**STL**

Standard Template Library, или STL /стандартна библиотека/, е набор от компоненти на С++, които използваме наготово. Те са описани във файлове, които включваме в нашата програма. Доброто овладяване на STL е предпоставка за създаване на значително по-кратки и по-лесно четими програми, като възможностите му позволяват и използването на някои иначе трудоемки за имплементиране алгоритми и структури данни.

Основните компоненти на библиотеката са класове, алгоритми и итератори. Тя осигурява много от основните алгоритми и структури от данни, необходими в работата на програмиста. Тъй като STL е библиотека с общо приложение, почти всички компоненти в нея са шаблони, което позволява да се използват с различни типове данни.

Класовете представляват структури от данни и функции за работа с тях и се наричат контейнери. Основните контейнери които STL предлага са vector, set, multiset, bitset, map, multimap, priority\_queue, queue и stack и т.н.

Основните алгоритми са sort, unique, count, reverse, next\_permutation, lower\_bound, upper\_bound и т.н.

Итераторите служат за достъп до елементите на структурите от данни.

**І. ИТЕРАТОРИ**

Итераторите са специални указатели към елементите на структурата. Декларират се с ключова дума iterator. Насочват се в началото на структурата с обект.begin(), или към края чрез обект.end(), където с обект е означено името на променлива от даден клас. Итераторът end() реално е след последния елемент. Когато итераторът стане равен на end(), това е индикация за край на елементите. Стойността на итераторите се взима със израза \*it. За итераторите са дефинирани операции ++ и --. Обхождането става като вървим от begin итератора към end итератора по този начин:

vector<int> v;

...

for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {

cout << \*it << endl;

}

Итераторите на на класовете string и vector позволяват произволен достъп до елемент от колекцията, например :

vector<int> v(100, 0);

cout << \*(v.begin() + 10) << endl;

Итераторите на колекциите map, set, multiset, multimap не предлагат произволен достъп , те имат предефинирани само операторите ++ и --, чрез които може да се придвижват към следващият или предишният елемент в контейнера .

**ІІ. АЛГОРИТМИ**

**а/ min (a, b), max (a, b)** – връщат минимален/максимален елемент измежду два a и b

Сложност: Константна.

Пример:

int a, b;

cin>>a>>b;

cout << "Min = " << min (a, b)<<endl;

**б/ swap (a, b)** – разменя стойностите на две променливи a и b. Сложност: Константна.

**в/ sort (начало, край)** – сортира в нарастващ ред масив a[] с n елемента или подрежда лексикографски знаците в даден низ. Изисква файл algorithm. Сложност: N\*logN.

Например: sort (a, a+n) – за масив a[n]

sort (s.begin(), s.end()) – за низ s

За сортиране в намаляващ ред или сортиране на структури по зададено поле, трябва да напишем булева функция, която да връща истина, ако елементите са подредени в реда, в който искаме. Тази булева функция става трети аргумент на функция sort.

Пример 1 – сортиране в намаляващ ред:

bool desc (int x, int y)

{

if (x>y) return true;

else return false;

}

int main ()

{

const int n = 5;

int a[] = {1,5, -2, 6, -6};

sort (a, a+n, desc);

for (int i=0; i<n; i++)

cout << a[i]<<endl;

return 0;

}

Пример 2 – сортиране на структура book (заглавие, автор, година, цена) по поле година в нарастващ ред. Условния оператор е заменен само с оператор return.

bool compareyear (book x, book y)

{

return x.year > y.year ;

}

В main(): sort (a, a+n, compareyear);

**г/ reverse (начало, край)** – обръща реда на елементите. Сложност: линейна – О(n).

Пример 1 – обръщане на елементите на масив: reverse(a, a+n);

Пример 2 – обръщане на низ: reverse(a.begin(), a.end());

**д/ count (начало, край, търсен елемент)** – връща броя елементи, равни на дадения. Изисква файл algorithm. Сложност: линейна – О(n).

Пример 1 – брой елементи в масив, равни на 0:

int A[] = { 2, 0, 4, 6, 0, 3, 1, -7 };

const int N = sizeof(A) / sizeof(int); //брой елементи в А

cout << "Number of zeros: " << count(A, A + N, 0) << endl;

Пример 2 – брой букви ‘а’ в низа s:

string s = "alabala";

cout << count(s.begin(), s.end(), 'a') << endl;

**e/ merge (начало1, край1, начало2, край2, нов масив)** – слива два подредени масива в трети – също подреден. Изисква файл algorithm. Сложност: линейна – О(n).

const int n = 5;

int a[] = {1, 2, 3, 9, 12};

int b[] = {0, 2, 5, 6, 10};

int c[2\*n];

merge(a, a+n, b, b+n, c);

for (int i=0; i<2\*n; i++)

cout << c[i]<<endl;

**ж/ binary\_search (начало, край, търсена стойност)** - двоично търсене, което проверява дали елемента е в сортираната последователност. Връща true, ако го има и false, ако го няма. Сложност:Логаритмична.

int main()

{

int A[] = { 1, 2, 3, 3, 3, 5, 8 };

const int N = sizeof(A) / sizeof(int);

for (int i = 1; i <= 10; ++i)

{

cout << "Searching for " << i << ": " << (binary\_search(A, A + N, i) ? "present" : "not present") << endl;

}

}

### з/ search (начало1, край1, начало2, край2) – търси подпоследователност в дадена последователност. Връща итератор към началото на първата подпоследователност или ако няма такава връща итератора на последния елемент на първия параметър. Сложност: Квадратична.

const char S1[] = "Hello, world!";

const char S2[] = "world";

const int N1 = sizeof(S1) - 1;

const int N2 = sizeof(S2) - 1;

const char\* p = search(S1, S1 + N1, S2, S2 + N2);

printf("Found subsequence \"%s\" at character %d of sequence \"%s\".\n", S2, p - S1, S1);

**ІІІ. КЛАСОВЕ**

**1. КЛАСЪТ string**

**2. КЛАСЪТ list**

– линеен двусвързан списък

– елементите се съхраняват на различни места в паметта,

– добавяне / премахване на елементи – на произволно място, константна сложност

– достъп до елементите – пряк до първия, последователен до останалите.

Трябва да включим файл list към програмата.

**а/ дефиниране на списък**  – Понеже list е шаблон, за да дефинираме списък от определен тип, трябва да го посочим по следния начин: list <тип> име;

Например: list <int> sp;

**б/ итератор на списък** – това е специална променлива, която служи за обхождане на списъка. Тя се дефинира така: list <тип>:: iterator име;

Например: list <int>::iterator i;

Инициализираме го с началото на списъка като използваме функция begin по следния начин i=име.begin ();. Тя връща началната позиция на списъка. Например: i=sp.begin ();

Допустими операции с итератора са:

* увеличаване с 1, т.е. i++
* извличане или промяна на стойността, която сочи, с операция \*, т.е. \*i.

**в/ обхождане на списъка** – започвайки от началото, увеличаваме итератора с 1, до достигането на края на списъка, т.е докато i != sp.end(), например:

for ( i=sp.begin (); i!=sp.end (); i++)

cout<< \*i<<endl;

**г/ включване на елемент в списъка**

* в началото – sp.push\_front(ст-ст)
* в края – sp.push\_back (ст-ст);
* преди указаната позиция: име.insert(итератор, стойност),

например: sp.insert (i, 5);

**д/ изтриване на елемент** **от списъка**

* sp.pop\_front() – изтрива първия елемент
* sp.pop\_back() – изтрива последния елемент
* sp.erase(i) – изтрива елемент на указантата от итератора позиция
* sp.clear() – изтрива всички елементи на списъка

**е/ достъп до първия и последния елемент –** sp.front(), sp.back()

**ж/ брой на елементите** – sp.size()

**з/ сортиране на елементите в списъка** – sp.sort();

**и/ обратен ред на елементите** – sp.reverse();

**Пример 1:** Създаване насписък с произволен брой цели числа, извеждане на елементите и техния брой.

#include <iostream>

#include <list>

using namespace std;

int main()

{ list <int> sp;

int a;

while( cin>>a)sp.push\_back (a);

list <int>::iterator i;

for ( i=sp.begin (); i!=sp.end (); i++)

cout<< \*i<<endl;

cout <<"Br. elem. = "<<sp.size ()<<endl;

return 0;

}

**3. КЛАСЪТ stack**

– LIFO (last-in first-out) структура, елементите се добавят и изваждат от края (върха)

– елементите се съхраняват на различни места в паметта,

– достъп до елементите – само до първия – пряк

За да я използваме, трябва да включим файл stack към програмата.

**а/ дефиниране** – stack <тип> име

**б/ поставяне на елементи в стека** – st.push(стойност)

**в/ изтриване на елемент от стека** – st.pop()

**г/ стойност на елемента навърха на стека** – st.top()

**д/ проверка за празен стек** – st.empty() – връща true, ако стека е празен, и false в противен случай. Използва се за обхождане на всички елементи: while(!st.empty())...

**е/ брой на елемента в стека** – st.size()

**Пример.** Дефиниране на стек с цели числа, въвеждане и извеждане на елементите му.

#include <iostream>

#include <stack>

using namespace std;

int main()

{

stack <int> st;

int x;

while (cin>>x) st.push(x);

while( !st.empty() )

{ cout << st.top() << endl;

st.pop();

}

return 0;

}

**4. КЛАСЪТ queue**

– FIFO (first-in first-out) структура – елементите се вмъкват от едната страна и се вадят от другата

– елементите се съхраняват на различни места в паметта,

– достъп до елементите – само до първия и последния – пряк

Трябва да включим файл queue към програмата.

**а/ дефиниране** – queue <тип> q

**б/ поставяне на елементи в опашката** – q.push(стойност)

**в/ изтриване на елемент от опашката** – q.pop()

**г/ стойност на първия елемент от опашката** – q.front()

**д/ стойност на последния елемент от опашката** – q.back()

**е/ брой на елементите в опашката** – q.size()

**5. КЛАСЪТ deque**

– опашка с два края / **d**ouble-**e**nded **que**ue/

– елементи се добавят и изтриват в двата края /push\_front(a), push\_back(a), pop\_front(), pop\_back()/

– Елементите се пазят в строга линейна последователност, разширява се автоматично

– Достъп до елементите – чрез индекс и итератор

Изисква включването на файл deque.

Пример:

#include <deque>

deque<int> mydeque

mydeque.push\_back (5), mydeque.pop\_back()

mydeque.push\_front (10)

deque<int>::iterator it

for ( it=mydeque.begin() ; it < mydeque.end(); it++ ) {…}

mydeque[1]

**6. КЛАСЪТ priority\_queue**

Приоритетната опашка – първият елемент е най-важният, според определената подредба. Стандартно на върха на приоритетната опашка в STL стои най-големия елемент – това поведение може лесно да се промени. Приоритетната опашка представлява реализация на двоична пирамида. Добавянето на елемент в опашката става с метода push, изваждането с pop, проверката дали е празна с empty.

Възможно е на опашката да бъде подаден функционален обект, или някои стандартен функционален обект (съдържащ се в хедъра functional), или да бъде предефиниран операторът < за сравнение, като по този начин се променя начина по които опашката сравнява елементи.

Просто използване на приоритетна опашка изглежда по този начин:

int num;

ifstream fin("chisla.in");

vector<int> v;

priority\_queue<int> pq;

while (fin >> num) pq.push(num);

while (!pq.empty()) {

v.push\_back(pq.top());

pq.pop();

}

В този пример се извършва сортиране на числа (по не особено ефективен начин).

Нека видим как може да променим поведението на priority\_queue, така че на върха на опашката да се намира най-малкия елемент – примерно за числа. Първия начин е с дефиниране на функционален обект:

template<class T>

class MyComparer {

public:

bool operator()(T a, T b) const {

return a > b;

}

};

priority\_queue<int, vector<int>, MyComparer<int> > pq;

Втория начин е с използване на стандартния функционален обект greater, който реално върши същата работа като класа MyComparer по-горе, тогава декларацията

На приоритетната опашка ще изглежда така:

priority\_queue<int, vector<int>, greater<int> > pq;

// важно е да има един интервал между > и >

Друг вариант е да използваме обекти, за които е предефиниран оператора <. Например клас за връх в граф, с който подреждаме върховете в графа по някаква ценова функция (например дължината на път до този връх спрямо някакъв начален връх):

struct Vertex {

int v, c;

Vertex() : v(0), c(0) {}

bool operator<(const Vertex& a, const Vertex& b) const {

return a.c > b.c;

}

};

**7. Класът vector**

Векторът е едномерен динамичен масив с произволен брой елементи.

Тази структура от данни прилича почти изцяло на списък – по същия начин се добавят, премахват, обхождат, извеждат стойностите на елементите. Разликата е в начина на представяне в паметта. При списъка елементите се разполагат на произволно място в динамичната памет и добавянето или премахването на елемент става много лесно. Елементите на вектора се разполагат в блок в динамичната памет. Добавянето на нов елемент и изтриване обикновено става в края, разширява се автоматично. Възможно е да добавим или изтрием елемент на произволно място, но това налага изместване на всички следващи елементи с една позиция, което е много тежка операция. Затова използваме vector когато работим с фиксиран брой елементи, /този брой може да се промени/, и добавянето става в края. Ако се налага вмъкване на произволно място, използваме list.

Друго предимство пред използването на свързан списък за подобна операция е това, че векторът заема много по-малко памет от свързания списък – при него не се пазят указатели.

Трябва да включим файл vector.

При създаването си векторът е празен. За него се заделя блок памет с подходящ размер. Ако елементите станат повече, автоматично се преоразмерява. Това е свързано с копиране на елементите в по-голям блок от паметта и забавя работата. Може при създаването да посочим размера, а при необходимост – да го преоразмерим.

Достъпът до елементите е по индекс (константна сложност) и с обхождане с итератор (линейна сложност).

За векторите са предифинирани оператораторите < и == т.е. те могат да бъдат сравнявани лексикографски и за еднаквост .

**а/ деклариране** – vector <тип> име; или vector <тип> име (брой);

В първия случай създава празен вектор, а във втория – вектор с указания брой елементи със стойност 0.

**б/добавяне на елементи** – v.push\_back (a); insert (it, x);

**в/ достъп до първия и последния елемент** – v.begin () и v.end();

**г/ изтриване** – v.pop\_back (); erase (i); clear();

**д/ брой елементи** – v.size();

**е/ преоразмеряване** – v.resize(n);

**ж/ размер на област от паметта, заделена за елементите на вектора** – v.capacity ();

**з/ заделяне на памет за n елемента** – v.reserve (n); Това означава, че планираме да добавяме до n елемента и предварително им заделяме място.

**и/ сортиране на елементите на вектор – sort(v.begin(), v.end());**

**к/ смяна на реда на елементите на вектор – reverse(v.begin(), v.end());**

Пример – въвеждане и извеждане на елементи във вектор:

#include <iostream>

#include <vector>

using namespace std;

int main ()

{ vector <int> v;

int a;

while (cin >>a)v.push\_back(a);

vector <int> :: iterator it;

for (it=v.begin(); it!=v.end();it++)

cout << \*it<<endl;

return 0;

}

Пример – вектор от вектори – в този случай трябва задължително да зададем размер на вектора от вектори чрез функция resize. При въвеждане трябва да съобразяваме, че трябва да укажем в кой от векторите добавяме нов елемент – трябва да зададем индекса на вектора и той трябва да е от 0 до капацитета на вектора -1:

#include <iostream>

#include <vector>

using namespace std;

int main ()

{

vector <vector <int> > g;

int a,b,n;

cin>>n;

g.resize(n);

for (int i=1; i<=n; i++)

{

cin>>a>>b;

g[a].push\_back(b);

}

return 0;

}

**8. КЛАСЪТ set, multiset**

Множество set се състои от елементи с уникални стойности, а в multiset се допуска въвеждането на повтарящи се елементи. Сложностите на операциите е O(logn). Реализира се като двоично дърво за претърсване.

Трябва да включим файла set в нашата програма: #include <set>.

Елементите които са в set не могат да бъдат променяни – невалидна е конструкция от вида на: set<int>::iterator it = s.begin(); \*it = 5; Решение на проблема е да изтрием елемента от set-a и да добавим нов елемент.

**a/ дефиниране –** set <тип> име; или

set <тип> име (име на дадено множество);

В първия случай създава празно множество, а във втория – създава ново множество, като копира в него елементите от даденото.

**б/ добавяне на елементи в множество** – име.insert(ст-ст). Елементът се добавя в множеството един път, независимо колко пъти ще изпълним оператора.

**в/ обхождане не елементите на множество и извеждане**

set<int>::iterator i;

for (i=m.begin (); i!=m.end (); i++)

cout <<\*i<<endl;

**г/ търсене на елемент в множеството –** име.find(ст-ст)или име.count(ст-ст)

Първата функция връща итератор - ако намери елемент с посочената стойност, връща итератор към него, ако не намери – връща итератор към края (име.end()). т.е. ако итераторът е != от име.end(), значи елементът е намерен.

Вторатa фунцкия count брои колко пъти се среща елементът – 0 или 1 път.

**д/ изтриване на елементи от множеството**

– по зададена стойност – име.erase(ст-ст)

– елемент сочен от итератора – или име.erase(итератор)

– всички елементи – име.clear();

– от един елемент на множеството до друг, като последния остава:

m.erase (m.find (1), m.find(4)); //Ако елементите са били 0, 1, 2, 3, 4, 5, 6, се изтриват 1, 2 и 3.

**е/ брой на елементите –** име.size()

**9. КЛАСЪТ pair**

Класа pair представлява двойка. Например pair<int, double> e двойка от елемент от тип int и елемент от тип double. За да достъпваме първия и втория елемент на двойка използваме членовете first и second на класа. Например:

pair<int, double> pr;

pr.first = 100;

pr.second = 52.5;

Друг вариант за инициализиране на двойка е с функцията make\_pair, например:

pair<int, int> myPoint = make\_pair(100, 20);

За pair класа е предефиниран оператора “<”, което е удобно ако искаме да сортираме елементи от този клас. Сортирането е по първия елемент чрез оператор “<” като при еднаквост се сравнява втория елемент пак със същият оператор.

**10. КЛАСЪТ map, mutlimap**

Контейнерите map и multimap съхраняват елементи с ключ и стойност (съпоставят на ключове съответни стойности). Задължително е ключовете да са сравними. При map ключът е уникален, а multimap позволява повтарящи се ключове.

**а/ деклариране** – map<тип на ключ, тип на стойност> mymap;

Например: map<int,string> mymap;

**б/ директен достъп до елементите по ключ** чрез използването на предефинирания оператор []. Достъпването не ключ, който не се намира в map-a предизвиква добавянето му като на него се съпоставя default стойността на елемента. Затова е препоръчително, ако не сме сигурни дали даден ключ съществува в map, да се проверява.

Например:

mymap[1]= "Anabel"; //добавя елемент с ключ 1 и указаната стойност

cout << mymap[1]<< endl; //извежда стойността Anabel

cout << mymap[2]<< endl; //тъй като няма елемент с такъв ключ, в момента на достъпване го добавя със стойност празния низ и го извежда.

**в/ добавяне на елементи**

- чрез директно присвояване

Например: mymap[1]= "Anabel";

string s; cin >> s; mymap[2]=s;

- чрез функция insert, но тъй като елементите на map-a са двойки, като параметър на функцията трябва да е двойка. Затова използваме make\_pair.

Например: mymap.insert(make\_pair(3,"Daniel"));

Ако съществува елемент с указания ключ, в map стойността му не се променя.

Например: mymap.insert(make\_pair(3,"Maria")); //няма да промени Daniel

**г/ обхождане и извеждане –**

- чрез итератор it – обръщението към ключа и стойността става съответно с (\*it).first и (\*it).second, или с it->first it->second.

for (it=mymap.begin(); it!=mymap.end(); it++)

//cout << it->first<< ". " <<it->second << endl;

cout << (\*it).first<< ". " <<(\*it).second << endl;

- ако ключът е цяло число и искаме да изведем само стойнсотта, може и както при масивите:

int n=mymap.size();

for (int i=1;i<=n;i++)

cout <<mymap[i]<<endl;

**д/ проверка за празен map –** mymap.empty() – връща истина/неистина

**е/ брой елементи** – mymap.size()

**ж/ брой елементи с указан ключ: mymap.count (ключ)** – връща 1, ако има и 0, в противен случай. В multimap е допустимо повтаряне на ключове, и функцията може да върне число, различно от 0 и 1.

**з/ търсене на елемент по ключ** – mymap.find(ключ) – връща указател към намерения елемент, или указател end, ако няма такъв елемент в map.

**и/ изтриване на елемент –** mymap.erase (итератор), mymap.erase (ключ) или mymap.erase (указател към началото на зона, указател към края на зона) – изтрива елемента, сочен от указател(итератор), или елемента с указания ключ, или елементите в указаната зона.

**й/ изтриване на всички елементи в map –** mymap.clear();

Пример 1: map <int, string>

#include <iostream>

#include <map>

using namespace std;

int main ()

{

map<int,string> mymap;

map<int,string>::iterator it;

mymap[1]="Anabel";

string s;cin>>s; mymap[2]=s;

mymap.insert(make\_pair(3,"Daniel"));

cout <<"Broi elements in mymap: " << mymap.size()<<endl;

cout << "Elements in mymap:" << endl;

for (it=mymap.begin(); it!=mymap.end(); it++)

//cout << it->first<< ". " <<it->second << endl;

cout << (\*it).first<< ". " <<(\*it).second << endl;

cout <<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

int n=mymap.size();

for (int i=1;i<=n;i++)

cout <<mymap[i]<<endl;

return 0;

}

Пример 2: map <string, int > - създава map с ключ от тип стринг, извежда елементите на map-a, търси елемент с указан ключ и извежда неговата стойност.

Ключовете се обхождат в нарастващ ред.

#include <iostream>

#include <map>

using namespace std;

int main ()

{ map<string,int> mymap;

map<string,int>::iterator it;

mymap["Anabel"]=1000;

string s;cin>>s; mymap[s]=2000;

mymap.insert(make\_pair("Daniel",3000));

for (it=mymap.begin(); it!=mymap.end(); it++)

cout << it->first<< ". " <<it->second << endl;

cout <<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

cout <<mymap.find("Daniel")->second <<endl;

return 0;

}

Пример 3. Изтриване на елементи

#include <iostream>

#include <map>

using namespace std;

int main ()

{ map<char,int> mymap;

map<char,int>::iterator it;

mymap['a']=10;

mymap['b']=20;

mymap['c']=30;

mymap['d']=40;

mymap['e']=50;

mymap['f']=60;

it=mymap.find('b');

mymap.erase (it); // erasing by iterator

mymap.erase ('c'); // erasing by key

it=mymap.find ('e');

mymap.erase ( it, mymap.end() ); // erasing by range

for ( it=mymap.begin() ; it != mymap.end(); it++ )

cout << (\*it).first << " => " << (\*it).second << endl;

return 0;

}

Пример 4. Въвежда поредица от думи и извежда честотата на всяка от тях

#include <iostream>

#include <map>

#include <string>

using namespace std;

int main() {

map<string, int> freq; // map of words and their frequencies

string word; // input buffer for words.

//--- Read words/tokens from input stream

while (cin >> word) { freq[word]++; }

//--- Write the count and the word.

map<string, int>::iterator iter;

for (iter=freq.begin(); iter != freq.end(); ++iter) {

cout << iter->second << " " << iter->first << endl;

}

return 0;

}//end main

Пример 5. multimap

#include <iostream>

#include <map>

using namespace std;

int main()

{ multimap<string, int> m;

multimap<string, int>::iterator it;

m.insert(pair<string, int>("a", 1));

m.insert(pair<string, int>("c", 2));

m.insert(pair<string, int>("b", 3));

m.insert(pair<string, int>("b", 4));

m.insert(pair<string, int>("a", 5));

m.insert(pair<string, int>("b", 6));

cout << "Number of elements with key a: " << m.count("a") << endl;

cout << "Number of elements with key b: " << m.count("b") << endl;

cout << "Number of elements with key c: " << m.count("c") << endl;

cout << "Elements in m: " << endl;

for (it = m.begin(); it != m.end(); it++)

cout << "[" << (\*it).first << ", " << (\*it).second << "]" << endl;

m.clear();

if (m.empty()) cout <<"Empty map"<<endl;

else cout <<"Not empty" << endl;

}

**11. КЛАСЪТ bitset, vector<bool>**

Този контейнер е свързан с побитовите операции, всеки негов елемент заема един бит. Предлага удобни методи за побитови манипулации. Задава се чрез bitset<n>, където броят на битовете n e константа. Класа bitset има два конструктора с параметри число или стринг.

Например: bitset<32> a(345534); илиbitset<32> b(“1010101”);

Има предифиниран оператор [] за достъп до отделните битове:

Например: bitset<5> a;

а[1] = 1; a[2] = 0;

Удобен е метода to\_ulong() , който връща представянето на битовете като 32 битово число.

bitset<10> a(5); int x = a.to\_ulong(); // x = 5

a[0] = 0;int x = a.to\_ulong(); // x = 4

bitset също така поддържа различните побитови операции:

bitset<100> a(1), b(“10010”), c(345);

c = a | b; а |= b;

c = a ^ b; a ^= b;

c = a & b; a &= b;

Еквивалент на bitset е vector<bool> - вектор, чийто елементи са битове, т.е. той реализира побитов масив и използва по-малко памет.

**ЗАДАЧИ**

**Зад. 1**. В празен стек(опашка) последователно постъпват елементите 2, 5, 7, 12, 1, 10:

а) Напишете програма, създаваща горния стек(опашка).

б) Да се отпечата онзи елемент от стека(опашката), до който има достъп. Да се изключат два елемента от стека(опашката) и да се отпечата елемента от върха на стека(опашката).

в) Да се добавят елементите 4, 8, да се изключат четири елемента и да се отпечата елемента от върха на стека(опашката).

**Зад. 2.** Да се състави програма, която поставя въвежда низ и го извежда в обратен ред чрез използване на стек.

string p;

cin>>p;

int i=0;

while (p[i]!=0) {

push(p[i]);i++;

};

while (!empty() )cout<< pop();

cout<<endl;

**Зад. 3.**  Напишете програма, която по зададено цяло число, като използва стек, получава число, записано със същите цифри, но в обратен ред.

**Зад. 4.** Напишете програма, която използвайки стек, определя двоичното представяне на едно естествено число *n*.

**Зад. 5.** Напишете програма, която:

а) По въведен от клавиатурата символен низ, съдържащ символите “(“ и “)”, и използвайки стек определя, дали скобите са вложени правилно. Тествайте програмата със следните низове “((( )( )) ( ))”, “((( )( )) ()))( )”.

б) По въведен от клавиатурата символен низ, съдържащ символите “(“, “)”, “[“, “]”, “{“ и “}”, и използвайки стек определя, дали трите вида скоби са вложени правилно. Тествайте програмата със следните низове “{( ) [( )( )] ( )}”, “( ) {[( )( )] ( )}[)”, “{{( ) [( )( )] ( )}”

в) Допълнете програмата от предната подточка, така че при несъответствие на скобите да издава подходящо съобщение за грешка. Тествайте програмата със следните низове

“{( ) [( )( )] ( )}”, “( ) {[( )( )] ( )}[)”, “{{( ) [( )( )] ( )}”, “{( ) [( )( )]} ( )]}”, “{( ) [( )( )] ( )](}”

**Зад. 6.** Напишете програма, която по въведен от клавиатурата символния низ, съдържащ аритметичен израз в постфиксен запис пресмята стойността на израза. Тествайте програмата със следните низове: “5 2 3 \* + 4 1 + –”, “9 3 / 2 \* 2 2 \* - ”, “2 2 2 2 2 \* \* + /”.

Алгоритъм:

- Четете израза отляво - надясно

- Когато срещнете операнд, поставете го в стека

- Когато срещнете операция, извадете два елемента от стека и извършете операцията

- Получения резултат върнете в стека

Решение:

string p;

int a,res,x,y;

cin>>p;

int n=p.length();

int i=0;

while (i<n)

{

if(p[i]>='0'&&p[i]<='9')

{ a=p[i]-'0';push(a);}

else

{ x=pop(); y = pop();

switch(p[i])

{ case '+': res=x+y; break;

case '-': res=y-x; break;

case '\*': res=x\*y; break;

}

push(res);

}

i++;}

if (!empty() )cout<<pop()<<endl;

**Зад. 7.** Напишете програма, която по въведен от клавиатурата символния низ, съдържащ логически израз в постфиксен запис пресмята стойността на израза. Тествайте програмата от предната подточка със следните низове: “0 1 0 v v 1 1 ^ v”, “1 0 1 0 1 0 v ^ v ^ v”, “1 0 ^ 1 v 1 ^ 0 v”.

**Зад. 8.** Да се състави програама, която при въвеждане на символен низ, състоящ се от символите “0” и “1”, използвайки стек определя, дали броят на единиците е равен на броя на нулите.

**Зад. 9.** Да се състави програма, която пресмята изрази от вида 6 7 *m* 5 8 2 *M* *m* *M*, където *m* е по-малкото от двете числа, а *M* е по-голямото.

**Зад. 10.** Да се състави програма, която пресмята изрази от вида 6 6 *A* 6 *G* 2 *A*, където *A* е средното аритметично на двете числа, а *G* е средното геометрично.

**Зад. 11.** Да се състави програма, която пресмята аритметични изрази от вида –( +(5,\*(2,3)), +(4,1)).

**Зад. 12.** Да се състави програма, която пресмята изрази от вида *M* ( *m* ( *M* (12,8), 5), *m* ( 6,7) ), където *m* е по-малкото от двете числа, а *M* е по-голямото.

**Зад. 13.** Да се състави програма, която пресмята логически изрази от вида *о а* 1 1 *о о* 0 1 0, където *а* е конюнкция, *о* - дизюнкция.

**Зад.14.** Да се намери броят на елементите на опашка.

**Зад.16.** Да се определи дли опашка от числа е наредена по намаляване на елементите й.

**Зад.17.** Даден е текст, балансиран по кръгли скоби. Като се използват опашка или стек да се да се определи за всяка двойка съответстващи си отваряща и затваряща скоба, позициите им в текста, подредени по нарастване на позициите на:

А) затварящите скоби;

Б) отварящите скоби.

Например за текста A+(TP-F(X)\*(B-C)) резултатът е:

А) 8, 10; 12 16; 3 17;

Б) 3 17; 8 10; 12 16.

**Зад.18.** Да се въведат n думи и да се преброи колко пъти се среща всяка дума.

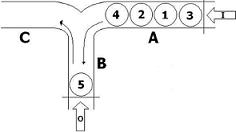
Упътване: Използвайте map. Де ключ, типа на ключа – string

Броя срещания е стойност, тип на стойността - int

**Зад.19.** Да се въведат n на брой рождени дати и да се изведе колко рожденика има във всеки месец.

Упътване: Използвайте map. Месец /ключ/ – брой дни /стойност/

**Зад.20.** Великият професор **Lotocus** измислил изобретението на картинката по-долу, което има следните функции:

* Отначало, **n** топки, номерирани с числата от **1** до **n** постъпват в областта **A** по случаен начин;
* Ако натиснем бутон **I,** първата топка от областта **A** влиза в област **B**;
* Ако натиснем бутон **О**, първата топка от област **B** се качваа в област **C**.

**Вход:**

* На първи ред се въвежда положително, различно от нула цяло число **n**;
* Вторият ред съдържа **n** положителни, различни от нула числа, разделени с интервал, изобразяващи подредбата в област А
* Третият ред съдържа **n** положителни, различни от нула числа, разделени с интервал, изобразяващи финалната конфигурация на топките в областта **C**.

|  |  |  |  |
| --- | --- | --- | --- |
| Вход | Изход | Вход | Изход |
| 5  5 4 2 1 3 2 4 1 5 3 | IIIOOIOOIO | 5  5 4 2 1 3 1 4 2 5 3 | imposibil |

**Изход:**

Да се изведе единствен ред , съдържащ символите **I** и **O** показващи последователността на нтискане на двата бутона, така че от стартовото разположение на топките, да се получи финалното. Ако не е възможно това, да се изведе съобщението **imposibil**.

**1 <= n <= 2000**

**Пример:**

**Зад. 21. / C3. ДУМИ/** На един голям лист хартия били написани *N* думи (1 < *N* < 1000), всяка на отделен ред. Някои от думите се срещали повече от веднъж. Иванчо задраскал някои от думите, така че останали възможно най-много неповтарящи се думи.

Напишете програма **WORDS**, която извежда на стандартния изход броя на останалите думи. Програмата трябва да прочете данните от стандартния вход. На първия ред е зададено числото *N* и на всеки от следващите *N* реда е записана по една дума с дължина до 100 знака, които са измежду малките латински букви.

**ПРИМЕР Вход Изход**

5 3

aaa

bbb

aaa

c

c

**Зад. 22. /ПОДДУМИ НОИ – 21.01.2007/**

Разглеждаме думи, образувани от главни латински букви. Последователност от една или повече съседни букви в думата наричаме „поддума”. Например, някои от думите, които са поддуми на думата TAPATOP са: T, A, APAT, TOP, TAPATOP. Напишете програма **SUBWORDS**, която намира броя на ***различните*** поддуми, които съдържа дадена дума.

От стандартния вход се въвежда един ред, който съдържа една дума с дължина, не по-голяма от 100. На стандартния изход да се изведе един ред с едно естествено число – броя на различните поддуми във въведената дума.

**ПРИМЕР Вход Изход**

CTPACT 18

***Обяснение:*** В думата CTPACT има 18 различни поддуми: C, T, P, A, CT, TP, PA, AC, CTP, TPA, PAC, ACT, CTPA, TPAC, PACT, CTPAC, TPACT и CTPACT.

#include <iostream>

#include <set>

#include <string>

using namespace std;

int main()

{ set <string> m;

string a;

cin >> a;

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

for(int j=1;j<a.size()+1; j++)

m.insert(a.substr(i,j));

cout << m.size();

return 0;

}

**Зад. 23. ТРИК / ЗМС 2011, автор: Николина Николова/**

Вероятно са ви въвличали в следния фокус-игра: *Намисли си едно число. А сега го събери с 3. Умножи го по 12 ...Колко получи? Значи... намисленото от теб число е ...*

Как „фокусникът“ познава кое е намисленото число? Много просто! Смята много бързо наум! :-)

Е, защо да смята като има кой да му го сметне! Напишете програма **trick**, която по дадена последователност от операции и краен резултат намира намисленото число.

**Вход:** От стандартния вход се въвежда последователност от числа или знаците **+, –** , **\*, =.** На всеки ред има точно едно число или знак за операция като числата и операциите се редуват. Последователността започва със знак за аритметична операция и завършва с число, предшествано от знака **=**.

**Изход:** В единствения ред на стандартния изход се извежда едно число – първоначално намисленото.

**Ограничения:** Крайният и всички междинни резултати могат да се представят с 32-битово цяло

число. Входната последователност се състои от не повече от 200 000 реда.

**Пример: Вход Изход**

+ 13

21

-

3

\*

4

=

124

**Решение:** Трябва в обратен ред да извършим противоположните изчисления. Затова ще използваме стек – един за числата, и втори – за знаците за аритметични операции. Въвеждаме едновременно знак и число и докато знака е различен от „=” ги поставяме в съответния стек. Последното въведено число след знака „=” ще е първия аргумент на операцията. Самата операция и втория аргумент прочитаме от стековете и в зависимост от операцията извършваме противоположното действие. Накрая извеждаме получената стойност.

#include <iostream>

#include <stack>

using namespace std;

int main()

{

stack <int> ch;

stack <char> zn;

char z;

int n, a;

while (cin>>z>>a && z!='=')

{

zn.push(z);

ch.push(a);

}

n = a;

while(!ch.empty())

{

a=ch.top();

ch.pop();

z=zn.top();

zn.pop();

if (z=='\*')n=n/a;

if (z=='+')n=n-a;

if (z=='-')n=n+a;

}

cout << n <<endl;

return 0;

}

**Зад. 24. КОНТЕКСТНО ТЪРСЕНЕ /ЗМС 2008/**

Всички съвременни интернет-търсачки показват резултатите от търсенето в контекст. Например, ако търсим с думата „състезания", търсачката извежда реда

**Зимните математически състезания ще се проведат в**

откъдето разбираме, че става въпрос именно за зимните математически състезания. Напишете програма CONTEXT, която по зададен текст и ключова дума извежда всички нейни срещания, заедно с контекста.

**Вход**: От първия ред на стандартния вход се въвежда едно число N. От втория ред се въвежда ключова дума или фраза, която може да съдържа интервал, латински букви, цифри, препинателни и математически знаци. От третия ред се въвежда текст, който може да съдържа интервали, латински букви, цифри, препинателни и математически знаци. Ако в началото и края на фразата и текста има интервали, те се игнорират.

**Изход**: Всяко срещане на ключовата дума или фраза се извежда заедно с контекста си на отделен ред, като сравнението в текста пренебрегва разликата между малки и главни букви. Дължината на всеки ред е равна на 2N + дължината на ключовата дума или фраза. За целта от двете страни на ключовата дума или фраза се добавят по N или края на текста) за някоя от страните, символите се допълват от другата страна на ключовата фраза до нужния размер. В случай, че текстът е по-кратък от 2N + дължината на ключовата дума или фраза, да се изведе целият текст.

Ограничения:0 < N < 100, Ключова дума - не повече от 50 символа Текст - не по-дълъг от 65 000 символа

**Пример:Вход**: 8

\_SNoW

\_\_\_Snow is a type of precipitation in the form of crystalline water ice, consisting of a multitude of snowflakes that fall from clouds. Since snow is composed of small ice particles, it is a granular material. Many winter sports, such as skiing and snowboarding depend on snow...\_\_\_

**Изход:**

Snow is a type of pr

tude of snowflakes t

. Since snow is comp

ing and snowboarding

ng depend on snow...

**Решение**: Решението на задачата се свежда до въвеждане на низовете – ще използваме тип string и ще ги въвеждаме с getline. Трябва да игнорираме Enter-а след числото на първия ред на входа. За да отстраним интервалите в началото и края използваме вградени функции, но може и с цикъл –докато срещнем символ различен от интервал. Другият момент в решението е да преобразуваме само в малки букви, като пазим оригиналния низ, за да извеждаме от него. Обхождаме дадения текст, извличаме подниз от текущата позиция с толкова знаци, колкото е дължината на търсения низ и сравняваме с търсения. Накрая трябва да съобразим, ако вляво или вдясно от намерения низ позициите са по-малко от n, да ги допълним от другия край на намерения низ, като трябва да изчислим с колко точно ще допълним.

#include <iostream>

#include <string>

using namespace std;

string a,b, alower, blower;

int main()

{ int n, i, j, sa, sb, ss;

cin>>n;cin.ignore();

getline(cin,a); getline(cin,b);

i=a.find\_first\_not\_of(" ");

j=a.find\_last\_not\_of(" ");

a = a.substr(i, j-i+1);

sa=a.size();

for (i=0;i<sa; i++)alower=alower+char(tolower(a[i]));

i=b.find\_first\_not\_of(" ");

j=b.find\_last\_not\_of(" ");

b = b.substr(i, j-i+1);

sb=b.size();

for (i=0;i<sb; i++)blower=blower+char(tolower(b[i]));

for(i=0;i<=sb-sa;i++)

if(alower==blower.substr(i,sa))

if(i<n)cout<<b.substr(0,sa+2\*n)<<endl;

else

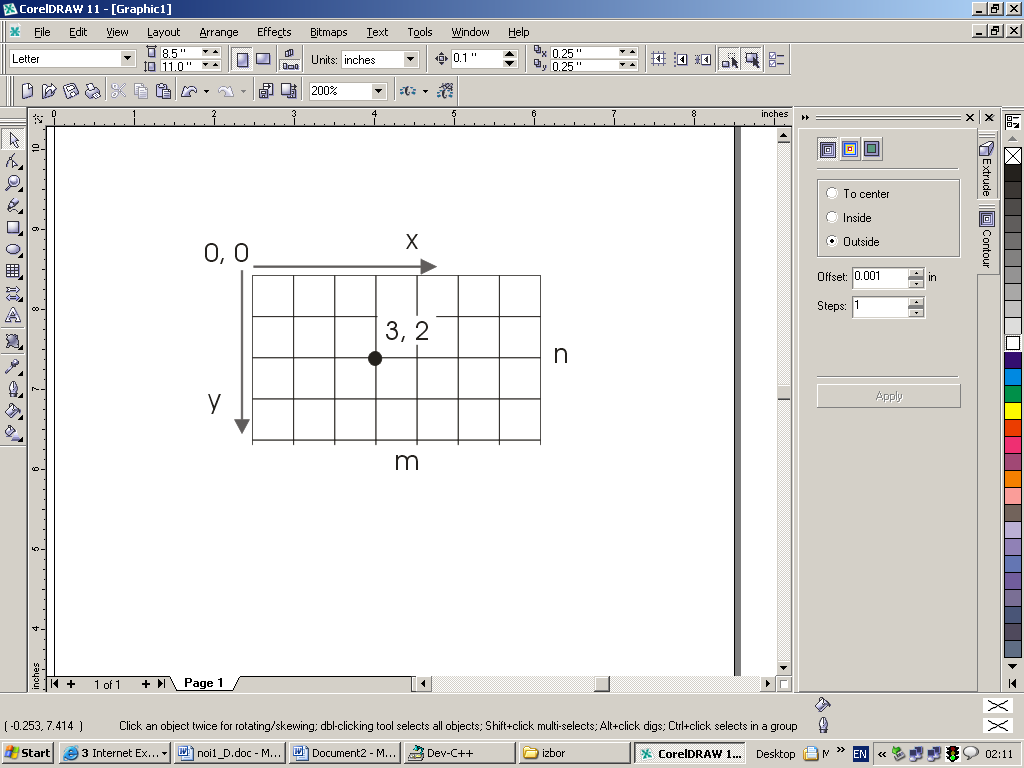
if(i+sa+n>sb) cout<<b.substr(sb-sa-2\*n,sa+2\*n)<<endl;

else cout<<b.substr(i-n,sa+2\*n)<<endl;

return 0;

}

**Зад. 25. /D3. ОТБОРИ – НОИ – 21.01.2007/**

Една моментна снимка от сателит показва разположението на играчите на терена в една нова игра „CF”. В тази игра има много играчи от различни отбори, които са с различни цветове екипи и също такива по цвят каски. Тази снимка на игралното поле е съхранена в “графичен” файл. Играч на отбор, който поддържате се намира с топката в долния десен ъгъл на терена. Вашата задача е да напишете програма **CF**, която намира колко отбора участват в играта. Ако играч от друг отбор е застрашил любимеца ви и се намира до него вляво изведете цвета му. Ако там се намира играч от същия отбор изведете 0, ако няма никакъв друг играч точно вляво изведете -1.

Един играч е най-близко вляво до любимеца ви, ако те имат една и съща координата по y и са на най-малко разстояние един от друг. Сигурно е, че никои двама играчи не се намират в този момент на едно и също място.

**Вход:** Въвежда се разположението на играчите на игралното поле. На първия ред N (N<100) – брой играчи без любимеца ви. На втория ред размерите на игрището m и n (1m<640, 1n<480).

На всеки следващ ред за всеки играч: се въвеждат по три цели числа мястото му по x, по y и цвета му c (0xm, 0yn, 0c255), отделени с един интервал.

На последния ред цвят на любимеца ви.

**Изход:** На първия ред се извежда число, показващо броя на отборите, които участват в играта. На втория ред се извежда едно число -1, 0 или цвят, показващо вида на играча най-близко до любимеца ви вляво.

**Примери**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Пример 1 | | Пример 2 | | Пример 3 | |
| Вход | 6  3 4  1 0 4  1 2 4  1 3 7  2 1 7  2 3 4  3 2 7  8 |  | 2  2 2  0 1 5  0 2 4  4 |  | 6  3 3  1 0 4  1 2 4  1 3 7  2 1 7  2 3 4  3 2 7  7 |  |
| Изход | 3  -1 | 2  0 | 2  4 |

Решение – Галина Момчева

Данните за всеки от играчите се съхраняват в масива t – представляващ полето за игра. Ако няма играч на дадени координати, то стойността на съответната клетка от t e -2, а ако има играч, то стойността на t е номера на цвета на екипа му. Преброяването на отборите е по-лесно, ако се използва структурата множество, в която поставяме различните цветове.

Намирането на играча отляво може да се получи с обхождане на n-тия ред, от играча в ъгъла - наляво. Ако се намери клетка със стойност различна от -2, то сме открили играч отляво и може да се изведе подходящата стойност – 0, ако е от същият отбор, или цвета на противника. Ако целият ред до началото има нулеви стойности, то никой не заплашва любимеца ни отляво.

#include <iostream>

#include <set>

using namespace std;

int main()

{ int N, k, i, m, n, j, col, fl=0; //в началото игрището е празно

int t[640][480];

for (i=0; i<640; i++)

for (j=0; j<480;j++)

t[i][j] = -2;

typedef set<int,greater<int> >IntSet;//дефиниране на структурата множество

IntSet set1;

cin >> N; cin >> m >> n;

for (k = 0; k < N; k++)

{ cin >> i;

cin >> j;

cin >> t[i][j];

set1.insert(t[i][j]);

}

cin >> t[m][n];

set1.insert(t[m][n]);

cout << set1.size() << endl; //извеждане на броят на отборите

//намиране на първият отляво

for (i=m-1; i>=0; i--)

if (t[i][n]!=-2)

if (t[i][n] == t[m][n]) {cout << 0; fl=1; break;}

else {cout << t[i][n]; fl=1;break;}

if (!fl) cout << - 1<< endl;// не е намерил никого вляво

return 0;

}

**Зад. 25. ДИНАСТИИ /НОИ2 – 2012, Задача С1, Автор: Емил Келеведжиев/**

Историци събирали данни за средновековни династии. Били оформени картончета, във всяко от които били записани по две имена на благородници – на баща и негов син. За съжаление, работата не била извършвана много прилежно и в получената картотека се срещали грешки и повторения. Все пак, било проверено, че в картотеката няма случаи, в които може да се окаже, че един човек е наследник на себе си. Напишете програма **dynasty**, която проверява дали в картотеката има грешки от вида: един син да е записан с различни имена на баща в две картончета. Ако няма такива грешки, програмата трябва да изведе името на благородник, който е оставил най-голям брой поколения. Според картотеката, броят на тези поколения не е по-голям от 100. Ако има повече от един такъв благородник, програмата трябва да изведе този, които има последно по азбучен ред име. Считаме, че различните хора са с различни имена.

**Вход.** На първия ред е записан броя на картончетата. Те могат да бъдат най-много 10 000 броя. За всяко картонче следва ред със записани два низа, всеки съставен от най-много 5 малки латински букви – имена на баща и син.

**Изход.** Името на първия открит син по реда на записите във вход, за който има картончета с различни имена на баща му. Ако не е открита такава грешка, програмата трябва да изведе името на описания по-горе благородник.

|  |  |
| --- | --- |
| **Пример 1**  **Вход**  8  aa ba  ga ha  ca aa  ca da  ia ca  ha bc  bc bd  aa ha  **Изход**  ha  Пояснение: ha е изведено, защото ha има две различни имена на бащи. | **Пример 2**  **Вход**  10  aa ba  ga ha  de da  ca aa  ia ca  ha bc  bc bd  iaa de  da vo  da vo  **Изход**  iaa  Пояснение: iaa има 3 последователни наследници: de, da и vo. Толкова наследници имат ia и ga, но имената им са преди името iaa по азбучната наредба. |

Решение – Емил Келеведжиев

Първата част на задачата може да се реши с метод на грубата сила, като за всеки син програмата провери, дали той се среща в два реда от входния файл, в които първите компоненти са различни. По този начин са необходими проверки от порядъка на *n*3, където *n* e броя на записите. В авторското решение тази проверка е реализирана с използване на map от STL, с което се постига брой на операциите от порядъка на *n* \* log *n*.

Втората част на задачата е решена, като за всеки син се намира неговият баща, след това – за този баща се намира неговият баща и т.н., докато се достигне случай, когато поредният баща не може да бъде намерен. Тогава името на последния намерен баща и съответната кратност се записват като двойка във вектор от STL. Накрая се намира най-големият елемент (двойка) в използвания вектор. Броят на операциите е от порядъка на *n*2 \* log *n*.

#include<iostream>

#include<map>

#include<vector>

#include<algorithm>

using namespace std;

int m;

map<string,string> p;

map<string,string>::iterator it;

vector<pair<int,string> > v;

void up(string b)

{ int c=0;

while(p.count(b)>0) {b=p[b]; c++;}

v.push\_back(pair<int,string>(c,b));

}

int main()

{cin >> m;

for(int i=1;i<=m;i++)

{ string a,b;

cin >> a >> b;

if(p.count(b)>0) if(p[b] != a) {cout << b << endl; return 0;}

p[b]=a;

}

for(it=p.begin();it!=p.end();it++) up((\*it).first);

cout << (\*max\_element(v.begin(),v.end())).second << endl;

}