

C++ Math & Random Library Rehberi

Kapsamlı Matematiksel ve Random Fonksiyonlar Kılavuzu

C++ Programcıları için Detaylı Referans

8 Eylül 2025

İçindekiler

1	Giriş	4
1.1	Kütüphane Dahil Etme	4
2	Temel Matematiksel Fonksiyonlar	4
2.1	Mutlak Değer Fonksiyonları	4
2.2	Üs ve Kök Fonksiyonları	5
3	Trigonometrik Fonksiyonlar	5
3.1	Temel Trigonometrik Fonksiyonlar	5
3.2	Hiperbolik Fonksiyonlar	6
4	Logaritmik Fonksiyonlar	7
5	Yuvarlama ve Taban Fonksiyonları	7
5.1	Yuvarlama Fonksiyonları	7
5.2	Modulo ve Kalan İşlemleri	8
6	Özel Matematiksel Fonksiyonlar	9
6.1	Faktöriyel ve Gamma Fonksiyonları	9
7	Sayı Özellikleri ve Kontrol Fonksiyonları	9
7.1	Sayı Sınıflandırma	9
8	Pratik Örnekler ve Kullanım Alanları	10
8.1	Geometrik Hesaplamalar	10
8.2	İstatistiksel Hesaplamalar	11
9	C++ Random Kütüphanesi	12
9.1	Modern Random Sayı Üretimi	12
9.2	Random Sayı Motorları (Engines)	12
9.3	Dağılım Fonksiyonları (Distributions)	13
9.3.1	Tam Sayı Dağılımları	13
9.3.2	Ondalık Sayı Dağılımları	14
9.4	Pratik Random Uygulamaları	15
9.4.1	Dizi Karıştırma ve Örneklem	15
9.4.2	Monte Carlo Simülasyonu	15
9.4.3	Random Walk Simülasyonu	16
9.5	Belirli Aralıklarda Random Sayı Üretimi	17
9.5.1	Tam Sayı Aralıkları	17
9.5.2	Ondalık Sayı Aralıkları	19
9.5.3	Özel Aralık Türleri	21
9.5.4	Performanslı Aralık Üretimi	23
9.6	Array'den Random Element Seçimi	26
9.6.1	Tekrarlı Seçim (Replacement ile)	26
9.6.2	Tekrarsız Seçim (Without Replacement)	28
9.6.3	Performance Karşılaştırması	31
9.7	Random Sayı Üretimi Best Practices	33

10 Performans İpuçları ve En İyi Uygulamalar	34
10.1 Hız Optimizasyonu	34
11 Hata Yönetimi ve Debugging	35
11.1 Matematiksel Hatalar	35
12 C++17 ve Sonrası Yeni Özellikler	36
12.1 Özel Matematik Fonksiyonları	36
13 Sonuç ve Özet	37
13.1 Referanslar ve Kaynaklar	37

1 Giriş

C++ programlama dilinin `<cmath>` ve `<random>` kütüphaneleri, matematiksel hesaplamalar ve rastgele sayı üretimi için kapsamlı fonksiyon koleksiyonları sunar. Bu rehber, C++ math ve random library'lerinin tüm özelliklerini detaylı bir şekilde açıklamaktadır.

1.1 Kütüphane Dahil Etme

```
1 #include <cmath>
2 #include <random>
3 #include <iostream>
4
5 int main() {
6     // Math ve Random fonksiyonlar art k kullan labilir
7     return 0;
8 }
```

Listing 1: Math ve Random kütüphanelerini dahil etme

2 Temel Matematiksel Fonksiyonlar

2.1 Mutlak Değer Fonksiyonları

=blue!5!white=blue!75!black=Mutlak Değer

Bir sayının mutlak değerini hesaplamak için kullanılır.

```
1 #include <cmath>
2 #include <iostream>
3
4 int main() {
5     // Farklı veri türleri için mutlak değer
6     int a = -5;
7     float b = -3.14f;
8     double c = -2.71828;
9
10    std::cout << "abs(" << a << ") = " << std::abs(a) << std::endl;
11    std::cout << "fabs(" << b << ") = " << std::fabs(b) << std::endl;
12    std::cout << "fabs(" << c << ") = " << std::fabs(c) << std::endl;
13
14    return 0;
15 }
```

Listing 2: Mutlak değer fonksiyonları

Fonksiyon Çeşitleri:

- `abs(int)` - Tam sayılar için
- `fabs(float/double)` - Ondalık sayılar için

- labs(long) - Long türü için
- llabs(long long) - Long long türü için

2.2 Üs ve Kök Fonksiyonları

=green!5!white=green!75!black=Üs ve Kök İşlemleri

Sayıların üssünü alma ve kök hesaplama işlemleri.

```

1 #include <cmath>
2 #include <iostream>
3
4 int main() {
5     double x = 16.0;
6     double y = 2.0;
7
8     // s i lemleri
9     std::cout << "pow(" << x << ", " << y << ") = " << std::pow(x,
10     y) << std::endl;
11     std::cout << "exp(" << y << ") = " << std::exp(y) << std::endl;
12     std::cout << "exp2(" << y << ") = " << std::exp2(y) << std::
13     endl;
14
15     // K k i lemleri
16     std::cout << "sqrt(" << x << ") = " << std::sqrt(x) << std::
17     endl;
18     std::cout << "cbrt(" << x << ") = " << std::cbrt(x) << std::
19     endl;
20     std::cout << "hypot(3, 4) = " << std::hypot(3, 4) << std::endl;
21
22     return 0;
23 }
```

Listing 3: Üs ve kök fonksiyonları

Fonksiyon Açıklamaları:

- pow(x, y) - x^y hesaplar
- exp(x) - e^x hesaplar
- exp2(x) - 2^x hesaplar
- sqrt(x) - \sqrt{x} hesaplar
- cbrt(x) - $\sqrt[3]{x}$ hesaplar
- hypot(x, y) - $\sqrt{x^2 + y^2}$ hesaplar

3 Trigonometrik Fonksiyonlar

3.1 Temel Trigonometrik Fonksiyonlar

=red!5!white=red!75!black=Trigonometri

Trigonometrik hesaplamalar için temel fonksiyonlar (radyan cinsinden).

```

1 #include <cmath>
2 #include <iostream>
3
4 int main() {
5     const double PI = std::acos(-1);
6     double angle = PI / 4; // 45 derece
7
8     // Temel trigonometrik fonksiyonlar
9     std::cout << "sin(" << angle << ") = " << std::sin(angle) <<
10         std::endl;
11     std::cout << "cos(" << angle << ") = " << std::cos(angle) <<
12         std::endl;
13     std::cout << "tan(" << angle << ") = " << std::tan(angle) <<
14         std::endl;
15
16     // Ters trigonometrik fonksiyonlar
17     double value = 0.707;
18     std::cout << "asin(" << value << ") = " << std::asin(value) <<
19         std::endl;
20     std::cout << "acos(" << value << ") = " << std::acos(value) <<
21         std::endl;
22     std::cout << "atan(" << value << ") = " << std::atan(value) <<
23         std::endl;
24     std::cout << "atan2(1, 1) = " << std::atan2(1, 1) << std::endl;
25
26     return 0;
27 }
```

Listing 4: Trigonometrik fonksiyonlar

3.2 Hiperbolik Fonksiyonlar

```

1 #include <cmath>
2 #include <iostream>
3
4 int main() {
5     double x = 1.0;
6
7     // Hiperbolik fonksiyonlar
8     std::cout << "sinh(" << x << ") = " << std::sinh(x) << std::
9         endl;
10    std::cout << "cosh(" << x << ") = " << std::cosh(x) << std::
11        endl;
12    std::cout << "tanh(" << x << ") = " << std::tanh(x) << std::
13        endl;
14
15    // Ters hiperbolik fonksiyonlar
16    std::cout << "asinh(" << x << ") = " << std::asinh(x) << std::
17        endl;
18    std::cout << "acosh(" << x + 1 << ") = " << std::acosh(x + 1)
19        << std::endl;
20    std::cout << "atanh(" << x/2 << ") = " << std::atanh(x/2) <<
21        std::endl;
22 }
```

```

16
17     return 0;
18 }

```

Listing 5: Hiperbolik fonksiyonlar

4 Logaritmik Fonksiyonlar

=orange!5!white=orange!75!black=Logaritma

Farklı tabanlarda logaritma hesaplama fonksiyonları.

```

1 #include <cmath>
2 #include <iostream>
3
4 int main() {
5     double x = 100.0;
6
7     // Farklı tabanlarda logaritma
8     std::cout << "log(" << x << ") = " << std::log(x) << std::endl;
9         // ln(x)
10    std::cout << "log10(" << x << ") = " << std::log10(x) << std::
11        endl; // log_10(x)
12    std::cout << "log2(" << x << ") = " << std::log2(x) << std::
13        endl; // log_2(x)
14
15    // zek logaritma fonksiyonlar
16    std::cout << "log1p(" << 0.1 << ") = " << std::log1p(0.1) <<
17        std::endl; // ln(1+x)
18    std::cout << "logb(" << x << ", 3) = " << std::logb(x) << std::
19        endl; // log_r(x)
20
21    return 0;
22 }

```

Listing 6: Logaritmik fonksiyonlar

5 Yuvarlama ve Taban Fonksiyonları

5.1 Yuvarlama Fonksiyonları

=purple!5!white=purple!75!black=Yuvarlama

Sayıları farklı şekillerde yuvarlama fonksiyonları.

```

1 #include <cmath>
2 #include <iostream>
3
4 int main() {
5     double x = 3.7;

```

```

6     double y = -3.7;
7
8     std::cout << "Pozitif say : " << x << std::endl;
9     std::cout << "ceil(" << x << ") = " << std::ceil(x) << std::
10    endl;    // Yukar yuvarla
11    std::cout << "floor(" << x << ") = " << std::floor(x) << std::
12    endl;    // A a yuvarla
13    std::cout << "round(" << x << ") = " << std::round(x) << std::
14    endl;    // En yak na yuvarla
15    std::cout << "trunc(" << x << ") = " << std::trunc(x) << std::
16    endl;    // Kesirli k sm at
17
18    std::cout << "\\nNegatif say : " << y << std::endl;
19    std::cout << "ceil(" << y << ") = " << std::ceil(y) << std::
20    endl;
21    std::cout << "floor(" << y << ") = " << std::floor(y) << std::
22    endl;
23    std::cout << "round(" << y << ") = " << std::round(y) << std::
24    endl;
25    std::cout << "trunc(" << y << ") = " << std::trunc(y) << std::
26    endl;
27
28    return 0;
29 }

```

Listing 7: Yuvarlama fonksiyonları

5.2 Modulo ve Kalan İşlemleri

```

1 #include <cmath>
2 #include <iostream>
3
4 int main() {
5     double x = 7.5;
6     double y = 2.3;
7
8     // Kalan i lemleri
9     std::cout << "fmod(" << x << ", " << y << ") = " << std::fmod(x
10    , y) << std::endl;
11    std::cout << "remainder(" << x << ", " << y << ") = " << std::
12    remainder(x, y) << std::endl;
13
14    // B l m ve kalan
15    int quotient;
16    double rem = std::remquo(x, y, &quotient);
17    std::cout << "remquo(" << x << ", " << y << ") = " << rem
18    << ", quotient = " << quotient << std::endl;
19
20    return 0;
21 }

```

Listing 8: Modulo ve kalan işlemleri

6 Özel Matematiksel Fonksiyonlar

6.1 Faktöriyel ve Gamma Fonksiyonları

=teal!5!white=teal!75!black=Özel Fonksiyonlar

Gelişmiş matematiksel hesaplamalar için özel fonksiyonlar.

```

1 #include <cmath>
2 #include <iostream>
3
4 int main() {
5     double x = 5.0;
6
7     // Gamma fonksiyonlar (C++17 ve sonrası)
8     #if __cplusplus >= 201703L
9     std::cout << "tgamma(" << x << ") = " << std::tgamma(x) << std
10     ::endl;          // (x)
11     std::cout << "lgamma(" << x << ") = " << std::lgamma(x) << std
12     ::endl;          // ln| (x)|
13     #endif
14
15     // Manuel faktöriyel hesaplama
16     auto factorial = [](int n) -> long long {
17         if (n <= 1) return 1;
18         return n * factorial(n - 1);
19     };
20
21     std::cout << "5! = " << factorial(5) << std::endl;
22
23     return 0;
24 }
```

Listing 9: Gamma ve faktöriyel fonksiyonları

7 Sayı Özellikleri ve Kontrol Fonksiyonları

7.1 Sayı Sınıflandırma

=yellow!5!white=yellow!75!black=Sayı Kontrolü

Sayıların özelliklerini kontrol etmek için kullanılan fonksiyonlar.

```

1 #include <cmath>
2 #include <iostream>
3 #include <limits>
4
5 int main() {
6     double finite_num = 42.0;
7     double inf_num = std::numeric_limits<double>::infinity();
8     double nan_num = std::numeric_limits<double>::quiet_NaN();
9 }
```

```

10 // Say zelliklerini kontrol et
11 std::cout << "Sonlu say kontrol :" << std::endl;
12 std::cout << "isfinite(" << finite_num << ") = " << std::
    isfinite(finite_num) << std::endl;
13 std::cout << "isinf(" << finite_num << ") = " << std::isinf(
    finite_num) << std::endl;
14 std::cout << "isnan(" << finite_num << ") = " << std::isnan(
    finite_num) << std::endl;
15
16 std::cout << "\\nSonsuz say kontrol :" << std::endl;
17 std::cout << "isfinite(inf) = " << std::isfinite(inf_num) <<
    std::endl;
18 std::cout << "isinf(inf) = " << std::isinf(inf_num) << std::
    endl;
19
20 std::cout << "\\nNaN kontrol :" << std::endl;
21 std::cout << "isnan(NaN) = " << std::isnan(nan_num) << std::
    endl;
22
23 // aret kontrol
24 std::cout << "\\n aret kontrol :" << std::endl;
25 std::cout << "signbit(-3.14) = " << std::signbit(-3.14) << std
    ::endl;
26 std::cout << "signbit(3.14) = " << std::signbit(3.14) << std::
    endl;
27
28 return 0;
29 }

```

Listing 10: Sayı sınıflandırma fonksiyonları

8 Pratik Örnekler ve Kullanım Alanları

8.1 Geometrik Hesaplamalar

```

1 #include <cmath>
2 #include <iostream>
3
4 class GeometryCalculator {
5 public:
6     // Daire alan hesaplama
7     static double circleArea(double radius) {
8         const double PI = std::acos(-1);
9         return PI * std::pow(radius, 2);
10    }
11
12    // ki nokta aras mesafe
13    static double distance(double x1, double y1, double x2, double
        y2) {
14        return std::hypot(x2 - x1, y2 - y1);
15    }
16
17    // gen alan (Heron form l )
18    static double triangleArea(double a, double b, double c) {
19        double s = (a + b + c) / 2; // Yar evre
20        return std::sqrt(s * (s - a) * (s - b) * (s - c));

```

```

21     }
22 };
23
24 int main() {
25     double radius = 5.0;
26     std::cout << "Daire alan (r=" << radius << "): "
27               << GeometryCalculator::circleArea(radius) << std::
28               endl;
29
30     double x1 = 0, y1 = 0, x2 = 3, y2 = 4;
31     std::cout << "Nokta mesafesi: "
32               << GeometryCalculator::distance(x1, y1, x2, y2) <<
33               std::endl;
34
35     double a = 3, b = 4, c = 5;
36     std::cout << "gen alan : "
37               << GeometryCalculator::triangleArea(a, b, c) << std::
38               endl;
39
40     return 0;
41 }

```

Listing 11: Geometrik hesaplama örneği

8.2 İstatistiksel Hesaplamalar

```

1  #include <cmath>
2  #include <iostream>
3  #include <vector>
4  #include <numeric>
5
6  class StatisticsCalculator {
7  public:
8      // Aritmetik ortalama
9      static double mean(const std::vector<double>& data) {
10         return std::accumulate(data.begin(), data.end(), 0.0) /
11         data.size();
12     }
13
14     // Standart sapma
15     static double standardDeviation(const std::vector<double>& data
16     ) {
17         double avg = mean(data);
18         double sum = 0.0;
19
20         for (double value : data) {
21             sum += std::pow(value - avg, 2);
22         }
23
24         return std::sqrt(sum / data.size());
25     }
26
27     // Varyans
28     static double variance(const std::vector<double>& data) {
29         double stdDev = standardDeviation(data);
30         return std::pow(stdDev, 2);
31     }
32 }

```

```

30 };
31
32 int main() {
33     std::vector<double> data = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
34                               8.0, 9.0, 10.0};
35
36     std::cout << "Veri seti: ";
37     for (double value : data) {
38         std::cout << value << " ";
39     }
40     std::cout << std::endl;
41
42     std::cout << "Ortalama: " << StatisticsCalculator::mean(data)
43               << std::endl;
44     std::cout << "Standart sapma: " << StatisticsCalculator::
45               standardDeviation(data) << std::endl;
46     std::cout << "Varyans: " << StatisticsCalculator::variance(data)
47               << std::endl;
48
49     return 0;
50 }

```

Listing 12: İstatistiksel hesaplamalar

9 C++ Random Kütüphanesi

9.1 Modern Random Sayı Üretimi

=cyan!5!white=cyan!75!black=Random Kütüphanesi

C++11 ile birlikte gelen modern ve güvenli rastgele sayı üretim sistemi.

C++ <random> kütüphanesi, eski `rand()` fonksiyonuna göre çok daha güçlü ve esnek bir rastgele sayı üretim sistemi sunar.

```

1 #include <random>
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5
6 int main() {
7     // Modern random sayı üretimi artık kullanılabiliyor
8     return 0;
9 }

```

Listing 13: Random kütüphanesini dahil etme

9.2 Random Sayı Motorları (Engines)

```

1 #include <random>
2 #include <iostream>
3 #include <chrono>
4

```

```

5 int main() {
6     // Farklı random motorlar
7     std::default_random_engine engine1;
8     std::mt19937 engine2;           // Mersenne Twister
9     std::mt19937_64 engine3;       // 64-bit Mersenne Twister
10    std::ranlux24 engine4;          // RANLUX generator
11
12    // Seed ayarlama
13    unsigned seed = std::chrono::system_clock::now().
        time_since_epoch().count();
14    std::mt19937 generator(seed);
15
16    // Basit say retimi
17    for (int i = 0; i < 5; ++i) {
18        std::cout << "Random: " << generator() << std::endl;
19    }
20
21    return 0;
22 }

```

Listing 14: Farklı random motorları

9.3 Dağılım Fonksiyonları (Distributions)

9.3.1 Tam Sayı Dağılımları

```

1 #include <random>
2 #include <iostream>
3
4 int main() {
5     std::random_device rd;
6     std::mt19937 gen(rd());
7
8     // Uniform dağılım (1-6 aras zar atma)
9     std::uniform_int_distribution<> dice(1, 6);
10    std::cout << "Zar atmalar: ";
11    for (int i = 0; i < 10; ++i) {
12        std::cout << dice(gen) << " ";
13    }
14    std::cout << std::endl;
15
16    // Bernoulli dağılımı (coin flip)
17    std::bernoulli_distribution coin(0.5);
18    std::cout << "Coin flips: ";
19    for (int i = 0; i < 10; ++i) {
20        std::cout << (coin(gen) ? "H" : "T") << " ";
21    }
22    std::cout << std::endl;
23
24    // Binomial dağılım
25    std::binomial_distribution<> binomial(10, 0.3);
26    std::cout << "Binomial (n=10, p=0.3): " << binomial(gen) << std::endl;
27
28    // Geometric dağılım
29    std::geometric_distribution<> geometric(0.2);

```

```

30     std::cout << "Geometric (p=0.2): " << geometric(gen) << std::
        endl;
31
32     return 0;
33 }

```

Listing 15: Tam sayı dağılımları

9.3.2 Ondalık Sayı Dağılımları

```

1  #include <random>
2  #include <iostream>
3  #include <iomanip>
4
5  int main() {
6      std::random_device rd;
7      std::mt19937 gen(rd());
8
9      // Uniform da 1 m (0.0 - 1.0 aras )
10     std::uniform_real_distribution<> uniform(0.0, 1.0);
11     std::cout << std::fixed << std::setprecision(4);
12     std::cout << "Uniform da 1 m: ";
13     for (int i = 0; i < 5; ++i) {
14         std::cout << uniform(gen) << " ";
15     }
16     std::cout << std::endl;
17
18     // Normal (Gaussian) da 1 m
19     std::normal_distribution<> normal(0.0, 1.0); // mean=0, stddev
        =1
20     std::cout << "Normal da 1 m: ";
21     for (int i = 0; i < 5; ++i) {
22         std::cout << normal(gen) << " ";
23     }
24     std::cout << std::endl;
25
26     // Exponential da 1 m
27     std::exponential_distribution<> exponential(1.0);
28     std::cout << "Exponential da 1 m: ";
29     for (int i = 0; i < 5; ++i) {
30         std::cout << exponential(gen) << " ";
31     }
32     std::cout << std::endl;
33
34     // Gamma da 1 m
35     std::gamma_distribution<> gamma(2.0, 1.0);
36     std::cout << "Gamma da 1 m: ";
37     for (int i = 0; i < 5; ++i) {
38         std::cout << gamma(gen) << " ";
39     }
40     std::cout << std::endl;
41
42     return 0;
43 }

```

Listing 16: Ondalık sayı dağılımları

9.4 Pratik Random Uygulamaları

9.4.1 Dizi Karıştırma ve Örnekleme

```

1 #include <random>
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5
6 int main() {
7     std::random_device rd;
8     std::mt19937 g(rd());
9
10    // Dizi karıştırma
11    std::vector<int> deck = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
12    std::cout << "Original: ";
13    for (int card : deck) std::cout << card << " ";
14    std::cout << std::endl;
15
16    std::shuffle(deck.begin(), deck.end(), g);
17    std::cout << "Karıştırılmış: ";
18    for (int card : deck) std::cout << card << " ";
19    std::cout << std::endl;
20
21    // Random örnekleme
22    std::vector<int> sample(3);
23    std::sample(deck.begin(), deck.end(), sample.begin(), 3, g);
24    std::cout << "3 eleman örneği: ";
25    for (int elem : sample) std::cout << elem << " ";
26    std::cout << std::endl;
27
28    return 0;
29 }

```

Listing 17: Dizi işlemleri

9.4.2 Monte Carlo Simülasyonu

```

1 #include <random>
2 #include <iostream>
3 #include <cmath>
4
5 class MonteCarloSimulation {
6 public:
7     static double estimatePi(int numSamples) {
8         std::random_device rd;
9         std::mt19937 gen(rd());
10        std::uniform_real_distribution<> dis(-1.0, 1.0);
11
12        int insideCircle = 0;
13
14        for (int i = 0; i < numSamples; ++i) {
15            double x = dis(gen);
16            double y = dis(gen);
17
18            if (x*x + y*y <= 1.0) {
19                insideCircle++;

```

```

20     }
21 }
22
23     return 4.0 * insideCircle / numSamples;
24 }
25 };
26
27 int main() {
28     std::vector<int> sampleSizes = {1000, 10000, 100000, 1000000};
29
30     for (int samples : sampleSizes) {
31         double piEstimate = MonteCarloSimulation::estimatePi(
32             samples);
33         double error = std::abs(piEstimate - M_PI);
34
35         std::cout << " rneklem : " << samples
36                 << ", Pi tahmini: " << piEstimate
37                 << ", Hata: " << error << std::endl;
38     }
39
40     return 0;
41 }

```

Listing 18: Monte Carlo Pi hesaplama

9.4.3 Random Walk Simülasyonu

```

1  #include <random>
2  #include <iostream>
3  #include <vector>
4  #include <cmath>
5
6  class RandomWalk {
7  private:
8      double x, y;
9      std::mt19937 gen;
10     std::uniform_real_distribution<> angleDist;
11     std::uniform_real_distribution<> stepDist;
12
13 public:
14     RandomWalk() : x(0.0), y(0.0), gen(std::random_device{}()),
15                 angleDist(0.0, 2 * M_PI), stepDist(0.5, 1.5) {}
16
17     void step() {
18         double angle = angleDist(gen);
19         double stepSize = stepDist(gen);
20
21         x += stepSize * std::cos(angle);
22     void step() {
23         double angle = angleDist(gen);
24         double stepSize = stepDist(gen);
25
26         x += stepSize * std::cos(angle);
27         y += stepSize * std::sin(angle);
28     }
29
30     double distanceFromOrigin() const {

```



```

31     return std::sqrt(x*x + y*y);
32 }
33
34 void reset() { x = 0.0; y = 0.0; }
35
36 std::pair<double, double> getPosition() const { return {x, y};
37 };
38
39 int main() {
40     RandomWalk walker;
41     int steps = 1000;
42
43     std::cout << "Random Walk Sim lasyonu (" << steps << " ad m):
44         " << std::endl;
45
46     for (int i = 0; i < steps; i += 100) {
47         auto [x, y] = walker.getPosition();
48         double distance = walker.distanceFromOrigin();
49
50         std::cout << "Ad m " << i << ": (" << x << ", " << y
51             << "), Mesafe: " << distance << std::endl;
52
53         for (int j = 0; j < 100; ++j) {
54             walker.step();
55         }
56
57         auto [finalX, finalY] = walker.getPosition();
58         std::cout << "Final pozisyon: (" << finalX << ", " << finalY
59             << "), Final mesafe: " << walker.distanceFromOrigin()
60             << std::endl;
61
62         return 0;
63     }

```

Listing 19: Random walk simülasyonu

9.5 Belirli Aralıklarda Random Sayı Üretimi

=lime!5!white=lime!75!black=Aralık Tabanlı Random Üretimi

Belirli aralıklarda tam sayı ve ondalıklı sayı üretme teknikleri.

9.5.1 Tam Sayı Aralıkları

```

1 #include <random>
2 #include <iostream>
3 #include <vector>
4
5 class IntegerRangeGenerator {
6 private:
7     std::mt19937 gen;
8
9 public:

```

```

10 IntegerRangeGenerator() : gen(std::random_device{}()) {}
11
12 // Basit aral k [min, max]
13 int generate(int min, int max) {
14     std::uniform_int_distribution<> dist(min, max);
15     return dist(gen);
16 }
17
18 // ift say lar aral
19 int generateEven(int min, int max) {
20     // Min ve max' ift yapmak i in ayarlama
21     if (min % 2 != 0) min++;
22     if (max % 2 != 0) max--;
23
24     if (min > max) return min; // Ge ersiz aral k
25
26     int evenCount = (max - min) / 2 + 1;
27     std::uniform_int_distribution<> dist(0, evenCount - 1);
28     return min + dist(gen) * 2;
29 }
30
31 // Tek say lar aral
32 int generateOdd(int min, int max) {
33     // Min ve max' tek yapmak i in ayarlama
34     if (min % 2 == 0) min++;
35     if (max % 2 == 0) max--;
36
37     if (min > max) return min; // Ge ersiz aral k
38
39     int oddCount = (max - min) / 2 + 1;
40     std::uniform_int_distribution<> dist(0, oddCount - 1);
41     return min + dist(gen) * 2;
42 }
43
44 // Katlar ( rn : 5'in katlar )
45 int generateMultiple(int min, int max, int multiple) {
46     if (multiple <= 0) return min;
47
48     int adjustedMin = ((min + multiple - 1) / multiple) *
49         multiple;
50     int adjustedMax = (max / multiple) * multiple;
51
52     if (adjustedMin > adjustedMax) return adjustedMin;
53
54     int count = (adjustedMax - adjustedMin) / multiple + 1;
55     std::uniform_int_distribution<> dist(0, count - 1);
56     return adjustedMin + dist(gen) * multiple;
57 }
58
59 int main() {
60     IntegerRangeGenerator generator;
61
62     std::cout << "=== Tam Say Aral k rnekleri ===" << std::
63         endl;
64
65     // Normal aral k
66     std::cout << "1-100 aras 10 say : ";

```

```

66     for (int i = 0; i < 10; ++i) {
67         std::cout << generator.generate(1, 100) << " ";
68     }
69     std::cout << std::endl;
70
71     // ift say lar
72     std::cout << "10-50 aras ift say lar: ";
73     for (int i = 0; i < 10; ++i) {
74         std::cout << generator.generateEven(10, 50) << " ";
75     }
76     std::cout << std::endl;
77
78     // Tek say lar
79     std::cout << "1-25 aras tek say lar: ";
80     for (int i = 0; i < 10; ++i) {
81         std::cout << generator.generateOdd(1, 25) << " ";
82     }
83     std::cout << std::endl;
84
85     // 5'in katlar
86     std::cout << "10-100 aras 5'in katlar : ";
87     for (int i = 0; i < 10; ++i) {
88         std::cout << generator.generateMultiple(10, 100, 5) << " ";
89     }
90     std::cout << std::endl;
91
92     return 0;
93 }

```

Listing 20: Farklı tam sayı aralıkları

9.5.2 Ondalık Sayı Aralıkları

```

1  #include <random>
2  #include <iostream>
3  #include <iomanip>
4  #include <cmath>
5
6  class FloatRangeGenerator {
7  private:
8      std::mt19937 gen;
9
10 public:
11     FloatRangeGenerator() : gen(std::random_device{}()) {}
12
13     // Basit ondalık aralık
14     double generate(double min, double max) {
15         std::uniform_real_distribution<> dist(min, max);
16         return dist(gen);
17     }
18
19     // Belirli hassasiyetle (rn : 2 ondalık basamak)
20     double generateWithPrecision(double min, double max, int
        precision) {
21         double multiplier = std::pow(10, precision);
22         int intMin = static_cast<int>(min * multiplier);
23         int intMax = static_cast<int>(max * multiplier);

```

```

24         std::uniform_int_distribution<> dist(intMin, intMax);
25         return dist(gen) / multiplier;
26     }
27
28     // Belirli ad mlarla ( rn : 0.5 ad mlarla)
29     double generateWithStep(double min, double max, double step) {
30         if (step <= 0) return min;
31
32         int steps = static_cast<int>((max - min) / step);
33         std::uniform_int_distribution<> dist(0, steps);
34         return min + dist(gen) * step;
35     }
36
37     // Logaritmik aral k
38     double generateLogarithmic(double min, double max) {
39         if (min <= 0 || max <= 0) return 1.0;
40
41         double logMin = std::log(min);
42         double logMax = std::log(max);
43         std::uniform_real_distribution<> dist(logMin, logMax);
44         return std::exp(dist(gen));
45     }
46 };
47
48 int main() {
49     FloatRangeGenerator generator;
50
51     std::cout << std::fixed << std::setprecision(3);
52     std::cout << "=== Ondal k l Say Aral k rnekleri ===" <<
53         std::endl;
54
55     // Normal aral k
56     std::cout << "0.0-1.0 aras : ";
57     for (int i = 0; i < 5; ++i) {
58         std::cout << generator.generate(0.0, 1.0) << " ";
59     }
60     std::cout << std::endl;
61
62     // Hassasiyet kontrol
63     std::cout << "1.0-10.0 aras (2 basamak): ";
64     for (int i = 0; i < 5; ++i) {
65         std::cout << generator.generateWithPrecision(1.0, 10.0, 2)
66             << " ";
67     }
68     std::cout << std::endl;
69
70     // Ad m kontrol
71     std::cout << "0.0-5.0 aras (0.5 ad m): ";
72     for (int i = 0; i < 5; ++i) {
73         std::cout << generator.generateWithStep(0.0, 5.0, 0.5) << "
74             ";
75     }
76     std::cout << std::endl;
77
78     // Logaritmik
79     std::cout << "1-1000 aras logaritmik: ";
80     for (int i = 0; i < 5; ++i) {

```

```

79         std::cout << generator.generateLogarithmic(1.0, 1000.0) <<
            " ";
80     }
81     std::cout << std::endl;
82
83     return 0;
84 }

```

Listing 21: Ondalıklı sayı aralıkları ve hassasiyet

9.5.3 Özel Aralık Türleri

```

1  #include <random>
2  #include <iostream>
3  #include <vector>
4  #include <set>
5  #include <algorithm>
6
7  class SpecialRangeGenerator {
8  private:
9      std::mt19937 gen;
10
11 public:
12     SpecialRangeGenerator() : gen(std::random_device{}()) {}
13
14     // Belirli say lar hari tutarak
15     int generateExcluding(int min, int max, const std::set<int>&
        excluded) {
16         std::vector<int> validNumbers;
17         for (int i = min; i <= max; ++i) {
18             if (excluded.find(i) == excluded.end()) {
19                 validNumbers.push_back(i);
20             }
21         }
22
23         if (validNumbers.empty()) return min;
24
25         std::uniform_int_distribution<> dist(0, validNumbers.size()
            - 1);
26         return validNumbers[dist(gen)];
27     }
28
29     // Sadece belirli say lar dahil ederek
30     int generateFromSet(const std::vector<int>& allowedNumbers) {
31         if (allowedNumbers.empty()) return 0;
32
33         std::uniform_int_distribution<> dist(0, allowedNumbers.size
            () - 1);
34         return allowedNumbers[dist(gen)];
35     }
36
37     // Gaussian da l m ile s n r l aral k
38     double generateGaussianInRange(double min, double max, double
        mean, double stddev) {
39         std::normal_distribution<> dist(mean, stddev);
40         double value;
41

```

```

42 // Aral k d na kana kadar tekrar dene
43 int attempts = 0;
44 do {
45     value = dist(gen);
46     attempts++;
47     if (attempts > 1000) { // Sonsuz d ng nleme
48         return (min + max) / 2.0;
49     }
50 } while (value < min || value > max);
51
52 return value;
53 }
54
55 // A rl kl aral k se imi
56 int generateWeighted(const std::vector<std::pair<std::pair<int,
57 int>, double>>& weightedRanges) {
58     if (weightedRanges.empty()) return 0;
59
60     // A rl klar topla
61     double totalWeight = 0.0;
62     for (const auto& range : weightedRanges) {
63         totalWeight += range.second;
64     }
65
66     // Random weight se
67     std::uniform_real_distribution<> weightDist(0.0,
68 totalWeight);
69     double selectedWeight = weightDist(gen);
70
71     // Hangi aral n se ildi ini bul
72     double currentWeight = 0.0;
73     for (const auto& range : weightedRanges) {
74         currentWeight += range.second;
75         if (selectedWeight <= currentWeight) {
76             // Bu aral ktan say ret
77             std::uniform_int_distribution<> rangeDist(range.
78 first.first, range.first.second);
79             return rangeDist(gen);
80         }
81     }
82
83     return weightedRanges[0].first.first; // Fallback
84 }
85 };
86
87 int main() {
88     SpecialRangeGenerator generator;
89
90     std::cout << "=== zel Aral k T rleri ===" << std::endl;
91
92     // Hari tutma
93     std::set<int> excluded = {13, 666, 777};
94     std::cout << "1-20 aras (13 hari ) : ";
95     for (int i = 0; i < 10; ++i) {
96         std::cout << generator.generateExcluding(1, 20, excluded)
97             << " ";
98     }
99     std::cout << std::endl;

```

```

96
97 // Belirli k meden se im
98 std::vector<int> primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
99     31};
100 std::cout << "Asal say lardan: ";
101 for (int i = 0; i < 10; ++i) {
102     std::cout << generator.generateFromSet(primes) << " ";
103 }
104 std::cout << std::endl;
105
106 // Gaussian s n rl
107 std::cout << std::fixed << std::setprecision(2);
108 std::cout << "Gaussian (0-10, mean=5, std=1.5): ";
109 for (int i = 0; i < 10; ++i) {
110     std::cout << generator.generateGaussianInRange(0.0, 10.0,
111         5.0, 1.5) << " ";
112 }
113 std::cout << std::endl;
114
115 // A r l k l aral klar
116 std::vector<std::pair<std::pair<int, int>, double>>
117     weightedRanges = {
118     {{1, 10}, 0.1}, // %10 ans
119     {{11, 50}, 0.6}, // %60 ans
120     {{51, 100}, 0.3} // %30 ans
121 };
122
123 std::cout << "A r l k l aral klar: ";
124 for (int i = 0; i < 15; ++i) {
125     std::cout << generator.generateWeighted(weightedRanges) <<
126         " ";
127 }
128 std::cout << std::endl;
129
130 return 0;
131 }

```

Listing 22: Özel aralık türleri ve filtreleme

9.5.4 Performanslı Aralık Üretimi

```

1 #include <random>
2 #include <iostream>
3 #include <vector>
4 #include <chrono>
5
6 class OptimizedRangeGenerator {
7 private:
8     mutable std::mt19937 gen;
9
10 public:
11     OptimizedRangeGenerator() : gen(std::random_device{}()) {}
12
13     // Batch retim - tek seferde birden fazla say
14     std::vector<int> generateBatch(int min, int max, size_t count)
15     {
16         std::uniform_int_distribution<> dist(min, max);

```

```

16     std::vector<int> result;
17     result.reserve(count);
18
19     for (size_t i = 0; i < count; ++i) {
20         result.push_back(dist(gen));
21     }
22
23     return result;
24 }
25
26 // Unique say lar retimi (tekrars z)
27 std::vector<int> generateUnique(int min, int max, size_t count)
28 {
29     if (count > static_cast<size_t>(max - min + 1)) {
30         count = max - min + 1; // Max m mk n say kadar
31     }
32
33     std::vector<int> range;
34     for (int i = min; i <= max; ++i) {
35         range.push_back(i);
36     }
37
38     std::shuffle(range.begin(), range.end(), gen);
39     range.resize(count);
40     return range;
41 }
42
43 // H z l boolean array i in
44 std::vector<bool> generateBoolArray(size_t size, double
45 trueRatio = 0.5) {
46     std::bernoulli_distribution dist(trueRatio);
47     std::vector<bool> result;
48     result.reserve(size);
49
50     for (size_t i = 0; i < size; ++i) {
51         result.push_back(dist(gen));
52     }
53
54     return result;
55 }
56
57 // Cached distribution kullan m
58 class CachedIntGenerator {
59 private:
60     std::uniform_int_distribution<> dist;
61     std::mt19937& gen_ref;
62
63 public:
64     CachedIntGenerator(int min, int max, std::mt19937& gen)
65         : dist(min, max), gen_ref(gen) {}
66
67     int operator()() {
68         return dist(gen_ref);
69     }
70 };
71
72 CachedIntGenerator getCachedGenerator(int min, int max) {
73     return CachedIntGenerator(min, max, gen);

```



```

72     }
73 };
74
75 // Performance test fonksiyonu
76 void performanceTest() {
77     OptimizedRangeGenerator generator;
78     const int iterations = 1000000;
79
80     auto start = std::chrono::high_resolution_clock::now();
81
82     // Normal retim
83     for (int i = 0; i < iterations; ++i) {
84         std::uniform_int_distribution<> dist(1, 100);
85         static std::mt19937 gen(std::random_device{}());
86         volatile int result = dist(gen); // volatile to prevent
            optimization
87         (void)result; // Suppress unused variable warning
88     }
89
90     auto end = std::chrono::high_resolution_clock::now();
91     auto duration1 = std::chrono::duration_cast<std::chrono::
        microseconds>(end - start);
92
93     start = std::chrono::high_resolution_clock::now();
94
95     // Cached retim
96     auto cachedGen = generator.getCachedGenerator(1, 100);
97     for (int i = 0; i < iterations; ++i) {
98         volatile int result = cachedGen();
99         (void)result;
100     }
101
102     end = std::chrono::high_resolution_clock::now();
103     auto duration2 = std::chrono::duration_cast<std::chrono::
        microseconds>(end - start);
104
105     std::cout << "Normal retim : " << duration1.count() << " s "
        << std::endl;
106     std::cout << "Cached retim : " << duration2.count() << " s "
        << std::endl;
107     std::cout << "H zlanma: " << static_cast<double>(duration1.
        count()) / duration2.count() << "x" << std::endl;
108 }
109
110 int main() {
111     OptimizedRangeGenerator generator;
112
113     std::cout << "=== Performansl Aral k retimi ===" << std::
        endl;
114
115     // Batch retim
116     auto batch = generator.generateBatch(1, 100, 10);
117     std::cout << "Batch (10 say ):";
118     for (int num : batch) {
119         std::cout << num << " ";
120     }
121     std::cout << std::endl;
122

```

```

123 // Unique say lar
124 auto unique = generator.generateUnique(1, 20, 8);
125 std::cout << "Unique say lar: ";
126 for (int num : unique) {
127     std::cout << num << " ";
128 }
129 std::cout << std::endl;
130
131 // Boolean array
132 auto bools = generator.generateBoolArray(15, 0.3);
133 std::cout << "Boolean array: ";
134 for (bool b : bools) {
135     std::cout << (b ? "T" : "F") << " ";
136 }
137 std::cout << std::endl;
138
139 // Performance test
140 std::cout << "\\n=== Performance Kar la t rmas ===" <<
    std::endl;
141 performanceTest();
142
143 return 0;
144 }

```

Listing 23: Optimized range generation

9.6 Array'den Random Element Seçimi

=magenta!5!white=magenta!75!black=Array Random Seçimi

Dizilerden istenen sayıda rastgele element seçme teknikleri.

9.6.1 Tekrarlı Seçim (Replacement ile)

```

1 #include <random>
2 #include <iostream>
3 #include <vector>
4 #include <array>
5
6 class ArrayRandomSelector {
7 private:
8     std::mt19937 gen;
9
10 public:
11     ArrayRandomSelector() : gen(std::random_device{}()) {}
12
13     // Vector'den tekrarlı seçim
14     template<typename T>
15     std::vector<T> selectWithReplacement(const std::vector<T>&
        source, size_t count) {
16         if (source.empty()) return {};
17
18         std::uniform_int_distribution<> dist(0, source.size() - 1);
19         std::vector<T> result;
20         result.reserve(count);

```

```

21
22     for (size_t i = 0; i < count; ++i) {
23         result.push_back(source[dist(gen)]);
24     }
25
26     return result;
27 }
28
29 // Array'den tekrarlı se im
30 template<typename T, size_t N>
31 std::vector<T> selectWithReplacement(const std::array<T, N>&
32     source, size_t count) {
33     if (N == 0) return {};
34
35     std::uniform_int_distribution<> dist(0, N - 1);
36     std::vector<T> result;
37     result.reserve(count);
38
39     for (size_t i = 0; i < count; ++i) {
40         result.push_back(source[dist(gen)]);
41     }
42
43     return result;
44 }
45
46 // C-style array'den tekrarlı se im
47 template<typename T>
48 std::vector<T> selectWithReplacement(const T* source, size_t
49     sourceSize, size_t count) {
50     if (sourceSize == 0) return {};
51
52     std::uniform_int_distribution<> dist(0, sourceSize - 1);
53     std::vector<T> result;
54     result.reserve(count);
55
56     for (size_t i = 0; i < count; ++i) {
57         result.push_back(source[dist(gen)]);
58     }
59
60     return result;
61 }
62 };
63
64 int main() {
65     ArrayRandomSelector selector;
66
67     // Vector rnei
68     std::vector<std::string> colors = {"K rm z ", "Mavi", "
69     Ye il", "Sar ", "Mor", "Turuncu"};
70     auto selectedColors = selector.selectWithReplacement(colors, 4)
71     ;
72
73     std::cout << "Vector'den 4 renk se imi (tekrarlı ): ";
74     for (const auto& color : selectedColors) {
75         std::cout << color << " ";
76     }
77     std::cout << std::endl;

```

```

75 // Array rnei
76 std::array<int, 8> numbers = {10, 20, 30, 40, 50, 60, 70, 80};
77 auto selectedNumbers = selector.selectWithReplacement(numbers,
78     5);
79
80 std::cout << "Array'den 5 say se imi (tekrarlı): ";
81 for (int num : selectedNumbers) {
82     std::cout << num << " ";
83 }
84 std::cout << std::endl;
85
86 // C-style array rnei
87 const char* fruits[] = {"Elma", "Armut", "Muz", "Portakal", "
88     zm "};
89 auto selectedFruits = selector.selectWithReplacement(fruits, 5,
90     3);
91
92 std::cout << "C-array'den 3 meyve se imi (tekrarlı): ";
93 for (const auto& fruit : selectedFruits) {
94     std::cout << fruit << " ";
95 }
96 std::cout << std::endl;
97
98 return 0;
99 }

```

Listing 24: Array'den tekrarlı random seçim

9.6.2 Tekrarsız Seçim (Without Replacement)

```

1 #include <random>
2 #include <iostream>
3 #include <vector>
4 #include <algorithm>
5 #include <set>
6
7 class UniqueArraySelector {
8 private:
9     std::mt19937 gen;
10
11 public:
12     UniqueArraySelector() : gen(std::random_device{}()) {}
13
14     // Shuffle y ntemi (h zlı , b y k diziler i in ideal)
15     template<typename T>
16     std::vector<T> selectUniqueShuffle(std::vector<T> source,
17         size_t count) {
18         if (count >= source.size()) return source;
19
20         std::shuffle(source.begin(), source.end(), gen);
21         source.resize(count);
22         return source;
23     }
24
25     // Sample y ntemi (C++17, k k se imler i in ideal)
26     template<typename T>

```

```

26     std::vector<T> selectUniqueSample(const std::vector<T>& source,
27                                     size_t count) {
28         if (count >= source.size()) return source;
29
30         std::vector<T> result;
31         result.reserve(count);
32         std::sample(source.begin(), source.end(), std::
33                     back_inserter(result), count, gen);
34         return result;
35     }
36
37     // Index-based se im (orjinal diziyi korur)
38     template<typename T>
39     std::vector<T> selectUniqueByIndex(const std::vector<T>& source
40                                     , size_t count) {
41         if (count >= source.size()) return source;
42
43         std::set<size_t> selectedIndices;
44         std::uniform_int_distribution<> dist(0, source.size() - 1);
45
46         // Unique indeksler se
47         while (selectedIndices.size() < count) {
48             selectedIndices.insert(dist(gen));
49         }
50
51         // Se ilen elementleri topla
52         std::vector<T> result;
53         result.reserve(count);
54         for (size_t index : selectedIndices) {
55             result.push_back(source[index]);
56         }
57
58         return result;
59     }
60
61     // Reservoir sampling (b y k diziler i in memory-efficient)
62     template<typename T>
63     std::vector<T> selectUniqueReservoir(const std::vector<T>&
64                                     source, size_t count) {
65         if (count >= source.size()) return source;
66
67         std::vector<T> reservoir;
68         reservoir.reserve(count);
69
70         // lk k elementi al
71         for (size_t i = 0; i < count && i < source.size(); ++i) {
72             reservoir.push_back(source[i]);
73         }
74
75         // Geri kalan elementler i in reservoir sampling
76         std::uniform_int_distribution<> dist;
77         for (size_t i = count; i < source.size(); ++i) {
78             dist.param(std::uniform_int_distribution<>::param_type
79                       (0, i));
80             size_t j = dist(gen);
81
82             if (j < count) {
83                 reservoir[j] = source[i];
84             }
85         }
86         return reservoir;
87     }

```

```

79         }
80     }
81
82     return reservoir;
83 }
84
85 // A r l k l s e i m
86 template<typename T>
87 std::vector<T> selectWeighted(const std::vector<T>& source,
88                             const std::vector<double>&
89                             weights,
90                             size_t count) {
91     if (source.size() != weights.size() || source.empty())
92         return {};
93
94     std::discrete_distribution<> dist(weights.begin(), weights.
95         end());
96     std::vector<T> result;
97     result.reserve(count);
98
99     for (size_t i = 0; i < count; ++i) {
100         result.push_back(source[dist(gen)]);
101     }
102
103     return result;
104 }
105 };
106
107 int main() {
108     UniqueArraySelector selector;
109
110     std::vector<std::string> students = {
111         "Ali", "Ay e ", "Mehmet", "Fatma", "Ahmet",
112         "Zeynep", "Mustafa", "Elif", " mer ", "Seda"
113     };
114
115     std::cout << "=== Tekrars z Se im Y ntemleri ===" << std::
116         endl;
117
118     // Shuffle y ntemi
119     auto shuffleResult = selector.selectUniqueShuffle(students, 4);
120     std::cout << "Shuffle y ntemi (4 renci ): ";
121     for (const auto& student : shuffleResult) {
122         std::cout << student << " ";
123     }
124     std::cout << std::endl;
125
126     // Sample y ntemi
127     auto sampleResult = selector.selectUniqueSample(students, 3);
128     std::cout << "Sample y ntemi (3 renci ): ";
129     for (const auto& student : sampleResult) {
130         std::cout << student << " ";
131     }
132     std::cout << std::endl;
133
134     // Index-based y ntem
135     auto indexResult = selector.selectUniqueByIndex(students, 5);
136     std::cout << "Index-based (5 renci ): ";

```

```

133     for (const auto& student : indexResult) {
134         std::cout << student << " ";
135     }
136     std::cout << std::endl;
137
138     // Reservoir sampling
139     auto reservoirResult = selector.selectUniqueReservoir(students,
140         3);
141     std::cout << "Reservoir sampling (3 rencia) : ";
142     for (const auto& student : reservoirResult) {
143         std::cout << student << " ";
144     }
145     std::cout << std::endl;
146
147     // A r l k l se im
148     std::vector<double> weights = {0.1, 0.2, 0.1, 0.3, 0.1, 0.2,
149         0.1, 0.4, 0.1, 0.2};
150     auto weightedResult = selector.selectWeighted(students, weights,
151         4);
152     std::cout << "A r l k l se im (4 rencia) : ";
153     for (const auto& student : weightedResult) {
154         std::cout << student << " ";
155     }
156     std::cout << std::endl;
157
158     return 0;
159 }

```

Listing 25: Array'den tekrarsız random seçim

9.6.3 Performance Karşılaştırması

```

1  #include <random>
2  #include <iostream>
3  #include <vector>
4  #include <chrono>
5  #include <algorithm>
6
7  class ArraySelectionBenchmark {
8  private:
9      std::mt19937 gen;
10
11  public:
12      ArraySelectionBenchmark() : gen(std::random_device{}()) {}
13
14      // Benchmark fonksiyonu
15      template<typename Func>
16      double measureTime(Func&& func, int iterations = 1000) {
17          auto start = std::chrono::high_resolution_clock::now();
18
19          for (int i = 0; i < iterations; ++i) {
20              func();
21          }
22
23          auto end = std::chrono::high_resolution_clock::now();
24          auto duration = std::chrono::duration_cast<std::chrono::
25              microseconds>(end - start);

```

```

25     return duration.count() / static_cast<double>(iterations);
26 }
27
28
29 void runBenchmarks() {
30     // Test verisi olu tur
31     std::vector<int> largeArray(10000);
32     std::iota(largeArray.begin(), largeArray.end(), 1);
33
34     const size_t selectCount = 100;
35
36     std::cout << "=== Performance Kar la t rmas ===" <<
37         std::endl;
38     std::cout << "Array boyutu: " << largeArray.size() << std::
39         endl;
40     std::cout << "Se ilecek eleman say s : " << selectCount
41         << std::endl;
42     std::cout << "Her test 1000 kez tekrarland \\n" << std::
43         endl;
44
45     // Shuffle y ntemi
46     auto shuffleTime = measureTime([&]() {
47         auto copy = largeArray;
48         std::shuffle(copy.begin(), copy.end(), gen);
49         copy.resize(selectCount);
50     });
51
52     // Sample y ntemi
53     auto sampleTime = measureTime([&]() {
54         std::vector<int> result;
55         result.reserve(selectCount);
56         std::sample(largeArray.begin(), largeArray.end(),
57             std::back_inserter(result), selectCount, gen
58         );
59     });
60
61     // Index-based y ntem
62     auto indexTime = measureTime([&]() {
63         std::set<size_t> indices;
64         std::uniform_int_distribution<> dist(0, largeArray.size
65             () - 1);
66
67         while (indices.size() < selectCount) {
68             indices.insert(dist(gen));
69         }
70
71         std::vector<int> result;
72         for (size_t idx : indices) {
73             result.push_back(largeArray[idx]);
74         }
75     });
76
77     // Sonu lar yazd r
78     std::cout << "Shuffle y ntemi:      " << shuffleTime << "
79         s " << std::endl;
80     std::cout << "Sample y ntemi:      " << sampleTime << "
81         s " << std::endl;

```



```

74         std::cout << "Index-based yöntem: " << indexTime << " s
           << std::endl;
75
76         std::cout << "\\n===   neriler   ===" << std::endl;
77         std::cout << "           K           k se imler i in: std::sample" <<
           std::endl;
78         std::cout << "           B y k se imler i in: shuffle" << std
           ::endl;
79         std::cout << "           Memory k s t l : index-based" << std::
           endl;
80         std::cout << "           ok b y k diziler: reservoir sampling"
           << std::endl;
81     }
82 };
83
84 int main() {
85     ArraySelectionBenchmark benchmark;
86     benchmark.runBenchmarks();
87
88     return 0;
89 }

```

Listing 26: Array seçim yöntemlerinin performans karşılaştırması

9.7 Random Sayı Üretimi Best Practices

=orange!5!white=orange!75!black=En İyi Uygulamalar

Random kütüphanesini verimli ve güvenli kullanmak için önemli ipuçları.

```

1  #include <random>
2  #include <iostream>
3  #include <thread>
4  #include <chrono>
5
6  class RandomNumberGenerator {
7  private:
8      static thread_local std::mt19937 generator;
9      static bool seeded;
10
11 public:
12     // Thread-safe singleton pattern
13     static std::mt19937& getGenerator() {
14         if (!seeded) {
15             auto seed = std::chrono::high_resolution_clock::now()
16                 .time_since_epoch().count();
17             generator.seed(static_cast<unsigned>(seed));
18             seeded = true;
19         }
20         return generator;
21     }
22
23     // Utility fonksiyonlar
24     static int randInt(int min, int max) {
25         std::uniform_int_distribution<> dist(min, max);

```

```

26     return dist(getGenerator());
27 }
28
29 static double randReal(double min = 0.0, double max = 1.0) {
30     std::uniform_real_distribution<> dist(min, max);
31     return dist(getGenerator());
32 }
33
34 static bool randBool(double probability = 0.5) {
35     std::bernoulli_distribution dist(probability);
36     return dist(getGenerator());
37 }
38 };
39
40 // Static member tan mlar
41 thread_local std::mt19937 RandomNumberGenerator::generator;
42 bool RandomNumberGenerator::seeded = false;
43
44 int main() {
45     // G venli ve kolay kullan m
46     std::cout << "Random int (1-10): " << RandomNumberGenerator::
47         randInt(1, 10) << std::endl;
48     std::cout << "Random double: " << RandomNumberGenerator::
49         randReal() << std::endl;
50     std::cout << "Random bool: " << RandomNumberGenerator::randBool
51         (0.7) << std::endl;
52
53     return 0;
54 }

```

Listing 27: Best practices örneği

Önemli Kurallar:

- `std::random_device` sadece seeding için kullanın
- `std::mt19937` gibi kaliteli generatorları tercih edin
- Generator ve distribution objelerini yeniden kullanın
- Thread safety için `thread_local` kullanın
- Eski `rand()` fonksiyonunu kullanmayın

10 Performans İpuçları ve En İyi Uygulamalar

10.1 Hız Optimizasyonu

=gray!10!white=gray!75!black=Performans İpuçları

Math kütüphanesi fonksiyonlarını verimli kullanmak için önemli ipuçları.

- **Gereksiz hesaplamalardan kaçının:** Döngü içinde sabit değerler için matematik fonksiyonlarını her seferinde çağırmayın.

- **Uygun veri türünü seçin:** Float ve double arasında performans farkı olabilir.
- **Compiler optimizasyonlarını kullanın:** -O2 veya -O3 bayraklarını kullanın.
- **Özel durumları kontrol edin:** NaN ve infinity değerleri için kontrol yapın.

```

1 #include <cmath>
2 #include <iostream>
3 #include <chrono>
4 #include <vector>
5
6 // Yava versiyon - her seferinde hesaplama
7 double slowCalculation(const std::vector<double>& data) {
8     double sum = 0.0;
9     for (size_t i = 0; i < data.size(); ++i) {
10         sum += std::pow(data[i], 2) / std::sqrt(data.size()); //
11         // Her seferinde sqrt hesaplan r
12     }
13     return sum;
14 }
15
16 // H z l versiyon - nceden hesaplama
17 double fastCalculation(const std::vector<double>& data) {
18     double sum = 0.0;
19     double sqrtSize = std::sqrt(data.size()); // Bir kez hesapla
20     for (size_t i = 0; i < data.size(); ++i) {
21         sum += (data[i] * data[i]) / sqrtSize; // pow yerine
22         // arpm
23     }
24     return sum;
25 }

```

Listing 28: Performans optimizasyonu örneği

11 Hata Yönetimi ve Debugging

11.1 Matematiksel Hatalar

```

1 #include <cmath>
2 #include <iostream>
3 #include <cerrno>
4 #include <cfenv>
5
6 void checkMathError(const std::string& operation) {
7     if (errno == EDOM) {
8         std::cout << "Domain error in " << operation << std::endl;
9         errno = 0;
10    } else if (errno == ERANGE) {
11        std::cout << "Range error in " << operation << std::endl;
12        errno = 0;
13    }
14 }
15
16 int main() {
17     // Domain error   rnei

```

```

18     double result1 = std::sqrt(-1.0);
19     checkMathError("sqrt(-1)");
20     if (std::isnan(result1)) {
21         std::cout << "sqrt(-1) returned NaN" << std::endl;
22     }
23
24     // Range error    rnei
25     double result2 = std::exp(1000.0);
26     checkMathError("exp(1000)");
27     if (std::isinf(result2)) {
28         std::cout << "exp(1000) returned infinity" << std::endl;
29     }
30
31     return 0;
32 }

```

Listing 29: Hata yönetimi

12 C++17 ve Sonrası Yeni Özellikler

12.1 Özel Matematik Fonksiyonları

C++17 ile birlikte gelen özel matematik fonksiyonları:

```

1  #include <cmath>
2  #include <iostream>
3
4  #if __cplusplus >= 201703L
5  int main() {
6      double x = 2.0;
7      int n = 3;
8
9      // Bessel fonksiyonlar
10     std::cout << "Bessel J0(" << x << ") = " << std::cyl_bessel_j
11         (0, x) << std::endl;
12     std::cout << "Bessel Y0(" << x << ") = " << std::cyl_neumann(0,
13         x) << std::endl;
14
15     // Legendre polinomlar
16     std::cout << "Legendre P" << n << "(" << 0.5 << ") = "
17         << std::legendre(n, 0.5) << std::endl;
18
19     // Laguerre polinomlar
20     std::cout << "Laguerre L" << n << "(" << x << ") = "
21         << std::laguerre(n, x) << std::endl;
22
23     return 0;
24 }
25 #else
26 int main() {
27     std::cout << "C++17 required for special math functions" << std
28         ::endl;
29     return 0;
30 }
31 #endif

```

Listing 30: C++17 özel matematik fonksiyonları

13 Sonuç ve Özet

C++ math ve random kütüphaneleri, matematiksel hesaplamalar ve rastgele sayı üretimi için kapsamlı ve güçlü araç setleri sunar. Bu rehberde ele alınan konular:

- Temel matematik fonksiyonları (mutlak değer, üs, kök)
- Trigonometrik ve hiperbolik fonksiyonlar
- Logaritmik işlemler
- Yuvarlama ve modulo işlemleri
- Özel matematik fonksiyonları
- Sayı özelliklerini kontrol fonksiyonları
- Modern random sayı üretimi ve dağılımları
- Belirli aralıklarda random sayı üretimi teknikleri
- Monte Carlo simülasyonları ve random walk uygulamaları
- Performans optimizasyonu ipuçları
- Hata yönetimi teknikleri

13.1 Referanslar ve Kaynaklar

- C++ Standard Library Documentation
- ISO C++17 Standard (ISO/IEC 14882:2017)
- cppreference.com matematik ve random fonksiyonları bölümü
- Numerical Recipes in C++ (Press, Teukolsky, Vetterling, Flannery)