Nesne Yönelimli Programlamaya Giriş

C++ OOP: Bagli Listeler, Yiginlar ve Kuyruklar

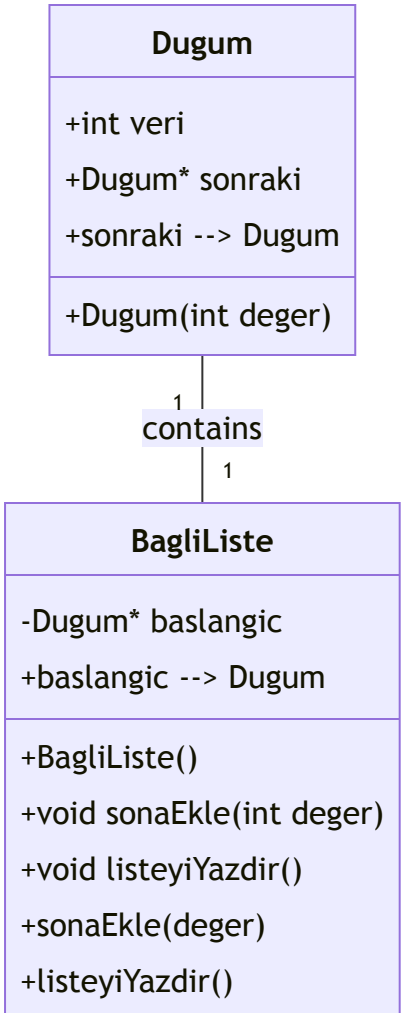
1. Bagli Listeler (Linked Lists)

1.1 Bagli Liste Nedir?

Bagli liste, elemanlarinin (dugumlerinin) her birinin bir sonraki elemanin adresini tuttugu veri yapisidir. Bagli listelerde, veri elemanlari ardisik olarak bellek icinde siralanmaz, her bir eleman bir "dugum" olarak temsil edilir ve bir sonraki dugume isaret eder.

1.2 Bagli Liste Turleri

- **Tek Yonlu Bagli Liste (Singly Linked List)**: Her dugum bir sonraki dugumun adresini tutar.
- **Cift Yonlu Bagli Liste (Doubly Linked List)**: Her dugum hem bir sonraki dugumun hem de bir onceki dugumun adresini tutar.
- **Cevrimli Bagli Liste (Circular Linked List)**: Son dugum, ilk dugume isaret eder.


Dugum
+int veri
+Dugum* sonraki
+sonraki --> Dugum
+Dugum(int deger)
1
contains
1
BagliListe
-Dugum* baslangic
+baslangic --> Dugum
+BagliListe()
+void sonaEkle(int deger)
+void listeyiYazdir()
+sonaEkle(deger)
+listeyiYazdir()

```cpp
#include <iostream>

// Düğüm yapısı (struct)
struct Dugum {
    int veri;       // Düğümde saklanacak veri
    Dugum* sonraki; // Bir sonraki düğümün adresi (gösterici)
};

int main() {
    // Başlangıçta liste boştur
    Dugum* baslangic = nullptr;

    // İlk düğümü oluşturalım (verisi 10 olsun)
    Dugum* ilkDugum = new Dugum(); // Bellekte yeni bir düğüm için yer ayır
    ilkDugum->veri = 10;           // Düğümün verisini 10 olarak ayarla
    ilkDugum->sonraki = nullptr;   // İlk düğümün sonraki düğümü yok (liste tek elemanlı)
    baslangic = ilkDugum;         // Başlangıç göstericisini ilk düğüme yönlendir

    // İkinci düğümü oluşturalım (verisi 20 olsun)
    Dugum* ikinciDugum = new Dugum();
    ikinciDugum->veri = 20;
    ikinciDugum->sonraki = nullptr;

    // İlk düğümün sonrasını ikinci düğüme bağlayalım
    ilkDugum->sonraki = ikinciDugum;
```

```
    // Üçüncü düğümü oluşturalım(verisi 30 olsun)
    Dugum* ucuncuDugum = new Dugum();
    ucuncuDugum->veri = 30;
    ucuncuDugum->sonraki = nullptr;

    //İkinci düğümün sonrasını üçüncü düğüme bağlayalım.
    ikinciDugum->sonraki = ucuncuDugum;

    // Listeyi yazdıralım
    Dugum* gecici = baslangic; // Geçici bir gösterici kullanarak listeyi dolaşacağız
    while (gecici != nullptr) { // Geçici gösterici null olana kadar döngü devam eder
        std::cout << gecici->veri << " "; // Düğümün verisini yazdır
        gecici = gecici->sonraki;       // Geçici göstericiyi bir sonraki düğüme ilerlet
    }
    std::cout << std::endl; // Yeni satıra geç

    // Belleği temizleyelim (önemli!)
    gecici = baslangic;
    while (gecici != nullptr) {
        Dugum* silinecek = gecici;
        gecici = gecici->sonraki;
        delete silinecek;
    }

    return 0;
}
```

1.3 C++ ile Bagli Liste Uygulaması

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

class Dugum {
public:
    int veri;
    Dugum* sonraki;
    Dugum(int deger) : veri(deger), sonraki(nullptr) {}
};
```

```
class BagliListe {
private:
    Dugum* baslangic;
public:
    BagliListe() : baslangic(nullptr) {}

    void sonaEkle(int deger) {
        Dugum* yeniDugum = new Dugum(deger);
        if (!baslangic) {
            baslangic = yeniDugum;
            return;
        }
        Dugum* gecici = baslangic;
        while (gecici->sonraki) {
            gecici = gecici->sonraki;
        }
        gecici->sonraki = yeniDugum;
    }
```

```
    void listeyiYazdir() {
        Dugum* gecici = baslangic;
        while (gecici) {
            cout << gecici->veri << " ";
            gecici = gecici->sonraki;
        }
        cout << endl;
    }
};

int main() {
    BagliListe liste;
    liste.sonaEkle(10);
    liste.sonaEkle(20);
    liste.sonaEkle(30);
    liste.listeyiYazdir();
    return 0;
}
```

2. Yiginlar (Stacks)

2.1 Yigin Nedir?

Yigin (stack), son giren ilk cikar (LIFO - Last In First Out) prensibine gore calisan bir veri yapisidir. Yiginlar genellikle islev cagriyalari, islem siralamalari gibi durumlarda kullanilir.

2.2 Yigin Operasyonları

- **push()**: Yigina eleman ekler.
- **pop()**: Yigindan eleman cikarir.
- **top()**: Yiginin tepe elemanini dondurur.
- **isEmpty()**: Yiginin bos olup olmadigini kontrol eder.

2.3 C++ ile Yigin Uygulaması

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

int main() {
    stack<int> yigin;

    yigin.push(10);
    yigin.push(20);
    yigin.push(30);

    cout << "Top element: " << yigin.top() << endl;

    yigin.pop();
    cout << "Top after pop: " << yigin.top() << endl;

    cout << "Is stack empty? " << (yigin.empty() ? "Yes" : "No") << endl;

    return 0;
}
```

3. Kuyruklar (Queues)

3.1 Kuyruk Nedir?

Kuyruk (queue), ilk giren ilk cikar (FIFO - First In First Out) prensibine gore calisan bir veri yapisidir. Kuyruklar genellikle isleme siralamasi gerektiren durumlarda kullanilir.

3.2 Kuyruk metodları

- `push(eleman)` : Kuyruğun sonuna bir eleman ekler.
- `pop()` : Kuyruğun başındaki elemanı çıkarır.
- `front()` : Kuyruğun başındaki elemana erişir (çıkarmaz).
- `back()` : Kuyruğun sonundaki elemana erişir.
- `empty()` : Kuyruğun boş olup olmadığını kontrol eder.
- `size()` : Kuyruktaki eleman sayısını döndürür.

3.3 Kuyruk Örnek:

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> kuyruk;

    kuyruk.push(10);
    kuyruk.push(20);
    kuyruk.push(30);

    cout << "Front element: " << kuyruk.front() << endl;

    kuyruk.pop();
    cout << "Front after pop: " << kuyruk.front() << endl;

    cout << "Is queue empty? " << (kuyruk.empty() ? "Yes" : "No") << endl;

    return 0;
}
```

Senaryo:

Bir robot kolunun hareketlerini kontrol eden bir sistemin parçası olarak, robotun kollarını hareket ettirmek için çeşitli sensör verileri ve motor komutları işlenmektedir. Robot kolunun hareketleri, verilen komutlara göre belirli bir sıra ile gerçekleştirilmelidir. Bu tür bir hareket sıralaması, bağlı listeler, yığınlar ve kuyruklar gibi veri yapıları kullanılarak yönetilebilir.

Örnek Uygulama: Robot Kolunun Hareket Sırasını Yönetme

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

class Komut {
public:
    string hareket;
    Komut* sonraki;
    Komut(string hareket) : hareket(hareket), sonraki(nullptr) {}
};
```

```
class HareketListesi {
private:
    Komut* baslangic;
public:
    HareketListesi() : baslangic(nullptr) {}

    void komutEkle(string hareket) {
        Komut* yeniKomut = new Komut(hareket);
        if (!baslangic) {
            baslangic = yeniKomut;
            return;
        }
        Komut* gecici = baslangic;
        while (gecici->sonraki) {
            gecici = gecici->sonraki;
        }
        gecici->sonraki = yeniKomut;
    }
```

```
    void komutlariYazdir() {
        Komut* gecici = baslangic;
        while (gecici) {
            cout << gecici->hareket << " ";
            gecici = gecici->sonraki;
        }
        cout << endl;
    }
};

int main() {
    HareketListesi hareketler;
    hareketler.komutEkle("Kolu yukari kaldır");
    hareketler.komutEkle("Kolu sağa hareket ettir");
    hareketler.komutEkle("Kolu aşağı indir");
    hareketler.komutlariYazdir();
    return 0;
}
```