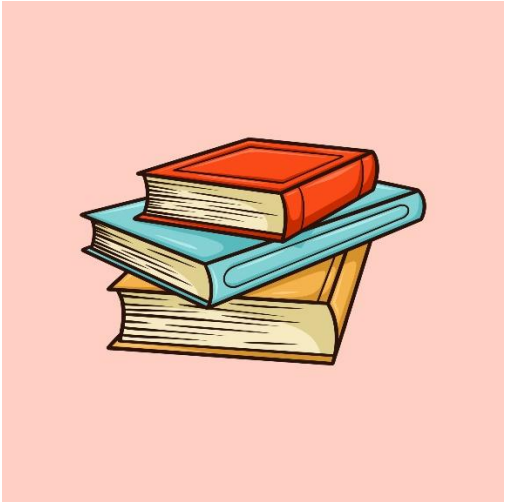
1. Ma’lumotlarning abstrak turlari (Stek va Navbat)

1.1 Stek

Stek – Stack inglizchadan uyum, g‘aram, dasta, bog‘lam degan ma’noni anglatadi. Stek - bu LIFO (last in – first out; oxirgi kelgan – birinchi ketadi) prinsipi bo‘yicha ishlaydigan ma’lumotlar strukturasi. Ushbu strategiyada, oxirgi kiritilgan element birinchi bo‘lib chiqadi. Haqiqiy misol sifatida siz bir-birining ustiga qo‘yilgan kitoblar to‘plamini olishingiz mumkin. Biz oxirgi qo‘ygan kitob tepada va pastda joylashgan kitobni olib tashlashz uchun, oxirgi qo‘yilgan kitob birinchi bo‘lib chiqadi deb aytishimiz mumkin.



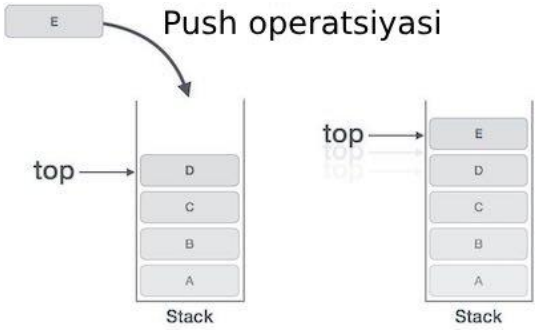
Rasm- 1 LIFO (last in – first out; oxirgi kelgan – birinchi ketadi) prinsipi

C++ dasturlash tili stekda turli amallarni bajarish uchun turli usullarni taqdim etadi.

Operatsiya	Tavsifi
push()	stekga element qo‘shadi
pop()	elementni stekdan olib tashlaydi
top()	stekning yuqori qismidagi elementni qaytaradi

size()	stekdagi elementlar sonini qaytaradi
empty()	stek bo‘sh bo‘lsa, true qiymatini qaytaradi

Push:stekga element qo‘shadi. Agar stek to‘lgan bo‘lsa, unda stekga element qo‘shib bolmaydi javobini qaytaradi.



Rasm- 2 Push operatsiyasining bajarilishi

Algoritm (Push)

```
Begin                //Boshlash
 if stack is full    //agar stek to‘lgan bo‘lsa
    return           // qaytarish
 endif
else                 // aks holda
 increment top       // elementni qo‘shish
 stack[top] assign value  // stekning eng tepasiga
end else
end procedure               // tugatish
```

Berilgan algoritm bo‘yicha C++ dasturlash tilida push funktsiyasining implementatsiyasi. C++ da stek yaratish uchun avvalo stek sarlavhasi faylini kiritishimiz kerak.

```
#include <stack>
```

Ushbu faylni import qilgandan so‘ng, biz quyidagi sintaksis yordamida stek yaratishimiz mumkin:

```
stack<type> st;
```

```
#include <iostream>
#include <stack>
using namespace std;

int main() {

  // Stek yaratish
  stack<string> colors;

  // Elementlarni stekka joylashtirish
  colors.push("Red");
  colors.push("Orange");

  cout << "Stack: ";

  // Stek elementlarini chiqarish
   while(!colors.empty()) {
    cout << colors.top() << ", ";
    colors.pop();
  }

  return 0;
}
```

Dastur na’tijasi:

```
Stack: Orange, Red,
Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Yuqoridagi misolda biz colors deb nomlangan satrlar stekini yaratdik. Keyin, biz stekka elementlarni qo‘shish uchun push() usulidan foydalandik.

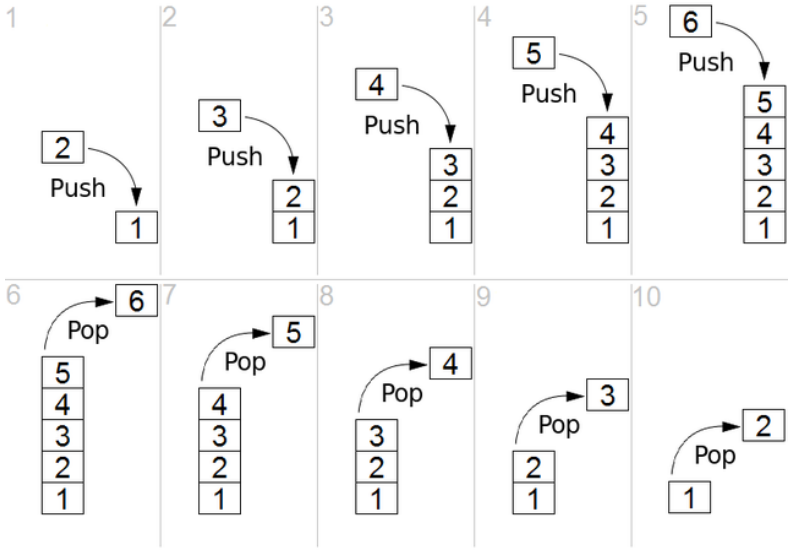
```
colors.push("Red");
colors.push("Orange");
```

Stack tarkibini to‘g’ridan-to‘g’ri chop etish o‘rniga, biz while takrorlanish operatoridan foydalandik.

```
while(!colors.empty()) {
  cout << colors.top() << ", ";
  colors.pop();
}
```

Stekning barcha elementlarini chop etish uchun biz tepa (top) elementini chiqaramaiz keyin tepa (top) elmenetini o‘chiramiz (pop). Ushbu jarayon stek bo‘sh bo‘lguncha qayta-qayta davom etadi.

Pop: Stekdan elementni olib tashlash uchun qo‘llaniladi. Buyumlar teskari tartibda silzitish kerak bo‘ladi. Agar stek bo‘sh bo‘lsa, unda xatolik qaytariladi.



Rasm- 3 Stekda Pop amalining bajarilishi

Algoritm (Pop)

```
Begin                          //Boshlash
 if stack is empty             //agar stek bo‘sh bo‘lsa
```

```
    return                    // qayarish
  endif
 else                         // aks holda
  store value of stack[top]   //
  decrement top               // stek tepasini kamaytirish
  return value                // qiymatni qaytarish
 end else
 end procedure               // protsedurani tugatish
```

Berilgan algoritm bo‘yicha C++ dasturlash tilida pop funktsiyasining implementatsiyasi.

```
#include <iostream>
#include <stack>
using namespace std;
// funktsiya prototipi
void display_stack(stack<string> st);
int main() {
  // string stekni yaratish
  stack<string> colors;
  // push yordamida elementlarni qo‘shish
  colors.push("Red");
  colors.push("Orange");
  colors.push("Blue");

  cout << "Berilgan massiv: ";
  // Stek elementlarini chiqarish
  display_stack(colors);

  // "Blue" rangini o‘chirish
  colors.pop();

  cout << "Stek elementni o‘chirgandan kiyin: ";

  // stek elementlarini chiqarish
  display_stack(colors);

  return 0;
}

// utility function to display stack elements
```

```
void display_stack(stack<string> st) {

  while(!st.empty()) {
    cout << st.top() << ", ";
    st.pop();
  }

  cout << endl;
}
```

Dastur natijasi:

```
/tmp/zmexBdo2kj.o
Berilgan massiv: Blue, Orange, Red,
Stek elementni o'chirgandan kiyin: Orange, Red,
```

Yuqoridagi misolda biz elementni stekdan olib tashlash uchun pop() usulidan foydalandik. Dastlab, stekning tarkibi {"ko‘k", "apelsin", "qizil"}.

```
// eng tepa (top) elementini o‘chirish
colors.pop()
```

Bu stekning yuqori qismidagi elementni, ya’ni oxirgi kiritilgan elementni olib tashlaydi, bu "Blue". Demak, yakuniy stek {"apelsin", "qizil"} ga aylanadi.

Top: stekdagi eng yuqori elementini qaytadi.

```
begin                        //Boshlash
  return stack[top]         // Eng tepadagi elementi qaytarish
end procedure                // Tugatish
```

Berilgan algoritm bo‘yicha C++ dasturlash tilida top funktsiyasining implementatsiyasi. Biz stekning yuqori qismidagi elementga top () usuli yordamida murojad qilamiz.

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
```

```
    // stek yaratish
    stack<string> colors;

    // Stekga elementlarni qo‘shish
    colors.push("Red");
    colors.push("Orange");
    colors.push("Blue");

    // tepadagi elementni topish
    string top = colors.top();

    cout << "Top Element: " << top;

    return 0;
}
```

Dastur na’tijasi:

```
/tmp/zmexBdo2kj.o
Top Element: Blue
```

Yuqoridagi misolda biz colors deb nomlangan satrlar to‘plamini yaratdik va quyidagi elementlarni qo‘shdik: "Red", "Yellow" va "Blue". Keyin biz yuqori elementga kirish uchun top () usulidan foydalandik:

```
    string top = colors.top();
```

Empty: Agar stek bo‘sh bo‘lsa rost (true) aks holda yolg’on (false) qaytaradi.

```
Begin               //Boshlash
 if top < 1        // agar eng tepadagi element 1 dan kichik
bo‘lsa
    return true    // true qaytarilsin
 else               // aks holda
    return false    // false qaytarilsin
end procedure       // Tugatish
```

Berilgan algoritm bo‘yicha C++ dasturlash tilida empty funktsiyasining implementatsiyasi. Stek bo‘sh yoki yo‘qligini tekshirish uchun empty() usulidan

foydalanamiz. Bu usul qaytadi: 1 (rost) - agar stack bo‘sh bo‘lsa,0 (yolg’on) - agar stack bo‘sh bo‘lmasa.

```
#include <iostream>
#include <stack>
using namespace std;

int main() {

  // duble tipda stek yaratish
  stack<double> nums;

  cout << "Stek bo‘shmi ? ";

  // stek bo‘shligini tekshirish
  if (nums.empty()) {
    cout << "Ha" << endl;
  }
  else {
    cout << "Yo‘q" << endl;
  }

  cout << "Elementlarni qo‘shing ..." << endl;

  // Elementlarni qo‘shish
  nums.push(2.3);
  nums.push(9.7);

  cout << "Stek bo‘shmi ? ";

  // Stek bo‘shligini tekshirish
  if (nums.empty()) {
    cout << "Ha";
  }
  else {
    cout << "Yo‘q";
  }

  return 0;
}
```

Dastur na’tijasi:

```
/tmp/zmexBdo2kj.o
Stek bo'shmi ? Ha
Elementlarni qo'shing ...
Stek bo'shmi ? Yo'q
```

Yuqoridagi misolda biz stek bo‘sh yoki yo‘qligini aniqlash uchun empty () funksiyasidan foydalandik.

```
if(nums.empty()) { // falsni qaytaradi
  cout << "Ha" << end;;
}
else {
  cout << "Yo‘q" << endl;
}
```

Dastlab, stekda hech qanday element yo‘q. Shunday qilib, `nums.empty()` “Ha” javobini qaytardi. Kiyin biz stekga element qo‘shtik va `nums.empty()` qaytadan foydalandik bu safar stek bo‘sh emas demak “Yo‘q” javobini qaytardi.

Size(): Stekdagi elementlar sonini olish uchun size() usulidan foydalanamiz.

```
#include <iostream>
#include <stack>
using namespace std;

int main() {

  // stek yaratish
  stack<int> prime_nums;

  // elementlarni stekga qo‘shish
  prime_nums.push(2);
  prime_nums.push(3);
  prime_nums.push(5);

  // stek hajmini topish
  int size = prime_nums.size();
  cout << "Stek hajmi: " << size;

  return 0;
}
```

Dastur na’tijasi:

```
/tmp/zmexBdo2kj.o
Stek hajmi: 3
```

Stekda bajariladigan amallarning vaqt murakkabligi:

Amallar	Vaqt murakkabligi
Push()	O(1)
Pop()	O(1)
Empty()	O(1)
Size()	O(1)

1.2 Stek turlari:

Ruxsat etilgan o‘lchamdagi stek: nomidan ko‘rinib turibdiki, berilgan o‘lchamdagi stek berilgan o‘lchamga ega va dinamik ravishda o‘sishi yoki qisqarishi mumkin emas. Agar stek to‘lgan bo‘lsa va unga element qo‘shishga harakat qilinsa, oqim xatosi paydo bo‘ladi. Agar stek bo‘sh bo‘lsa va undan elementni olib tashlashga harakat qilinsa, oqim xatosi paydo bo‘ladi.

Dinamik o‘lchamdagi stek: dinamik o‘lchamdagi stek dinamik ravishda o‘sishi yoki qisqarishi mumkin. Stek to‘lganida, u yangi elementni joylashtirish uchun avtomatik ravishda hajmini oshiradi va stek bo‘sh bo‘lganda uning hajmini kamaytiradi. Ushbu turdagi stek bog’langan ro‘yxat yordamida amalga oshiriladi, chunki bu stekning o‘lchamini oson o‘zgartirishga imkon beradi.

Ushbu ikkita asosiy turga qo‘shimcha ravishda, Steklarning bir nechta boshqa variantlari mavjud, jumladan:

- Infix to Postfix Stack: ushbu turdagi stek infix iboralarini postfix iboralariga aylantirish uchun ishlatiladi.

- Ifodani baholash to‘plami (Expression Evaluation Stack): ushbu turdagi stek postfiks ifodalarini baholash uchun ishlatiladi.
- Rekursiyaviy stek: ushbu turdagi stek kompyuter dasturidagi funktsiya chaqiruvlarini kuzatib borish va funktsiya qaytganda boshqaruvni to‘g’ri funktsiyaga qaytarish uchun ishlatiladi.
- Xotirani boshqarish to‘plami: ushbu turdagi stek dastur hisoblagichining qiymatlarini va registrlarning qiymatlarini kompyuter dasturida saqlash uchun ishlatiladi, bu funktsiya qaytganda dasturning avvalgi holatiga qaytishiga imkon beradi.

Stekning ilovalari:

- Tahrirlovchilar, photoshop kabi ko‘plab joylarda qayta tiklash xususiyatlarni.
- Veb-brauzerlarda oldinga va orqaga harakatlanish xususiyatlari.
- Xanoy minorasi, daraxt shpallari, Stok oralig’i muammolari va gistogramma muammolari kabi ko‘plab algoritmlarda qo‘llaniladi.
- Topologik saralash va bog’langan komponentlari kuchli bo‘lgan Graf algoritmlarda
- Xotirani boshqarishda har qanday zamonaviy kompyuter ishlaydigan maqsad uchun asosiy boshqaruv sifatida stekdan foydalanadi. Kompyuter tizimida ishlaydigan har bir dastur o‘z xotira taqsimotiga ega.

Stackning afzalliklari:

- *Implementatsiya oson:* stek ma’lumotlar tuzilishini massivlar yoki bog’langan ro‘yxatlar yordamida amalga oshirish va uning operatsiyalarini tushunish va amalga oshirish oson.
- *Xotiradan samarali foydalanish:* Stack qo‘shni xotira blokidan foydalanadi, bu esa uni boshqa ma’lumotlar tuzilmalariga nisbatan xotiradan foydalanishda samaraliroq bo‘lishiga sabab bo‘ladi.

- *Tez kirish vaqti**:*** Stack ma’lumotlar tuzilishi elementlarni qo‘shish va olib tashlash uchun tezkor kirish vaqtini ta’minlaydi, chunki elementlar qo‘shiladi va stekning yuqori qismidan chiqariladi.
- *Funktsiyalar uchun qulay:* stek ma’lumotlar tuzilishi funktsiya chaqiruvlarini va ularning holatlarini saqlash uchun ishlatiladi, bu esa rekursiv funktsiya chaqiruvlarini samarali amalga oshirishga yordam beradi.
- *Backtracking-ni qo‘llab-quvvatlaydi:* Stack data tuzilmasi oldingi holatlarni saqlash orqali barcha mumkin bo‘lgan echimlarni o‘rganish uchun muammolarni hal qilishda ishlatiladigan backtracking algoritmlarini qo‘llab-quvvatlaydi.
- *Kompilyator dizaynida ishlatiladi:* stek ma’lumotlar tuzilishi kompilyator dizaynida dasturlash tillarini tahlil qilish va sintaksis tahlil qilish uchun ishlatiladi.
- *Bekor qilish/qayta bajarish operatsiyalari:* stek ma’lumotlar tuzilishi matn muharrirlari, Graf dizayn vositalari va dasturiy ta’minotni ishlab chiqish muhiti kabi turli xil dasturlarda bekor qilish va qayta ishlashni yoqish uchun ishlatiladi.

Stekning kamchiliklari:

- *Cheklangan imkoniyatlar**:*** Stek ma’lumotlar tuzilishi cheklangan imkoniyatlarga ega, chunki u faqat belgilangan miqdordagi elementlarni ushlab turishi mumkin. Agar stek to‘la bo‘lsa, yangi elementlarni qo‘shish stekning to‘lib ketishiga olib kelishi mumkin va bu ma’lumotlarning yo‘qolishiga olib keladi.
- *Tasodifiy kirish yo‘q:* stek ma’lumotlar tuzilishi uning elementlariga tasodifiy kirishga imkon bermaydi va u faqat stekning yuqori qismidan elementlarni qo‘shish va olib tashlashga imkon beradi. Stekning o‘rtasida joylashgan

elementga kirish uchun uning ustidagi barcha elementlarni olib tashlash kerak.

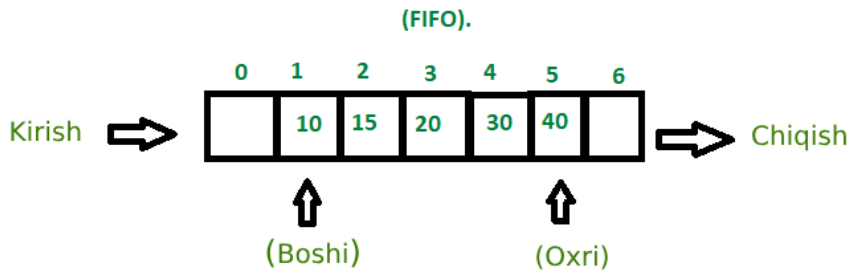
- *Xotirani boshqarish:* stek ma’lumotlar tuzilishi qo‘shni xotira blokidan foydalanadi, natijada elementlar tez-tez qo‘shilsa va olib tashlansa, xotira parchalanishi mumkin.
- *Muayyan ilovalar uchun mos emas:* stek ma’lumotlar tuzilishi algoritmlarni qidirish yoki saralash kabi stekning o‘rtasida joylashgan elementlarga kirishni talab qiladigan ilovalar uchun mos emas.
- *Stekni to‘ldirish va oqim:* stek ma’lumotlari tuzilishi stek ustiga juda ko‘p elementlar surilsa, stekning to‘lib ketishiga olib kelishi mumkin va agar stekdan juda ko‘p elementlar chiqsa, stekning pasayishiga olib kelishi mumkin.
- *Rekursiv funktsiya cheklovlarni chaqiradi:* stek ma’lumotlar tuzilishi rekursiv funktsiya chaqiruvlarini qo‘llab-quvvatlasa-da, juda ko‘p rekursiv funktsiya chaqiruvlari Stack toshib ketishiga olib kelishi mumkin, natijada dastur tugatiladi.

1.3 Navbat

Navbat dasturlashda foydali ma’lumotlar tuzilmasi hisoblanadi. Navbat birinchi kirgan birinchi ketadi (FIFO- First In First Out) qoidasiga amal qiladi - birinchi bo‘lib kiradigan element birinchi bo‘lib chiqadigan elementdir. Bu kinozal tashqarisidagi chiptalar navbatiga o‘xshaydi, bu erda navbatga birinchi kirgan kishi chiptani birinchi bo‘lib oladi(22-rasm).

Rasm- 4 Navbat ma’lumotlar tuzilmasi

Xizmat qilishga tayyor navbatdagi yozuvning holati, ya’ni navbatdan olib tashlanadigan birinchi yozuv navbatning old qismi(ba’zan navbatning boshi) deb ataladi, xuddi shunday, navbatdagi oxirgi yozuvning holati, ya’ni eng ko‘p yaqinda qo‘shilgan, navbatning orqa (yoki dumi) deb ataladi. Quyidagi rasmga qarang.



Rasm- 5 Navbatda birinchi kirgan ma’lumot birinchi chiqadi (FIFO- First In First Out) qoidasi

1.4 Navbatlarning turlari:

- *Kirish cheklangan navbat:* bu oddiy navbat. Ushbu turdagi navbatda kirishni faqat bitta uchidan olish mumkin, ammo o‘chirish har qanday uchidan amalga oshirilishi mumkin.
- *Chiqish cheklangan navbat:* bu ham oddiy navbat. Ushbu turdagi navbatda kirishni ikkala uchidan olish mumkin, ammo o‘chirish faqat bitta uchidan amalga oshirilishi mumkin.

- *Dairesel navbat:* bu oxirgi pozitsiya birinchi holatga qaytariladigan navbatning maxsus turi. Bu erda ham operatsiyalar FIFO tartibida amalga oshiriladi.
- *Ikki tomonlama navbat (Dequeue):* ikki tomonlama navbatda qo‘shish va o‘chirish operatsiyalari, ikkalasi ham ikkala uchidan ham bajarilishi mumkin.
- *Ustuvor navbat:* ustuvor navbat-bu elementlarga ularga berilgan ustuvorlik asosida kiradigan maxsus navbat. Ko‘proq bilish uchun bu murojaat.

C++ da navbat yaratish uchun avvalo navbat sarlavhasi faylini kiritishimiz kerak.

```
#include <queue>
```

Ushbu faylni import qilgandan so‘ng, biz quyidagi sintaksis yordamida navbat yaratishimiz mumkin:

```
queue<type> q;
```

Ma’lumotlar tuzilmasidagi navbat uchun ba’zi asosiy operatsiyalar:

Operatsiya	Tavsifi
enqueue ()	elementni navbatning orqa tomoniga kiritadi
dequeue ()	elementni navbatning old qismidan olib tashlaydi
Front()	navbatning birinchi elementini qaytaradi
Back()	navbatning oxirgi elementini qaytaradi
size()	navbatdagi elementlar sonini qaytaradi
empty()	agar navbat bo‘sh bo‘lsa, true qiymatini qaytaradi

1. Enqueue():

Enqueue () navbatning oxiriga element qo‘shadi.

Navbatga element kiritish uchun quyidagi amallarni bajarish kerak:

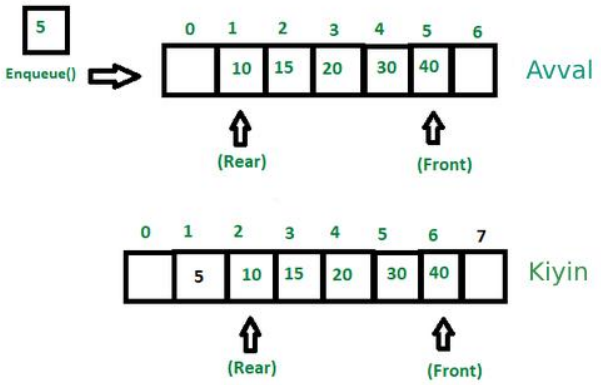
1-qadam: Navbat to‘lganligini tekshiring;

2-qadam: Agar navbat to‘la bo‘lsa xatolik qaytarish;

3-qadam: agar navbat to‘la bo‘lmasa, navbatning oxiridan joy ajratish;

4-qadam: Yaratilgan bo‘sh joyga elementni qo‘shish;

5-qadam: muvaffaqiyatni qaytarish;



Rasm- 6 Enqueue () operatsiyasi

Enqueue ning C++ da implementatsiyasi:

```
void queueEnqueue(int data)       //Enqueue funktsiyasi
{
     // Navbatning bo‘sh yoki to‘lganligini tekshirish
     if (capacity == rear) {
          printf("\nNavbat to‘liq: \n");
          return;
     }

     // Elementni eng oxiriga qo‘shish
     else {
          queue[rear] = data;
          rear++;
     }
     return;
}
```

2. Dequeue(): Navbatda birinshi elementini olib tashlash (yoki murojat qilish).Dequeue operatsiyasini bajarish uchun quyidagi qadamlar qo‘yiladi:

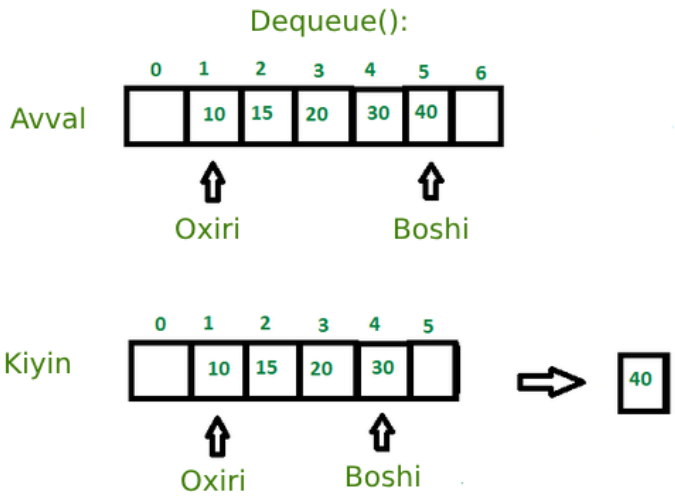
1-qadam: navbat bo‘sh yoki yo‘qligini tekshiring;

2-qadam: agar navbat bo‘sh bo‘lsa, oqim xatosini qaytaring;

3-qadam: agar navbat bo‘sh bo‘lmasa, old tomoni ko‘rsatgan ma’lumotlarga murojat qilining;

4-qadam: keyingi mavjud ma’lumotlar elementiga ishora qilish uchun old ko‘rsatgichni oshiring;

Qadam 5: muvaffaqiyatni qaytarish;



Rasm- 7 Dequeue() operatsiyasi

C++ da implementatsiyasi

```
void queueDequeue()
{
      // If queue is empty
      if (front == rear) {
            printf("\nQueue is empty\n");
            return;
      }

      // Shift all the elements from index 2
      // till rear to the left by one
```

```
        else {
            for (int i = 0; i < rear - 1; i++) {
                queue[i] = queue[i + 1];
            }

            // decrement rear
            rear--;
        }
        return;
    }
```

3. front (): bu operatsiya navbatning eng boshidagi elementini qaytaradi.

```
    // boshidagi elementni chiqarish
    int front(Queue* queue)
    {
        if (isempty(queue))
            return INT_MIN;
        return queue->arr[queue->front];
    }
```

4.rear():bu operatsiya navbarning eng oxiridagi elementi qaytaradi .

```
    int rear(queue<int>& myQueue)
    {
        queue<int> tempQueue = myQueue;

        while (tempQueue.size() > 1) {
            tempQueue.pop();
        }

        return tempQueue.front();
    }
```

5.isEmpty(): Navbat bo‘sh yoki bo‘sh emasligini tekshiradi va true yoki falsi javobiniq qaytaradi.

```
      // Bu funktsiya navbatning bo‘sh yoki yo‘qligini
    tekshiradi
      bool isEmpty()
      {
```

```
        if (front == -1)
             return true;
        else
             return false;
    }
```

C++ dasturlash tilida navbatning to‘liq implementatsiyasi.

```
    // Navbatning implementatsiyasi
    #include <iostream>
    #include <climits>
    using namespace std;

    class Queue {
    public:
       int front, rear, size;
       unsigned cap;
       int* arr;
    };

    Queue* createQueue(unsigned cap)
    {
       Queue* queue = new Queue();
       queue->cap = cap;
       queue->front = queue->size = 0;

       queue->rear = cap - 1;
       queue->arr = new int[(queue->cap * sizeof(int))];
       return queue;
    }

    int isFull(Queue* queue)
    {
       return (queue->size == queue->cap);
    }

    int isempty(Queue* queue) { return (queue->size ==
0); }
    // navbatga element qo‘shish funktsiyasi
    //Bu navbatning oxirini va hajmini o‘zgartiradi
    void enqueue(Queue* queue, int item)
```

```
{
   if (isFull(queue))
        return;
   queue->rear = (queue->rear + 1) % queue->cap;
   queue->arr[queue->rear] = item;
   queue->size = queue->size + 1;
   cout << item << " navbatga element qo‘shildi: \n";
}
// navbatdan element o‘shirish funktsiasi
// Bu navbatning hajmi ni o‘zgartiradi
int dequeue(Queue* queue)
{
   if (isempty(queue))
        return INT_MIN;
   int item = queue->arr[queue->front];
   queue->front = (queue->front + 1) % queue->cap;
   queue->size = queue->size - 1;
   return item;
}
int front(Queue* queue)
{
   if (isempty(queue))
        return INT_MIN;
   return queue->arr[queue->front];
}
int rear(Queue* queue)
{
   if (isempty(queue))
        return INT_MIN;
   return queue->arr[queue->rear];
}

// Bosh funktsiya
int main()
{
   Queue* queue = createQueue(1000);
   enqueue(queue, 10);
   enqueue(queue, 20);
   enqueue(queue, 30);
   enqueue(queue, 40);
   cout << dequeue(queue);
```

```
        cout << " navbatdan element ochirildi\n";
        cout << "Boshidagi element " << front(queue) <<
    endl;
        cout << "Oxirgi element " << rear(queue);
        return 0;}
```

Dastur na’tijasi:

```
/tmp/4JS4qyX5qm.o
10 navbatga element qo'shildi:
20 navbatga element qo'shildi:
30 navbatga element qo'shildi:
40 navbatga element qo'shildi:
10 navbatdan element ochirildi
Boshidagi element 20
Oxirgi element 40
```

Navbatning qo‘llanishi:

- *Multi programming:* ko‘p dasturlash asosiy xotirada bir nechta dastur ishlayotganligini anglatadi. Ushbu bir nechta dasturlarni tashkil qilish juda muhim va bu bir nechta dasturlar navbat sifatida tashkil etilgan.
- *Tarmoq:* tarmoqda navbat yo‘riqnoma yoki kalit kabi qurilmalarda ishlatiladi. navbatning yana bir ilovasi-bu pochta xabarlari uchun ma’lumotlarni saqlaydigan va fayllarni boshqaradigan katalog bo‘lgan pochta navbati.
- *Ishni rejalashtirish:* kompyuterda birin-ketin bajarilishi rejalashtirilgan ma’lum miqdordagi ishlarni bajarish vazifasi bor. Ushbu ishlar navbat yordamida tashkil etilgan protsessorga birma-bir tayinlanadi.
- *Umumiy manbalar:* navbatlar bitta umumiy resurs uchun kutish ro‘yxati sifatida ishlatiladi.

Navbatning afzalliklari:

- Katta hajmdagi ma’lumotlarni osonlik bilan samarali boshqarish mumkin.
- Qo‘shish va o‘chirish kabi operatsiyalar osonlik bilan bajarilishi mumkin, chunki u birinchi navbatda birinchi qoidaga amal qiladi.
- Navbatlar ma’lum bir xizmat bir nechta iste’molchilar tomonidan ishlatilganda foydalidir.

- Ma’lumotlar jarayonlararo aloqa uchun navbatlar tez.
- Navbatlardan boshqa ma’lumotlar tuzilmalarini amalga oshirishda foydalanish mumkin.

Navbatning kamchiliklari:

- Elementlarni o‘rtadan kiritish va o‘chirish kabi operatsiyalar ko‘p vaqt talab etadi.
- Cheklangan xotira.
- Klassik navbatda yangi element faqat mavjud elementlar navbatdan o‘chirilganda kiritilishi mumkin.
- Elementni qidirish O(N) vaqtini oladi.
- Navbatning maksimal hajmi oldindan belgilanishi kerak.