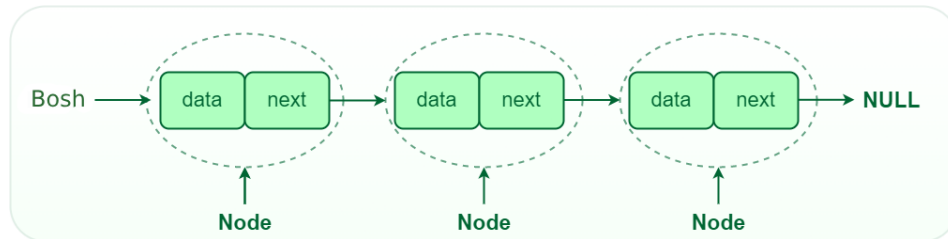


## 1. Bog'langan ro'yxat (Linked list)

Bog'langan ro'yxat (Linked list) - bu chiziqli ma'lumotlar strukturasi bo'lib, unda elementlar qo'shni joyda saqlanmaydi, aksincha ular ko'rsatkichlar yordamida bog'lanadi. Bog'langan ro'yxat bir qator bog'langan tugun (node)larni hosil qiladi va har bir tugun ma'lumotlarni va keyingi tugunning manzilini saqlaydi.



*Rasm- 1 Bog'langan ro'yxat ma'lumotlar tuzilamsi*

Linked list: bog'langan ro'yxatdagi tugun odatda ikkita komponentdan iborat:

- Data: U tugun bilan bog'liq haqiqiy qiymat yoki ma'lumotlarni saqlaydi.
- Next ko'rsatkich: u keyingi tugunning xotira manzilini (ma'lumotnomasini) ketma-ketlikda saqlaydi.

Bosh va Null: bog'langan ro'yxatga ro'yxatdagi birinchi tugunga ishora qiluvchi bosh tugun orqali kirish mumkin. Ro'yxatdagi oxirgi tugun ro'yxatning oxirini ko'rsatib, NULL yoki nullptr ga ishora qiladi. Ushbu tugun oxirgi tuguni sifatida tanilgan.

***Quyida bog'langan ro'yxatning bir nechta afzalliklari keltirilgan:***

- Dinamik ma'lumotlar tuzilishi: xotira hajmi kiritish yoki chiqarish operatsiyalarini bajarish paytida bevosita ish vaqtida ajratish mumkin.
- Qo'shish/o'chirish qulayligi: elementlarni kiritish va o'chirish massivlarga qaraganda sodda, chunki qo'shish va o'chirishdan keyin hech qanday elementni almashtirish kerak emas, faqat manzillarni yangilanish zarur.

- Xotiradan samarali foydalanish: biz bilganimizdek bog'langan ro'yxat dinamik ma'lumotlar tuzilmasi bo'lib, hajmi talabga muvofiq ortadi yoki kamayadi, shuning uchun bu xotira isrof bo'lishining oldini oladi.
- Amalga oshirish: turli xil rivojlangan ma'lumotlar tuzilmalari stek, navbat, Graf, Xash xaritalar va boshqalar kabi bog'langan ro'yxat yordamida amalga oshirilishi mumkin.

Keling, bog'langan ro'yxatning har bir tugun (node)ining qanday ifodalanishini ko'rib chiqaylik. Har bir tugun quyidagilardan iborat:

```
struct node{  
    int data;  
    struct node *next;  
};
```

Biz ma'lumotlar elementini ham, keyingi tugun (node) ma'lumotnomasini ham shunday tuzilishga o'tkazamiz:

Har bir tuzilma tugunda ma'lumotlar elementi va boshqa struktura tuguniga ko'rsatkich mavjud. Keling, bu qanday ishlashini tushunish uchun uchta elementdan iborat oddiy bog'langan ro'yxat yarataylik:

Bog'langan ro'yxat ( Linked list) implementatsiyasi.

```
#include <iostream>  
using namespace std;  
  
// node kiritish  
class Node {  
    public:  
    int value;  
    Node* next;  
};  
  
int main() {  
    Node* head;  
    Node* one = NULL;  
    Node* two = NULL;
```

```

Node* three = NULL;

// 3 ta node ni ajratish
one = new Node();
two = new Node();
three = new Node();

// qiymatlar berish
one->value = 1;
two->value = 2;
three->value = 3;

// node larni bog'lash
one->next = two;
two->next = three;
three->next = NULL;

// linked list qiymatini chiqarish
head = one;
while (head != NULL) {
    cout << head->value;
    head = head->next;
}
}

```

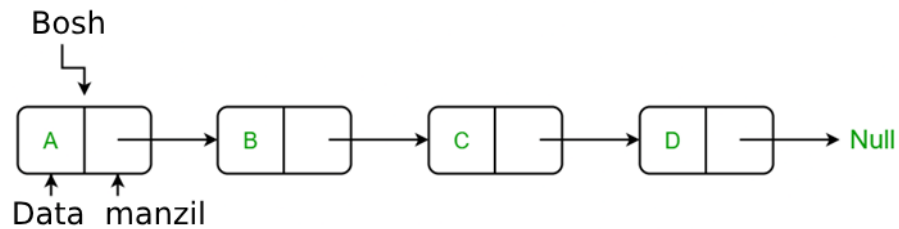
### 1.1 Bog'langan ro'yxatning (Linked list) turlari.

Bog'langan ro'yxatlarning (Linked list) asosan uch turi mavjud:

- Yagona bog'langan ro'yxat (Single-linked list)
- Ikkilangan bog'langan ro'yxat (Double linked list )
- Doiraviy bog'langan ro'yxat (Circular linked list)

#### ***1. Bir taraflama(Yagona) bog'langan ro'yxat (Single-linked list):***

Bir taraflama bog'langan ro'yxat - bu chiziqli ma'lumotlar strukturasi bo'lib, unda elementlar qo'shni xotira joylarida saqlanmaydi va har bir element faqat ko'rsatgich yordamida keyingi elementiga ulanadi.



*Rasm- 2 Yagona bog'langan ro'yxatning bir-biriga ulanishi*

Tugun (node) quyidagicha ifodalanadi:

```
struct node {
    int data;
    struct node *next;
}
```

Uch a'zodan iborat yakka bog'langan ro'yxat quyidagicha tuzilishi mumkin:

```
/* nodeni inisilizatsiyalash */
struct node *head;
struct node *one = NULL;
struct node *two = NULL;
struct node *three = NULL;

/* Xotirada joy ajratish */
one = malloc(sizeof(struct node));
two = malloc(sizeof(struct node));
three = malloc(sizeof(struct node));

/* Qiymatlar berih */
one->data = 1;
two->data = 2;
three->data = 3;

/* nodelarni bog'lash */
one->next = two;
two->next = three;
three->next = NULL;

/* 1-chi node ning adresini saqlash */
head = one;
```

**2. Ikkilik bog'langan (Double linked list)** ro'yxatda har bir tugun keyingi va oldingi tugunlar havolalarni o'z ichiga oladi. Bu oldinga va orqaga yo'nalishda harakatlanish imkonini beradi, lekin orqaga havola uchun qo'shimcha xotira talab qiladi.



*Rasm- 3 Ikkilik bog'langan ro'yxat*

Tugun (node) quyidagicha ifodalanadi:

```
struct node {
    int data;
    struct node *next;
    struct node *prev;
}
```

Uch a'zodan iborat yakka bog'langan ro'yxat quyidagicha tuzilishi mumkin:

```
/* Initialize nodes */
struct node *head;
struct node *one = NULL;
struct node *two = NULL;
struct node *three = NULL;

/* Allocate memory */
one = malloc(sizeof(struct node));
two = malloc(sizeof(struct node));
three = malloc(sizeof(struct node));

/* Assign data values */
one->data = 1;
two->data = 2;
three->data = 3;

/* Connect nodes */
```

```

one->next = two;
one->prev = NULL;

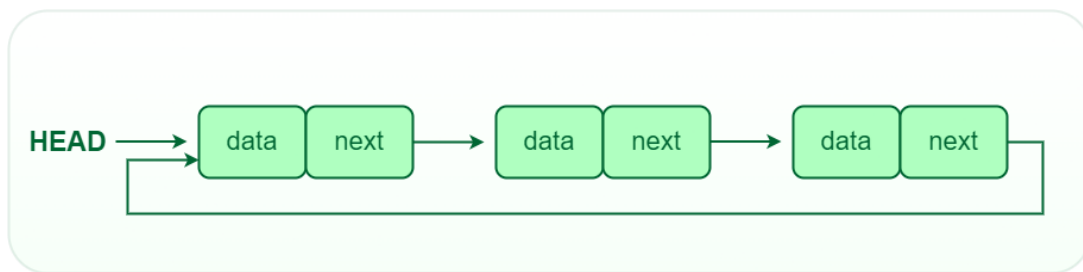
two->next = three;
two->prev = one;

three->next = NULL;
three->prev = two;

/* Save address of first node in head */
head = one;

```

**3. Doiraviy bog'langan (Circular linked list) ro'yxatda** oxirgi tugun bosh tugunga qaytib, doiraviy tuzilmani yaratadi. U yakka yoki ikkilamchi bog'langan bo'lishi mumkin.



*Rasm- 4 Doiraviy bog'langan ro'yxat*

Uch a'zodan iborat doiraviy yakka bog'langan ro'yxat quyidagicha tuzilishi mumkin:

```

struct node *head;
struct node *one = NULL;
struct node *two = NULL;
struct node *three = NULL;

/* Allocate memory */
one = malloc(sizeof(struct node));
two = malloc(sizeof(struct node));
three = malloc(sizeof(struct node));

/* Assign data values */
one->data = 1;

```

```
two->data = 2;
three->data = 3;

/* Connect nodes */
one->next = two;
two->next = three;
three->next = one;

/* Save address of first node in head */
head = one;
```

## 1.2 Bog'langan ro'yxatlar usitda majariladigan amallar

1. O'tish (Traversal)- bog'langan ro'yxatning har bir elementiga murojat.
2. Qo'shish (Insertion): Bog'langan ro'yxatga yangi tugun qo'shish to'g'ri ketma-ketlikni saqlash uchun mavjud tugularning ko'rsatkichlarini sozlashni o'z ichiga oladi. Qo'shish ro'yxatning boshida, oxirida yoki istalgan pozitsiyada amalga oshirilishi mumkin
3. O'chirish (Deletion): Tugunni bog'langan ro'yxatdan olib tashlash o'chirilgan tugun qoldirgan bo'shliqni to'ldirish uchun qo'shni tugunlarning ko'rsatkichlarini sozlashni talab qiladi. O'chirish ro'yxatning boshida, oxirida yoki istalgan pozitsiyada amalga oshirilishi mumkin.
4. Qidirish (Searching): Bog'langan ro'yxatda ma'lum bir qiymatni izlash ro'yxatni bosh tugundan qiymat topilmaguncha yoki ro'yxat oxirigacha aylanib o'tishni o'z ichiga oladi.
5. Saralash (sort) - bog'langan ro'yxatning tugunlarini tartiblash

### ***O'tish (Traversal)***

Bog'langan ro'yxatning mazmunini ko'rsatish juda oddiy. Biz vaqtinchalik tugunni keyingisiga o'tkazamiz va uning mazmunini ko'rsatamiz.

Temp NULL bo'lsa, biz bog'langan ro'yxatning oxiriga yetganimizni bilamiz while siklidan chiqamiz.

```
struct node *temp = head;
```

```
printf("\n\nList elements are - \n");
while(temp != NULL) {
    printf("%d --->", temp->data);
    temp = temp->next;
}
```

### ***Qo'shish (Insertion)***

Bog'larga ro'yxatning boshiga, o'rtasiga yoki oxiriga elementlar qo'shishingiz mumkin.

#### 1) Boshiga kiritish:

- Yangi tugun (node) uchun xotira ajrating;
- Ma'lumotlarni saqlang;
- Yangi tugunning keyingi qismini boshga ishora qilib o'zgartiring;
- Oxirida yaratilgan tugunga boshni o'zgartiring;

```
struct node *newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

newNode->next = head;

head = newNode;
```

#### 2) Oxirida kiritish

- Yangi tugun (node) uchun xotira ajrating;
- Ma'lumotlarni saqlang;
- Oxirgi tugunga o'tish;
- Oxirgi tugunning keyingisini yaqinda yaratilgan tugunga o'zgartiring;

```
- struct node *newNode;
- newNode = malloc(sizeof(struct node));
- newNode->data = 4;
- newNode->next = NULL;
-
```



```

- struct node *temp = head;
- while(temp->next != NULL) {
- temp = temp->next;
- }
-
- temp->next = newNode;
-

```

### 3) O'rtasiga kiritish

- Xotirani ajrating va yangi tugun uchun ma'lumotlarni saqlang
- Yangi tugunning kerakli pozitsiyasidan oldin tugunga o'ting;
- O'rtasiga yangi tugun qo'shish uchun keyingi ko'rsatkichlarni o'zgartiring;

```

struct node *newNode;
newNode = malloc(sizeof(struct node));
newNode->data = 4;

struct node *temp = head;

for(int i=2; i < position; i++) {
    if(temp->next != NULL) {
        temp = temp->next;
    }
}
newNode->next = temp->next;
temp->next = newNode;

```

### ***O'chirish (Deletion)***

Linked list boshidan, oxiridan yoki ma'lum bir pozitsiyadan o'chirishingiz mumkin.

#### 1) Boshidan o'chirish:

- Boshni ikkinchi tugun ga yo'naltiring;

```
head = head->next;
```

#### 2) Oxiridan o'chirish

- Ikkinchi oxirgi elementga o'ting;

- Keyingi ko'rsatkichni null ga o'zgartiring;

```
struct node* temp = head;
while (temp->next->next!=NULL) {
    temp = temp->next;
}
temp->next = NULL;
```

### 3) O'rtadan o'chirish:

- O'chiriladigan elementdan oldin elementga o'ting;
- Tugunni zanjirdan chiqarib tashlash uchun keyingi ko'rsatkichlarni o'zgartiring;

```
for(int i=2; i< position; i++) {
    if(temp->next!=NULL) {
        temp = temp->next;
    }
}

temp->next = temp->next->next;
```

### ***Linked list elementlarini saralash (sort)***

Bog'langan ro'yxat elementlarini o'sish tartibida saralash uchun oddiy tartiblash algoritmidan Bubble Sort dan foydalanamiz.

- Boshni joriy tugun sifatida elon qiling va keyinroq foydalanish uchun boshqa tugun indeksini yarating;
- Agar bosh null bo'lsa, qayting;
- Aks holda, oxirgi tugungacha (ya'ni NULL) siklni bajaring;
- Keyingi oqim tugunini indeksda saqlang;
- Joriy tugun ma'lumotlari keyingi tugundan kattaroq yoki yo'qligini tekshiring. Agar u kattaroq bo'lsa, joriy va indeksni almashtiring.;

```
// Sort the linked list
void sortLinkedList(struct Node** head_ref) {
    struct Node *current = *head_ref, *index = NULL;
    int temp;
```

```

    if (head_ref == NULL) {
        return;
    } else {
        while (current != NULL) {
            // index points to the node next to current
            index = current->next;

            while (index != NULL) {
                if (current->data > index->data) {
                    temp = current->data;
                    current->data = index->data;
                    index->data = temp;
                }
                index = index->next;
            }
            current = current->next;
        }
    }
}

```

Bog'langan ro'yxatta (Linked list) chiqarish,o'chirish,saralash,qidirish operatsiyalarni bajarish.

```

#include <iostream>
using namespace std;
// node (tugun (node)) yaratish
struct Node {
    int data;
    struct Node* next;
};

void insertAtBeginning(struct Node** head_ref, int new_data) {
    // node ka xotiradan joy ajratish
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

    // malumotni kiritish
    new_node->data = new_data;
    new_node->next = (*head_ref);

    // boshni yangi node ga qoshish
}

```

```

    (*head_ref) = new_node;
}

// node tan kiyin yandi node yaratish
void insertAfter(struct Node* prev_node, int new_data) {
    if (prev_node == NULL) {
        cout << "the given previous node cannot be NULL";
        return;
    }

    struct Node* new_node = (struct
Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
}

// Insert at the end
void insertAtEnd(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct
Node*)malloc(sizeof(struct Node));
    struct Node* last = *head_ref; /* used in step 5*/

    new_node->data = new_data;
    new_node->next = NULL;

    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }

    while (last->next != NULL) last = last->next;

    last->next = new_node;
    return;
}

// node ni o'chirish
void deleteNode(struct Node** head_ref, int key) {
    struct Node *temp = *head_ref, *prev;

    if (temp != NULL && temp->data == key) {
        *head_ref = temp->next;
        free(temp);
        return;
    }

    // ochirmoqchi bo'lgan kalitni topish

```

```

while (temp != NULL && temp->data != key) {
    prev = temp;
    temp = temp->next;
}

    if (temp == NULL) return;

// Node ni o'chirish
prev->next = temp->next;

free(temp);
}

// Qidiruv
bool searchNode(struct Node** head_ref, int key) {
    struct Node* current = *head_ref;

    while (current != NULL) {
        if (current->data == key) return true;
        current = current->next;
    }
    return false;
}

// linked listni saralash
void sortLinkedList(struct Node** head_ref) {
    struct Node *current = *head_ref, *index = NULL;
    int temp;

    if (head_ref == NULL) {
        return;
    } else {
        while (current != NULL) {
            // index
            index = current->next;

            while (index != NULL) {
                if (current->data > index->data) {
                    temp = current->data;
                    current->data = index->data;
                    index->data = temp;
                }
                index = index->next;
            }
            current = current->next;
        }
    }
}

```

```

}

// Barcha linked listni chop etish
void printList(struct Node* node) {
    while (node != NULL) {
        cout << node->data << " ";
        node = node->next;
    }
}

// Dasturni yugirtirish
int main() {
    struct Node* head = NULL;

    insertAtEnd(&head, 1);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 3);
    insertAtEnd(&head, 4);
    insertAfter(head->next, 5);

    cout << "Linked list: ";
    printList(head);

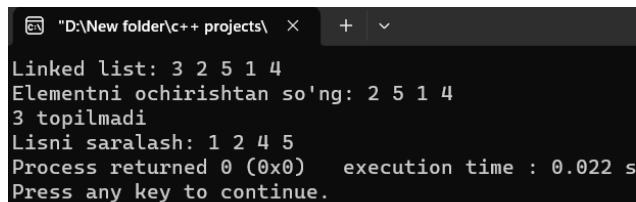
    cout << "\nElementni o'chirishtan so'ng: ";
    deleteNode(&head, 3);
    printList(head);

    int item_to_find = 3;
    if (searchNode(&head, item_to_find)) {
        cout << endl << item_to_find << " topildi";
    } else {
        cout << endl << item_to_find << " topilmadi";
    }

    sortLinkedList(&head);
    cout << "\nLisni saralash: ";
    printList(head);
}

Dastur natijasi:

```



The screenshot shows a terminal window with the following output:

```

D:\New folder\c++ projects\ >
Linked list: 3 2 5 1 4
Elementni ochirishtan so'ng: 2 5 1 4
3 topilmadi
Lisni saralash: 1 2 4 5
Process returned 0 (0x0)   execution time : 0.022 s
Press any key to continue.

```

### 1.3 Bog'langan ro'yxatlarning avfzalliklari va kamchiliklari.

#### Avfzalliklari

- Dinamik o'lcham: Bog'langan ro'yxatlar dinamik ravishda o'sishi yoki qisqarishi mumkin, chunki xotira taqsimoti ish vaqtida amalga oshiriladi.
- Qo'shish va o'chirish: Bog'langan ro'yxatga elementlarni qo'shish yoki o'chirish samarali, ayniqsa katta ro'yxatlar uchun.
- Moslashuvchanlik: Bog'langan ro'yxatlar qo'shni xotira blokini talab qilmasdan osongina qayta tashkil etilishi va o'zgartirilishi mumkin.

#### Kamchiliklari

- Tasodifiy kirish: massivlardan farqli o'laroq, bog'langan ro'yxatlar indeks bo'yicha elementlarga to'g'ridan-to'g'ri kirishga ruxsat bermaydi. Muayyan tugun (node)ga erishish uchun o'tish kerak.
- Qo'shimcha xotira: Bog'langan ro'yxatlar massivlarga nisbatan ko'rsatkichlarni saqlash uchun qo'shimcha xotira talab qiladi.

Massiv (Array)	Bog'langan ro'yxat (Linked list)
Massiv – bir xil ma'lumotlar turidagi elementlar to'plami.	Bog'langan ro'yxat-bu har bir element ko'rsatkichlar yordamida keyingisiga ulangan bir xil turdagi elementlarning buyurtma qilingan to'plami.
Massiv elementlariga massiv indeksi yordamida tasodifiy kirish mumkin.	Bog'langan ro'yxatlarda tasodifiy kirish mumkin emas. Elementlarga ketma-ket kirish kerak bo'ladi.
Ma'lumotlar elementlari xotirada qo'shni joylarda saqlanadi.	Yangi elementlar har qanday joyda saqlanishi mumkin va

	ko'rsatkichlar yordamida yangi element uchun ma'lumotnoma yaratiladi.
Qo'shish va o'chirish operatsiyalari qimmatroq, chunki xotira joylari ketma-ket joylashgan.	Qo'shish va o'chirish operatsiyalari bog'langan ro'yxatda tez va oson.
Xotira kompilyatsiya vaqtida ajratiladi (statik xotira ajratish).	Xotira run-time ajratilgan (dinamik xotira ajratish).
Qator hajmi array deklaratsiyasi vaqtida belgilangan bo'lishi kerak	Bog'langan ro'yxay hajmi oshishi mumkin agar yangi element qo'shilsa

### **Mavzu yuzasidan savollar:**

1. Qisqacha bog'langan ro'yxat ma'lumotlar tuzilmasini tushuntiring.
2. Bog'langan ro'yxat ma'lumotlar tuzilmasini grafik ko'rinishda ifodalang.
3. Bog'langan ro'yxat ma'lumotlar tuzilmasining qancha turi mavjud?
4. Oddiy bog'langan ro'yxatni amalga oshirish uchun qancha ko'rsatgich kerak?
5. Massivlarni bog'langan ro'yxatlar bilan Solishtiring.