>` - C zaglavlje `stdio.h`
- `<cmath>` - zaglavlje biblioteke za rad sa matematičkim operacijama (C zaglavlje `math.h`)

- `<cstdlib>` - C zaglavlje `stdlib.h`
- `<vector>` - zaglavlje biblioteke u kojoj je definisana struktura vektor
- `<string>` - zaglavlje biblioteke za rad sa niskama
- `<utility>` - zaglavlje biblioteke za rad sa parovima objekata, funkcije za zamenu objekata,...
- `<algorithm>` - zaglavlje koje sadrži funkcije za rad sa skupovima elemenata (kojima se pristupa preko iteratora)

3 Ispisivanje i učitavanje podataka

Zaglavlje biblioteke `<iostream>` obezbeđuje rad sa ulaznim (`std::cin`) i izlaznim tokom (`std::cout`).

Ispisivanje podataka. Podaci se ispisuju usmeravanjem podataka, izlaznim operatorom `<<`. Ispis više podataka se vrši ulančavanjem više operatora `<<` i podataka za ispis. Prelazak u novi red se dobija pomoću `endl`.

```
#include<iostream>
int main() {
  std::cout << "Ispis poruke iz " << 2 << "dela." << std::endl;
  return 0;
}
```

Korišćenje naredbe `using namespace std;` omogućava pristupu standardnom entitetu (`namespace`) koji se naziva `std` (moze se izbeći ponavljanje imenskog prostora `std`)

```
#include<iostream>
using namespace std;
int main() {
  cout << "Ispis poruke iz " << 2 << "dela." << endl;
  return 0;
}
```

Učitavanje podataka. Učitavanje podataka vrši se usmeravanjem ulaznog toka `std::cin`, operatorom `>>`, ka promenljivima u kojima će ulazni podaci biti smešteni.

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

int main() {

  int d, m, g;
  cout << "Unesite danasnji datum razdvojen razmakom: " << endl;

  cin >> d >> m >> g;

  cout << "Uneli ste: " << d << "." << m << "." << g << "." << endl;
  return 0;
}
```

4 Vector

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

Struktura vektor. Struktura vektor predstavlja niz susednih elemenata (u memoriji). Struktura je definisana u okviru biblioteke `<vector>`. Pri deklaraciji, tip elemenata vektora se navodi unutar `<>` zagrada. Elementima vektora se pristupa operatorom `[]`.

Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.

Memorijsko zauzeće. Internally, vectors use a dynamically allocated array to store their elements. This array may need to be reallocated in order to grow in size when new elements are inserted, which implies allocating a new array and moving all elements to it. This is a relatively expensive task in terms of processing time, and thus, vectors do not reallocate each time an element is added to the container.

Instead, vector containers may allocate some extra storage to accommodate for possible growth, and thus the container may have an actual capacity greater than the storage strictly needed to contain its elements (i.e., its size). Libraries can implement different strategies for growth to balance between memory usage and reallocations, but in any case, reallocations should only happen at logarithmically growing intervals of size so that the insertion of individual elements at the end of the vector can be provided with amortized constant time complexity (see push_back).

Therefore, compared to arrays, vectors consume more memory in exchange for the ability to manage storage and grow dynamically in an efficient way.

Compared to the other dynamic sequence containers (deques, lists and forward_lists), vectors are very efficient accessing its elements (just like arrays) and relatively efficient adding or removing elements from its end. For operations that involve inserting or removing elements at positions other than the end, they perform worse than the others, and have less consistent iterators and references than lists and forward_lists.

Operator []. Vraća element na traženoj poziciji. Portable programs should never call this function with an argument that is out of range, since this causes undefined behavior.

```
#include<iostream>
#include<vector>

using namespace std;

int main() {

  //promenljiva tipa vektor koja sadrzi 5 elemenata int
  vector<int> niz(5);

  //ucitavanje elemenata vektora
  for(int i=0; i<5; i++)
    cin >> niz[i];

  //ispisivanje elemenata vektora
  for(int i=0; i<5; i++)
    cout << niz[i] << " ";
  cout << endl;

  return 0;

}
```

Dinamička alokacija. Moguća je i dinamička alokacija broja elemenata u toku izvršavanja.

Range based for loops. Od standarda 2011. moguće je koristiti for petlje na pojednostavljen način. Ova opcija zahteva korišćenje opcije `-std=c++11` prilikom prevođenja programa.

Range based loops work best when all you want to do is simply iterate over the elements of a vector.

```
#include<iostream>
#include<vector>

using namespace std;

int main() {

  //dinamicka inicijalizacija broja elemenata u toku izvrsavanja

  int n;
  cout << "Unesite broj elemenata: ";
  cin >> n;

  //inicijalizacija broja elemenata vektora na vrednost n, ucitana u
  //fazi izvrsavanja
  vector<int> niz(n);

  //unosenje elemenata niza
  for(int i=0; i<n; i++)
    cin >> nix[i];

  //drugi nacin prolaza kroz vektor
  for(int x : niz)
    cout << x << ' ';
  cout << endl;

}
```

Funkcija *push_back*. Dodaje novi element na kraj vektora.
Adds a new element at the end of the vector, after its current last element. The content of val is copied (or moved) to the new element.

This effectively increases the container size by one, which causes an automatic reallocation of the allocated storage space if -and only if- the new vector size surpasses the current vector capacity.

If a reallocation happens, all iterators, pointers and references related to the container are invalidated. Otherwise, only the end iterator is invalidated, and all iterators, pointers and references to elements are guaranteed to keep referring to the same elements they were referring to before the call.

Funkcija *size*. Vraća broj elemenata u vektoru. This is the number of actual objects held in the vector, which is not necessarily equal to its storage capacity.

```
//Primer: operator [], funkcije push_back, pop_back i size

#include <iostream>
#include <vector>

using namespace std;

int main ()
{
  vector<int> myvector;
  int myint;

  cout << "Please enter some integers (enter 0 to end):" << endl;

  do {
    cin >> myint;
    myvector.push_back(myint);
  } while (myint);

  cout << "Size of myvector is: " << myvector.size() << endl;

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

  myvector.pop_back();

  cout << "at the end myvector contains: ";
  for(int i=0; i < myvector.size(); i++)
    cout << myvector[i] << " ";
  cout << endl;

  return 0;
}

Please enter some integers (enter 0 to end):
3 1 8 4 0
Size of myvector is: 5
myvector contains: 3 1 8 4 0
after pop_back:
myvector contains: 3 1 8 4
```

Funkcija *capacity*. Returns the size of the storage space currently allocated for the vector, expressed in terms of elements.

This capacity is not necessarily equal to the vector size. It can be equal or greater, with the extra space allowing to accommodate for growth without the need to reallocate on each insertion.

Notice that this capacity does not suppose a limit on the size of the vector. When this capacity is exhausted and more is needed, it is automatically expanded by the container (reallocating it storage space). The theoretical limit on the size of a vector is given by member max_size.

The capacity of a vector can be explicitly altered by calling member vector::reserve.

Funkcija *max_size*. Returns the maximum number of elements that the vector can hold.

This is the maximum potential size the container can reach due to known system or library implementation limitations, but the container is by no means guaranteed to be able to reach that size: it can still fail to allocate storage at any point before that size is reached.

```
//Primer: funkcije size, capacity i max_size

#include <iostream>
#include <vector>

using namespace std;
int main (){

  vector<int> myvector;
  // set some content in the vector:
  for (int i=0; i<10; i++)
    myvector.push_back(i);

  cout << "size: " << myvector.size() << endl;
  cout << "capacity: " << myvector.capacity() << endl;
  cout << "max_size: " << myvector.max_size() << endl;

  return 0;
}

size: 10
capacity: 16
max_size: 1073741823
```

Funkcija *resize*. Funkcija koja menja sadržaj vektora tako da sadrži tačno n elemenata (vrednost `size` će biti postavljena na vrednost n).

Resizes the container so that it contains n elements.

If new size — n is smaller than the current container size, the content is reduced to its first n elements, removing those beyond (and destroying them).

If n is greater than the current container size, the content is expanded by inserting at the end as many elements as needed to reach a size of n. If *val* is specified, the new elements are initialized as copies of *val*, otherwise, they are value-initialized.

If n is also greater than the current container capacity, an automatic reallocation of the allocated storage space takes place.

Notice that this function changes the actual content of the container by inserting or erasing elements from it.

```
// resizing vector
#include <iostream>
#include <vector>

using namespace std;

int main ()
{
  vector<int> myvector;

  // set some initial content:
  for (int i=1;i<10;i++)
    myvector.push_back(i);

  cout << "size: " << myvector.size() << endl;
  myvector.resize(5);
  cout << "size: " << myvector.size() << endl;
  myvector.resize(8,100);
  cout << "size: " << myvector.size() << endl;
  myvector.resize(12);
  cout << "size: " << myvector.size() << endl;

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

  return 0;
}

size: 9
size: 5
size: 8
size: 12
myvector contains: 1 2 3 4 5 100 100 100 0 0 0 0
```

Funkcija *reserve*. Requests that the vector capacity be at least enough to contain n elements.

If n is greater than the current vector capacity, the function causes the container to reallocate its storage increasing its capacity to n (or greater).

In all other cases, the function call does not cause a reallocation and the vector capacity is not affected.

This function has no effect on the vector size and cannot alter its elements.

```
// vector::reserve
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int>::size_type sz;

  std::vector<int> foo;
  sz = foo.capacity();
  std::cout << "making foo grow:\n";
  for (int i=0; i<100; ++i) {
    foo.push_back(i);
    if (sz!=foo.capacity()) {
      sz = foo.capacity();
      std::cout << "capacity changed: " << sz << '\n';
    }
  }

  std::vector<int> bar;
  sz = bar.capacity();
  bar.reserve(100);   // this is the only difference with foo above
  std::cout << "making bar grow:\n";
  for (int i=0; i<100; ++i) {
    bar.push_back(i);
    if (sz!=bar.capacity()) {
      sz = bar.capacity();
      std::cout << "capacity changed: " << sz << '\n';
    }
  }
  return 0;
}
```

```
making foo grow:
capacity changed: 1
capacity changed: 2
capacity changed: 4
capacity changed: 8
capacity changed: 16
capacity changed: 32
capacity changed: 64
capacity changed: 128
making bar grow:
capacity changed: 100
```

5 Iteratori nad vektorima

Iteratori su posebna vrsta pokazivača koji se koriste za prolazak (iteraciju) kroz kolekciju elemenata. Svaka kolekcija elemenata poseduje iterator koji pokazuje na početni element kolekcije i iterator koji pokazuje na kraj kolekcije. Prolazak kroz kolekciju se obavlja počevši od početnog iteratora, inkrementacijom iteratora sve dok ne dostigne iterator za kraj. Promenljiva koja sadrži iterator ima tip iteratora kolekcije kroz koju prolazi.

begin. Returns an iterator pointing to the first element in the vector.

Notice that, unlike member `vector::front`, which returns a reference to the first element, this function returns a random access iterator pointing to it.

end. Returns an iterator referring to the past-the-end element in the vector container.

The past-the-end element is the theoretical element that would follow the last element in the vector. It does not point to any element, and thus shall not be dereferenced.

Because the ranges used by functions of the standard library do not include the element pointed by their closing iterator, this function is often used in combination with vector::begin to specify a range including all the elements in the container.

If the container is empty, this function returns the same as vector::begin.

```
// vector::begin/end
#include <iostream>
#include <vector>

using namespace std;

int main ()
{
  vector<int> myvector;
  vector<int>::iterator it;

  for (int i=1; i<=5; i++)
    myvector.push_back(i);

  cout << "myvector contains:";
  for (it = myvector.begin() ; it != myvector.end(); it++)
    cout << ' ' << *it;
  cout << '\n';

  return 0;
}
```

front. Returns a reference to the first element in the vector.

Unlike member vector::begin, which returns an iterator to this same element, this function returns a direct reference.

back. Returns a reference to the last element in the vector.

Unlike member vector::end, which returns an iterator just past this element, this function returns a direct reference.

```
// vector::front, back
#include <iostream>
#include <vector>

using namespace::std;

int main ()
{
  vector<int> myvector;

  myvector.push_back(78);
  myvector.push_back(16);

  // now front equals 78, and back 16
  myvector.front() -= myvector.back();

  cout << "myvector.front() is now " << myvector.front() << '\n';

  return 0;
}
```

at. Returns a reference to the element at position n in the vector.

The function automatically checks whether n is within the bounds of valid elements in the vector, throwing an out_of_range exception if it is not (i.e., if n is greater than, or equal to, its size). This is in contrast with member operator[], that does not check against bounds.

```
// vector::at
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector (10);   // 10 zero-initialized ints

  // assign some values:
  for (unsigned i=0; i<myvector.size(); i++)
    myvector.at(i)=i;

  std::cout << "myvector contains:";
  for (unsigned i=0; i<myvector.size(); i++)
    std::cout << ' ' << myvector.at(i);
  std::cout << '\n';

  return 0;
}

myvector contains: 0 1 2 3 4 5 6 7 8 9
```

rbegin. Returns a reverse iterator pointing to the last element in the vector (i.e., its reverse beginning).

Reverse iterators iterate backwards: increasing them moves them towards the beginning of the container.

`rbegin` points to the element right before the one that would be pointed to by member end.

Notice that unlike member `vector::back`, which returns a reference to this same element, this function returns a reverse random access iterator.

rend. Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (which is considered its reverse end).

The range between `vector::rbegin` and `vector::rend` contains all the elements of the vector (in reverse order).

```
// vector::rbegin/rend
#include <iostream>
#include <vector>

using namespace::std;

int main ()
{
  vector<int> myvector (5);  // 5 default-constructed ints

  int i=0;

  vector<int>::reverse_iterator rit;
  vector<int>::iterator it;

  for (it = myvector.begin(); it != myvector.end(); it++)
    *it = ++i;

  cout << "myvector contains (in reverse):";
  for (rit=myvector.rbegin(); rit!= myvector.rend(); rit++)
    cout << *rit << ' ';
  cout << endl;

  return 0;
}

myvector contains (in reverse): 5 4 3 2 1
```

advance. Advances the iterator it by n element positions.

If it is a random-access iterator, the function uses just once operator+ or operator-. Otherwise, the function uses repeatedly the increase or decrease operator (operator++ or operator–) until n elements have been advanced.

Izuzetak: Inicijalizacija niza u narednom programu zahteva korišćenje opcije `-std=c++11` (`vector<int> ar = { 1, 2, 3, 4, 5 };`).

```
// advance()
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main()
{
    vector<int> ar = { 1, 2, 3, 4, 5 };

    // Declaring iterator to a vector
    vector<int>::iterator ptr = ar.begin();

    // Using advance() to increment iterator position
    // points to 4
    advance(ptr, 3);

    // Displaying iterator position
    cout << "The position of iterator after advancing is : ";
    cout << *ptr << " ";

    return 0;

}

The position of iterator after advancing is : 4
```

next. Returns an iterator pointing to the element that it would be pointing to if advanced n positions.
it is not modified.

prev. Returns an iterator pointing to the element that it would be pointing to if advanced -n positions.

auto. Kako bi se izbeglo pisanje dugog naziva tipa iteratora, od standarda iz 2011. godine, podržano je korišćenje ključne reči `auto`, kojom se kompajleru ostavlja da prepozna tip promenljive. Zahteva korišćenje opcije `-std=c++11` prilikom prevođenja programa.

```
// next() and prev()
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
int main()
{
    vector<int> ar = { 1, 2, 3, 4, 5 };

    // Declaring iterators to a vector
    vector<int>::iterator ptr = ar.begin();
    vector<int>::iterator ftr = ar.end();


    // Using next() to return new iterator
    // points to 4
    auto it = next(ptr, 3);

    // Using prev() to return new iterator
    // points to 3
    auto it1 = prev(ftr, 3);

    // Displaying iterator position
    cout << "The position of new iterator using next() is : ";
    cout << *it << " ";
    cout << endl;

    // Displaying iterator position
    cout << "The position of new iterator using prev() is : ";
    cout << *it1 << " ";
    cout << endl;

    return 0;
}

The position of new iterator using next() is : 4
The position of new iterator using prev() is : 3
```

6 String

`http://www.cplusplus.com/reference/string/string/?kw=string`

Strings are objects that represent sequences of characters. String class stores the characters as a sequence of bytes with a functionality of allowing access to single byte character.

Operator []. Returns a reference to the character at position pos in the string.

Funkcija lenght. Returns the length of the string, in terms of bytes.

This is the number of actual bytes that conform the contents of the string, which is not necessarily equal to its storage capacity. Both `string::size` and `string::length` are synonyms and return the exact same value.

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

using namespace::std;

int main ()
{
  string str = "Test string";

  cout << str << endl;

  for (int i=0; i<str.length(); i++)
    cout << str[i];
  cout << endl;

  return 0;
}
```

Operator =. Assigns a new value to the string, replacing its current contents.

Operator +=. Extends the string by appending additional characters at the end of its current value.

Operator +. Returns a newly constructed string object with its value being the concatenation of the characters in the left string followed by those of the second string.

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

using namespace::std;

int main ()
{
  string str1, str2, str3;
  str1 = "Test string: ";   // c-string
  str2 = 'x';               // single character
  str3 = str1 + str2;       // string

  cout << str3  << '\n';

  string name ("John");
  string family ("Smith");
  name += " K. ";         // c-string
  name += family;         // string
  name += '\n';           // character

  cout << name;

  return 0;
}
```

Funkcija getline. This function is used to store a stream of characters as entered by the user.

Funkcije *push_back*, *pop_back*, *capacity*, funkcije sa iteratorima se koriste na isti način kao kod vektora.

```
#include<iostream>
#include<string> // for string class
using namespace std;
int main()
{
    string str;                  // Declaring string

    getline(cin,str);            // Taking string input using getline()

    cout << "The initial string is : ";
    cout << str << endl;

    str.push_back('s');

    cout << "The string after push_back operation is : ";
    cout << str << endl;

    str.pop_back();

    cout << "The string after pop_back operation is : ";
    cout << str << endl;

    return 0;
}

Input: abc
The initial string is : abc
The string after push_back operation is : abcs
The string after pop_back operation is : abc
```

Funkcija resize. This function changes the size of string, the size can be increased or decreased.

Funkcija shrink_to_fit. This function decreases the capacity of the string and makes it equal to its size.

Izuzetak: Funkcije `pop_back` i `shrink_to_fit` zahtevaju korišćenje opcije `-std=c++11` (introduced from C++11 for strings).

```
#include<iostream>
#include<string> // for string class
using namespace std;
int main()
{
    // Initializing string
    string str = "geeksforgeeks is for geeks";

    // Displaying string
    cout << "The initial string is : ";
    cout << str << endl;

    // Resizing string using resize()
    str.resize(13);

    // Displaying string
    cout << "The string after resize operation is : ";
    cout << str << endl;

    // Displaying capacity of string
    cout << "The capacity of string is : ";
    cout << str.capacity() << endl;

    // Decreasing the capacity of string
    // using shrink_to_fit()
    str.shrink_to_fit();

    // Displaying string
    cout << "The new capacity after shrinking is : ";
    cout << str.capacity() << endl;

    return 0;
}

The initial string is : geeksforgeeks is for geeks
The string after resize operation is : geeksforgeeks
The capacity of string is : 26
The new capacity after shrinking is : 13
```

Iteratorske funkcije.

begin. This function returns an iterator to beginning of the string.

end. This function returns an iterator to end of the string.

rbegin. This function returns a reverse iterator pointing at the end of string.

rend. This function returns a reverse iterator pointing at beginning of string.

```
#include<iostream>
#include<string> // for string class
using namespace std;
int main()
{
    // Initializing string‘
    string str = "geeksforgeeks";

    // Declaring iterator
    std::string::iterator it;

    // Declaring reverse iterator
    std::string::reverse_iterator it1;

    // Displaying string
    cout << "The string using forward iterators is : ";
    for (it=str.begin(); it!=str.end(); it++)
      cout << *it;
    cout << endl;

    // Displaying reverse string
    cout << "The reverse string using reverse iterators is : ";
    for (it1=str.rbegin(); it1!=str.rend(); it1++)
      cout << *it1;
    cout << endl;

    return 0;

}
```

Dodatne funkcije za rad sa stringovima

copy. This function copies the substring in target character array mentioned in its arguments. It takes 3 arguments, target char array, length to be copied and starting position in string to start copying.

swap. This function swaps one string with other.

```
#include<iostream>
#include<string> // for string class
using namespace std;
int main()
{
    // Initializing 1st string
    string str1 = "geeksforgeeks is for geeks";

    // Declaring 2nd string
    string str2 = "geeksforgeeks rocks";

    // Declaring character array
    char ch[80];

    // using copy() to copy elements into char array
    // copies "geeksforgeeks"
    str1.copy(ch,13,0);

    // Diplaying char array
    cout << "The new copied character array is : ";
    cout << ch << endl << endl;

    // Displaying strings before swapping
    cout << "The 1st string before swapping is : ";
    cout << str1 << endl;
    cout << "The 2nd string before swapping is : ";
    cout << str2 << endl;

    // using swap() to swap string content
    str1.swap(str2);

        // Displaying strings after swapping
    cout << "The 1st string after swapping is : ";
    cout << str1 << endl;
    cout << "The 2nd string after swapping is : ";
    cout << str2 << endl;

    return 0;

}

The new copied character array is : geeksforgeeks

The 1st string before swapping is : geeksforgeeks is for geeks
The 2nd string before swapping is : geeksforgeeks rocks
The 1st string after swapping is : geeksforgeeks rocks
The 2nd string after swapping is : geeksforgeeks is for geeks
```

7 Parovi i n-torke

https://www.geeksforgeeks.org/pair-in-cpp-stl/
https://www.geeksforgeeks.org/tuples-in-c/

Klasa *pair*. This class couples together a pair of values, which may be of different types (T1 and T2). The individual values can be accessed through its public members *first* and *second*.

Pairs are a particular case of *tuple*.

Funkcija *make_pair*. Constructs a pair object with its first element set to x and its second element set to y.

```
#nekoliko nacina inicijalizacije
#include <iostream>
#include <utility>
using namespace std;

int main()
{
    pair <int, char> PAIR1 ;
    PAIR1.first = 100;
    PAIR1.second = 'G' ;

    cout << PAIR1.first << " " ;
    cout << PAIR1.second << endl ;

    pair <string,double> PAIR2 ("GeeksForGeeks", 1.23);

    cout << PAIR2.first << " " ;
    cout << PAIR2.second << endl ;

    pair <string, double> PAIR3 ;
    PAIR3 = make_pair ("GeeksForGeeks is Best",4.56);

    return 0;
}
```

Klasa tuple. Tuples are objects that pack elements of -possibly- different types together in a single object, just like pair objects do for pairs of elements, but generalized for any number of elements.

Conceptually, they are similar to plain old data structures (C-like structs) but instead of having named data members, its elements are accessed by their order in the tuple.

Funkcija make_tupple. Constructs an object of the appropriate tuple type to contain the specified elements.

Funkcija get. Returns a reference to the I-th element of tuple.

```
#include<iostream>
#include<tuple> // for tuple
using namespace std;
int main()
{
    // Declaring tuple
    tuple <char, int, float> geek;

    // Assigning values to tuple using make_tuple()
    geek = make_tuple('a', 10, 15.5);

    // Printing initial tuple values using get()
    cout << "The initial values of tuple are : ";
    cout << get<0>(geek) << " " << get<1>(geek);
    cout << " " << get<2>(geek) << endl;

    // Use of get() to change values of tuple
    get<0>(geek) = 'b';
    get<2>(geek) =  20.5;

     // Printing modified tuple values
    cout << "The modified values of tuple are : ";
    cout << get<0>(geek) << " " << get<1>(geek);
    cout << " " << get<2>(geek) << endl;

    return 0;
}
```

8 Algorithm

`http://www.cplusplus.com/reference/algorithm/?kw=algorithm`

Zaglavlje `<algorithm>` sadrži funkcije za rad sa skupovima elemenata (kojima se pristupa preko iteratora).

8.1 Funkcije koje ne menjaju podatke.

find. Returns an iterator to the first element in the range [first,last) that compares equal to val. If no such element is found, the function returns last.

The function uses operator == to compare the individual elements to val.

Return value: An iterator to the first element in the range that compares equal to val. If no elements match, the function returns last.

```
// find example
#include <iostream>     // std::cout
#include <algorithm>    // std::find
#include <vector>       // std::vector

int main () {
  // using std::find with array and pointer:
  int myints[] = { 10, 20, 30, 40 };
  int * p;

  p = std::find (myints, myints+4, 30);
  if (p != myints+4)
    std::cout << "Element found in myints: " << *p << '\n';
  else
    std::cout << "Element not found in myints\n";

  // using std::find with vector and iterator:
  std::vector<int> myvector (myints,myints+4);
  std::vector<int>::iterator it;

  it = find (myvector.begin(), myvector.end(), 30);
  if (it != myvector.end())
    std::cout << "Element found in myvector: " << *it << '\n';
  else
    std::cout << "Element not found in myvector\n";

  return 0;
}

Izlaz:
Element found in myints: 30
Element found in myvector: 30
```

find_if. Returns an iterator to the first element in the range [first,last) for which pred returns true. If no such element is found, the function returns last.

Return value: An iterator to the first element in the range for which pred does not return false. If pred is false for all elements, the function returns last.

first, last Input iterators to the initial and final positions in a sequence. The range used is [first,last), which contains all the elements between first and last, including the element pointed by first but not the element pointed by last.

pred Unary function that accepts an element in the range as argument and returns a value convertible to bool. The value returned indicates whether the element is considered a match in the context of this function. The function shall not modify its argument. This can either be a function pointer or a function object.

```
// find_if example
#include <iostream>     // std::cout
#include <algorithm>    // std::find_if
#include <vector>       // std::vector

bool IsOdd (int i) {
  return ((i%2)==1);
}

int main () {
  std::vector<int> myvector;
  std::vector<int>::iterator it;

  myvector.push_back(10);
  myvector.push_back(25);
  myvector.push_back(40);
  myvector.push_back(55);

  it = std::find_if (myvector.begin(), myvector.end(), IsOdd);
  std::cout << "The first odd value is " << *it << '\n';

  return 0;
}

Izlaz:
The first odd value is 25
```

count. Returns the number of elements in the range [first,last) that compare equal to val.

The function uses operator == to compare the individual elements to val.
`http://www.cplusplus.com/reference/algorithm/count/`

count_if. `http://www.cplusplus.com/reference/algorithm/count_if/`

8.2 Funkcije koje menjaju podatke.

copy. Copies the elements in the range [first,last) into the range beginning at result.

```
// copy algorithm example
#include <iostream>     // std::cout
#include <algorithm>    // std::copy
#include <vector>       // std::vector

int main () {
  int myints[]={10,20,30,40,50,60,70};
  std::vector<int> myvector (7);
  std::vector<int>::iterator it;

  std::copy ( myints, myints+7, myvector.begin() );

  std::cout << "myvector contains:";
  for (it = myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;

  std::cout << '\n';

  return 0;
}

Izlaz:
myvector contains: 10 20 30 40 50 60 70
```

copy_if. Funkcija koja iz opsega kopira samo određene elemente.
`http://www.cplusplus.com/reference/algorithm/copy_if/`

```
// copy_if example
#include <iostream>     // std::cout
#include <algorithm>    // std::copy_if, std::distance
#include <vector>       // std::vector

int main () {
  std::vector<int> foo = {25,15,5,-5,-15};
  std::vector<int> bar (foo.size());

  // copy only positive numbers:
  auto it = std::copy_if (foo.begin(), foo.end(), bar.begin(),
                        [](int i){return !(i<0);} );
  bar.resize(std::distance(bar.begin(),it));
  // shrink container to new size

  std::cout << "bar contains:";
  for (int& x: bar)
    std::cout << ' ' << x;
  std::cout << '\n';

  return 0;
}

Izlaz:
bar contains: 25 15 5
```

transform. Funkcija koja nad podacima iz zadatih opsega (jednog ili dva opsega) primenjuje datu operaciju. `http://www.cplusplus.com/reference/algorithm/transform/`

```
// transform algorithm example
#include <iostream>     // std::cout
#include <algorithm>    // std::transform
#include <vector>       // std::vector
#include <functional>   // std::plus

int op_increase (int i) { return ++i; }

int main () {
  std::vector<int> foo;
  std::vector<int> bar;

  // set some values:
  for (int i=1; i<6; i++)
    foo.push_back (i*10);                    // foo: 10 20 30 40 50

  bar.resize(foo.size());                    // allocate space

  std::transform (foo.begin(), foo.end(), bar.begin(), op_increase);
                                             // bar: 11 21 31 41 51

  // std::plus adds together its two arguments:
  std::transform (foo.begin(), foo.end(), bar.begin(), foo.begin(),
                  std::plus<int>());
                                             // foo: 21 41 61 81 101

  std::cout << "foo contains:";
  for (std::vector<int>::iterator it=foo.begin(); it!=foo.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}

foo contains: 21 41 61 81 101
```

8.3 Funkcije za sortiranje.

sort. Funkcija koja sortira elemente u rastućem poretku.

Sorts the elements in the range [first,last) into ascending order.

The elements are compared using operator $<$ for the first version, and *comp* for the second.

```
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector

bool myfunction (int i,int j) { return (i<j); }

struct myclass {
  bool operator() (int i,int j) { return (i<j);}
} myobject;

int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  std::vector<int> myvector (myints, myints+8);
  // 32 71 12 45 26 80 53 33

  // using default comparison (operator <):
  std::sort (myvector.begin(), myvector.begin()+4);
  //(12 32 45 71)26 80 53 33

  // using function as comp
  std::sort (myvector.begin()+4, myvector.end(), myfunction);
  // 12 32 45 71(26 33 53 80)

  // using object as comp
  std::sort (myvector.begin(), myvector.end(), myobject);
  //(12 26 32 33 45 53 71 80)

  // print out content:
  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it=myvector.begin();
       it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}

Izlaz:
myvector contains: 12 26 32 33 45 53 71 80
```

8.4 Binarna pretraga.

lower_bound. Returns an iterator pointing to the first element in the range [first,last) which does not compare less than val.

The elements are compared using operator $<$ for the first version, and comp for the second. The elements in the range shall already be sorted according to this same criterion (operator $<$ or comp), or at least partitioned with respect to val.

first, last Forward iterators to the initial and final positions of a sorted (or properly partitioned) sequence. The range used is [first,last), which contains all the elements between first and last, including the element pointed by first but not the element pointed by last.

val Value of the lower bound to search for in the range. For (1), T shall be a type supporting being compared with elements of the range [first,last) as the right-hand side operand of operator¡.

comp Binary function that accepts two arguments (the first of the type pointed by ForwardIterator, and the second, always val), and returns a value convertible to bool. The value returned indicates whether the first argument is considered to go before the second. The function shall not modify any of its arguments. This can either be a function pointer or a function object.

upper_bound. Returns an iterator pointing to the first element in the range [first,last) which compares greater than val.

```
#include <iostream>     // cout
#include <algorithm>    // lower_bound, std::upper_bound, std::sort
#include <vector>       // vector

using namespace::std;

int main () {
  int myints[] = {10,20,30,30,20,10,10,20};
  vector<int> v(myints,myints+8);           //10 20 30 30 20 10 10 20

  sort (v.begin(), v.end());                //10 10 10 20 20 20 30 30

  vector<int>::iterator low,up;
  low=lower_bound (v.begin(), v.end(), 20); //         ^
  up= upper_bound (v.begin(), v.end(), 20); //                  ^

  cout << "lower_bound at position " << (low- v.begin()) << '\n';
  cout << "upper_bound at position " << (up - v.begin()) << '\n';

  return 0;
}

Izlaz:
lower_bound at position 3
upper_bound at position 6
```

equal_range. Returns the bounds of the subrange that includes all the elements of the range [first,last) with values equivalent to val.

The elements in the range shall already be sorted according to this same criterion (operator < or comp), or at least partitioned with respect to val.

If val is not equivalent to any value in the range, the subrange returned has a length of zero, with both iterators pointing to the nearest value greater than val, if any, or to last, if val compares greater than all the elements in the range.

Return value: A pair object, whose member pair::first is an iterator to the lower bound of the subrange of equivalent values, and pair::second its upper bound. The values are the same as those that would be returned by functions lower_bound and upper_bound respectively.

```
// equal_range example
// equal_range example
#include <iostream>     // cout
#include <algorithm>    // equal_range, std::sort
#include <vector>       // vector

using namespace::std;

bool mygreater (int i,int j) { return (i>j); }

int main () {
  int myints[] = {10,20,30,30,20,10,10,20};
  vector<int> v(myints,myints+8);
  // 10 20 30 30 20 10 10 20

  pair<vector<int>::iterator,vector<int>::iterator> bounds;

  // using default comparison:
  sort (v.begin(), v.end());
  bounds=equal_range (v.begin(), v.end(), 20);
  // 10 10 10 20 20 20 30 30
  //          ^        ^

  // using "mygreater" as comp:
  sort (v.begin(), v.end(), mygreater);
  bounds=equal_range (v.begin(), v.end(), 20, mygreater);

  // 30 30 20 20 20 10 10 10
  //       ^        ^

  cout << "bounds at positions " << (bounds.first - v.begin());
  cout << " and " << (bounds.second - v.begin()) << '\n';

  return 0;
}

Izlaz:
bounds at positions 2 and 5
```

binary_search. Funkcija koja pretražuje vrednost u sortiranom nizu.

Returns true if any element in the range [*first*,*last*) is equivalent to *val*, and false otherwise.

The elements are compared using operator¡ for the first version, and comp for the second. Two elements, a and b are considered equivalent if `(!(a<b) && !(b<a))` or if `(!comp(a,b) && !comp(b,a))`.

The elements in the range shall already be sorted according to this same criterion (operator¡ or comp),

```
#include <iostream>     // std::cout
#include <algorithm>    // std::binary_search, std::sort
#include <vector>       // std::vector

bool myfunction (int i,int j) { return (i<j); }

int main () {
  int myints[] = {1,2,3,4,5,4,3,2,1};
  std::vector<int> v(myints,myints+9);
  // 1 2 3 4 5 4 3 2 1

  // using default comparison:
  std::sort (v.begin(), v.end());

  std::cout << "looking for a 3... ";
  if (std::binary_search (v.begin(), v.end(), 3))
    std::cout << "found!\n"; else std::cout << "not found.\n";

  // using myfunction as comp:
  std::sort (v.begin(), v.end(), myfunction);

  std::cout << "looking for a 6... ";
  if (std::binary_search (v.begin(), v.end(), 6, myfunction))
    std::cout << "found!\n"; else std::cout << "not found.\n";

  return 0;
}

Izlaz:
looking for a 3... found!
looking for a 6... not found.
```

8.5 Funkcije minimuma i maksimuma.

min. Funkcija koja vraća manju vrednost od dva prosleđena argumenta. Ako su vrednosti iste, vratiće prvi argument.

max. Funkcija koja vraća veću vrednost od dva prosleđena argumenta. Ako su vrednosti iste, vratiće prvi argument.

```
// min example
#include <iostream>     // std::cout
#include <algorithm>    // std::min

int main () {
  std::cout << "min(1,2)==" << std::min(1,2) << '\n';
  std::cout << "min(2,1)==" << std::min(2,1) << '\n';
  std::cout << "min('a','z')==" << std::min('a','z') << '\n';
  std::cout << "min(3.14,2.72)==" << std::min(3.14,2.72) << '\n';

  std::cout << "max(1,2)==" << std::max(1,2) << '\n';
  std::cout << "max(2,1)==" << std::max(2,1) << '\n';
  std::cout << "max('a','z')==" << std::max('a','z') << '\n';
  std::cout << "max(3.14,2.73)==" << std::max(3.14,2.73) << '\n';

  return 0;
}

Izlaz:
min(1,2)==1
min(2,1)==1
min('a','z')==a
min(3.14,2.72)==2.72
max(1,2)==2
max(2,1)==2
max('a','z')==z
max(3.14,2.73)==3.14
```

Funkcije nad opsegom podataka. The versions for initializer lists return the smallest (for min) of all the elements in the list. Returning the first of them if these are more than one.

The function uses operator < (or *comp*, binary function used for comparison, if provided) to compare the values.

minimax. Funkcija koja vraća i najmanji i najveći element. Returns a pair with the smallest of a and b as first element, and the largest as second. If both are equivalent, the function returns `make_pair(a,b)`.

The versions for *initializer lists* return a pair with the smallest of all the elements in the list as first element (the first of them, if there are more than one), and the largest as second (the last of them, if there are more than one).

```
#include <iostream>     // std::cout
#include <algorithm>    // std::minmax

int main () {
  auto result = std::minmax({1,2,3,4,5});

  std::cout << "minmax({1,2,3,4,5}): ";
  std::cout << result.first << ' ' << result.second << '\n';
  return 0;
}

minmax({1,2,3,4,5}): 1 5
```

min_element. Vraća najmanji element u traženom opsegu.

Returns an iterator pointing to the element with the smallest value in the range [first,last) (or last if the range is empty).

The comparisons are performed using either operator $<$ for the first version, or *comp* for the second; An element is the smallest if no other element compares less than it. If more than one element fulfills this condition, the iterator returned points to the first of such elements.

Parameters:

first, last Input iterators to the initial and final positions of the sequence to compare. The range used is [first,last), which contains all the elements between first and last, including the element pointed by first but not the element pointed by last.

comp Binary function that accepts two elements in the range as arguments, and returns a value convertible to bool. The value returned indicates whether the element passed as first argument is considered less than the second. The function shall not modify any of its arguments. This can either be a function pointer or a function object.

max_element. Returns an iterator pointing to the element with the largest value in the range [first,last).

The comparisons are performed using either operator¡ for the first version, or comp for the second; An element is the largest if no other element does not compare less than it. If more than one element fulfills this condition, the iterator returned points to the first of such elements.

```
// min_element/max_element example
#include <iostream>     // std::cout
#include <algorithm>    // std::min_element, std::max_element

bool myfn(int i, int j) {
  return i<j;
}

struct myclass {
  bool operator() (int i,int j) {
    return i<j;
  }
} myobj;

int main () {
  int myints[] = {3,7,2,5,6,4,9};

  // using default comparison:
  std::cout << "The smallest element is ";
  std::cout << *std::min_element(myints,myints+7) << '\n';
  std::cout << "The largest element is ";
  std::cout << *std::max_element(myints,myints+7) << '\n';

  // using function myfn as comp:
  std::cout << "The smallest element is ";
  std::cout << *std::min_element(myints,myints+7,myfn) << '\n';
  std::cout << "The largest element is ";
  std::cout << *std::max_element(myints,myints+7,myfn) << '\n';

  // using object myobj as comp:
  std::cout << "The smallest element is ";
  std::cout << *std::min_element(myints,myints+7,myobj) << '\n';
  std::cout << "The largest element is ";
  std::cout  << *std::max_element(myints,myints+7,myobj) << '\n';

  return 0;
}
```