# **Стандарттуу шаблондордун китепканасы**

Стандарттуу шаблондордун китепканасы белгилүү жана кеңири колдонгон алгоритмдерди жана класстарды ишке ашыруу үчүн стандарттуу класстарды жана функцияларды камтыйт. Мисалы, стандарттуу китепканада бизге белгилүү болгон векторлор, тизмелер, кезектер жана стектер иштөө функциялары бар. Ошондой эле ал жерде мындай түзүмдөргө жетүү процедуралары аныкталган.

## **Стандарттуу шаблондор китепканасы менен таанышуу**

Стандарттуу шаблондордун китепканасы өтө чоң жана анын синтаксиси өтө татаал болгондуктан, ал кандайча түзүлгөндүгү жана кандай элементтерден турганын түшүнүү керек. Стандарттуу шаблондор китепканасынын маңызы үч элементтен турат: контейнер, алгоритм жана итератор. Бул элементтер өз ара тыгыз байланышта иштейт.

Башка объектерди сактоо үчүн колдонулган объектерди ***контейнер*** д.а. Контейнерлер ар кандай типте болот. Мисалы, ***vector*** (вектор) классында динамик массиви, ***queue*** (кезек) классында кезек, ***list*** (тизме) классында тизме аныкталат. Базалык контейнерлерден сырткары стандарттуу шаблондор китепканасында ***ассоциатив контейнерлер*** аныкталган. Ассоциатив контейнерлерде ачкыч/маани түгөйлөр сакталат. Мындай контейнерлер ***ачкычтардын*** (*keys*) жардамы менен ал жерде сакталган маанилерди тез алууга мүмкүндүк берет.

Ар бир контейнер-класста ал контейнер менен иштөө үчүн функциялардын тобу бар. Мисалы, тизмеде элементтерди кыстаруу, алып салуу жана кошумчалоо функциялары бар. Стекте элементти стекке жайгаштыруу жана стектен алуу функциялары бар.

***Алгоритмдер*** контейнерде жайгашкан элементтер менен амалдарды аткарат. Контейнердин мазмуну алдын-ала аныктоо, иреттөө, издөө жана алмаштыруу алгоритмдерди камтыйт. Көпчүлүк алгоритмдер удаалаштык менен иштөөгө багытталат. Удаалаштык контейнердин ичинде жайгашкан элементтердин сызыктуу тизмеси.

Контейнерлерге карата көрсөткүчтөрдүн ролун аткарган объектерди ***итератор*** д.а. Массивтин элементтерине көрсөткүч аркылуу кайрылгандай, итераторлор дал ошондой контейнердин мазмунуна кайрыла алат. Итераторлордун беш тиби:

|  |  |
| --- | --- |
| **Итератор** | **Байандамасы** |
| Эркин кайрылуу (random access) | Маанилерди окуу жана жазуу үчүн колдонулат. Элементтерге эркин кайрылат. |
| Эки багыттуу (bidirectional) | Маанилерди окуу жана жазуу үчүн колдонулат. Контейнерде эки багытта жүрө алат. |
| Бир багыттуу (forward) | Маанилерди окуу жана жазуу үчүн колдонулат. Контейнерде бир багытта жүрө алат. |
| Киргизүү (input) | Маанилерди окуу үчүн колдонулат. Контейнерде бир багытта жүрө алат |
| Чыгаруу (output) | Маанилерди жазуу үчүн колдонот. Контейнерде бир багытта жүрө алат |

Бул жерде *киргизүү* жана *чыгаруу* итераторлору тиешелүү түрдө маалыматты контейнерден киргизүү жана контейнерге чыгаруу дегенди түшүндүрөт.

Итераторлор менен көрсөткүчтөргө окшош эле иштесе болот. Алар үстүнөн инкремент, декремент амалдары аткарылат жана \* амалы колдонулат. Итератордун тибин ар кайсы контейнерде аныкталган ***iterator*** тиби катары жарыяланат.

Стандарттуу шаблондордун китепканасында ***тескер итераторлор*** бар. Мындай итераторлор удаалаштыкты тескер багытта өтөт.

Шаблондорду байандаганда ар кайсы типтеги итераторлор үчүн төмөнкү терминдерди колдонобуз:

|  |  |
| --- | --- |
| **Термин** | **Итератордун тиби** |
| Randlter | Эркин кирүү(random access) |
| Bilter | Эки багыттуу (bidirectional) |
| Forlter | Бир багыттуу (forward) |
| Inlter | Киргизүү (input) |
| Oulter | Чыгаруу (output) |

Контейнерлер, алгоритмдер жана итераторлордон сырткары стандарттуу шаблондор китепканасында кошумча дагы бир нече стандарттуу компоненттер камтылган. Алардын негизгилери эс бөлүштүрүүчүлөр, предикаттар жана салыштыруу функциялары.

Ар бир контейнер *эс бөлүштүрүүчүгө* ээ жана контейнерге эсти бөлүү процессин башкарат. Унчукпаганда эс бөлүштүргүч катары *allocator* классынын объектиси алынат. Бирок колдонуучу атайын эс бөлүштүргүч аныктап, ага кошумча милдеттерди жүктөй алат.

Кээ бир алгоритмдерде жана контейнерлерде *предикат* д.а. өзгөчө түрдөгү функциялар колдонулат. Предикат унардык же бинардык болушу мүмкүн. Унардык предикат бир, бинардык эки аргументке ээ. Бул функциялар чын же жалган деген маани кайтарып берет. Унардык предикаттар *UnPred* типке, бинардык *BinPred* типке ээ болушат. Бинардык предикаттын аргументтери ирети менен жайгашат:*биринчи, экинчи.* Унардык жана бинардык предикаттардын аргументтери контейнерде сакталган объектердин тибине дал келет.

Кээ бир алгоритмдерде жана класстарда эки элементти салыштыруу үчүн колдонулган атайын типтеги бинардык предикат кездешет. Мындай *предикатты салыштыруу функциясы* д.а. Эгерде биринчи аргумент экинчи аргументтен кичине болсо, функция чын деген маани кайтарат. Салыштыруу функция *Comp* тибинде болот.

Шаблон-класстарды колдоо үчүн Си++ тилинин стандарттуу китепканасы **<***utility***>** жана **<***functional***>** файлдарын камтыйт. Мисалы, **<***utility***>** файлы *pair* (түгөй) шаблон-классын аныктамасын камтыйт. **<***functional***>** файлында төмөндө аталган бир катар объект-функцияларды аныктайт:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| plus | divides | equal\_to | greater\_equal | logical\_and |
| minus | modulus | not\_equal\_to | less | logical\_or |
| multiplies | negate | greater | less\_equal | logical\_not |

## **Контейнер класстар**

Төмөнкү жадыбалда стандарттуу шаблондор китепканасында аныкталган контейнерлер аталган:

|  |  |  |
| --- | --- | --- |
| **Контейнер** | **Байандама** | **Файл** |
| bitset | Биттердин көптүгү | <bitset> |
| deque | Эки жактуу кезек | <deque> |
| list | Сызыктуу тизме | <list> |
| map | Түгөйлөрдү сактоо үчүн ассоциатив тизме. Ар бир ачкыч менен бир эле маани байланат. | <map> |
| multimap | Түгөйлөрдү сактоо үчүн ассоциатив тизме. Ар бир ачкыч менен бир же бир нече маани байланат. | <map> |
| multiset | Элементтери сөзсүз түрдө уникалдуу болбогон көптүк | <set> |
| priority\_queue | Биринчилиги бар кезек | <queue> |
| queue | Кезек | <queue> |
| set | Элементтери уникалдуу болгон көптүк | <set> |
| stack | Стек | <stack> |
| vector | Динамик массиви | <vector> |

Шаблон-класстын жарыялоосуна кирген элементтердин типтери ар кандай болгондуктан, класс-контейнерлерде *typedef* кызматчы сөзү аркылуу ал типтердин макулдашылган типтери жарыяланат. Төмөндө *typedef* кызматчы сөзү аркылуу айкындалган жыш кезиккен типтер келтирилген:

|  |  |
| --- | --- |
| **Макулдашылган типтин ысымы** | **Байандамасы** |
| size\_type | Интегралдуу тип, *size\_t*типке дал келет |
| reference | Элементке шилтеме |
| const\_reference | Элементке турактуу шилтеме |
| iterator | Итератор |
| const\_iterator | Турактуу итератор |
| reverse\_iterator | Кери итератор |
| const\_reverse\_iterator | Турактуу кери итератор |
| value\_type | Контейнерде сакталган маани тиби |
| allocator\_type | Эс бөлүштүргүч тиби |
| key\_type | Ачкыч тиби |
| key\_compare | Эки ачкычты салыштырган функция тиби |
| value\_compare | Эки маанини салыштырган функция тиби |

## **Векторлор**

Эң популяр контейнер болуп вектор эсептелет. *vector* классы динамик массивтерин колдойт. Зарыл болгон учурда өлчөмүн өзгөрткөн массивди *динамик массиви*деп аташат. Белгилүү болгондой Си++ тилинде компилөө учурунда массив аныкталган өлчөмдө болот. Мындай жол массивди аныктоонун эффективдүү ыкмасы болгону менен, ал чектелген, себеби програм аткарылып жаткан учурда массивдин өлчөмүн учурдун талабына жараша өзгөртө албайт. Бул маселени вектор чечет. Вектор динамик массиви болгону менен, элементтерине кайрылуу үчүн жөнөкөй индекстүү нотация колдонот. Төмөндө *vector* классы үчүн шаблондун спецификациясы келтирилген:

template <class T, class Allocator=allocator<T>> class vector

мында Т – контейнерде сакталуучу берилиштердин тиби, *Allocator* кызматчы сөзү эс бөлүштүргүчтү аныктайт. *vector* классында төмөнкү конструкторлор аныкталган:

explicit vector(const Allocator &a=Allocator());  
explicit vector(size\_type сан, const T & маани=T(), const Allocator &a=Alloator());  
vector (const vector<T, Allocator> &объект);  
template<class InIter>vector(Inlter башы<Inlter айагы>, const Allocator &a=Allocator());

Биринчи форма бош вектордун конструктору. Конструктордун экинчи калыбында элементтердин саны *сан* аркылуу аныкталат, ал эми ар бир элемент *маани* маанисине барабар. Конструктордун үчүнчү калыбында вектор бирдей элементер үчүн колдонот, ар бир элемент *объект*менен аныкталат. Конструктордун төртүнчү калыбы элементтердин арымын камтыйт. Арым *башы* жана *айагы* менен аныкталат.

Вектордо сакталган ар бир объект үчүн унчукпаган конструктор аныкталыш керек. Андан сырткары, объект үчүн **<** жана **==** операторлору аныкталыш керек. Кээ бир компилегичтер үчүн башка салыштыруу операторлорун аныктоого туура келип калышы мүмкүн. Берилиштердин кыстарылган типтери үчүн аталган талаптар автоматтык түрдө аткарылат.

Синтаксиси татаал көрүнгөнү менен, векторду жарыялоодо эч кандай кыйынчылык болбойт. Төмөндө андай жарыялоолорго бир нече мисал келтирилген:

|  |  |
| --- | --- |
| vector <int> vektor;  vector<char> char\_vektor(5);  vector<char>char\_vektor(5,’z’);  vector<int> int\_vektor(vektor); | /\*узундугу нөл болгон бүтүн типтер үчүн вектор\*/ /\*5 элементтен турган белгилер үчүн вектор\*/ /\*белгилер үчүн вектор жана аны аныктоо\*/ /\*Бүтүн векторлордон бүтүн векторду түзүү\*/ |

*vector* классы үчүн төмөнкү салыштыруу операторлору аныкталат:

**==, < , <=, !=, >, >=**

Ошондой эле *vector* классы үчүн индекс**[]** оператору аныкталат. Ал вектордун элементтерине кадимки индекстүү нотация аркылуу кайрылууга мүмкүндүк берет.

Төмөнкү жадыбалда **vector** классынын мүчө-функциялары келтирилген.

|  |  |
| --- | --- |
| Мүчө-функция | Байандамасы |
| template<class Inlter>void assign(Inlter,башы Inlter айагы); | башы, айагы итераторлор менен аныкталган удаалаштыкты векторго менчиктейт |
| template<class Size, class T> void assign (size сан, const T&маани =T); | Векторго сан элементти менчиктейт, ар бир элементтин мааниси мааниге барабар. |
| reference at(size\_type i);  const reference at(size\_type i) const; | i параметри менен берилген элементке шилтемени кайтарат. |
| reference back();  const\_reference back() const; | Вектордун акыркы элементине шилтемени кайтарат. |
| iterator begin();  const\_iterator begin(); | Вектордун биринчи элементинин итераторун кайтарат. |
| size\_type capacity() const; | Вектордун учурдагы сыйымдуулугун кайтарат. |
| void clear(); | Вектордун бардык элементтерин жойот |
| bool empty() const; | Чакырылуучу вектор бош болсо, чын деген маани, болбосо *жалган* деген маани кайтарат. |
| iterator end();  const\_iterator end() const; | Вектордун айагынын итераторун чакырат. |
| iterator erase(iterator i); | i итератору шилтеген элементти жойот. Жойулган элементтен кийин турган элементтин итераторун кайтарат. |
| iterator erase (iterator башы, iterator айагы); | *башы, айагы* итераторлордун арасында жайгашкан элементтерди жойот. Акыркы жойулган элементтен кийин турган элементтин итераторун кайтарып берет. |
| reference front();  const\_ reference front() const; | Вектордун биринчи элементине шилтемени кайтарат. |
| allocator\_type\_get\_allocator() const; | Вектордун эс бөлүштүргүчүн кайтарат. |
| iterator insert(iterator i, const T&маани=T()); | *маани* параметрин i итератору менен берилген элементтин алдына кыстарат. Кыстарылган элементтин итераторун кайтарат. |
| void insert(iterator i, size\_type сан, const T & маани); | i итератору менен аныкталган элементтин алдына маани маанисин сан жолу кыстарат |
| template <class Inlter> void insert(iterator i, Inlter башы, Inlter айагы); | i итератору менен аныкталган элементтин алдына *башы, айагы* аралыкта аныкталган удаалаштыкты кыстарат. |
| size\_type max\_size() const; | Вектордо сакталуучу максималдуу элементтердин санын кайтарат. |
| reference operator[] (size\_type i) const;  const\_ reference operator [](size\_type i) const; | i параметри менен аныкталган элементтин шилтемесин кайтарат. |
| void pop\_back(); | Вектордун акыркы элементин жойот |
| void push\_back(const T&маани); | Мааниси *маани*  параметрге барабар болгон элементти вектордун айагына кошот. |
| reverse\_iterator rbegin();  const\_reverse\_iterator rbegin() const; | Вектордун айагынын кери итераторун кайтарат. |
| reverse\_iterator rend();  const\_reverse\_iterator rend() const; | Вектордун башынын кери итераторун кайтарат. |
| void reserve (size\_type сан); | Вектордун сыйымдуулугун санга барабар кылат. |
| void resize(size\_type сан, T маани=T()); | Вектордун өлчөмүн сан өлчөмүнө өзгөртөт.  Эгерде вектор узарса, анда айагына кошулуучу элементтер маани маанини кабыл алат. |
| size\_type size() const; | Учурда вектордо сакталган элементтердин санын кайтарат. |
| void swap(vector<Allocator> &объект); | Вектордо сакталган элементтерди объектте сакталган элементтер менен орундарын алмаштырат. |

*vector*классынын эң маанилүү мүчө-функциялары катары *size(), begin(), end(), push\_back(), insert()*жана *erase()*саналат. *size()*функциясы вектордун учурдагы өлчөмүн билдирет. *begin()* функциясы вектордун башталышынын итераторун берет,*end()*функциясы вектордун айагынын итераторун берет. *push\_back()*функциясы маанини вектордун айагына кыстарат. Вектордун каалаган жерине элементти*insert()*функциясы жайгаштырат. Вектордун элементтин*erase()*функциясы аркылуу өчүрсө болот.

Төмөнкү мисалда векторлор менен жүргүзүлүүчү негизги амалдар көрсөтүлгөн.

#include <iostream>  
#include <vector>  
using namespace std;  
  
    int main()  
    {  
        // узундугу нөл болгон векторду түзүү  
        vector <int> vektor;  
        int i;  
        cout<<"Вектордун өлчөмү="<<vektor.size()<<"\n";  
        //Вектордун айагына маанилерди кыстаруу.  
        //Зарыл болгон учурда вектордун көлөмү өсөт  
        for(i=0;i<5;i++) vektor.push\_back(i);  
        cout<<"Вектордун жаңы өлчөмү="<<vektor.size()<<"\n";  
        //Вектордун мазмунун экранга чыгаруу  
        cout<<"Вектордун учурдагы мазмуну \n";  
        for(i=0;i<vektor.size();i++)  
            cout<<vektor[i]<<' ';  
        cout<<"\n";  
        //Вектордун айагына жаңы маанилерди кыстаруу.  
        //Зарыл болгон учурда вектордун көлөмү өсөт  
        for(i=0;i<5;i++) vektor.push\_back(i+5);  
        cout<<"Вектордун жаңы өлчөмү="<<vektor.size()<<"\n";  
        //Вектордун мазмунун экранга чыгаруу  
        cout<<"Вектордун жаңы мазмуну \n";  
        for(i=0;i<vektor.size();i++)  
            cout<<vektor[i]<<' ';  
        cout<<"\n";  
        //Вектордун мазмунун өзгөртүү  
        for(i=0;i<vektor.size();i++) vektor[i]=vektor[i]+vektor[i];  
        //Вектордун жаңы мазмунун экранга чыгаруу  
        cout<<"Вектордун  мазмуну \n";  
        for(i=0;i<vektor.size();i++)  
            cout<<vektor[i]<<' ';  
        cout<<"\n";  
        return 0;  
}

Програмда бүтүн маанилерди сактоо үчүн *vektor* вектору түзүлөт. Алдын-ала аныктоо жок болгондуктан, анын сыйымдуулугу нөлгө барабар. *push\_back()* мүчө-функция аркылуу *vektor* векторунун айагына беш элемент кошулат. Жаңы элементтерди кыстаруу үчүн, *vektor* векторунун өлчөмү чоңойот.

Програм аткарылгандан кийин экранга төмөнкү жыйынтык чыгат:

Вектордун өлчөмү=0  
Вектордун жаңы өлчөмү=5  
Вектордун учурдагы мазмуну  
0 1 2 3 4  
Вектордун жаңы өлчөмү=10  
Вектордун жаңы мазмуну  
0 1 2 3 4 5 6 7 8 9  
Вектордун  мазмуну  
0 2 4 6 8 10 12 14 16 18

Си++ тилинде массивдер жана көрсөткүчтөр өз ара тыгыз байланышта. Массивке индекс же болбосо көрсөткүч аркылуу кайрылат. Ушул сыйактуу стандарттуу шаблондор китепканасында векторлор менен итераторлордун ортосунда тыгыз байланыш бар. Вектордун мүчөсүнө оператор же итератор аркылуу кайрылуу төмөнкү мисалда көрсөтүлгөн.

#include <iostream>  
#include <vector>  
using namespace std;  
  
    int main()  
    {  
        // узундугу нөл болгон векторду түзүү  
        vector <int> vektor;  
        int i;  
       //Вектордун айагына маанилерди кыстаруу.  
        for(i=0;i<5;i++) vektor.push\_back(i);  
      //Вектордун мазмунун индекс оператору аркылуу чыгаруу  
         for(i=0;i<5;i++)  
            cout<<vektor[i]<<' ';  
        cout<<"\n";  
        //Вектордун мазмунун итератор аркылуу чыгаруу  
         vector <int>::iterator p=vektor.begin();  
         while(p!=vektor.end())  
         {cout<<\*p<<' ';  
             p++;  
         }  
         return 0;  
}

Програм төмөнкү жыйынтыкты берет:

0 1 2 3 4  
0 1 2 3 4

*р*итераторунун жарыялоосуна көңүл буралы. *iterator* тиби класс-контейнер аркылуу аныкталып жатат. Тандалган контейнерге итераторду алуу үчүн, аны дал ушундай жарыялаш керек, б.а. *iterator* тибинин алдына контейнердин ысымын жазыш керек. *begin() мүчө*-функция аркылуу итератор аныкталып, вектордун башын көрсөтөт. Итераторго инкремент операторун колдонуп, вектордун каалаган элементине кайрылса болот. Бул процесс массивдин элементтерине кайрылууда көрсөткүчтү колдонууга окшош. *end()* мүчө-функция аркылуу вектордун айагына жетүү фактысы аныкталат.

*insert()*мүчө-функция аркылуу элементтерди вектордун каалган жерине жайгаштырса болот. Элементтер вектордон *erase()* функциясы аркылуу жойулат. Мисалы,

#include <iostream>  
#include <vector>  
using namespace std;  
  
    int main()  
    {  
  // узундугу беш, элементтери бирге барабар болгон  
  // векторду түзүү  
        vector <int> vektor(5,1);  
        int i;  
  //Вектордун узундугун жана мазмунун чыгаруу  
        cout<<"Өлчөм="<<vektor.size()<<"\n";  
        cout<<"Мазмуну:"<<"\n";  
         for(i=0;i<vektor.size();i++)  
            cout<<vektor[i]<<' ';  
        cout<<"\n";  
        vector<int>::iterator p=vektor.begin();  
        p=p+2;//үчүнчү элементти көрсөтөт  
  //р итератору көрсөткөн жерге беш жаңы элемент кыстаруу  
  //ар бир элемент сегизге барабар  
         vektor.insert(p,5,8);  
  //Вектордун узундугун жана мазмунун чыгаруу  
        cout<<"Өлчөм="<<vektor.size()<<"\n";  
        cout<<"Мазмуну:"<<"\n";  
         for(i=0;i<vektor.size();i++)  
            cout<<vektor[i]<<' ';  
        cout<<"\n";  
  //Кыстарылган элементтерди жойу  
        p=vektor.begin();  
        p=p+2;//үчүнчү элементти көрсөтөт  
        vektor.erase(p,p+5);  
  //Вектордун узундугун жана мазмунун чыгаруу  
        cout<<"Өлчөм="<<vektor.size()<<"\n";  
        cout<<"Мазмуну:"<<"\n";  
         for(i=0;i<vektor.size();i++)  
            cout<<vektor[i]<<' ';  
        cout<<"\n";  
         return 0;  
}

Бул програм төмөнкүнү берет:

Өлчөм=5  
Мазмуну:  
1 1 1 1 1  
Өлчөм=10  
Мазмуну:  
1 1 8 8 8 8 8 1 1 1  
Өлчөм=5  
Мазмуну:  
1 1 1 1 1

Төмөнкү мисалда вектор колдонуучу тарабынан аныкталган класстын объектилерин сактоо үчүн колдонулат. Класста унчукпаганда конструктор жана **<** оператору менен **==** операторлору кайрадан жүктөлгөн версиялары аныкталат. Компилятордо стандарттуу шаблондордун китепканасы ишке ашырылганына жараша, калган салыштыруу операторлорун аныктоого туура келип калышы мүмкүн.

#include <iostream>  
#include <vector>  
using namespace std;  
  
    class misal{  
        double d;  
        public:  
            misal(){d=0.0;}  
            misal(double x){d=x;}  
            misal &operator=(double x)  
            {d=x; return \*this;}  
            double kaytar(){ return d;}  
        };  
        bool operator <(misal i, misal j)  
        {return i.kaytar()<j.kaytar();}  
        bool operator==(misal i, misal j)  
        {return i.kaytar()==j.kaytar();}  
  
    int main()  
    {vector <misal> vektor;  
        int i;  
        for(i=0;i<5;i++)  
            vektor.push\_back(misal(i/3.0));  
        for(i=0;i<vektor.size();i++)  
            cout<<vektor[i].kaytar()<<' ';  
        cout<<"\n";  
        for(i=0;i<vektor.size();i++)  
            vektor[i]=vektor[i].kaytar()+3.5;  
        for(i=0;i<vektor.size();i++)  
            cout<<vektor[i].kaytar()<<' ';  
        cout<<"\n";  
        return 0;  
}

Бул програм төмөнкү натыйжаны берет:

0  0.333333  0.666667  1  1.33333  
3.5  3.83333  4.16667  4.5  4.83333

## **Тизмелер**

*list* классы эки багыттуу сызыктуу тизмени колдойт. Вектордон айырмаланып тизменин элементтерине кайрылуу удалааш жүргүзүлөт. Тизмелер эки багыттуу болгондуктан, элементтерге эки жагынан тең кайрылса болот.

Төмөндө *list* классынын шаблонун спецификациясы көрсөтүлгөн:

template<class T, class Allocator=allocator<T>> class list;

мында *Т* тизмеде сакталуучу берилиштердин тиби, *Allocator*эс бөлүштүргүч. *list* классында төмөнкү конструкторлор аныкталган:

explicit list(const Allocator &a=Allocator());  
explicit list(size\_type сан, const T& маани=T(), const Alloator &a=Allocator());  
list(const\_list<T,Allocator>&объект);  
template <class Inlter>list(Inlter башы, Inlter  айагы, const Allocator &a=Allocator());

Биринчи калып бош тизменин конструктору. Экинчи калыпта маанилери ***маани***ге барабар болгон **а** элементи бар тизменин конструкторун аныктайт. Конструктордун үчүнчү калыбы элементтери ***объект***ге барабар болгон бирдей элементтерин тизмесин түзөт. ***башы, айагы*** итераторлору менен аныкталган диапозондогу элементтерден турган тизмени төртүнчү калыптагы конструктор аныктайт.

*list* классы үчүн төмөнкү катыш амалдары аныкталган:

**==, <, <=, !=, >, >=**

Төмөнкү жадыбалда *list* классынын мүчө-функциялары келтирилген.

|  |  |
| --- | --- |
| **Мүчө-функция** | **Байандамасы** |
| *template<class Inlter> void assign(Inlter башы, Inlter айагы);* | ***башы, айагы*** менен аныкталган удаалаштыкты тизмеге менчиктейт. |
| *template<class size, class T> void assign(Size сан, const T маани=T());* | Ар бир элементи ***маани***ге барабар болгон ***сан*** элементти тизмеге менчиктейт. |
| *reference back();*  *const\_reference back() const;* | Тизменин акыркы элементинин шилтемесин кайтарат. |
| *iterator begin();*  *const\_ iterator begin() const;* | Тизменин биринчи элементинин итераторун кайтарат. |
| *void clear();* | Тизменин бардык элементтерин жойот. |
| *bool empty() const;* | Тизме бош болсо чынды кайтарат, болбосо жалганды кайтарат. |
| *iterator end();*  *const\_iterator end() const;* | Тизменин айагынын итераторун кайтарат. |
| *iterator erase(iterator i);* | *i*итератору көрсөткөн элементти жоет. Жойулган элементтен кийин жайгашкан элементтин итераторун берет. |
| *iterator erase(iterator башы, iterator айагы);* | ***башы, айагы*** итератору менен белгиленген элементтерди жойот. Жойулган элементтен кийин жайгашкан элементтин итераторун берет. |
| *reference front();*  *const\_reference front() const;* | Тизменин биринчи элементине шилтемени кайтарат. |
| *allocator\_type get\_allocator() const;* | Тизменин эс бөлүштүргүчүн кайтарат. |
| *iterator insert(iterator i, const T&маани=T());* | *i*итератору менен белгиленген элементтин алдына ***маани*** параметрин кыстарат. |
| *void insert(iterator I, size\_type сан, const T& маани);* | *i*итератору менен белгиленген элементтин алдына ***маани***нин ***сан*** көчүрмөсүн кыстарат. |
| *template<class Inlter> void insert(iterator i, Inlter башы, Inkter айагы);* | *i*итератору менен белгиленген элементтин алдына ***башы, айагы*** менен аныкталган удаалаштыкты кыстарат. |
| *size\_type max\_size() const;* | Тизмеде сакталуучу максималдуу  элементтердин санын кайтарат. |
| *void merge(list<T, Allocator> &объект);*  *template<class Comp> void merge(list<T, Allocator>& объект, Comp сал\_функциясы);* | ***объект***те сакталган иреттелген тизмени чакырылган иреттелген тизме менен бириктирет. Жыйынтык иреттелет. Бириктирүүдөн кийин ***объект***те сакталган тизме бош болот. Экинчи калыпта бир элементтин мааниси экинчи элементтен кичине болгондугун текшерүү үчүн ***сал\_функциясы***салыштыруу функциясы колдонот. |
| *void pop\_back();* | Тизменин акыркы элементин жойот. |
| *void pop\_front();* | Тизменин биринчи элементин жойот. |
| *void push\_back(const T & маани);* | Мааниси ***маани***ге барабар болгон элементти тизменин айагына кошот. |
| *void push\_front(const T & маани);* | Мааниси ***маани***ге барабар болгон элементти тизменин башына кошот. |
| *reverce\_iterator rbegin();*  *const\_reverce\_iterator rbegin() const;* | Тизменин айагынын кери итераторун кайтарат. |
| *void remove(const T & маани);* | ***маани*** параметрине барабар болгон элементтерди тизмеден жоет. |
| *template <class UnPred> void remove\_if(UnPred байандооч);* | ***байандооч*** предикаттын мааниси чын болгон маанилерди тизмеден жоет. |
| *reverse\_iterator rend();*  *const\_reverse\_iterator rend() const;* | Тизменин башынын кери итераторун кайтарат. |
| *void resize(size\_type сан, T маани=T());* | ***сан***га жараша тизменин өлчөмүн өзгөртөт. Эгерде тизме узарса, анда айагына кошулган элементтер ***маани*** маанисин алышат. |
| *void reverse();* | Элементтерди тескер тартипте жайгашытарат. |
| *size\_type size() const;* | Учурда тизмеде сакталган элементтердин санын берет. |
| *void sort();*  *template <class Comp> void sort (Comp сал\_функциясы);* | Тизмени иреттейт. Экинчи формада бир элементтин мааниси экинчи элементтен кичине болгондугун текшерүү үчүн ***сал\_функциясы***салыштыруу функциясы колдонот. |
| *void splice(iterator i, list<T, Allocator &объект);* | Тизмеге ***объект*** объектисинин мазмунун кыстарат. Кыстаруу жери *i*итератору менен аныкталат. Кыстаруудан кийин ***объект***те сакталган тизме бош болот. |
| *void splice(iterator i, list<T, Allocator &объект, iterator элемент);* | ***объект*** объектисинде сакталган тизмеден ***элемент*** итератору көрсөткөн элементти тизмеден жойот жана аны чакырган тизмеге сактайт. |
| *void splice(iterator i , list<T, Allocator &объект, iterator башы, iterator айагы);* | ***объект***объектисинде сакталган жана ***башы, айагы*** итераторлору менен белгиленген  элементтердин удаалаштыгын жойот жана аларды чакырган тизмеге сактайт. Кыстаруу жери *i*итератору менен аныкталат. |
| *void swap(list<T, Allocator> &объект);* | Чакырган тизменин элементтери менен ***объект*** объектисинде сакталган элементтерди орундары менен алмаштырат. |
| *void unique();*  *template <class BinPred> void unique(BinPred байандооч);* | Чакырган тизмеден түгөй элементтерди жойот. Экинчи формада элементтердин уникалдуулугун аныктоо үчүн ***байандооч*** предикаты колдонот. |

Төмөндө жөнөкөй тизмеге мисал келтирилген.

#include <iostream>  
#include <list>  
using namespace std;  
    int main()  
    {  
        //Бош тизмени түзүү  
        list<char> tizme;  
        int i;  
        for(i=0;i<5;i++) tizme.push\_back('A'+i);  
        cout<<"Өлчөм="<<tizme.size()<<'\n';  
        list<char>::iterator p;  
        cout<<"Тизменин мазмуну:\n";  
        while(!tizme.empty())  
        {p=tizme.begin();  
            cout<<\*p<<' ';  
            tizme.pop\_front();  
        }  
        return 0;  
}

Бул програмда белгилердин тизмеси түзүлүп жатат. Алды менен бош тизме түзүлөт. Андан кийин ал жерге беш белги кыстарылат. Бул амал*push\_back()* функция аркылуу ишке ашырылат. Функция кезектеги белгини тизменин айагына кыстарат. Тизменин өлчөмү экранга чыккандан кийин, тизменин мазмуну экранга чыга баштайт. Ал үчүн ар бир элемент экранга чыккандан кийин, ал элемент тизмеден жойулат. Бул процесс тизме бош болгонго чейин кайталанат. Програм аткарылып бүткөндөн кийин экранга төмөнкүлөр чыгат:

Өлчөм=5  
Тизменин мазмуну:  
A B C D E

Бул мисалда тизмени башынан айагына чейин өтүүдө биз тизмени бошоттук. Бирок андай шарт зарыл эмес. Төмөндө програмдын өзгөртүлгөн версиясы келтирилген.

#include <iostream>  
#include <list>  
using namespace std;  
  
    int main()  
    {  
        //Бош тизмени түзүү  
        list<char> tizme;  
        int i;  
        for(i=0;i<5;i++) tizme.push\_back('A'+i);  
        cout<<"Өлчөм="<<tizme.size()<<'\n';  
        list<char>::iterator p=tizme.begin();  
        cout<<"Тизменин мазмуну:\n";  
        while(p!=tizme.end())  
        {  
            cout<<\*p<<' ';  
            p++;  
        }  
        return 0;  
}

Бул учурда *р* итератору тизменин башын көрсөткөндөй кылып алдын-ала аныкталат. Циклдын ар бир кадамында анын мааниси бирге чоңойот дагы, ал кийинки элементке шилтейт. *р* итератору тизменин айагын шилтегенде, цикл токтойт.

Тизме эки багыттуу болгондуктан, элементтерди тизменин башынан же айагынан баштап жайгаштырса болот. Төмөнкү мисалда эки тизме түзүлөт. Экинчи тизмеде биринчи тизменин элементтери кери тартипте жайгашышат.

#include <iostream>  
#include <list>  
using namespace std;  
  
    int main()  
    {  
  //Бош тизмени түзүү  
        list<char> tizme;  
        list<char> keri\_tizme;  
        int i;  
        for(i=0;i<5;i++) tizme.push\_back('A'+i);  
        cout<<"Тизменин өлчөмү="<<tizme.size()<<'\n';  
        cout<<"Тизменин мазмуну:\n";  
        list<char>::iterator p;  
   //Тизмеден элементтерди жойу  
   //жана аларды кери тартипте

   //экинчи тизмеде жайгаштыруу  
        while(!tizme.empty())  
        {  
            p=tizme.begin();  
            cout<<\*p<<' ';  
            tizme.pop\_front();  
            keri\_tizme.push\_front(\*p);  
        }  
  cout<<'\n';  
  cout<<"Кери тизменин өлчөмү="<<keri\_tizme.size()<<'\n';  
  cout<<"Кери тизменин мазмуну:\n";  
        p=keri\_tizme.begin();  
        while(p!=keri\_tizme.end())  
        {  
            cout<<\*p<<' ';  
            p++;  
        }  
        return 0;  
}

Бул мисалда *tizme* тизменин элементерин кери тартипте жайгаштыруу төмөнкүчө ишке ашырылат: tizme тизмесинен элементтер башынан алынып кезеги менен keri\_tizme тизмеге жайгаштырылат. Програм төмөнкү жыйынтыкты берет:

Тизменин өлчөмү=5  
Тизменин мазмуну:  
A B C D E  
Кери тизменин өлчөмү=5  
Кери тизменин мазмуну:  
E D C B A

*sort()* мүчө-функция аркылуу тизмени иреттесе болот. Төмөнкү програмда кокус белгилер топтолуп, иреттелет.

#include <iostream>  
#include <list>  
#include <cstdlib>  
using namespace std;  
  
    int main()  
    {  
        //Бош тизмени түзүү  
        list<char> tizme;  
        int i;  
        //Тизмени кокус белгилер менен толтуруу  
        for(i=0;i<10;i++) tizme.push\_back('A'+(rand()%26));  
        cout<<"Тизменин өлчөмү="<<tizme.size()<<'\n';  
        cout<<"Тизменин мазмуну:\n";  
        list<char>::iterator p=tizme.begin();  
        while(p!=tizme.end())  
        {  
            cout<<\*p<<' ';  
            p++;  
        }  
        cout<<'\n';  
        //Тизмени иреттөө  
        tizme.sort();  
        cout<<"Иреттелген тизме\n";  
        p=tizme.begin();  
        while(p!=tizme.end())  
        {  
            cout<<\*p<<' ';  
            p++;  
        }  
        return 0;  
}

Бул програм төмөнкү жыйынтыкты берет:

Тизменин өлчөмү=10  
Тизменин мазмуну:  
A K Y I T T T Q E I  
Иреттелген тизме  
A E I I K Q T T T Y

Иреттелген тизмени башка тизме менен бириктирсе болот. Натыйжада эки тизмеден алынган элементтери иреттелип жаңы тизме түзүлөт. Жаңы тизме чакырган тизмеде сакталат, ал эми экинчи тизме бош тизмеге айланат. Мисалы, биринчи тизмеде **ACEGI** белгилери, экинчи тизмеде **BDFHJ** белгилери жайгашса, бириктирүүдөн кийин **ABCDEFGHIJ** тизмесин алабыз.

#include <iostream>  
#include <list>  
using namespace std;  
  
    int main()  
    {  
        //Бош тизмени түзүү  
        list<char> tizme\_1, tizme\_2;  
        int i;  
        //Тизмени кокус белгилер менен толтуруу  
        for(i=0;i<10;i+=2) tizme\_1.push\_back('A'+i);  
        for(i=1;i<10;i+=2) tizme\_2.push\_back('A'+i);  
        cout<<"Биринчи тизменин мазмуну:\n";  
        list<char>::iterator p=tizme\_1.begin();  
        while(p!=tizme\_1.end())  
        {  
            cout<<\*p<<' ';  
            p++;  
        }  
        cout<<'\n';  
        cout<<"Экинчи тизменин мазмуну:\n";  
        p=tizme\_2.begin();  
        while(p!=tizme\_2.end())  
        {  
            cout<<\*p<<' ';  
            p++;  
        }  
        cout<<'\n';  
        //Тизмелерди бириктирүү  
        tizme\_1.merge(tizme\_2);  
        if(tizme\_2.empty())  
 cout<<"Эми экинчи тизме бош\n";  
 cout<<"Бириктиргенден кийин биринчи тизменин мазмуну:\n";  
        p=tizme\_1.begin();  
        while(p!=tizme\_1.end())  
        {  
            cout<<\*p<<' ';  
            p++;  
        }  
        return 0;  
}

Програм төмөнкү жыйынтыкты берет:

Биринчи тизменин мазмуну:  
A C E G I  
Экинчи тизменин мазмуну:  
B D F H J  
Эми экинчи тизме бош  
Бириктиргенден кийин биринчи тизменин мазмуну:  
A B C D E F G H I J

## **Ассоциатив тизмелер**

*map* классы ар бир маани уникалдуу ачкычты камтыган ассоциатив контейнерди колдойт. Чынында ачкычты мааниге берилген ысым катары караса болот. Маани контейнерге жайгаштырылгандан кийин, аны ачкычтын жардамы менен сууруп алса болот. Демек ассоциатив контейнерди ачкыч/маани түгөй катары караса болот. Ассоциатив тизмелердин артыкчылыгы берилген ачкыч бойунча керектүү маанини алууда турат.

Жогоруда айтылгандай, ассоциатив контейнерде уникалдуу гана ачкычтарды сактаса болот. Ачкычтарды кайталоого болбойт. Төмөндө *map* классынын шаблону келтирилген:

template<class Key, class T, class Comp=less<Key>, class Allocator=allocator<T>class map;

мында *Key* ачкыч тибиндеги берилиштер, *T* сакталуу берилиштердин тиби, *Comp* эки ачкычты салыштыруу функциясы (унчукпаганда *less()* функциясы), *Allocator* эс бөлүштүргүч (унчукпаганда allocator).

*map* классында төмөнкү конструкторлор аныкталган:

explicit map(const Comp &сал\_функциясы=Comp(), const Allocator &a=Allocator());  
map(const map<Key, T, Comp, Allocator>&объект);  
template<class Inlter>map<Inlter башы, Inlter айагы, const Comp & сал\_функциясы = Comp(), const Allocator &a=Allocator());

Биринчи калыбы бош ассоциатив тизменин конструкторун билдирет. Экинчи калып бирдей элементтерден турган ассоциатив тизмени аныктайт. Ар бир элемент ***объект*** болот. Үчүнчү калыпта ***башы, айагы*** итераторлору менен аныкталган элементтердин аралыгын аныктаган конструктор. ***сал\_функциясы***(бар болсо) элементтердин иреттөө багытын аныктайт.

Ачкыч катары берилген ар бир объект үчүн унчукпаганда конструктор жана бир нече катыш белгилери аныкталыш керек. *map* классы үчүн төмөнкү салыштыруу операторлору аныкталат:

**==, <, <=, !=, >, >=**

Төмөнкү жадыбалда *map* классынын мүчө-функциялары келтирилген. Жадыбалда *key\_t* ачкычтын тиби, *key\_v* ачкыч/маани түгөйдүн тиби (*pair<key,T>* тиби).

|  |  |
| --- | --- |
| Мүчө-функция | **Байандамасы** |
| *iterator begin();*  *const\_iterator begin() const;* | Ассоциатив тизменин биринчи элементинин итераторун кайтарат. |
| *void clear();* | Ассоциатив тизменин бардык элементтерин жоет. |
| *size\_type count (const key\_t &k) const;* | ***k*** ачкычы ассоциативдүү тизмеде жолукканына жараша 0 же 1 маанини кайтарат. |
| *bool empty() const;* | Ассоциативдүү тизме бош болсо чын маанисин кайтарат, болбосо жалган деген маанини кайтарат. |
| *iterator end();*  *const\_iterator end const;* | Ассоциативдүү тизменин айагынын итераторун кайтарат. |
| *pair<iterator, iterator> equal\_range(const key\_t &k);*  *pair<const\_iterator, const\_iterator> equal\_range(const key\_t &k) const;* | ***k*** ачкычын камтыган, ассоциативдүү тизменин биринчи жана акыркы элементин көрсөткөн түгөй итераторду кайтарат. |
| *void erase(iterator i);* | ***i*** итератору көрсөткөн элементти жойот. |
| *void erase(iterator башы, itaretor айагы);* | ***башы, айагы*** итераторлору көрсөткөн аралыктагы элементтерди жойот. |
| *size\_type erase(const key\_t &k);* | ***k*** ачкытын маанисине дал келген элементтерди жойот. |
| *iterator find(const key\_t &k);*  *const\_iteartor find(const key\_t &k) const;* | ***k*** ачкычы менен берилген итераторду кайтарат. Эгерде ачкыч табылбаса, ассоциатив тизменин айагынын итераторун берет. |
| *allocator\_type get\_allocator() const;* | Ассоциатив тизменин эс бөлүштүргүчүн кайтарат. |
| *iterator insert(iterator i, const value\_type &маани);* | ***i*** итератору көрсөткөн же андан кийинки орунга  ***маани*** параметрин кыстарат.  Кыстарылган элементтин итераторун кайтарат. |
| *template<class Inlter> void insert(Inlter башы, Inlter айагы);* | ***башы, айагы*** итераторлору менен аныкталган элементтердин удаалаштыгын кыстарат. |
| *pair <iterator, bool> insert(const value\_type & маани);* | Ассоциатив тизмеге ***маани*** параметрин кыстарат. Кыстарылган элементтин итераторун кайтарат. Кыстарылуучу элемент тизмеде жок болгон учурда гана кыстарылат. Элемент кыстарылган учурда функция **pair <iterator, true>** маанини кайтарат, болбосо **pair <iterator, false>** маанини кайтарат. |
| *key\_compare key\_comp() const;* | Ачкычтарды салыштырган объект-функцияны кайтарат. |
| *iterator lower\_bound(const key\_t &k);*  *const\_iterator lower\_bound(const key\_t &k) const;* | Ассоциатив тизменин ачкычы ***k*** параметринен чоң же барабар болгон элементтердин биринчи элементинин итераторун кайтарат. |
| *size\_type max\_size() const;* | Ассоциатив тизмеде сакталуучу максималдуу элементтердин санын берет. |
| *reference operator[](const key\_t &i);* | ***i*** ачкычына дал келген элементке шилтемени кайтарат. Эгерде андай элемент жок болсо, анда ал ассоциатив тизмеге кыстарылат. |
| *reverse\_iterator rbegin();*  *const\_reverse\_iterator rbegin() const;* | Ассоциатив тизменин айагынын кери итераторун кайтарат. |
| *reverse\_iterator rend();*  *const\_reverse\_iterator rend() const;* | Ассоциатив тизменин башталышынын кери итераторун кайтарат. |
| *size\_type size() const;* | Учурда ассоциатив тизмеде сакталган элементтердин санын берет. |
| *void swap(map<key, T, Comp, Allocator> &объект);* | Ассоциатив тизмедеги элементтерди ***объект*** объектисиндеги элементтер менен орундарын алмаштырат. |
| *iterator upper\_bound(const key\_t &k);*  *const\_iterator upper\_bound(const key\_t &k) const;* | Ачкычы ***k*** параметринен чоң болгон ассоциатив тизменин биринчи элементинин итераторун кайтарат. |
| *value\_compare value\_comp() const;* | Маанилерди салыштырган объект-функцияны кайтарат. |

Ассоциатив тизмелерде ачкыч/маани түгөйлөр *pair* тибиндеги объект түрүндө сакталат. *pair* тибиндеги объектин шаблону төмөнкү түргө ээ:

template<class Ktype, class Vtype> struct pair{  
typedef Ktype тип\_1; //ачкычтын тиби  
typedef Vtype тип\_2; //маанинин тиби  
Ktype бир; // ачкычты тутат  
Vtype эки; // маанини тутат}  
Конструкторлору  төмөнкүчө жарыяланат:  
pair();  
pair(const Ktype &k, const Vtype &v);  
template <class A, class B> pair(const<A,B> &объект);

Ошондой эле ачкыч/маани түгөйлөрдү *make\_pair()* функция аркылуу түзсө болот. Бул функция берилиштердин тибин параметр катары колдонуп, *pair* тибиндеги объектилерди түзөт. Прототиби төмөнкүдөй:

temaplate <class Ktype, class Vtype>  
pair <Ktype, Vtype> make\_pair(const Ktype &k, const Vtype &v);

жана функция *pair* тибиндеги объектини кайтарат.

Төмөнкү мисалда ачкыч/маани түгөйлүрдү сактаган ассоциатив тизмени колдонуунун негиздери келтирилген. Ачкыч катары белги, маани катары бүтүн сан алынып ачкыч/маани түгөйлөр төмөнкүчө сакталат:

|  |  |
| --- | --- |
| A | 0 |
| B | 1 |
| C | 2 |
| D | 3 |
| E | 4 |

Түгөйлөр ушул калыпта сакталган үчүн, колдонуучу ачкычты (тизмедеги бир тамга) киргизгенде, экранга ага дал келген маани чыгат.

#include <iostream>  
#include <map>  
using namespace std;  
  
    int main()  
    {  
        map<char, int> tizme;  
        int i;  
        //Түгөйлөрдү ассоциатив тизмеде жайгаштыруу  
        for(i=0;i<5;i++)  
            tizme.insert(pair<char, int> ('A'+i,i));  
        char ch;  
        cout<<"Ачкычты киргизгиле: ";  
        cin>>ch;  
        map<char, int>::iterator p;  
        //Берилген ачкыч бойунча маанини издөө  
        p=tizme.find(ch);  
        if(p!=tizme.end())  
            cout<<"\nМаани= "<<p->second;  
        else  
       cout<<"Ассоциатив тизмеде андай ачкыч жок\n";  
    eturn 0;  
}

Ачкыч/маани түгөйлөрдү түзүү үчүн *pair* класс-шаблонго көңүл буралы: *pair* класс-шаблондо көрсөтүлгөн берилиштер тиби ассоциатив тизмеде сакталган берилиштердин типтерине дал келиш керек. Ассоциатив тизме ачкыч/маани түгөйлөр аркылуу аныкталгандан кийин, керектүү маанини *find()* функциясы аркылуу тапса болот. *find()* функциясы ачкычка дал келген элементтин итераторун кайтарат. Тиешелүү ачкыч табылбаса, ассоциатив тизменин айагынын итераторун берет. Ачкычка дал келген маани табылган учурда, ал *pair* шаблонунун экинчи мүчөсү катары сакталат.

Жогорку мисалда ачкыч/маани түгөйлөрдүн типтери айкын түрдө *pair<char, int>* түзүмдө берилди. Бул көз караш туура болгону менен, практикада *make\_pair()* функцияны колдонуу ыңгайлуу болот. Функция объектер түгөйлөрүн өзүнүн параметрлеринин типтеринин негизинде түзөт.

#include <iostream>  
#include <map>  
using namespace std;  
  
    int main()  
    {  
        map<char, int> tizme;  
        int i;  
   //Түгөйлөрдү ассоциатив тизмеде жайгаштыруу  
        for(i=0;i<5;i++)  
            tizme.insert(make\_pair ('A'+i,i));  
        char ch;  
        cout<<"Ачкычты киргизгиле: ";  
        cin>>ch;  
        map<char, int>::iterator p;  
        //Берилген ачкыч бойунча маанини издөө  
        p=tizme.find(ch);  
     if(p!=tizme.end())  
            cout<<"\nМаани= "<<p->second;  
     else  
       cout<<"Ассоциативдүү тизмеде андай ачкыч жок\n";  
   return 0;  
}

Мисал мурдагы мисалдан *tizme.insert(make\_pair ('A'+i,i));* сабы менен гана айырмаланат.

Ассоциатив тизмелеринде колдонуучу тарабынан аныкталган типтерди сактаса болот. Төмөнкү мисалда эки класс түзүлөт: *soz* (сөз) жана *antonim* (антоним). Ассоциатив тизмелер ачкычтардын иреттелген тизмесин колдогон үчүн, програмда *soz* тибиндеги объектер үчүн **<** оператору аныкталат. Эгерде класстын объектиси ачкыч катары колдонсо, анда, эреже бойунча, **<** операторун бардык класстар үчүн кайрадан жүктөш керек.

#include <iostream>  
#include <map>  
#include <cstring>  
using namespace std;  
  
class soz{  
    char sap[20];  
    public:  
        soz(){strcpy(sap,"");}  
        soz(char \*s){strcpy(sap,s);}  
        char \*kaytar(){return sap;}  
    };  
  
    bool operator <(soz x, soz y)  
    {return strcmp(x.kaytar(), y.kaytar())<0;}  
    class antonim{  
        char sap[20];  
        public:  
            antonim(){strcmp(sap,"");}  
            antonim(char \*s){strcpy(sap,s);}  
            char \*kaytar(){return sap;}  
        };  
  
    int main()  
    {  
        map<soz,antonim> tizme;  
 //Сөз жана антонимдерди ассоциатив тизмеде жайгаштыруу  
        tizme.insert(pair<soz,antonim>(soz("ооба"),antonim("жок")));  
        tizme.insert(pair<soz,antonim>(soz("ак"),antonim("кара")));  
        tizme.insert(pair<soz,antonim>(soz("чын"),antonim("калп")));  
        tizme.insert(pair<soz,antonim>(soz("жакшы"),antonim("жаман")));  
        //Берилген сөз менен антонимди издөө  
        char sap[30];  
        cout<<"Сөз киргиз:";  
        cin>>sap;  
        map<soz, antonim>::iterator p;  
        p=tizme.find(soz(sap));  
        if(p!=tizme.end())  
            cout<<"Антоним: "<<p->second.kaytar();  
        else  
            cout<<"Тизмеде андай сөз жок!\n";  
        return 0;  
}

## **Алгоритмдер**

Алгоритмдер контейнерлерди иштетүүгө багытталган. Ар  контейнер өзүнүн базалык амалдарына ээ болгону менен, стандарттуу алгоритмдер кеңири, комлекстүү амалдарды жүргүзүүгө жардам берет. Андан сырткары алгоритмдер эки башка типтеги контейнерлер менен иштөөгө мүмкүндүк берет. Стандарттуу шаблондорунун китепканасындагы алгоритмдерге кайрылуу үчүн програм башына **<algorithm>** файлын кошуу керек.

Стандарттуу шаблондор китепканасында абдан көп алгоритмдер аныкталган. Бардык алгоритмдер функция-шаблон түрүндө ишке ашырылган, ошондуктан аларды каалгандай типтеги контейнерлер менен колдонсо болот. Төмөнкү жадыбалда алгоритмдердин тизмеси келтирилген.

|  |  |
| --- | --- |
| **Алгоримт** | **Милдети** |
| *adjacent\_find* | Удаалаштыкта туташ түгөйлөрдү издейт. Биринчи түгөйдүн итераторун кайтарат. |
| *binary\_search* | Иреттелген удаалаштыкта экилик издөө жүргүзөт. |
| *copy* | Удаалаштыктын көчүрмөсүн алат. |
| *copy\_backward* | Айагында жайгашкан элементтерди башталышына алып келип, көчүрмөсүн алат. |
| *count* | Удаалаштыктагы элементтердин санын берет. |
| *count\_if* | Удаалаштыктын кандайдыр бир шартка баш ийген элементтердин санын берет. |
| *equal* | Эки аралыктын  бирдейлигин аныктайт. |
| *equal\_range* | Удаалаштыкта элементтердин жайгашуу иретин бузбай жаңы элементи кошууга мүмкүн болгон аралыкты кайтарат. |
| *fill*  *fill\_n* | Аралыкты берилген маани менен толтурат. |
| *find* | Маани үчүн аралыкты издеп, биринчи табылган элементти кайтарат. |
| *find\_end* | Удаалаштык үчүн аралыкты издейт. Аралыктын ичиндеги удаалаштыктын акыркы элементинин итераторун кайтарат. |
| *find\_first\_of* | Удаалаштыктын ичинен аралыктын элементине түгөй болгон биринчи элементти издейт. |
| *find\_if* | Шартка жараша элементке аралыкты издейт. |
| *for\_each* | Элементтер аралыгы үчүн функцияны дайындайт. |
| *generate*  *generate\_n* | Функция кайтарып берген маанини аралыктагы элементтерге менчиктейт. |
| *includes* | Бир удаалаштыктын бардык элементтери башка удаалаштыкка толугу менен киргендигин аныктайт. |
| *inplace\_mergr* | Бир аралыкты башка аралык менен бириктирет. Эки аралык тең иреттелген болуш керек. Жыйынтык иреттелет. |
| *iter\_swap* | Эки итератор көрсөткөн маанилерди орундары менен алмыштырат. Итераторлор функциянын аргументтери болот. |
| *lexicographical\_compare* | Эки удаалаштыкты алфавит тартибинде салыштырат. |
| *lower\_bound* | Удаалаштыкта берилген мааниден кичине болбогон маанини табат. |
| *make\_heap* | Удаалаштыкка пирамида сорттоону колдонот. |
| *max* | Эки маанинин чоңун кайтарат. |
| *max\_element* | Аралыктын эң чоң элементинин итераторун кайтарат. |
| *merge* | Иреттелген эки удаалаштыкты бириктирип, жыйынтыгын үчүнчү удаалаштыкка жайгаштырат. |
| *min* | Эки маанинин кичинесин берет. |
| *min\_element* | Аралыктын эң кичине элементинин итераторун кайтарат. |
| *mismatch* | Эки удаалаштыктын биринчи дал келбеген элементтеринин итераторлорун кайтарат. |
| *next\_permutation* | Удаалаштыктын кийинки орун алмаштыруусун түзөт. |
| *next\_element* | Берлиген элементтен кичине элементтерди ал элементке чейин, калгандарын ал элементтен кийин жайгаштырат. |
| *partial\_sort* | Аралыкты иреттейт. |
| *partial\_sort\_copy* | Аралыкты иреттейт дагы, жыйынтыктуу удаалаштыкка сыйган элементтердин көчүрмөсүн көчүрөт. |
| *partition* | Удаалаштыкта берилген шартка баш ийген элементтер алды менен, баш ийбеген элементтер кийин жайгашкандай кылып иреттейт. |
| *pop\_heap* | Биринчи жана акыркы элементтин алдында турган элементтерди орундары менен алмаштырат. |
| *prev\_permutation* | Удаалаштыкка алдыда жүргүзүлгөн орун алмаштырууну түзөт. |
| *push\_heap* | Элементти пирамиданын айагына жайгаштырат. |
| *random\_shuffle* | Удаалаштыкты баш аламан кылат. |
| *remove*  *remove\_if*  *remove\_copy*  *remove\_copy\_if* | Берилген аралыктан элементти жойот. |
| *replace*  *replace\_if*  *replace\_copy*  *replace\_copy\_if* | Аралыкта элементтерди алмаштырат |
| *reverce*  *reverce\_copy* | Элементерди иреттөө багытын кери багытка алмаштырат. |
| *rotate* | Аралыктагы элементтердин бардыгын сол тарапка жылдырат. |
| *rotate\_copy* | Удаалаштыкты удаалаштыктын ичинен издейт. |
| *search*  *search\_n* | Берилген сандагы бирдей элементтердин удаалаштыгын издейт. |
| *set\_difference* | Эки иреттелген удаалаштыктын окшош болбогон бөлүктөрүнөн турган жаңы удаалаштыкты түзөт. |
| *set\_intersection* | Эки иреттелген удаалаштыктын окшош болгон бөлүктөрүнөн турган жаңы удаалаштыкты түзөт. |
| *set\_symmetric\_difference* | Эки иреттелген удаалаштыктын симметриялык окшош болбогон бөлүктөрүнөн турган жаңы удаалаштыкты түзөт. |
| *set\_union* | Эки иреттелген удаалаштыктын бириктирүүсүнөн турган жаңы удаалаштыкты түзөт. |
| *sort* | Аралыкты иреттейт. |
| *sort\_heap* | Аралыктын ичинен пирамиданы иреттейт. |
| *stable\_partition* | Шартка баш ийген элементтерди шартка баш ийбеген элементтердин алдына жайгаштырат. |
| *stable\_sort* | Аралыкты иреттейт. Окшош элементтер орундары менен алмаштырылбайт. |
| *swap\_ranges* | Эки маанинин орундарын алмаштырат. |
| *transform* | Аралыктагы элементтердин орундарын  алмаштырат. |
| *unique* | Элементтер аралыгына функцияны белгилейт дагы, жыйынтыкты жаңы удаалаштыкка сактайт. |
| *unique\_copy* | Аралыктан кайталанган элементтерди жойот |
| *upper\_bound* | Удаалаштыкта берилген мааниден чоң болбогон эң акыркы  маанини аныктайт. |

Кээ бир алгоритмдерге мисал келтирели.

Эң жөнөкөй алгоритм катары *count()* жана *count)\_if* алгоритмдерин караса болот. Төмөндө алардын негизги калыптары көрсөтүлгөн:

template<class Inlter, class T>  
size\_t count(Inlter башы, Inlter айагы, const T& маани);  
template<class Inlter, class T>  
size\_t count\_if(Inlter башы, Inlter айагы, UnPred &байандооч\_функц);

*count()* алгоритми удаалаштыкта ***башы*** итератору менен башталган, ***айагы*** итератору менен бүткөн жана ***маани*** параметрине барабар болгон элементтердин санын кайтарат. *count()\_if* алгоритми удаалаштыкта ***башы*** итератору менен башталган, ***айагы*** итератору менен бүткөн жана ***байандооч\_функц*** предикатында чын деген мааниге ээ болгон элементтердин санын кайтарат. Төмөнкү програм *count()*жана *count()\_if* алгоритмдерди көрсөтөт.

#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  //Унардуу предикат маанинин так же жуп экендигин аныктайт  
    bool jup(int i)  
     {return !(i%2);}  
  
     int main()  
    {vector<int> vektor;  
        int i;  
        for(i=0;i<10;i++)  
        {if (i%2) vektor.push\_back(1);  
            else  
                vektor.push\_back(2);  
        }  
        cout<<"Удаалаштык:\n";  
        for(i=0;i<vektor.size();i++)  
            cout<<vektor[i]<<' ';  
        cout<<'\n';  
        int k;  
        k=count(vektor.begin(), vektor.end(),1);  
        cout<<k<<" элементи так.\n";  
        k=count\_if(vektor.begin(), vektor.end(),jup);  
        cout<<k<<" элементи жуп.\n";  
        return 0;  
}

Програм төмөнкү жыйынтыкты берет:

Удаалаштык:  
2 1 2 1 2 1 2 1 2 1  
5 элементи так.  
5 элементи жуп.

Кээ бир учурларда берилген удаалаштыктын аныкталган үзүндүлөөрүнөн турган жаңы удаалаштыкты алууга туура келет. Бул максатты ишке ашыруу үчүн *remove\_copy()*алгоритми колдонот. Алгоритм төмөнкү шаблонго ээ:

template<class Inlter, class Outlter, class T>  
Outlter remove\_copy(Inlter башы, Inlter айагы, Outlter жыйынтык, const T &маани);

Демек*remove\_copy()*алгоритми***башы*** жана ***айагы*** итераторлору менен белгиленген аралыктан ***маани*** параметрине барабар болгон элементтерди ***жыйынтык*** итератору менен белгиленген удаалаштыкка жайгаштырат. Алгоритм түзүлгөн жаңы удаалаштыктын айагынын итераторун кайтарат. Төмөнкү мисалда алды менен удаалаштыкта 1 жана 2 сандары кезек-кезек менен жайгаштырылат. Андан кийин удаалаштыктан 1 цифрасы жойулат.

#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
    int main()  
    {vector<int> vektor\_1, vektor\_2(10);  
        int i;  
        for(i=0;i<5;i++)  
        {if (i%2) vektor\_1.push\_back(1);  
            else  
                vektor\_1.push\_back(2);  
        }  
        cout<<"Удаалаштык:\n";  
        for(i=0;i<vektor\_1.size();i++)  
            cout<<vektor\_1[i]<<' ';  
        cout<<'\n';  
        //1 жойу  
        remove\_copy(vektor\_1.begin(),vektor\_1.end(),vektor\_2.begin(),1);  
        cout<<"Жаңы удаалаштык:\n";  
        for(i=0;i<vektor\_2.size();i++)  
            cout<<vektor\_2[i]<<' ';  
        cout<<'\n';  
        return 0;  
}

Жыйынтык төмөнкүдөй болот:

Удаалаштык:  
2 1 2 1 2  
Жаңы удаалаштык:  
2 2 2 0 0 0 0 0 0 0

*reverse()* алгоритми удаалаштыктагы элементтердин жайгашуу тартибин кери тартипке алмаштырат. Негизги калыбы төмөнкүдөй:

template <class Bilter> void reverse(bilter башы, Bilter айагы);

Бул алгоримт ***башы, айагы***аралыкта жайгашкан элементтердин тартибин кери тартипке алмаштырат. Мисалы,

#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
    int main()  
    {vector<int> vektor;  
       int i;  
        for(i=0;i<5;i++)  
         vektor.push\_back(i);  
        cout<<"Удаалаштык:\n";  
        for(i=0;i<vektor.size();i++)  
            cout<<vektor[i]<<' ';  
        cout<<'\n';  
        //Тартибин өзгөртүү  
        reverse(vektor.begin(),vektor.end());  
        cout<<"Кери тартип:\n";  
        for(i=0;i<vektor.size();i++)  
            cout<<vektor[i]<<' ';  
        cout<<'\n';  
        return 0;  
}

Жыйынтыгы төмөнкү болот:

Удаалаштык:  
0 1 2 3 4  
Кери тартип:  
4 3 2 1 0

Кызыктуу алгоритмдердин бири катары *transform()* алгоритмин караса болот. Анда берилген аралыктын элементтерин берилген функция аркылуу өзгөртөт. *transform()* алгоритми эки негизги калыпка ээ:

template<class Inlter, class Outlter, class Func>  
Outlter transform(Inlter башы, Inlter айагы, Outlter жыйынтык, Func унардык\_функция);  
  
template<class Inlter\_11, class Inlter\_22, class Outlter, class Func>  
Outlter transform(Inlter башы\_1, Inlter айагы\_1, Inlter башы\_2, Outlter  жыйынтык, Func бинардык\_функция);

*transform()* алгоритми функцияны аралыктын элементтерине колдонот дагы, жыйынтыгын ***жыйынтык*** итератору менен аныкталган жерге сактайт. Биринчи калыпта аралык ***башы, айагы*** итераторлор аркылуу берилет, ал эми колдонуучу функция катары  ***унардык\_функция*** колдонот. Бул функция параметр катары элементтин маанисин алат дагы өзгөртүлгөн элементти кайтарып берет. Экинчи калыпта өзгөртүү бинардык ***бинардык\_функция*** аркылуу жүргүзүлөт. Ал биринчи параметр катары өзгөртүүгө арналган элементтин маанисин алат, ал эми экинчи параметр катары экинчи удаалаштыктын элементин алат.

Төмөнкү програмда өзгөртүү жүргүзүү үчүн тизменин элементтерин квадратка көтөргөн *ozgort()* функциясы колдонот. Жыйынтыктоочу удаалаштык баштапкы удаалаштыкта сакталганына көңүл бургула.

#include <iostream>  
#include <list>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
    //өзгөртүү функциясы  
    int ozgort(int i){return i\*i;}  
  
    int main()  
    {list<int> tizme;  
        int i;  
        for(i=0;i<5;i++)  
         tizme.push\_back(i);  
        cout<<"Тизменин алгачкы маанилери:\n";  
        list<int>::iterator p=tizme.begin();  
        while(p!=tizme.end())  
        {  
          cout<<\*p<<' ';  
            p++;  
        }  
        cout<<'\n';  
        //Элементтерди өзгөртүү  
 p=transform(tizme.begin(), tizme.end(), tizme.begin(),ozgort);  
        cout<<"Өзгөртүлгөн тизме:\n";  
        p=tizme.begin();  
        while(p!=tizme.end())  
        {  
          cout<<\*p<<' ';  
            p++;  
        }  
   return 0;  
}

Жыйынтыкта көрүнүп тургандай элементтер квадратка көтөрүлдү:

Тизменин алгачкы маанилери:  
0 1 2 3 4  
Өзгөртүлгөн тизме:  
0 1 4 9 16

## **Сап классы**

Си++ тилине кыстарылган сап тибин жекече өзүн колдобойт. Бирок саптарды иштетүү үчүн эки мүмкүнчүлүккө ээ. Биринчиден, бизге белгилүү болгон белгилердин массивин колдонуу, экинчиден, *string*тибиндиеги объектрди пайдалануу.

Чындыгында *string*классы *basic\_string*жалпы шаблон-класстын бир бөлүгү болот. *basic\_string*класстын эки туунду классы бар: белгилердин 8-разряддуу саптарын колдогон *string*классы жана белгилердин узун саптарын колдогон *wstring*классы. Практикалык программалоодо көпчүлүк учурда 8-разряддуу белгилер менен иш жүргүзүлгөндүктөн *basic\_string*классынын *string* версиясын карайбыз.

Буга чейин сап катары нөл белгиси менен айактаган массивди колдондук. Бирок мындай массивдерди Си++ тилинин стандарттуу операторлору менен иштетүүгө болбойт жана алар кадимки туйунтмалардын бөлүктөрү боло албайт. Програмдын төмөнкү үзүндүсүн карап көрөлү:

char sap\_1[20], sap\_2[20], sap\_3[30];  
sap\_1=”бир”; // мүмкүн эмес  
sap\_1=” эки”; // мүмкүн эмес  
sap\_3=sap\_1+sap\_2;// // ката

Комментарийлерде көрүнүп тургандай, Си++ тилинде белгилер массивине жаңы маанини берүү үчүн менчиктөө операторун жана саптарды бириктирүү үчүн кошуу операторун колдонууга болбойт. Мындай амалдарды төмөндө көрсөтүлгөн китепканалык функциялар аркылуу ишке ашырылат:

strcpy(sap\_1,”бир”);  
strcpy(sap\_2,” эки”);  
strcpy(s3,s1);  
strcpy(s3,s2);

Нөл менен айактаган сап массивтери берилиштердин тиби боло албайт, аларга Си++ тилинин операторлорун колдонууга болбойт. Ошондой эле нөл менен айактаган белгилер массиви менен иштөөдө, ал массивтин чегарасынан чыгып кетүү учурлары кезигет. Мисалы, стандарттуу *strcpy()* функциясында массивдин чегарасын текшерген эч кандай атрибут жок.

Ошентип Си++ тилине  *string* стандарттуу классын кошууга үч себеби бар экен: сыйышкычтык (эми сап берилиштер тиби болот), ыңгайлуулук (Си++ тилинин стандарттуу операторлорун колдонсо болот), коопсуздук (массивдин чегаралары бузулбайт).

 Адатта сап классын стандарттуу шаблондор китепканасынын бөлүгү катары каралбаганы менен, сап классы Си++ тилинде аныкталган класс-контейнер болот.  Демек сап классына карата алдынкы бөлүктө байандалган алгоритмдерди колдонсо болот экен. Андан сырткары, саптарды иштетүү үчүн кошумча мүмкүнчүлүктөр бар. *string* классына кирүүгө мүмкүндүк алуу үчүн програмдын башына *string**бөрк*файлын кошуу керек.

*string*классы өтө чоң. Анын ичинде көптөгөн сандагы конструкторлор жана мүчө-функциялар бар. Кээ бир мүчө-функциялар бир катар кайрадан жүктөлгөн калыптарга ээ. Бул бөлүктө *string* классынын баардык мүчөлөрү жөнүндө айтууга мүмкүн эмес. Ал үчүн биз негизги мүмкүнчүлүктөрүн изилдейли.

*string*классы бир нече конструкторго ээ. Төмөндө эң көп колдонгондордун прототиптери келтирилген:

strıng();  
string(const char \* сап);  
string(const string &сап);

Биринчи калыпта *string* тибиндеги бош объект түзүлөт. Экинчи калыпта *string* тибиндеги объект түзүлөт. Ал ***сап*** көрсөткүчү менен белгиленген жана нөл менен айактаган сап. Бул калып нөл менен айактаган сапты *string* тибиндеги объектиге айландырат. Үчүнчү калыпта *string* тибиндеги объект башка *string* тибиндеги объектиден түзүлөт.

Төмөндө *string* тибиндеги объектилер менен иштөөгө жөнү бар операторлор келтирилген:

|  |  |
| --- | --- |
| **Оператор** | **Маани** |
| **=** | Менчиктөө |
| **+** | Бириктирүү |
| **+=** | Бириктирүү менен менчиктөө |
| **==** | Барабар |
| **!=** | Барабар эмес |
| **<** | Кичине |
| **<=** | Кичине же барабар |
| **>** | Чоң |
| **>=** | Чоң же барабар |
| **[]** | Индекс |
| **<<** | Чыгаруу |
| **>>** | Киргизүү |

Аталган операторлор *string* тибиндеги объектерди кадимки туйунтмаларда колдонууга мүмкүнчүлүк берет жана атайын фукнцияларды чакырууну танууга жол ачат.

*string* тибиндеги объектини башка *string* тибиндеги объект менен бириктирүү үчүн же *string* тибиндеги объекти Си тилиндеги сап тиби менен бириктирүү үчүн **+** оператору колдонулат:

string +  string  
string + С\_ string  
С\_ string +  string

Мындан сырткары **+** операторун саптын айагына жалгыз белгини бириктирүү үчүн колдонуу каралган.

*string* классында *npos* турактуусу аныкталган. Адатта анын мааниси -1 барабар. Ал турактуу саптын мүмкүн болгон узундугун аныктайт.

Саптар менен көпчүлүк амалдарды сап операторлор аркылуу ишке ашырса болот. Бирок кээ бир амалдар үчүн *string* классынын мүчө-функциялары талап кылынат. Мисалы, бир сапты башка сапка менчиктөө үчүн *assign()* функциясы колдонот. Анын негизги эки калыбы төмөнкүлөр:

string fiassgn(const string &сап\_объектиси, size\_type  башы, size\_type сан);  
string &assign (const char \* сап, size\_type сан);

Биринчи калыпта ***сап\_объектиси*** объектисинин башталышынан башталып, ***сан*** параметрине чейин барабар болгон белгилер тобу чакырган объектиге менчиктелет. Экинчи калыпта нөл менен айактаган ***сап*** сабынан ***сан*** параметрине барабар болгон белгилердин тобу чакырган объектиге менчиктелет. Эки учурда тең функция чакырган объектиге шилтемени кайтарат. Албетте, бир сапты башка сапка толугу менен менчиктөө үчүн **=** операторун колдонуу ыңгайлуу. *assign()* функциясын саптын бир бөлүгүн менчиктөөдө колдонот.

Бир саптын бөлүгүн башка сапка чиркөөнү *append()* мүчө-функция аркылуу ишке ашырса болот. Төмөндө бул функциянын негизги эки калыбы келтирилген:

string &append(const string &сап\_объектиси, size\_type  башы, size\_type сан);  
string &append (const char \* сап, size\_type сан);

Биринчи калыпта ***сап\_объектиси*** объектинин башынан башталып, ***сан*** параметрине чейин барабар болгон белгилер тобу чакырган объектиге чиркелет. Экинчи калыпта нөл менен айактаган ***сап*** сабынан ***сан*** параметрине барабар болгон белгилердин тобу чакырган объектиге бириктирилет. Эки учурда тең функция чакырган объектиге шилтемени кайтарат. Албетте, бир сапты башка сапка толугу менен чиркөө үчүн **+** операторун колдонуу ыңгайлуу. *assign()*функциясын саптын бир бөлүгүн бириктирүүдө колдонот.

*insert()* функциясы аркылуу сапка белгилерди кыстарса болот. Функциянын негизги калыптары төмөнкүлөр:

string &insert(size\_type башы, consr string & сап\_объектиси);  
string &insert(size\_type башы, consr string & сап\_объектиси, size\_type кыст\_башы, syze\_type сан);

Биринчи калыпта чакырууну козгогон сапка ***башы*** индексинен баштап ***сап\_объектиси*** кыстарылат. Экинчи калыпта ***сап\_объектиси*** объектисинин ***кыст\_башы*** индексинен баштап ***сан*** белгиси  чакырууну козгогон сапка кыстарылат.

*replace()* функциясы аркылуу саптагы белгилер алмаштырылат. Прототиптери төмөнкүлөр:

string &replace(size\_type башы, size\_type сан, const string &сап\_объектиси);  
string fireplace(size\_type башы, size\_type сан, const string &сап\_объектиси, size\_type алм\_башы, size\_type алм\_саны);

Биринчи калыпта чакырууну козгогон сапта ***башы*** индексинен баштап, ***сан***белгиси ***сап\_объектиси*** менен алмаштырылат. Экинчи калыпта чакырууну козгогон сапта***башы*** индексинен башталып ***сан*** белгиси алмаштырылат.  ***сап\_объектиси*** объектен ***алм\_башы*** индексинен баштап ***алм\_саны*** белги алынат. Эки учурда тең функция чакырган объектиге шилтемени кайтарат.

Саптан белгилерди *erase()* функция аркылуу жойот. Төмөндө бул функциянын бир калыбы келтирилген:

string &erase(size\_type башы==0, size\_type сан=npos);

Функция ***башы*** индексинен баштап ***сан*** белгини жойот.

**string** классында саптарды издөөгө багытталган бир нече мүчө-функция бар. Негизги функциялардын прототиптери төмөндө келтирилген:

size\_type find(const string &сап\_объект, size\_type башы=0) const;  
size\_type rfind(const string &сап\_объект, size\_type башы=npos) const;

*find()* функциясы саптан ***башы*** индексинен баштап ***сап\_объект*** камтылган сапты издейт. Эгерде изделип жаткан сап табылса, анда функция ал сап жайгашкан индексти кайтарат. Эгерде сап табылбаса, анда *функция npos* маанини кайтарат. *rfind()* функциясы ***башы*** индексинен баштап берилген сапты кери багытта издейт.  Эгерде изделип жаткан сап табылса, анда функция ал сап жайгашкан индексти кайтарат. Эгерде сап табылбаса, анда функция *npos* маанини кайтарат.

Эки сап объектерин салыштыруу үчүн жогоруда байандалган катыш операторлор колдонулат. Ал эми саптардын бөлүктөрүн салыштыруу үчүн *compare()* функциясы пайдаланылат. Ал төмөнкү прототипке ээ:

int compare(size\_type башы, size\_type сан, cinst string & сап\_объект) const;

Мында ***сап\_объект***объектисинин ***башы*** индексинен башталган ***сан*** белгиси чакырууну козгогон сап менен салыштырылат. Эгерде чакырууну козгогон сап  ***сап\_объект***ден кичине болсо, функция терс маанини кайтарат. Эгерде чакырууну козгогон сап  ***сап\_объект***ден чоң болсо, функция оң маанини кайтарат. Эгерде чакырууну козгогон сап  ***сап\_объект***ге барабар болсо, функция нөлду кайтарат.

*string* тибиндеги объекттер  ыңгайлуу болгону менен, кээ бир учурларда нөл менен айактаган белгилердин массивин колдонууга туура келет. Мисалы, *string* тибиндеги объекти файлдын ысымын түзүү үчүн колдонулат. Бирок файлды ачууда нөл менен айактаган стандарттуу сапка көрсөткүчтү колдонууга туура келет. Бул маселени чечүү үчүн *string* классы *c\_str()* мүчө-функцияны камтыйт. Прототиби төмөнкү:

const char \*c\_str() const;

*string* классы контейнер болгон үчүн, анда сап башынын итераторун кайтарган  *begin()* жана саптын айагынын итераторун кайтарган *end()* функцияларды камтыйт.

Төмөнкү програмда сап менен жүргүзүүлүчү кээ бир амалдар көрсөтүлгөн:

#include <iostream>  
#include <string>  
using namespace std;  
    int main()  
    {  
        string sap\_1("Биринчи сап");  
        string sap\_2("Экинчи сап");  
        string sap\_3;  
        //Саптарды менчиктөө  
        sap\_3=sap\_1;  
        cout<<sap\_1<<'\n'<<sap\_2<<'\n' <<sap\_3<<'\n';  
        //Саптарды бириктирүү  
        sap\_3=sap\_1+sap\_2;  
        cout<<sap\_3<<'\n';  
        //Саптарды салыштыруу  
  if(sap\_3>sap\_1) cout<<"Сап\_3 > Сап\_1 \n";  
  if(sap\_3==sap\_1+sap\_2) cout<<"Сап\_3 == Сап\_1+Сап\_2 \n";  
   //Объектиге жөнөкөй сапты менчиктесе болот  
        sap\_1="Бул жөнөкөй сап";  
        cout<<sap\_1<<'\n';  
        //Бир сап объект негизинде  
        // башка сап объектини түзүү  
        string sap\_4(sap\_1);  
        cout<<sap\_4<<'\n';  
        //Сапты киргизүү  
        cout<<"Сап киргиз:";  
        cin>>sap\_4;  
        cout<<sap\_4<<'\n';  
        return 0;  
}

Програм аткарылып, төмөнкү жыйынтыкты берет:

Биринчи сап  
Экинчи сап  
Биринчи сап  
Биринчи сапЭкинчи сап  
Сап\_3 > Сап\_1  
Сап\_3 == Сап\_1+Сап\_2  
Бул жөнөкөй сап  
Бул жөнөкөй сап  
Сап киргиз:Салам  
Салам

*string* тибиндеги объектер менен Си++ тилинин кыстарылган типтери сыйактуу мамиле аткарылат. Програмдан байкалып тургандай, саптардын узундуктары берилбейт.  *string* тибиндеги объектер автоматтык түрдө керектүү узундуктагы саптарды сактоого мүмкүндүк алат.

Төмөнкү програмда *insert(), erase()*жана *replace()* функцияларды колдонууга мисалдар келтирилген.

#include <iostream>  
#include <string>  
using namespace std;  
    int main()  
    {  
        string sap\_1("Биринчи сап");  
        string sap\_2("XYZ");  
        cout<<sap\_1<<'\n'<<sap\_2<<'\n';  
        //insert функциясы  
        cout<<"Биринчи сапты экинчи сапка кыстаруу\n";  
        sap\_1.insert(8,sap\_2);  
        cout<<sap\_1<<'\n';  
        //erase функциясы  
        cout<<"Саптан алты белгини өчүрүү\n";  
        sap\_1.erase(8,6);  
        cout<<sap\_1<<'\n';  
        //replаce функциясы  
        cout<<"Биринчи саптын белгилерин\n";  
        cout<<"экинчи саптын белгилери менен алмаштыруу\n";  
        sap\_1.replace(8,3,sap\_2);  
        cout<<sap\_1<<'\n';  
        return 0;  
 }

Програмдын жыйынтыгы төмөнкү:

Биринчи сап  
XYZ  
Биринчи сапты экинчи сапка кыстаруу  
Биринчи XYZсап  
Саптан алты белгини өчүрүү  
Биринчи  
Биринчи саптын белгилерин  
экинчи саптын белгилери менен алмаштыруу  
Биринчи XYZ

*string* классы берилиштердин тибин аныктаган үчүн, *string* тибиндеги объектерди сактоо үчүн контейнерлерди түзүүгө мүмкүнчүлүк ачылып жатат. Мисалы, төмөндө 5-бөлүктө каралган сөздөрдү жана антонимдерди сактоого ассоциатив тизмени түзүү програмдын кайрадан каралган версиясы келтирилген.

#include <iostream>  
#include <map>  
#include <string>  
using namespace std;  
    int main()  
    {  
        map<string,string> tizme;  
  //Сөз жана антонимдерди ассоциатив тизмеде жайгаштыруу  
     tizme.insert(pair<string,string>("ооба","жок"));  
     tizme.insert(pair<string,string>("ак","кара"));  
     tizme.insert(pair<string,string>("чын","калп"));  
     tizme.insert(pair<string,string>("жакшы","жаман"));  
     //Берилген сөз менен антонимди издөө  
        string sap;  
        cout<<"Сөз киргиз:";  
        cin>>sap;  
        map<string,string>::iterator p;  
        p=tizme.find(sap);  
        if(p!=tizme.end())  
            cout<<"Антоним: "<<p->second;  
        else  
            cout<<"Тизмеде андай сөз жок!\n";  
        return 0;  
}