Лекция 8-9

FIFO, LIFO, STL STACK ADAPTER-ДІ ҚОЛДАНУ

Бірінші кірді, бірінші шықты (FIFO).

Кезектер ***first-in, first-out*** сызбасын қолданатын деректердің сызықты құрылымы болып табылады. ***First-in, first-out*** кезекте енгізілген бірінші элемент кезектен өшірілетін бірінші элемент болып табылатынын білдіреді, яғни элементтер кезек басынан өшіріліп кезек соңына енгізіледі.

Адамдар кезекке артынан кіріп, кезектен тек кезек басына жетіп билеттерін сатып алғаннан кейін ғана өшірілетін. Ұшақтар аэропорттар ұшуы немесе қонуы үшін өз кезектерін күтеді. Кез келген жерде "First Come, First Served" кездесетін болса, онда кезек жұмыс іске қосылатынын айтуға болады.

Программалық жабдықтамаларды өңдеу және информатикада кезектің әртүрлі қосымшалары кездеседі. ***Queue*** буфер ретінде қолданылуы мүмкін. Буфер кейінірек өңделуі тиіс ақпараттарды уақытша сақтау орнымен жабдықтайды. Маршрутизаторлардың желілік буфері алдымен маршрутизаторларға шығыс мәліметтерді жүзеге асыру үшін мәліметтердің кіріс пакеттерін кезекке қояды. Операциялық жүйелер жоспарлау саясатын жүзеге асыру үшін кезектерді қолдануы мүмкін. Ағындық аудио және видео интернет арқылы пакеттердің шамалы санын уақыттық шектеуде байланыстыру жылдамдығын ескеру үшін кезекке буферлейді.

Queue (кезектер) адаптері ***STL*** контейнерлер үшін интерфейспен жабдықтайтын адаптерлерден тұрады. ***Queue*** (кезектер) адаптері ***queue*** деректер құрылымына сәйкес келетін интерфейспен жабдықтайды және ***deque*** контейнерінде үнсіздік бойынша негізделеді. ***Queue*** типті объект ***deque*** типті объектісін қолданғанда интерфейс арқылы кезекке тиімді болып қашықталады. 1-листинг ***queue*** адаптерлі интерфейсін бейнелейді.

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

using namespace std;

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

    queue<int> q;

    // push and pop
    q.push(1);
    q.pop();

    // front and back
    q.push(1);
    q.push(2);
    cout << q.front() << endl;
    cout << q.back() << endl;

    // size and empty
    cout << q.size() << endl;
    cout << q.empty() << endl;

    return EXIT_SUCCESS;
}
```

1-листинг

Queue адаптерлі интерфейстермен жұмыс істейтін алты функция мүшелері бар. ***Push*** және ***pop*** әдістері элементтерді кезекке енгізеді және өшіреді, кезектің басы мен соңында сақталатын деректерге қолжетімділікті қамтамасыз етеді. STL контейнеріндегі ***size*** әдісі кезекте сақталған элементтер санын қайтарады және егер кезек бос болса, онда ***empty*** әдісі ***true*** болады, кері жағдайда ***false*** болады. Жоғарыда келтірілген мысалдың 4-жолында ***queue*** адаптерін қолдану үшін кітапханасы енгізілген.

Queue адаптерінің кемшілігі ***iterators-***ты қолдануы болып табылады. Тек ***iterators*** көмегімен ***queue*** адаптері ***queue-***де сақталған бүкіл элементті аралауы мүмкін. Шын мәнінде ***queue*** өзінде сақталған алғашқы және соңғы элементтен басқасын жасырады. Егер қосымша ***queue-***ні, бірақ сонымен қатар кезекте сақталған элементтерге қол жетімділікті талап етсе, онда орнына ***deque*** қолданған жөн.

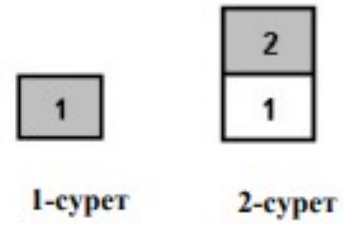
1-кестеде queue класының функция мүшелері және олардың ***deque-***дегі эквиваленттері бейнеленген.

queue member	**deque equivalent**
push	push_back
pop	pop_front
front	front
back	back
size	size
empty	empty

Соңынан кірді, бірінші шықты

Стек соңғы енгізілген элементке ғана қолжетімділікпен жабдықтайтын деректердің сызықты құрылымы болып табылады. Стекте элементтер тек бір шетінен ғана енгізілуі және жойылуы мүмкін, яғни элементтерді енгізу және өшіру саясаты – «***Last-in, First-out***". Басқаша айтқанда стектен өшірілуі мүмкін келесі элемент барлық уақытта стекке соңғы енгізілген элемент болып табылады. Біз стекке элементтерді енгізе аламыз, бірақ элементтерді өшіру кезінде біз тек соңғы енгізілген элементті ғана өшіре аламыз.

Асханадағы тәрелкелер мен подностар стектің деректер құрылымына мысал болып табылады. Подностар стегі қалай қолданылатынын қарастырайық. Поднос қажет болған жағдайда барлығы оны стектің жоғарғы жағынан бастап алады. Подносты төменгі жағынан алу қауіпті, ол стектегі бүкіл ыдысты бұлдіруі мүмкін. Подностарды жинақтағанда олар стектің жоғарғы жағына орналастырылады.



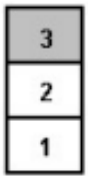
1-сурет **2-сурет**

Алдымен бос стекке бірнеше элементті қосуды қарастырып көрейік. 1-мәніне тең элементті енгіземіз. 1-сурет осы мысалды бейнелейді.

Әрі қарай стекке екінші элементті қоямыз.

2-сурет стекке 2 мәніне тең элементті енгізгеннен кейінгі күйін көрсетеді.

Сұр түске боялған элемент стектің жоғарғы жағында орналасқан. Екінші элементті қосу бірінші элементті толығымен жасырады, себебі екінші элемент жойылуы тиіс жалғыз элемент болып табылады. Сонымен қатар қолжетімді болатын стектегі жалғыз элемент. Бұл – стектің векторлар мен қосбетті кезектерден айырмашылығы. Стекте деректер құрылымында сақталған элементтерге қолжетімділік бар, бірақ олардың мәндеріне қолжетімсіз болып келеді. Біз стекті және тек жоғарғы бөлігінде ғана тұрған элементтің мәнін көре аламыз.



3-сурет

3-сурет үшінші элемент қосылғаннан кейінгі стекті көрсетеді. Енгізілген элементтің мәні 3-ке тең. Егер стектен элемент өшіру қажет болса, онда стек 2-суреттегі бейнеге енеді.

STL stack Adapter-ін қолдану

STL стек адаптері стек ретінде қолданылатын интерфейспен жабдықтайды.

```
 1: #include <iostream>
 2: #include <string>
 3: #include <cstdlib>
 4: #include <stack>
 5:
 6: using namespace std;
 7:
 8: int main(int argc, char* argv[]) {
 9:
10:     stack<int> s;
11:
12:     // push and pop
13:     s.push(1);
14:     s.pop();
15:
16:     // top
17:     s.push(10);
18:     s.push(11);
19:     cout << s.top() << endl;
20:
21:     // size and empty
22:     cout << s.size() << endl;
23:     cout << s.empty() << endl;
24:
25:     return EXIT_SUCCESS;
26: }
```

2-листинг

Стек адаптері интерфейсімен жұмыс істейтін алты функция мүшелер бар. ***push and pop*** әдістері стектегі сәйкесінше элементтерді енгізеді, жояды, ***top*** әдісі стектің жоғарғы жағында сақталған ақпараттарға сілтеме жасайды. ***Queue*** адаптері секілді ***size*** әдісі стекте сақталған элементтерді қайтарады және ***empty*** әдісі ***true*** мәнін береді, егер стек бос болса, кері жағдайда ***false*** мәнін береді.

Программада ***STL*** стек адаптерін қолдану үшін программалаушы кітапханасын қосуы керек. Бұл 1-листингтің 4-жолында көрсетілген. 2-листинг стек адаптерінің тәжірибеде кездесетін формасын көрсетеді. Бұл тізімдер текстік файлдар қатарын кері ретпен экранға шығарады. Стек бұл мәселені шешу үшін деректер құрылымымен қамтамасыз етеді.

```
 1: // open file specified by command-line argument
 2: ifstream inf(argv[1]);
 3:
 4: if ( !inf ) {
 5:  cerr << "cannot open " << filename << "for input"<< endl;
 6:  return EXIT_FAILURE;
 7: }
 8:
 9: stack<string> s;
10: string line;
11:
12: // read file line by line
13: while (getline(inf, line)) {
14:     s.push(line);
15: }
16:
17: inf.close();
18:
19: // print lines in reverse
20: while (!s.empty()) {
21:     cout << s.top() << endl;
22:     s.pop();
23: }
```

3-листинг

РЕКУРСИЯ, РЕКУРСИВТІ ФУНКЦИЯЛАР, СТЕКТІ ШАҚЫРУ

Рекурсивті функциялар

Рекурсивті функция – өзін-өзі шақыра алатын функция. Рекурсивті функцияларды қолдану күрделі есептерді шешуге мүмкіндік береді. C++ тілі өзге көптеген тілдер секілді рекурсиямен жұмыс істейді. Өзін-өзі шексіз шақыратын функция құрылмас үшін рекурсивті функцияларды абайлап қолдану ұсынылады. Типтік рекурсияның псевдокоды келесі түрде болады.

```
if (simplest case) then
    solve directly
else
    make recursive call to a simpler case
```

1-мысал

Тиімді рекурсияларды құру және қолдану бұл үлкенді кішіге бөлу болып табылады. Нәтижесінде проблема рекурсияны қолданбай шығаруға мүмкіндік беретіндей қысқарады. Ол ***base case*** шақырады.

Факториалды шешу рекурсияны шешудің мысалы болып табылады. N санының факториалы N-нен 1-ге дейінгі сандардың көбейтіндісіне тең. Мысалы, 5 санының факториалы келесідей болады 5 * 4 * 3 * 2 * 1. Ол 120-ға тең. 3 санының факториалы келесідей болады 3 * 2 * 1. Ол 6-ға тең. Леп (!) белгісі факториал санын береді.

Факториалдарды рекурсивті түрде бейнелейік. Мысалы, 5 санының факториалын табайық.

2-мысалдан 5! = 5 * 4 * 3 * 2 * 1 екенін көреміз. Бірақ 4 саны үшін 4! = 4 * 3 * 2 * 1 болатынын білеміз. Олай болса, рекурсивті түрде 5! = 5 * 4! жазуға болады.

3-мысалда 2-мысалдағы сандардың рекурсивті анықталуы бейнеленген. Біз нөлдік факториалды рекурсивті бейнелей алмағандықтан, ол – ***base case***.

$5! = 5 * 4 * 3 * 2 * 1 = 120$
$4! = 4 * 3 * 2 * 1 = 24$
$3! = 3 * 2 * 1 = 6$
$2! = 2 * 1 = 2$
$1! = 1$
$0! = 1$

2-мысал

$5! = 5 * 4!$
$4! = 4 * 3!$
$3! = 3 * 2!$
$2! = 2 * 1!$
$1! = 1 * 0!$
$0! = 1$

3-мысал

Төмендегі листингте C++ тілінде факториалды рекурсивті есептеу коды берілген

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

using namespace std;

int factorial(int n) {

    if (n == 0) {
        // base case
        return 1;
    }
    else {
        // recursive call
        int value = factorial(n - 1);
        return n * value;
    }
}

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

    cout << factorial(5) << endl;
    return EXIT_SUCCESS;
}
```

Жоғарыдағы листинг нәтижесінде экранға 120 саны шығады. Бірақ шығарылу жолы түсініксіз, сондықтан экранға шығару операторын енгіземіз. 2-листинг факториалдың жаңартылған түрі болып табылады.

```
int factorial(int n) {

    cerr << "factorial(" << n << ") begin" << endl;

    if (n == 0) {
        cerr << "factorial(" << n << ") returns 1" << endl;
        return 1;  // base case
    }
    else {
     int ret = n * factorial(n - 1); // recursive call
     cerr << "factorial(" << n << ") returns "<< ret << endl;
     return ret;
    }
}
```

2-листингтің нәтижесі:

```
factorial(5) begin
factorial(4) begin
factorial(3) begin
factorial(2) begin
factorial(1) begin
factorial(0) begin
factorial(0) returns 1
factorial(1) returns 1
factorial(2) returns 2
factorial(3) returns 6
factorial(4) returns 24
factorial(5) returns 120
120
```

Нәтижесінен көріп тұрғанымыздай, программа алдымен параметрі 5-ке тең ***factorial*** функциясын шақырады. ***Main*** функциясы алғашқы осы шақыруды орындайды. Орындалу барысында ***factorial*** (5) аргументі 4-ке тең ***factorial*** функциясын шақырады. ***factorial*** (4) орындалып, ***factorial*** (3)-ті шақырады, дәл осылай ***factorial*** (0) шамасы 1-ге тең мәнді қайтарғанша алоритм жалғаса береді. Содан соң ***factorial*** (1) ***factorial*** (2)-ге шамасы 1-ге тең мәнді, ***factorial*** (2) ***factorial*** (3)-ке шамасы 2-ге тең мәнді қайтарады, дәл осылай ***factorial*** (5) ***main*** функциясына 120 мәнін қайтарғанша цикл жалғаса береді

Стекті шақыру

Call Stack – орындалатын функцияларды және кезекте тұрған өзге де операциялардың орындалуын басқару үшін қолданатын программаның жады аймағы. 3-сурет «***кезектегі функцияны шақыру***» деп аталатын түсінікті бейнелейді.

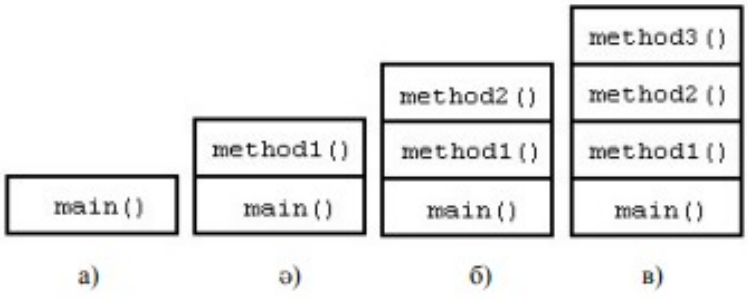
1-суретте параметрі 5-ке тең ***factorial*** функциясының көшірмесі программаның орындалуының алғашқы және соңғы нүктесі болғаны көрініп тұр. Программа ***factorial*** (5) орындауын ***factorial*** (4) функциясы жұмысын аяқтағаннан соң тоқтатады. Сол секілді ***factorial*** (4) функциясы ***factorial*** (3) функциясының орындалып біткенін күтеді. Осы амалдардың барлығы ***стекті шақыру*** арқылы басқарылады.

Мысал ретінде C++ тілінде жазылған келесі программаны қарастырайық:

```
 1:  #include <iostream>
 2:  #include <cstdlib>
 3:
 4:  using namespace std;
 5:
 6:  void method3(void) {
 7:      cout << "Method 3" << endl;
 8:  }
 9:
10:  void method2(void) {
11:      method3();
12:  }
13:
14:  void method1(void) {
15:      method2();
16:  }
17:
18:  int main(int argc, char* argv[]) {
19:      method1();
20:      return EXIT_SUCCESS;
21:  }
```

Функцияларды шақыру арқылы амалдары орындау барысында программадағы шақыруларды басқару үшін программа стекті шақыруды қалай қолданатыны 3-листингте көрсетілген.

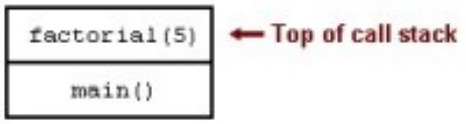
C++ тілінде орындалатын алғашқы амалдардың барлығы ***main*** функциясында жүзеге асырылады. 2-а суретте ***main*** функциясы жұмысын бастап ***method1*** функциясын шақырғанға дейінгі стектің қалып күйін бейнеленген. ***Main*** функциясы ішкі программа болғандықтан және орындалып жатқандықтан ол бірінші ақпаратты оқиды, бұл – стекті шақыру шегінде тұрған функция. Функцияның локальды айнымалыларын қосатын ақпаратты ***белсенділік жазуы*** деп атайды.



2-сурет

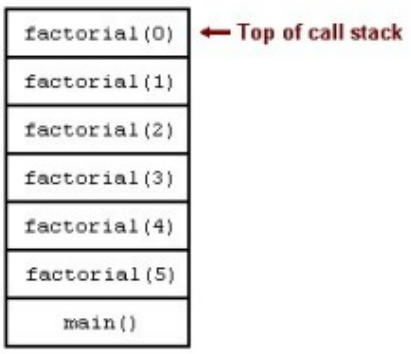
main функциясы ***method1*** шақырғанда орындалу уақытының жүйесі стекті шақыру шегінде ***method1-***ге жазуды белсенді етеді. Содан соң орындалу уақытының жүйесі ***method1*** аяқталуын күтіп ***main*** функциясының орындалуын тоқтатады. 11.2-ә суретте программаның осы нүктесіндегі стектің күйін бейнелейді. Содан соң ***method1*** функциясы ***method2*** функциясын шақырады. ***method1*** функциясының жұмысы ***method2*** функциясының жұмысының аяқталғанын күтеді. Бұл 1-б суретінде бейнеленген. ***method2*** функциясы жұмысын аяқтағаннан соң ***method3*** функциясын шақырады, ол 11.2-в суретінде бейнеленген

Стектерді шақыру рекурсивті функциялармен осы әдіспен жұмыс істейді. Стектерді шақыру рекурсивті ***factorial*** функциясымен басқарылатындықтан стекті шақыруды қайталайық. Программаның орындалуы ***main*** функциясынан басталады. ***Main*** функциясы параметрі 5-ке тең ***factorial*** функциясын шақырады. Жүйелі ***run-time*** орындалуы параметрі 5-ке тең факториалды стектің жоғарғы жағынан бастауы тиіс. Осы нүктедегі стек шақыруы 3-суретте бейнеленген.



3-сурет

factorial (5) функциясын шақыру өз кезегінде ***factorial*** (3) функциясын шақыратын ***factorial*** (4) функциясын шақырады, дәл осылай ***factorial*** (0) функциясы өз жұмысын бастағанша жалғаса береді. Стекті осы нүктеде шақыру 4-суретте бейнеленген.



4-сурет

factorial (0) функциясы ***base case*** рекурсия болып табылады. Функцияның бұл көшірмесі жұмысын аяқтағаннан соң ол өзінің төменгі жағында орналасқан жазу белсенділігіне 1 сан мәнін қайтарады, содан соң орындалу уақытының жүйесі жазу белсенділігін ***factorial*** (0) жылжытып, ***factorial*** (1) функциясын орындауға кіріседі. Осылайша, ***main*** функциясы жұмысын аяқтағанша цикл жалғаса береді, яғни программа аяқталады.