Obyektyönlü proqramlaşdırma

C++

Dərs №7

C++ dili ilə obyektyönlü proqramlaşdırma

Mündəricat

Dinamik verilənlər strukturu anlayışı.....3

Stek.....5

Stek üzərində əsas əməliyyatlar və onun elementləri.....5
Stekin reallaşdırılması.....6

Növbə.....10

Növbənin reallaşdırılması.....11
Dairəvi növbə.....15
Dairəvi növbənin reallaşdırılması.....15
Prioritetli növbə.....19
Prioritetli növbənin reallaşdırılması.....20

Ev tapşırığı.....26

Dinamik verilənlər strukturu anlayışı

Bu gün biz dinamik verilənlər strukturları konsepsiyası ilə tanış oluruq. İndiyədək biz yalnız proqram icrası zamanı dəyişməyən, statik strukturlu verilənlər ilə işləmişik. Paradoks, hətda dinamik massiv o qədər də dinamik deyil, onun ölçüsünü dəyişdirmək üçün onu yenidən yaratmaq lazım gəlir. Başqa sözlə desək, proqram işlədiyi zaman elementlərin qiymətini yalnız dəyişdirmək daha asandır, elementlərin sayını dəyişdirmək isə bir çox monoton prosedurlara gətirib çıxardır. Bu isə hər zaman rahat iş deyil.

Hesab edək ki, sinif şagirdlərin məlumatlarını daxil və emal etmək üçün proqramda verilənləri yadda saxlamaq üçün statik massivlər istifadə edilir. Massivin ölçüsünü təyin etmək üçün proqramçıya sinifdəki şagirdlərin orta və ya maksimum sayını əsas götürmək lazım gəlir. Bu halda, əgər şagirdlərin sayı nəzərdə tutduğumuzdan azdırsa, kompüterin yaddaşı səmərəli istifadə olunmur, çox olduqda isə, proqramın istifadəsi mümkün deyil (bu halda koda dəyişikliklər edilməli və kompilyasiya olunmalıdır). Dinamik massivin yaradılması çıxış yolu ola bilər, ancaq kodu yazmaqda bir çox çətinliklər yaradacaq.

Təbiəti etibarilə dinamik olan verilənləri emal edən məsələləri dinamik struktur ilə həll etmək daha asandır. Hal-hazırda bu üsül ilə işləməyi öyrənəcəyik.

Yuxarıda dediklərimizi nəzərə alsaq, təxmin edə bilərik ki, dinamik struktur bir konstruksiyadır ki, hansı ki, proqram icra edildikdə ehtiyyac olduqda yeni elementlər üçün müəyyən yaddaş ayırır və ya lazımsız elementlər üçün ayrılmış yaddaşı silir. Dinamik strukturlu verilənlərin ünvanlaşdırılması problemin həlli üçün bir metod istifadə olunur, adı isə yaddaşın dinamik paylanması adlanır, yəni, yaddaş o zaman paylanılır ki, proqram kompilyasiya yox, icra olunur. Kompilyator bu zaman elementin özünü yox, onun ünvanını yadda saxlamaq üçün müəyyən yaddaş ayırır.

Bir neçə növ dinamik strukturlu verilənlər mövcuddur. Onların üstünlüyü olduğu qədər çatışmamazlıqları da var. Bu səbəbdən, hansının istifadə edilməsi ancaq qarşıya qoyulan məsələdən asılıdır.

Stek

Stek – dinamik verilənlər strukturudur, hardakı yeni elementlərin əlavə edilməsi və mövcud olan elementlərin silinməsi bir ucdan baş verir - stekin zirvəsindən.

Həmçinin, stekdə baza ünvanı mövcuddur. Bu anlayış altında stekin yerləşdiyi yaddaşın başlanğıc ünvanı nəzərdə tutulur.

Elementlər stekdən müəyyən ardıcıllıqla çıxarılır, yəni burada bu prinsip işləyir LIFO (*Last In First Out*) və ya «axırıncı gələn birinci gedər».

Stek misalı kimi uşaq piramidasını misal çəkmək olar. Burada halqaların əlavə olunması və çıxarılması bu qayda üzrə gedir.

Steki yadda saxlamaq üçün yaddaşda müəyyən sahə ayrılır. Onun sərhəd ünvanı stekin fiziki strukturunun parametridir.

Əgər steki doldurarkən göstərici yuxarıdadırsa, sahənin kənarlarından qırağa çıxırsa, o zaman stekin daşması baş verir. Belə olduqda bu halın aradan qaldırılması tələb olunur.

Stek və elementləri üzərində əsas əməliyyatlar

1. Elementin stekə əlavə olunması.
2. Elementin stekdən silinməsi.

3. Yoxlama, stek boşdurmu (stek o zaman boş sayılır ki, zirvənin göstəricisi aşağı sərhəd göstəricisi ilə eyni olsun)
4. Zirvədə olan elementə silməmiş baxmaq.

Təlimat. *Dinamik strukturlu stek çox vaxt ifadələrin sintaksis analizi üçün istifadə edilir.*

Qeyd. Yeri gəlmişkən, hər bir proqramın öz steki var. Burada kompilyator lokal dəyişənləri yaradır. Həmçinin, stek üzərində parametrlər funksiyaya keçid edir.

Stekin reallaşdırılması

```
#include <iostream>
#include <string.h>
#include <time.h>
using namespace std;
```



```
class Stack
{//Stekin yuxarı və aşağı sərhədləri
     enum {EMPTY = -1, FULL = 20};
    //Verilənləri yadda saxlamaq üçün massiv
     char st[FULL + 1];
    //Stekin yuxarı göstərəcisi
    int top;
public:
    //Konstruktor
    Stack();
    //Elementin əlavə edilməsi
    void Push(char c);
    //Elementin çıxarılması
    char Pop();
    //Stekin boşaldılması
     void Clear();
    //Elementin stekdə olub olmamasının
    //yoxlanılması
    bool IsEmpty();
    //Stekin dolmasının yoxlanılması
    bool IsFull();
    //Stekdə elementlərin sayı
    int GetCount();
};
Stack::Stack()
{
//Əvvəlcədən stek boşdur
  top = EMPTY;
}
```

```
void Stack::Clear()
{
    //Stekin səmərəli təmizlənməsi
    //(verilənlər massivdə hələ də mövcuddur,
    //ancaq stekin zirvəsi ilə işləyən
    //siniflərin funksiyaları
    //onları nəzərə almayacaq)

    top = EMPTY;

bool Stack::IsEmpty()
{
    //Boşdur?
    return top == EMPTY;
}

bool Stack::IsFull()
{
    //Doludur?
    return top == FULL;
}

int Stack::GetCount()
{
    //Stekdə elementlərin sayı
    return top + 1;
}
void Stack::Push(char c)
{
    //Əgər stekdə yer varsa, stekin zirvəsinin
    //göstəricini artırırıq və yeni element əlavə
    //edirik
    if(!IsFull())
        st[++top] = c;
}
char Stack::Pop()
{
    //Əgər stekdə element varsa, üstdəkini
```

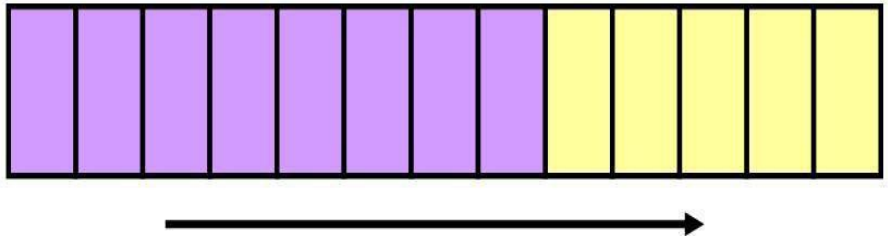


```
    //qaytarırıq və göstəricini bir vahid azaldırıq
    if(!IsEmpty())
        return st[top--];
    else //Əgər stekdə elementlər yoxdursa
        return 0;
}

void main()
{
    srand(time(0));
    Stack ST;
    char c;
    //hələ ki stek dolmamışdır
    while(!ST.IsFull()){
    c=rand()%4+2;
        ST.Push(c);
    }
    //hələ ki stek boşalmamışdır
    while(c=ST.Pop()){
        cout<<c<<" ";
    }
    cout<<"\n\n";
}
```

Növbə

Növbəti dinamik struktur sadə növbədir. Praktikada növbəni, stekdə olduğu kimi, massiv vasitəsilə reallaşdırmaq olar.

Növbə — uzunluğu fərqli ola bilən elementlər ardıcılığından ibarətdir. Növbəyə elementlərin əlavə olunması bir tərəfdən baş verir, silinmə isə digər tərəfdən. Bu konstruksiya bu ideologiya üzərində işləyir: FIFO (First In — First Out), yəni «birinci gələn — birinci gedər». Növbə üçün elementlərin sonlu ardıcıllığını ayırmaq olar. Bu halda zamanın hər bir anında növbənin elementləri yalnız ardicil elementlərin bir hissəsi məşğuldur.



Prinsipcə, adi növbə ilə biz hər an qarşılaşırıq. Burada bir neçə misal verilib:

Mavzoley növbəsi, printerdə çap növbəsi, həmçinin xətti alqoritm də növbə ilə işləyir.

Növbənin reallşdırılması

```
#include <iostream>
#include <string.h>
#include <time.h>
using namespace std;

class Queue
{
    //Növbə
    int * Wait;
    //Növbənin maksimal ölçüsü
    int MaxQueueLength;
    //Hal-hazırdakı növbənin ölçüsü
    int QueueLength;

public:
    //Konstruktor
    Queue(int m);
   //Destruktor
    ~Queue();
   //Elementlərin əlavə olunması
void Add(int c);
//Elementlərin çıxarılması
int Extract();
//Növbənin təmizlənməsi
void Clear();
//Növbədə olan elementlərin
yoxlanılması
bool IsEmpty();
    //Növbənin dolu olmasının yoxlanışı
    bool IsFull();
```

```
    //Növbədə elementlərin sayı
    int GetCount();
    //Növbənin nümayiş etdirilməsi
    void Show();
};
void Queue::Show(){_________________________________
    cout<<"\n                                    \n";
    //Növbənin nümayiş etdirilməsi
    for(int i=0;i<QueueLength;i++){
        cout<<Wait[i]<<" ";
    }_____________________________________
    cout<<"\n                                    \n";
}

Queue::~Queue()
{
    //Növbənin silinməsi
    delete[]Wait;
}

Queue::Queue(int m)
{
    //ölçünü alırıq
    MaxQueueLength=m;
    //növbə yaradırıq
    Wait=new int[MaxQueueLength];
    //əvvəlcədən növbə boşdur
    QueueLength = 0;
}

void Queue::Clear()
{
    //Növbənin səmərəli təmizlənməsi
    QueueLength = 0;
}
```



```
bool Queue::IsEmpty()
{
    //Boşdur?
    return QueueLength == 0;
}
bool Queue::IsFull()
{//Növbə doludur?
    return QueueLength == MaxQueueLength;
}

int Queue::GetCount()
{//Növbədə elementlərin sayı
return QueueLength;
}
void Queue::Add(int c)
{
    //Əgər növbədə yer varsa, o zaman
    //elementlərin sayını bir vahid
    //artiririq
    if(!IsFull())
        Wait[QueueLength++] = c;}
int Queue::Extract()
{//əgər növbədə element varsa,
//o zaman birinci gələni
//qaytarırıq və növbəni sürüşdürürük
if(!IsEmpty()){
        //birincini yadda saxla
        int temp=Wait[0];
        //bütün elememntləri sürüşdür
        for(int i=1;i<QueueLength;i++)
            Wait[i-1]=Wait[i];
```

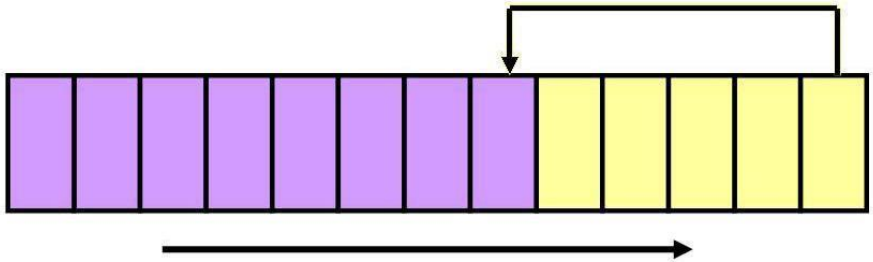
```
        //sayı azaltmaq
        QueueLength--;

        //birincini
        qaytarmaq(sifirinci)
        return temp;
    }

    else //Əgər stekdə element
        //yoxdursa
         return -1;
}

void main()
{
    srand(time(0));
    //Növbənin yaradılması
    Queue QU(25);

    //Elementlərin bir hissəsinin
    //əlavə edilməsi
    for(int i=0;i<5;i++){
        QU.Add(rand()%50);
    }
    //Növbənin göstərilməsi
    QU.Show();

    //Elementin çıxarılması
    QU.Extract();
    //Növbənin göstərilməsi
    QU.Show();
}
```


13

Dairəvi növbə

Dairəvi növbə sadə növbəyə çox oxşayır. O da bu ideologiya ilə işləyir FIFO,
birinci daxil olan, birinci çıxar.

Fərqi ondadır ki, əvvəldən çıxan element axıra yerləşdirilir.



Ən sadə misal olaraq, suyun təbiətdə burulğanı, tramvaylar, hansıların ki, dairəvi marşrutu olur.

Dairəvi növbənin reallşdırılması

```
#include <iostream>
#include <string.h>
#include <time.h>
using namespace std;

class QueueRing
{
    //Növbə
    int * Wait;
    //Növbənin maksimal ölçüsü
    int MaxQueueLength;
```

```
      //Növbənin hal-hazırdakı ölçüsü
      int QueueLength;
 public:
      //Konstruktor
      QueueRing(int m);
      //Destruktor
      ~QueueRing();
      //Elementlərin əlavə olunması
      void Add(int c);
      //Elementlərin çıxarılmas
      bool Extract();
//Növbənin təmizlənməsi
void Clear();
      //Növbədə elementlərin olmasının
      yoxlanılması
      bool IsEmpty();
      //Növbənin dolu olmasının yoxlanılması
      bool IsFull();
      //Növbədə elementlərinin sayı
      int GetCount();
//Növbənin nümayiş etdirilməsi
void Show();
 };
 void QueueRing::Show(){_________________________________
      cout<<"\n                                        \n";
      //Növbənin nümayiş etdirilməsi
      for(int i=0;i<QueueLength;i++){
          cout<<Wait[i]<<" ";
      }
```

```
cout<<"\n---------------------------------- \n";
}

QueueRing::~QueueRing()
{
    //növbənin silinməsi
    delete[]Wait;
}

QueueRing::QueueRing(int m)
{
    //ölçünü alırıq
    MaxQueueLength=m;
    //növbə yaradırıq
    Wait=new int[MaxQueueLength];
    //növbə əvvəldən boşdur
    QueueLength = 0;
}

void QueueRing::Clear()
{
    //Növbənin səmərəli təmizlənməsi
    QueueLength = 0;
}

bool QueueRing::IsEmpty()
{
    //Boşdur?
    return QueueLength == 0;
}

bool QueueRing::IsFull()
{
    // Doludur?
    return QueueLength == MaxQueueLength;
}
```

```
int QueueRing::GetCount()
{
    //Növbədə element sayı
    return QueueLength;
}

void QueueRing::Add(int c)
{
    //Əgər növbədə boş yer varsa, dəyəri
    //artırırıq və yeni element əlavə edirik
    if(!IsFull())
        Wait[QueueLength++] = c;
}

bool QueueRing::Extract()
{
    //Əgər növbədə element varsa, ilk gələni
    qaytarırıq və növbəni sürüşdürürük
    if(!IsEmpty()){
        //birinci yadda saxla
        int temp=Wait[0];
        //bütün elementləri sürüşdür
        for(int i=1;i<QueueLength;i++)
            Wait[i-1]=Wait[i];

        //birinci çıxarılmış elementi sona
        //yerləşdiririk
        Wait[QueueLength-
        1]=temp; return 1;
    }

    else return 0;
}

void main()
{
    srand(time(0));
```



```
    //növbə yaratmaq
    QueueRing QUR(25);

    //elementlərin bir qisminin əlavə edilməsi
    for(int i=0;i<5;i++){
        QUR.Add(rand()%50);
    }
    //növbənin nümayiş etdirilməsi
    QUR.Show();

    //elementin çıxarılması
    QUR.Extract();

    // növbənin nümayiş etdirilməsi
    QUR.Show();
}
```

Prioritetli növbə

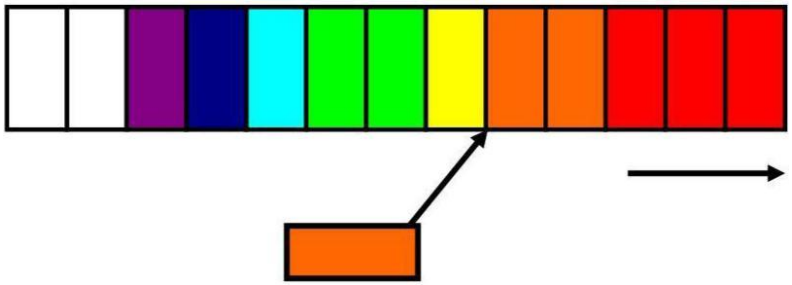
Biz artıq 2 növ növbə ilə taniş olduq, onlar çox asan idilər. Ancaq bir növbə də var— **prioritetli ilə növbə**.

Bəzən elə növbə qurmaq lazım gəlir ki, elementlərin çıxışı müəyyən prioritetə görə olsun. Prioritetlər yerində rəqəm, sabit (konstant) və s. ola bilər. Çıxış zamanı prioriteti daha yüksək olan element seçilir ki.

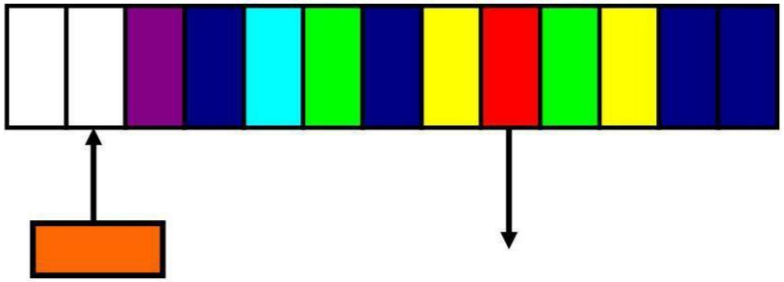
Bir neçə növ prioritetli növbə var:

1. **Prioritetli daxil etmə növbəsi** — elementlərin ardıcıllığı ciddi şəkildə nizamlanıb. Başqa sözlə, hər element növbəyə daxil olarkən,

növbədə prioritetə görə nizamlanır. Çıxış zamanı isə element başlanğıcdan çıxarılır.



B **Prioritetli çıxış növbəsi** — element növbənin sonuna əlavə edilir, çıxış zamanı ən prioritetli element çıxır, hansıkı daha sonra növbədən silinir



Prioritetli növbənin reallaşdırılması

```
#include <iostream>
#include <string.h>
#include <time.h>
using namespace std;
class QueuePriority
{
        //Növ
    bə
        int *
    Wait;
        //Pri
    oritet int
    * Pri;
        //Maksimal ölçü
        int MaxQueueLength;

        //Növbənin hal-hazırki ölçüsü
        int QueueLength;

      public:
        //Konstruktor
        QueuePriority(int m);
        //Destruktor
        ~QueuePriority();
        //Elementin əlavə
    edilməsi void Add(int c,int
    p);
        //Elemenlərin
    çıxarılması int Extract();
        //Növbənin
    təmizlənməsi void
    Clear();

        //Elementlərin növbədə olmasının
    yoxlanılması
        bool IsEmpty();

        //Növbənin dolmasının yoxlanılması
        bool IsFull();

        //Növbədə elementlərin sayı
        int GetCount();

        //növbənin nümayiş etdirilməsi void
    Show();
};
```

```
void QueuePriority::Show(){--------------------------------
    cout<<"\n                                          \n";
    // növbənin nümayiş etdirilməsi
    for(int i=0;i<QueueLength;i++)
             { cout<<Wait[i]<<" - "<<Pri[i]<<"\n\n";
    }----------------------------------
    cout<<"\n                                          \n";
}

QueuePriority::~QueuePriority()
{
    //növbənin silinməsi
    delete[]Wait;
    delete[]Pri;
}

QueuePriority::QueuePriority(int m)
{
    //ölçünü alırıq
    MaxQueueLength=m;
    //növbə yaradırıq
    Wait=new int[MaxQueueLength];
    Pri=new int[MaxQueueLength];
    //əvvəlcədən növbə boşdur
    QueueLength = 0;
}

void QueuePriority::Clear()
{
   //Növbənin səmərəli təmizlənməsi
    QueueLength = 0;
}

bool QueuePriority::IsEmpty()
{
    //Boşdur?
    return QueueLength == 0;
}
```

```
bool QueuePriority::IsFull()
{
    //Doludur?
    return QueueLength == MaxQueueLength;
}

int QueuePriority::GetCount()
{
    //Növbədə olan elementlərin sayı
    return QueueLength;
}

void QueuePriority::Add(int c,int p)
{
    //Əgər növbədə boş yer varsa,
    //O zaman sayı artıb, elementi əlavə edirik
    if(!IsFull()){
        Wait[QueueLength] = c;
        Pri[QueueLength] = p;
        QueueLength++;
    }
}

int QueuePriority::Extract()
{
    //Əgər növbədə element varsa, o zaman prioriteti
    //yüksək olan element qaytarılır və növbə
    //sürüşdürülür
    if(!IsEmpty()){
        //prioritet elementi 0 götürək
        int max_pri=Pri[0];
        //prioritet indeksi isə =

        0 int pos_max_pri=0;

        //prioriteti axtaraq
        for(int i=1;i<QueueLength;i++)
```

```
            //əgər daha prioritetli element tapılsa:
            if(max_pri<Pri[i]){
                max_pri=Pri[i]
                pos_max_pri=i;
            }
        //prioritetli elementi çıxardırıq
        int temp1=Wait[pos_max_pri];
        int temp2=Pri[pos_max_pri];

        //bütün elementləri sürüşdür
        for(int i=pos_max_pri;i<QueueLength-1;i++)
        { Wait[i]=Wait[i+1];
            Pri[i]=Pri[i+1];
        }
        //sayı azaldırıq
        QueueLength--;
        //çıxışa verilən elementi qaytarır

        return temp1;

    }
    else return -1;
 }
void main()
 {srand(time(0));
//növbənin yaradılması
    QueuePriority QUP(25);
    //elementlərin bir qismini

    //növbəyə yerləşdirilməsi

    for(int i=0;i<5;i++){

    //qiymət 0-dan 99-a

    //prioritet 0-dan 11-ə

        QUP.Add(rand()%100,rand()%12);

    //növbənin nümayiş etdirilməsi
    QUP.Show();

    //elementin çıxarılması
    QUP.Extract();

    //növbənin nümayiş etdirilməsi
    QUP.Show();
}
```

Ev tapşırığı

1. “Birəlli quldur” oynu immmitasiya edən oyun yarat. Məsələn, “Enter” düyməsinə basdıqda 3 dənə çarx dövr etməyə başlayır (təbii ki, hər birisinin dövr etmə sayı təsadüfən seçilir) hər birində müxtəlif işarələr var, əgər hər hansı kombinasiya alınsa, oyunçu hədiyyə qazanır.
2. "Marşrut taksilərin dayanacağı"nın imitasiya modelini qurun. Aşağıdakı məlumatlar daxil edilməlidir: günün müxtəlif vaxtlarında avtobus dayanacağına sərnişinlərin gəlməsinin orta vaxtı, günün müxtəlif vaxtlarında avtobus dayanacağına marşrutların gəlməsinin orta vaxtı, dayanacaq növü (son dayanacaq, ya da yox). Müəyyən etmək lazımdır: sərnişinin dayanacaqda dayanmasının orta vaxtı, marşrutların gəlməsinin intervalı, dayanacaqda ən azı N sərnişin olmalıdırlar. Marşrutda boş yerlərin sayı təsadüfi bir qiymətdir.
3. Printerdə çapın növbəsini imitasiya edən proqram yazın. Hər birinin öz prioriteti olan müştərilər olmalıdır və onlar printerə sorğu göndərirlər. Hər yeni müştəri prioritetindən asılı olaraq növbəyə düşür. Çapın statistikasını (istifadəçi, vaxt) digər növbədə yadda saxlamaq vacibdir. Statistikanın ekrana verilməsini nəzərə almaq.