Лекция 5

СЫЗЫҚТЫ ҚҰРЫЛЫМДАР, STL ЖӘНЕ БАЗАЛЫҚ КОНТЕЙНЕРЛЕР

С тілінде *char* типті массивтер символдық қатарлардан тұрады. C++ тілінде бұл механизм ***C-style strings*** деп аталады. Төменде C-style *strings*-тің "apple" сөзінің жеке символдарының қалай жазылғаны көрсетілген.

'a'	'p'	'p'	'l'	'e'	'\0'

Оның басты ерекшелігі массив соңында *C-style strings*-тің аяқталғанын білдіретін арнайы *Null character* немесе *null terminator* символы бар. Сондықтан *C-style strings* жадыдан символдық қатарлар санына қосымша бір байт орынды талап етеді. Мысалы, "apple" сөзіндегі бес әріпті сақтау үшін *C-style strings* 6-ұяшықты қолданады, бір ұяшық нөлдік символды сақтау үшін қолданылады. Осы курстың соңына дейін біз стандартты *string* класын қолданамыз. *C-style strings*-ді *string* объектісіне көшіру *string* класының конструкторы арқылы жүзеге асырылады. 1-листинг осындай конструкторды бейнелейді.

```
 1:  int main(int argc, char* argv[]) {
 2:
 3:      string s1(argv[0]);  // convert from char*
 4:
 5:      char apple[] = "apple";
 6:      string s2(apple);    // convert from char[]
 7:
 8:      cout << s1 << endl;
 9:      cout << s2 << endl;
10:
11:      return EXIT_SUCCESS;
12:  }
```

String класының қосымша операциялары

String класы символдар қатарының кеңейтілген қолданысын қамтамасыз етеді. *String* класы қатарлармен жұмыс істеуге қажетті күрделі деңгейлі функциялар жиынтығымен жұмыс істеуге мүмкіндік береді.

– ***erase*** - string символдар тізбегін жояды;

– ***find*** - string-ті басқа string негіздейді;

– ***substr -*** string-ті басқа string-тің бөлігі ретінде қайтарады;

– ***Replace*** қатардағы символдарды өзге қатардың символымен алмастырады;

– ***insert*** бір string-ті басқа string-ке қояды.

Төменде жоғарыда аталған функция мүшелерінің тізімі көрсетілген

```
1:   string s1("Demonstrating all the advanced");
2:   string s2("string functions!!!");
3:
4:   // erase the exclamation marks
5:   s2.erase(16, 3);
6:
7:   // replace 'all the' with 'some'
8:   s1.replace(14, 7, "some");
9:
10:  // insert a space at the beginning of s2
11:  s2.insert(0, " ");
12:
13:  // extract everything after "some" from s1
14:  int index = s1.find("some");
15:  string s3 = s1.substr(index);
16:
17:  cout << s1 << s2 << endl << s3 << endl;
```

Standard Template Library-ге кіріспе

Стандартты C++ тілі көптеген кітапханадан, соның ішінде *Standard Template Library* (STL) тұрады. STL программалаудың жалпыланған есептерін орындауға арналған жалпы компоненттермен жабдықтайды. Бұл компоненттер өзінің икемділігімен, тиімділігімен және теориялық негізделгендігімен сипатталады.

Кітапхана негізгі үш абстракциядан тұрады.

– ***контейнерлер***;

– ***итераторлар***;

– ***алгоритмдер***.

Контейнерлер *strings, vectors, lists, sets, stacks*, және т.б. тұрады. Контейнерлер тәуелсіз кластар жиыны ретінде ұйымдастырылған. STL контейнердегі барлық кластар кластардың шаблондары болып табылады және кез келген типтегі *items*-ті орната алады. Бірақ жеке кластардың қолданушы интерфейсінде контейнерлер аз болады. Оның орнына көптеген есептерді шығаруға мүмкіндік беретін үлкен және жақсы ұйымдастырылған STL алгоритмдер коллекциясы бар.

Мысалы, контейнерлердің барлық түрімен жұмыс істейтін *remove*() әмбебап функциясы бар.

STL концепциясының тағы ***ерекшелігі*** итератор алгоритмдері мен контейнерлер арасында байланыспен қамтамасыз етеді. Контейнерлердің басқа кітапханалардан ***айырмашылығы*** алгоритмдік абстракцияларда жинақталады. STL компоненттерінің тиімділігі дәстүрлі C++ тілінің кодтары арқылы құрылған компоненттер концентрациясына тең болады.

C++ тілінде жұмыс істейтін программалаушының ***STL*** жақсы меңгеруі тиіс және бір себеп, *STL* теориялық негіздерден тұрады және қолданушы анықтаған деректер құрылымы мен алгоритмдерді жүзеге асыруға бағыттауыш ретінде қолданылуы мүмкін концептуалды біліммен қамтамасыз етеді. STL-де құрылатын идеялар көп жағдайда басқа мақсатта қолданылуы мүмкін. Мысалы, кейбір кластарда сақталған элементтерге колжетімді болу үшін итераторларды қолдану дәстүрлі әдістерге қарағанда тиімді шешім болып келеді.

Ақпараттарды итератор арқылы алып, алгоритмдерді құру көп жағдайда оны икемді етеді және кодтарды жөндеу барысында тиімді болып келеді: итеративті тәсіл алгоритмдерді сипаттауыш ерекшеліктерін көруге мүмкіндік береді.

КОНТЕЙНЕРЛЕР

Стандартты контейнерлер атаулары сәйкесінше контейнерлердің типтерімен берілген келесі файлдар арқылы қолжетімді.

```
#include <string>       // strings
#include <vector>       // arrays
#include <list>         // cyclic doubly linked lists
#include <deque>        // hybrid list/array
#include <queue>        // queue
#include <stack>        // stack
#include <bitset>       // bit-vectors
#include <set>          // general sets
#include <map>          // associative arrays
```

```
#include <complex>      // complex numbers
#include <valarray>     // numerical arrays
#include <numeric>     // numerical algorithms
#include <cmath>        // math functions
```

Өкінішке орай, қазіргі уақытта C++ тілінің барлығында STL толығымен орнатылмаған, мысалы, заманауи қолданыстағы C++ тілінде болмауы мүмкін. Бір жағынан, STL орындалатын кейбір амалдар төмендегідей кеңейтілімдерде қосылған.

```
#include <hash_set>     // hash set
#include <hash_map>     // hash map
#include <slist>          // singly
linked list
```

Олар стандартты кітапхана бөлігі болып табылмайды және оларды қолдану барысында абай болған жөн.

ИТЕРАТОРЛАР

Итераторлар STL-дегі алгоритмдер мен контейнерлер арасындағы жалғыз интерфейс болып табылады. Итераторлар C++ тіліндегі нұсқауыштарға ұқсас болып келеді. Мысалы, төменде келтірілген стандартты идиомалар контейнер құрамына кіреді және оның элементтерін жазады.

```
string A = "This is a string";

string::iterator it;   // create iterator
for (it = A.begin(); it != A.end(); ++it) {
    cout << *it << endl;
}
```

Мұндағы *it* A контейнерінің итераторы болып табылады. *A.begin()* және *A.end()* функция – мүшелері A бірінші элементінің алғашқы нұсқауыштары, ал қалғандары "o*ne-past-the-end*" нұсқауыштары, яғни егер контейнерге соңғы элемент енгізілген болса, онда осы элементтің гипотикалық позициясы болатынына назар аударыңыз. Итераторлар соңғы контейнерлердегі кейбір пунктерге қолжетімді болу үшін қолданылады. Олар, сонымен қатар пунктер блогына қолжетімділік мүмкіндігін береді: *біз кез келген уақытта*

итератор қадамдарының көмегімен қандай да бір бастапқы шамадан соңғы шамаға өте аламыз.

АЛГОРИТМДЕР

Алгоритмдердің жалпыланған жиыны *STL* ядросы болып табылады. *STL* итераторлары контейнерлер мен алгоритмдер арасындағы өзара байланыс формасы болғандықтан, алгоритмдер бос функция ретінде құрылған. Функция – мүшесінің орнына бос функцияларды қолдану олардың сандарын қысқартуға тиімді болып келеді. Кейбір ерекше жағдайларда итераторлар тиімділік параметрі бойынша қажетті ақпараттарды бермейтіндіктен, функция мүшелерді қолдануға болмайды.

String контейнерлері үшін *replace* және *reverse* функцияларын оқыған жөн. *Replace* функциясы барлық енгізілген элементтерді басқа шамамен алмастырады. Төменде қатардағы символдардың барлығы дефиспен алмастырылған. *Reverse* функциясы контейнердің бір бөлігін өзгертеді. Төменде тізімде бүкіл қатарды өзгертеді.

```
#include <string>
#include <cstdlib>
#include <iostream>
#include <algorithm>

using namespace std;

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

    string s = "This is a test";
    cout << s << endl;

    // replace spaces with hyphens
    replace(s.begin(), s.end(), ' ', '-');
    cout << s << endl;

    // reverse the string
    reverse(s.begin(), s.end());
    cout << s << endl;

    return EXIT_SUCCESS;
}
```

STL vector Container-ін қолдану

Vector класы массивтерді қауіпсіз және көпфункционалды альтернативпен жабдықтайды. Массив секілді вектор – бұл бірдей деректер типінен тұратын объектілердің тізбегі. *STL* контейнері секілді *vector* класы жалпыланған программалауда қолданылады. Векторлар жоғары деңгейлі есептерді шығаруға мүмкіндік беретін функция мүшесі болғандықтан, массивтердің қауіпсіз альтернативі болып келеді.

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

using namespace std;

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

    vector<int> v1;
    vector<double> v2;
    vector<bool> v3;
    vector<string> v4;

    return EXIT_SUCCESS;
}
```

Жоғарыдағы листинг *vector* класының объектілер деклорациясын көрсетеді. *Vector* типті объектілерді қолдану үшін вектор класын анықтайтын кітапхананы қосуымыз керектігіне назар аударыңыз. Бұл 4-жолда көрсетілген. Біз векторды вектор объектісінің ішінде жариялай аламыз. Бұл матрица секілді екіөлшемді деректер қорында көрсетілген.

Келесіде вектор ішінде хабарлау көрсетілген:

```
1:     vector<vector<int> > matrix;
```

Vector класы программалаушы вектордың алғашқы өлшемі мен элементтерді бастапқы мәнін беру үшін қолдана алатын конструкторлар жиынынан тұрады. Келесі листинг осының мысалы болып табылады:

```
 1: #include <string>
 2: #include <cstdlib>
 3: #include <iostream>
 4: #include <vector>
 5:
 6: using namespace std;
 7:
 8: int main(int argc, char* argv[]) {
 9:
10:     vector<int> v1;       // initially empty
11:     vector<int> v2(5);    // 5 elements, initialized to 0
12:     vector<int> v3(10, 1);// 10 elements, initialized to 1
13:     vector<int> v4(v3);   // v4 is a copy of v3
14:
15:     return EXIT_SUCCESS;
16: }
```

Векторда сақталған элементтерге әртүрлі тәсілмен қол жеткізуге болады. Бірінші *vector* класында *overloads operator*[] операторы бар. Ол массивке қолжетімділікті қамтамасыз етеді. Шекараларын тексеруді талап етпейді. Массивтің алғашқы және соңғы элементтеріне қол жеткізудің бірнеше тәсілдері бар. Олар төмендегі листингте көрсетілген.

```
1:     vector<int> v(10);
2:
3:     v[1] = 2;
4:     v.at(2) = 45;
5:     v.front() = v.back();
```