

# C++'ta Pointer ile Dizilerde Sıralama ve Aşırı Yüklü Fonksiyonlar

Prof.Dr.Kasım Kurt

1 Pointer ile Dizilerde Sıralama

2 Aşırı Yüklü Fonksiyonlar

# Amaç

Bu bölümde:

- C++'ta pointer kavramını kısaca hatırlayacağız.
- Diziler ile pointer ilişkisinin nasıl kurulduğunu göreceğiz.
- Bir diziyi **pointer kullanarak** küçükten büyüğe sıralayan bir fonksiyon yazacağız.
- Kullanılan algoritma: **Selection Sort** (seçmeli sıralama).

# Pointer ve Dizi İlişkisi

- C++'ta bir dizi ismi, çoğu durumda **ilk elemanın adresi** gibi davranışır.
- Örnek:
  - `int dizi[5];`
  - `int *p = dizi; (p, &dizi[0] adresini tutar)`
- Pointer aritmetiği ile dizi elemanlarına erişim:
  - `*(p + i)` ifadesi, `dizi[i]` ile aynıdır.
- Fonksiyonlara dizi göndermek için genellikle:
  - `int *p` veya `int dizi[]` şeklinde parametre yazılır.

# Selection Sort (Seçmeli Sıralama)

- Amaç: Diziyi **küçükten büyüğe** sıralamak.
- Temel adımlar:
  - ① İlk elemanı (indis 0) geçici olarak en küçük kabul et.
  - ② Dizinin geri kalanında (1, 2, ..., n-1) daha küçük bir eleman var mı bak.
  - ③ Daha küçük bir eleman bulunursa, minimumu güncelle.
  - ④ Taranma bittiğinde, bulunan en küçük eleman ile baştaki elemanın yerini değiştir.
  - ⑤ Sonra 2. eleman için aynı işlemi tekrarla, bu şekilde sona kadar git.
- Karmaşıklık:  $O(n^2)$
- Biz bu algoritmayı **pointer aritmetiği** ile yazacağız.

# Sıralama Fonksiyonu (Pointer ile)

```
void sirala(int *p, int n)
{
    for (int i = 0; i < n - 1; ++i)
    {
        int *pMin = p + i;      // i. eleman en küçük kabul
        int *p2   = p + i + 1; // aramaya buradan başla
        for (; p2 < p + n; ++p2) // Geri kalan elemanlar içinde en küçük
        {
            if (*p2 < *pMin)
            {
                pMin = p2;      // yeni minimum adresi
            }
        } // i. eleman ile bulunan minimumu yer değiştir
        int temp = *(p + i);
        *(p + i) = *pMin;
        *pMin = temp;
    }
}
```

# Ana Program (Pointer ile Dizi Sıralama)

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

void sirala(int *p, int n); // prototip

int main()
{
    int dizi[100];
    int n;

    cout << "Kac eleman gireceksiniz? ";
    cin >> n;

    cout << n << " adet tamsayi giriniz:\n";
    for (int i = 0; i < n; ++i)
    {
        cin >> dizi[i];
    }
}
```

# Örnek Çalışma

- Giriş:
  - $n = 5$
  - Dizi elemanları: 8 3 10 1 6
- `sirala(dizi, 5)` çağrısından sonra:

Çıktı

Sıralanmış dizi: 1 3 6 8 10

- Bu örnekte sıralama işlemi boyunca dizinin elemanlarına **doğrudan pointer aritmetiği** ile erişilmiştir (`*(p+i)`, `pMin`, `p2` vb.).

# Aşırı Yüklü Fonksiyon Nedir?

- C++'ta **aynı isimli** birden fazla fonksiyon tanımlanabilir.
- Bu fonksiyonların:
  - Parametre sayıları farklı olabilir,
  - Parametre türleri farklı olabilir,
  - Parametrelerin tür sıralamaları farklı olabilir.
- Bu olaya **function overloading** (aşırı yükleme) denir.
- Derleyici, **çağrı anında** gönderilen argümanlara bakarak hangi fonksiyonun çağrılabileceğine karar verir.

- Sadece **dönüş türünü** (return type) değiştirerek fonksiyon aşırı yüklenemez.
- Örnek (HATA):
  - int f(int x);
  - double f(int x); **(Yanlış!)**
- Çünkü derleyici, çağrı anında sadece fonksiyon ismine ve parametre listesine bakar, dönüş türüne bakmaz.

## Örnek 1: topla Fonksiyonunu Aşırı Yüklemek

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

// 1) İki int toplayan fonksiyon
int topla(int a, int b) {
    cout << "int + int fonksiyonu cagirildi\n";
    return a + b;
}

// 2) Üç int toplayan fonksiyon
int topla(int a, int b, int c) {
    cout << "int + int + int fonksiyonu cagirildi\n";
    return a + b + c;
}

// 3) İki double toplayan fonksiyon
double topla(double x, double y) {
    cout << "double + double fonksiyonu cagirildi\n";
    return x + y;
```

## Örnek 1: Ana Program ve Çıktı

```
int main() {  
    int     a = 3, b = 5, c = 10;  
    double x = 2.5, y = 4.7;  
  
    cout << "topla(a, b)    = " << topla(a, b)    << endl;  
    cout << "topla(a, b, c)= " << topla(a, b, c) << endl;  
    cout << "topla(x, y)    = " << topla(x, y)    << endl;  
  
    return 0;  
}
```

## Örnek 1: Ana Program ve Çıktı

```
int main() {  
    int     a = 3, b = 5, c = 10;  
    double x = 2.5, y = 4.7;  
  
    cout << "topla(a, b)    = " << topla(a, b)    << endl;  
    cout << "topla(a, b, c)= " << topla(a, b, c) << endl;  
    cout << "topla(x, y)    = " << topla(x, y)    << endl;  
  
    return 0;  
}
```

### Olası Çıktı:

- int + int fonksiyonu çağırıldı
- topla(a, b) = 8
- int + int + int fonksiyonu çağırıldı
- topla(a, b, c)= 18
- double + double fonksiyonu çağırıldı

- `topla(a, b)` çağrılarında:
  - İki adet int gönderildiği için `int topla(int, int)` fonksiyonu seçilir.
- `topla(a, b, c)` çağrılarında:
  - Üç adet int gönderildiği için `int topla(int, int, int)` fonksiyonu seçilir.
- `topla(x, y)` çağrılarında:
  - İki adet double gönderildiği için `double topla(double, double)` fonksiyonu seçilir.
- Derleyici, uygun fonksiyonu **derleme zamanında** seçer (compile-time polymorphism).

## Örnek 2: Farklı Türlere Göre Aşırı Yükleme

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

void yazdir(int deger) {
    cout << "int deger: " << deger << endl;
}

void yazdir(double deger) {
    cout << "double deger: " << deger << endl;
}

void yazdir(const char* metin) {
    cout << "metin: " << metin << endl;
}

int main() {
    yazdir(10);           // int versiyonu
    yazdir(3.14);         // double versiyonu
    yazdir("Merhaba");   // const char* versiyonu
```

## Örnek 2: Yorum

- `yazdir(10)` çağrılarında:
  - Parametre türü `int` olduğu için `void yazdir(int)` fonksiyonu çağrılır.
- `yazdir(3.14)` çağrılarında:
  - Parametre türü `double` olduğu için `void yazdir(double)` fonksiyonu çağrılır.
- `yazdir("Merhaba")` çağrılarında:
  - Parametre türü `const char*` olduğu için `void yazdir(const char*)` fonksiyonu çağrılır.
- Aynı isimli fonksiyonlar, **parametre tipine göre** seçilerek çağrılır.

## Pointer ile Dizilerde Sıralama

- Dizi ismi, çoğu durumda ilk elemanın adresi gibi davranış gösterir.
- Pointer aritmetiği ( $p+i$ ,  $*(p+i)$ ) ile dizide dolaşılabilir.
- Selection sort gibi algoritmalar pointer kullanılarak fonksiyonlar içinde uygulanabilir.

## Aşırı Yüklü Fonksiyonlar

- Aynı isimli fonksiyonlar, farklı parametre listeleri ile tanımlanabilir.
- Derleyici, çağrıda kullanılan argümanların tür ve sayısına göre uygun fonksiyonu seçer.
- Sadece dönüş türünü değiştirerek aşırı yükleme yapılamaz.

# Teşekkürler