C++ Math & Random Library Rehberi

Kapsamlı Matematiksel ve Random Fonksiyonlar Kılavuzu

C++ Programcıları için Detaylı Referans

8 Eylül 2025

İçindekiler

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

```cpp
#include <cmath>
#include <random>
#include <iostream>

int main() {
    // Math ve Random fonksiyonlar  art k  kullan labilir
    return 0;
}
```

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.

```cpp
#include <cmath>
#include <iostream>

int main() {
    // Farkl  veri t rleri i in mutlak de er
    int a = -5;
    float b = -3.14f;
    double c = -2.71828;

    std::cout << "abs(" << a << ") = " << std::abs(a) << std::endl;
    std::cout << "fabs(" << b << ") = " << std::fabs(b) << std::endl;
    std::cout << "fabs(" << c << ") = " << std::fabs(c) << std::endl;

    return 0;
}
```

Listing 2: Mutlak değer fonksiyonları

Fonksiyon Çeşitleri:

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

```cpp
#include <cmath>
#include <iostream>

int main() {
    double x = 16.0;
    double y = 2.0;

    //  s  i lemleri
    std::cout << "pow(" << x << ", " << y << ") = " << std::pow(x, y) << std::endl;
    std::cout << "exp(" << y << ") = " << std::exp(y) << std::endl;
    std::cout << "exp2(" << y << ") = " << std::exp2(y) << std::endl;

    // K k i lemleri
    std::cout << "sqrt(" << x << ") = " << std::sqrt(x) << std::endl;
    std::cout << "cbrt(" << x << ") = " << std::cbrt(x) << std::endl;
    std::cout << "hypot(3, 4) = " << std::hypot(3, 4) << std::endl;

    return 0;
}
```

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).

```
#include <cmath>
#include <iostream>

int main() {
    const double PI = std::acos(-1);
    double angle = PI / 4; // 45 derece

    // Temel trigonometrik fonksiyonlar
    std::cout << "sin(" << angle << ") = " << std::sin(angle) <<
        std::endl;
    std::cout << "cos(" << angle << ") = " << std::cos(angle) <<
        std::endl;
    std::cout << "tan(" << angle << ") = " << std::tan(angle) <<
        std::endl;

    // Ters trigonometrik fonksiyonlar
    double value = 0.707;
    std::cout << "asin(" << value << ") = " << std::asin(value) <<
        std::endl;
    std::cout << "acos(" << value << ") = " << std::acos(value) <<
        std::endl;
    std::cout << "atan(" << value << ") = " << std::atan(value) <<
        std::endl;
    std::cout << "atan2(1, 1) = " << std::atan2(1, 1) << std::endl;

    return 0;
}
```

Listing 4: Trigonometrik fonksiyonlar

3.2 Hiperbolik Fonksiyonlar

```
#include <cmath>
#include <iostream>

int main() {
    double x = 1.0;

    // Hiperbolik fonksiyonlar
    std::cout << "sinh(" << x << ") = " << std::sinh(x) << std::
        endl;
    std::cout << "cosh(" << x << ") = " << std::cosh(x) << std::
        endl;
    std::cout << "tanh(" << x << ") = " << std::tanh(x) << std::
        endl;

    // Ters hiperbolik fonksiyonlar
    std::cout << "asinh(" << x << ") = " << std::asinh(x) << std::
        endl;
    std::cout << "acosh(" << x + 1 << ") = " << std::acosh(x + 1)
        << std::endl;
    std::cout << "atanh(" << x/2 << ") = " << std::atanh(x/2) <<
        std::endl;

    return 0;
}
```

Listing 5: Hiperbolik fonksiyonlar

4 Logaritmik Fonksiyonlar

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

Farklı tabanlarda logaritma hesaplama fonksiyonları.

```
#include <cmath>
#include <iostream>

int main() {
    double x = 100.0;

    // Farkl  tabanlarda logaritma
    std::cout << "log(" << x << ") = " << std::log(x) << std::endl;
             // ln(x)
    std::cout << "log10(" << x << ") = " << std::log10(x) << std::
        endl;  // log_10(x)
    std::cout << "log2(" << x << ") = " << std::log2(x) << std::
        endl;    // log_2(x)

    //  zel  logaritma fonksiyonlar
    std::cout << "log1p(" << 0.1 << ") = " << std::log1p(0.1) <<
        std::endl; // ln(1+x)
    std::cout << "logb(" << x << ", 3) = " << std::logb(x) << std::
        endl;     // log_r(x)

    return 0;
}
```

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.

```
#include <cmath>
#include <iostream>

int main() {
    double x = 3.7;
    double y = -3.7;

    std::cout << "Pozitif say : " << x << std::endl;
    std::cout << "ceil(" << x << ") = " << std::ceil(x) << std::endl;     // Yukar  yuvarla
    std::cout << "floor(" << x << ") = " << std::floor(x) << std::endl;   // A  a    yuvarla
    std::cout << "round(" << x << ") = " << std::round(x) << std::endl;   // En yak na yuvarla
    std::cout << "trunc(" << x << ") = " << std::trunc(x) << std::endl;   // Kesirli k s m  at

    std::cout << "\\nNegatif say : " << y << std::endl;
    std::cout << "ceil(" << y << ") = " << std::ceil(y) << std::endl;
    std::cout << "floor(" << y << ") = " << std::floor(y) << std::endl;
    std::cout << "round(" << y << ") = " << std::round(y) << std::endl;
    std::cout << "trunc(" << y << ") = " << std::trunc(y) << std::endl;

    return 0;
}
```

Listing 7: Yuvarlama fonksiyonları

5.2 Modulo ve Kalan İşlemleri

```
#include <cmath>
#include <iostream>

int main() {
    double x = 7.5;
    double y = 2.3;

    // Kalan i lemleri
    std::cout << "fmod(" << x << ", " << y << ") = " << std::fmod(x, y) << std::endl;
    std::cout << "remainder(" << x << ", " << y << ") = " << std::remainder(x, y) << std::endl;

    // B l m ve kalan
    int quotient;
    double rem = std::remquo(x, y, &quotient);
    std::cout << "remquo(" << x << ", " << y << ") = " << rem
              << ", quotient = " << quotient << std::endl;

    return 0;
}
```

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.

```cpp
#include <cmath>
#include <iostream>

int main() {
    double x = 5.0;

    // Gamma fonksiyonlar  (C++17 ve sonras )
    #if __cplusplus >= 201703L
    std::cout << "tgamma(" << x << ") = " << std::tgamma(x) << std::endl;        //   (x)
    std::cout << "lgamma(" << x << ") = " << std::lgamma(x) << std::endl;        // ln|  (x)|
    #endif

    // Manuel fakt riyel hesaplama
    auto factorial = [](int n) -> long long {
        if (n <= 1) return 1;
        return n * factorial(n - 1);
    };

    std::cout << "5! = " << factorial(5) << std::endl;

    return 0;
}
```

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.

```cpp
#include <cmath>
#include <iostream>
#include <limits>

int main() {
    double finite_num = 42.0;
    double inf_num = std::numeric_limits<double>::infinity();
    double nan_num = std::numeric_limits<double>::quiet_NaN();

    // Say  zelliklerini  kontrol et
    std::cout << "Sonlu say  kontrol :" << std::endl;
    std::cout << "isfinite(" << finite_num << ") = " << std::
        isfinite(finite_num) << std::endl;
    std::cout << "isinf(" << finite_num << ") = " << std::isinf(
        finite_num) << std::endl;
    std::cout << "isnan(" << finite_num << ") = " << std::isnan(
        finite_num) << std::endl;

    std::cout << "\\nSonsuz say  kontrol :" << std::endl;
    std::cout << "isfinite(inf) = " << std::isfinite(inf_num) <<
        std::endl;
    std::cout << "isinf(inf) = " << std::isinf(inf_num) << std::
        endl;

    std::cout << "\\nNaN kontrol :" << std::endl;
    std::cout << "isnan(NaN) = " << std::isnan(nan_num) << std::
        endl;

    //  aret  kontrol
    std::cout << "\\n  aret kontrol :" << std::endl;
    std::cout << "signbit(-3.14) = " << std::signbit(-3.14) << std
        ::endl;
    std::cout << "signbit(3.14) = " << std::signbit(3.14) << std::
        endl;

    return 0;
}
```

Listing 10: Sayı sınıflandırma fonksiyonları

8 Pratik Örnekler ve Kullanım Alanları

8.1 Geometrik Hesaplamalar

```
#include <cmath>
#include <iostream>

class GeometryCalculator {
public:
    // Daire alan  hesaplama
    static double circleArea(double radius) {
        const double PI = std::acos(-1);
        return PI * std::pow(radius, 2);
    }

    //  ki nokta aras  mesafe
    static double distance(double x1, double y1, double x2, double
        y2) {
        return std::hypot(x2 - x1, y2 - y1);
    }

    //   gen  alan  (Heron form l )
    static double triangleArea(double a, double b, double c) {
        double s = (a + b + c) / 2; // Yar   evre
        return std::sqrt(s * (s - a) * (s - b) * (s - c));
```

```cpp
    }
};

int main() {
    double radius = 5.0;
    std::cout << "Daire alan  (r=" << radius << "): "
              << GeometryCalculator::circleArea(radius) << std::endl;

    double x1 = 0, y1 = 0, x2 = 3, y2 = 4;
    std::cout << "Nokta mesafesi: "
              << GeometryCalculator::distance(x1, y1, x2, y2) << std::endl;

    double a = 3, b = 4, c = 5;
    std::cout << "  gen   alan : "
              << GeometryCalculator::triangleArea(a, b, c) << std::endl;

    return 0;
}
```

Listing 11: Geometrik hesaplama örneği

8.2 İstatistiksel Hesaplamalar

```cpp
#include <cmath>
#include <iostream>
#include <vector>
#include <numeric>

class StatisticsCalculator {
public:
    // Aritmetik ortalama
    static double mean(const std::vector<double>& data) {
        return std::accumulate(data.begin(), data.end(), 0.0) / data.size();
    }

    // Standart sapma
    static double standardDeviation(const std::vector<double>& data) {
        double avg = mean(data);
        double sum = 0.0;

        for (double value : data) {
            sum += std::pow(value - avg, 2);
        }

        return std::sqrt(sum / data.size());
    }

    // Varyans
    static double variance(const std::vector<double>& data) {
        double stdDev = standardDeviation(data);
        return std::pow(stdDev, 2);
    }
};

int main() {
    std::vector<double> data = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0};

    std::cout << "Veri seti: ";
    for (double value : data) {
        std::cout << value << " ";
    }
    std::cout << std::endl;

    std::cout << "Ortalama: " << StatisticsCalculator::mean(data) << std::endl;
    std::cout << "Standart sapma: " << StatisticsCalculator::standardDeviation(data) << std::endl;
    std::cout << "Varyans: " << StatisticsCalculator::variance(data) << std::endl;

    return 0;
}
```

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.

```cpp
#include <random>
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    // Modern random say  retimi  art k kullan labilir
    return 0;
}
```

Listing 13: Random kütüphanesini dahil etme

9.2 Random Sayı Motorları (Engines)

```cpp
#include <random>
#include <iostream>
#include <chrono>

```

```
int main() {
    // Farkl  random motorlar
    std::default_random_engine engine1;
    std::mt19937 engine2;              // Mersenne Twister
    std::mt19937_64 engine3;           // 64-bit Mersenne Twister
    std::ranlux24 engine4;             // RANLUX generator

    // Seed ayarlama
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::mt19937 generator(seed);

    // Basit say   retimi
    for (int i = 0; i < 5; ++i) {
        std::cout << "Random: " << generator() << std::endl;
    }

    return 0;
}
```

Listing 14: Farklı random motorları

9.3 Dağılım Fonksiyonları (Distributions)

9.3.1 Tam Sayı Dağılımları

```
#include <random>
#include <iostream>

int main() {
    std::random_device rd;
    std::mt19937 gen(rd());

    // Uniform da  l m (1-6 aras  zar at    )
    std::uniform_int_distribution<> dice(1, 6);
    std::cout << "Zar at  lar : ";
    for (int i = 0; i < 10; ++i) {
        std::cout << dice(gen) << " ";
    }
    std::cout << std::endl;

    // Bernoulli da  l m  (coin flip)
    std::bernoulli_distribution coin(0.5);
    std::cout << "Coin flips: ";
    for (int i = 0; i < 10; ++i) {
        std::cout << (coin(gen) ? "H" : "T") << " ";
    }
    std::cout << std::endl;

    // Binomial da  l m
    std::binomial_distribution<> binomial(10, 0.3);
    std::cout << "Binomial (n=10, p=0.3): " << binomial(gen) << std::endl;

    // Geometric da  l m
    std::geometric_distribution<> geometric(0.2);
    std::cout << "Geometric (p=0.2): " << geometric(gen) << std::endl;

    return 0;
}
```

Listing 15: Tam sayı dağılımları

9.3.2 Ondalıklı Sayı Dağılımları

```
#include <random>
#include <iostream>
#include <iomanip>

int main() {
    std::random_device rd;
    std::mt19937 gen(rd());

    // Uniform da   l  m (0.0 - 1.0 aras )
    std::uniform_real_distribution<> uniform(0.0, 1.0);
    std::cout << std::fixed << std::setprecision(4);
    std::cout << "Uniform da   l  m: ";
    for (int i = 0; i < 5; ++i) {
        std::cout << uniform(gen) << " ";
    }
    std::cout << std::endl;

    // Normal (Gaussian) da   l  m
    std::normal_distribution<> normal(0.0, 1.0); // mean=0, stddev=1
    std::cout << "Normal da   l  m: ";
    for (int i = 0; i < 5; ++i) {
        std::cout << normal(gen) << " ";
    }
    std::cout << std::endl;

    // Exponential da   l  m
    std::exponential_distribution<> exponential(1.0);
    std::cout << "Exponential da   l  m: ";
    for (int i = 0; i < 5; ++i) {
        std::cout << exponential(gen) << " ";
    }
    std::cout << std::endl;

    // Gamma da   l  m
    std::gamma_distribution<> gamma(2.0, 1.0);
    std::cout << "Gamma da   l  m: ";
    for (int i = 0; i < 5; ++i) {
        std::cout << gamma(gen) << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Listing 16: Ondalıklı sayı dağılımları

9.4 Pratik Random Uygulamaları

9.4.1 Dizi Karıştırma ve Örnekleme

```cpp
#include <random>
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::random_device rd;
    std::mt19937 g(rd());

    // Dizi kar  t rma
    std::vector<int> deck = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    std::cout << "Orijinal: ";
    for (int card : deck) std::cout << card << " ";
    std::cout << std::endl;

    std::shuffle(deck.begin(), deck.end(), g);
    std::cout << "Kar  t r lm  : ";
    for (int card : deck) std::cout << card << " ";
    std::cout << std::endl;

    // Random  rnekleme
    std::vector<int> sample(3);
    std::sample(deck.begin(), deck.end(), sample.begin(), 3, g);
    std::cout << "3 eleman  rnei  : ";
    for (int elem : sample) std::cout << elem << " ";
    std::cout << std::endl;

    return 0;
}
```

Listing 17: Dizi işlemleri

9.4.2 Monte Carlo Simülasyonu

```cpp
#include <random>
#include <iostream>
#include <cmath>

class MonteCarloSimulation {
public:
    static double estimatePi(int numSamples) {
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_real_distribution<> dis(-1.0, 1.0);

        int insideCircle = 0;

        for (int i = 0; i < numSamples; ++i) {
            double x = dis(gen);
            double y = dis(gen);

            if (x*x + y*y <= 1.0) {
                insideCircle++;
            }
        }

        return 4.0 * insideCircle / numSamples;
    }
};

int main() {
    std::vector<int> sampleSizes = {1000, 10000, 100000, 1000000};

    for (int samples : sampleSizes) {
        double piEstimate = MonteCarloSimulation::estimatePi(
           samples);
        double error = std::abs(piEstimate - M_PI);

        std::cout << " rneklem : " << samples
                  << ", Pi tahmini: " << piEstimate
                  << ", Hata: " << error << std::endl;
    }

    return 0;
}
```

Listing 18: Monte Carlo Pi hesaplama

9.4.3 Random Walk Simülasyonu

```
#include <random>
#include <iostream>
#include <vector>
#include <cmath>

class RandomWalk {
private:
    double x, y;
    std::mt19937 gen;
    std::uniform_real_distribution<> angleDist;
    std::uniform_real_distribution<> stepDist;

public:
    RandomWalk() : x(0.0), y(0.0), gen(std::random_device{}()),
                   angleDist(0.0, 2 * M_PI), stepDist(0.5, 1.5) {}

    void step() {
        double angle = angleDist(gen);
        double stepSize = stepDist(gen);

        x += stepSize * std::cos(angle);
    void step() {
        double angle = angleDist(gen);
        double stepSize = stepDist(gen);

        x += stepSize * std::cos(angle);
        y += stepSize * std::sin(angle);
    }

    double distanceFromOrigin() const {
        return std::sqrt(x*x + y*y);
    }

    void reset() { x = 0.0; y = 0.0; }

    std::pair<double, double> getPosition() const { return {x, y};
       }
};

int main() {
    RandomWalk walker;
    int steps = 1000;

    std::cout << "Random Walk Sim lasyonu (" << steps << " ad m):
       " << std::endl;

    for (int i = 0; i < steps; i += 100) {
        auto [x, y] = walker.getPosition();
        double distance = walker.distanceFromOrigin();

        std::cout << "Ad m " << i << ": (" << x << ", " << y
                  << "), Mesafe: " << distance << std::endl;

        for (int j = 0; j < 100; ++j) {
            walker.step();
        }
    }

    auto [finalX, finalY] = walker.getPosition();
    std::cout << "Final pozisyon: (" << finalX << ", " << finalY
              << "), Final mesafe: " << walker.distanceFromOrigin()
                  << std::endl;

    return 0;
}
```

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

```
#include <random>
#include <iostream>
#include <vector>

class IntegerRangeGenerator {
private:
    std::mt19937 gen;

public:
```

```cpp
    IntegerRangeGenerator() : gen(std::random_device{}()) {}

    // Basit aral k [min, max]
    int generate(int min, int max) {
        std::uniform_int_distribution<> dist(min, max);
        return dist(gen);
    }

    //  ift say lar aral
    int generateEven(int min, int max) {
        // Min ve max'  ift yapmak i in ayarlama
        if (min % 2 != 0) min++;
        if (max % 2 != 0) max--;

        if (min > max) return min; // Ge ersiz aral k

        int evenCount = (max - min) / 2 + 1;
        std::uniform_int_distribution<> dist(0, evenCount - 1);
        return min + dist(gen) * 2;
    }

    // Tek say lar aral
    int generateOdd(int min, int max) {
        // Min ve max'  tek yapmak i in ayarlama
        if (min % 2 == 0) min++;
        if (max % 2 == 0) max--;

        if (min > max) return min; // Ge ersiz aral k

        int oddCount = (max - min) / 2 + 1;
        std::uniform_int_distribution<> dist(0, oddCount - 1);
        return min + dist(gen) * 2;
    }

    // Katlar ( rn : 5'in katlar )
    int generateMultiple(int min, int max, int multiple) {
        if (multiple <= 0) return min;

        int adjustedMin = ((min + multiple - 1) / multiple) * multiple;
        int adjustedMax = (max / multiple) * multiple;

        if (adjustedMin > adjustedMax) return adjustedMin;

        int count = (adjustedMax - adjustedMin) / multiple + 1;
        std::uniform_int_distribution<> dist(0, count - 1);
        return adjustedMin + dist(gen) * multiple;
    }
};

int main() {
    IntegerRangeGenerator generator;

    std::cout << "=== Tam Say  Aral k  rnekleri  ===" << std::endl;

    // Normal aral k
    std::cout << "1-100 aras  10 say : ";
    for (int i = 0; i < 10; ++i) {
        std::cout << generator.generate(1, 100) << " ";
    }
    std::cout << std::endl;

    //  ift  say lar
    std::cout << "10-50 aras   ift  say lar: ";
    for (int i = 0; i < 10; ++i) {
        std::cout << generator.generateEven(10, 50) << " ";
    }
    std::cout << std::endl;

    // Tek say lar
    std::cout << "1-25 aras  tek say lar: ";
    for (int i = 0; i < 10; ++i) {
        std::cout << generator.generateOdd(1, 25) << " ";
    }
    std::cout << std::endl;

    // 5'in katlar
    std::cout << "10-100 aras  5'in katlar : ";
    for (int i = 0; i < 10; ++i) {
        std::cout << generator.generateMultiple(10, 100, 5) << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Listing 20: Farklı tam sayı aralıkları

9.5.2 Ondalıklı Sayı Aralıkları

```
#include <random>
#include <iostream>
#include <iomanip>
#include <cmath>

class FloatRangeGenerator {
private:
    std::mt19937 gen;

public:
    FloatRangeGenerator() : gen(std::random_device{}()) {}

    // Basit ondal kl  aral k
    double generate(double min, double max) {
        std::uniform_real_distribution<> dist(min, max);
        return dist(gen);
    }

    // Belirli hassasiyetle ( rn : 2 ondal k basamak)
    double generateWithPrecision(double min, double max, int
        precision) {
        double multiplier = std::pow(10, precision);
        int intMin = static_cast<int>(min * multiplier);
        int intMax = static_cast<int>(max * multiplier);

        std::uniform_int_distribution<> dist(intMin, intMax);
        return dist(gen) / multiplier;
    }

    // Belirli ad mlarla ( rn : 0.5 ad mlarla)
    double generateWithStep(double min, double max, double step) {
        if (step <= 0) return min;

        int steps = static_cast<int>((max - min) / step);
        std::uniform_int_distribution<> dist(0, steps);
        return min + dist(gen) * step;
    }

    // Logaritmik aral k
    double generateLogarithmic(double min, double max) {
        if (min <= 0 || max <= 0) return 1.0;

        double logMin = std::log(min);
        double logMax = std::log(max);
        std::uniform_real_distribution<> dist(logMin, logMax);
        return std::exp(dist(gen));
    }
};

int main() {
    FloatRangeGenerator generator;

    std::cout << std::fixed << std::setprecision(3);
    std::cout << "=== Ondal kl  Say  Aral k  rnekleri  ===" <<
        std::endl;

    // Normal aral k
    std::cout << "0.0-1.0 aras : ";
    for (int i = 0; i < 5; ++i) {
        std::cout << generator.generate(0.0, 1.0) << " ";
    }
    std::cout << std::endl;

    // Hassasiyet kontrol
    std::cout << "1.0-10.0 aras  (2 basamak): ";
    for (int i = 0; i < 5; ++i) {
        std::cout << generator.generateWithPrecision(1.0, 10.0, 2)
            << " ";
    }
    std::cout << std::endl;

    // Ad m kontrol
    std::cout << "0.0-5.0 aras  (0.5 ad m): ";
    for (int i = 0; i < 5; ++i) {
        std::cout << generator.generateWithStep(0.0, 5.0, 0.5) << "
             ";
    }
    std::cout << std::endl;

    // Logaritmik
    std::cout << "1-1000 aras  logaritmik: ";
    for (int i = 0; i < 5; ++i) {
```

```cpp
        std::cout << generator.generateLogarithmic(1.0, 1000.0) << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Listing 21: Ondalıklı sayı aralıkları ve hassasiyet

9.5.3 Özel Aralık Türleri

```cpp
#include <random>
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>

class SpecialRangeGenerator {
private:
    std::mt19937 gen;

public:
    SpecialRangeGenerator() : gen(std::random_device{}()) {}

    // Belirli say lar hari tutarak
    int generateExcluding(int min, int max, const std::set<int>& excluded) {
        std::vector<int> validNumbers;
        for (int i = min; i <= max; ++i) {
            if (excluded.find(i) == excluded.end()) {
                validNumbers.push_back(i);
            }
        }

        if (validNumbers.empty()) return min;

        std::uniform_int_distribution<> dist(0, validNumbers.size() - 1);
        return validNumbers[dist(gen)];
    }

    // Sadece belirli say lar dahil ederek
    int generateFromSet(const std::vector<int>& allowedNumbers) {
        if (allowedNumbers.empty()) return 0;

        std::uniform_int_distribution<> dist(0, allowedNumbers.size() - 1);
        return allowedNumbers[dist(gen)];
    }

    // Gaussian da l m ile s n rl aral k
    double generateGaussianInRange(double min, double max, double mean, double stddev) {
        std::normal_distribution<> dist(mean, stddev);
        double value;

        // Aral k d   n a   kana   kadar tekrar dene
        int attempts = 0;
        do {
            value = dist(gen);
            attempts++;
            if (attempts > 1000) { // Sonsuz d ng   nleme
                return (min + max) / 2.0;
            }
        } while (value < min || value > max);

        return value;
    }

    // A  rl kl  aral k se imi
    int generateWeighted(const std::vector<std::pair<std::pair<int,
        int>, double>>& weightedRanges) {
        if (weightedRanges.empty()) return 0;

        // A  rl klar  topla
        double totalWeight = 0.0;
        for (const auto& range : weightedRanges) {
            totalWeight += range.second;
        }

        // Random weight se
        std::uniform_real_distribution<> weightDist(0.0,
           totalWeight);
        double selectedWeight = weightDist(gen);

        // Hangi aral   n se ildi ini bul
        double currentWeight = 0.0;
        for (const auto& range : weightedRanges) {
            currentWeight += range.second;
            if (selectedWeight <= currentWeight) {
                // Bu aral ktan say   ret
                std::uniform_int_distribution<> rangeDist(range.
                   first.first, range.first.second);
                return rangeDist(gen);
            }
        }

        return weightedRanges[0].first.first; // Fallback
    }
};

int main() {
    SpecialRangeGenerator generator;

    std::cout << "===  zel  Aral k T rleri ===" << std::endl;

    // Hari  tutma
    std::set<int> excluded = {13, 666, 777};
    std::cout << "1-20 aras  (13 hari ): ";
    for (int i = 0; i < 10; ++i) {
        std::cout << generator.generateExcluding(1, 20, excluded)
           << " ";
    }
    std::cout << std::endl;

    // Belirli k meden se im
    std::vector<int> primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
       31};
    std::cout << "Asal say lardan: ";
    for (int i = 0; i < 10; ++i) {
        std::cout << generator.generateFromSet(primes) << " ";
    }
    std::cout << std::endl;

    // Gaussian s n rl
    std::cout << std::fixed << std::setprecision(2);
    std::cout << "Gaussian (0-10, mean=5, std=1.5): ";
    for (int i = 0; i < 10; ++i) {
        std::cout << generator.generateGaussianInRange(0.0, 10.0,
           5.0, 1.5) << " ";
    }
    std::cout << std::endl;

    // A  rl kl  aral klar
    std::vector<std::pair<std::pair<int, int>, double>>
       weightedRanges = {
        {{1, 10}, 0.1},     // %10  ans
        {{11, 50}, 0.6},    // %60  ans
        {{51, 100}, 0.3}    // %30  ans
    };

    std::cout << " A  rl kl  aral klar: ";
    for (int i = 0; i < 15; ++i) {
        std::cout << generator.generateWeighted(weightedRanges) <<
           " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Listing 22: Özel aralık türleri ve filtreleme

9.5.4 Performanslı Aralık Üretimi

```
#include <random>
#include <iostream>
#include <vector>
#include <chrono>

class OptimizedRangeGenerator {
private:
    mutable std::mt19937 gen;

public:
    OptimizedRangeGenerator() : gen(std::random_device{}()) {}

    // Batch  retim - tek seferde birden fazla say
    std::vector<int> generateBatch(int min, int max, size_t count)
       {
        std::uniform_int_distribution<> dist(min, max);
        std::vector<int> result;
        result.reserve(count);

        for (size_t i = 0; i < count; ++i) {
            result.push_back(dist(gen));
        }

        return result;
    }

    // Unique say lar  retimi  (tekrars z)
    std::vector<int> generateUnique(int min, int max, size_t count)
        {
        if (count > static_cast<size_t>(max - min + 1)) {
            count = max - min + 1; // Max m mk n say  kadar
        }

        std::vector<int> range;
        for (int i = min; i <= max; ++i) {
            range.push_back(i);
        }

        std::shuffle(range.begin(), range.end(), gen);
        range.resize(count);
        return range;
    }

    // H zl  boolean array i in
    std::vector<bool> generateBoolArray(size_t size, double
       trueRatio = 0.5) {
        std::bernoulli_distribution dist(trueRatio);
        std::vector<bool> result;
        result.reserve(size);

        for (size_t i = 0; i < size; ++i) {
            result.push_back(dist(gen));
        }

        return result;
    }

    // Cached distribution kullan m
    class CachedIntGenerator {
    private:
        std::uniform_int_distribution<> dist;
        std::mt19937& gen_ref;

    public:
        CachedIntGenerator(int min, int max, std::mt19937& gen)
            : dist(min, max), gen_ref(gen) {}

        int operator()() {
            return dist(gen_ref);
        }
    };

    CachedIntGenerator getCachedGenerator(int min, int max) {
        return CachedIntGenerator(min, max, gen);
```

```cpp
    }
};

// Performance test fonksiyonu
void performanceTest() {
    OptimizedRangeGenerator generator;
    const int iterations = 1000000;

    auto start = std::chrono::high_resolution_clock::now();

    // Normal  retim
    for (int i = 0; i < iterations; ++i) {
        std::uniform_int_distribution<> dist(1, 100);
        static std::mt19937 gen(std::random_device{}());
        volatile int result = dist(gen); // volatile to prevent optimization
        (void)result; // Suppress unused variable warning
    }

    auto end = std::chrono::high_resolution_clock::now();
    auto duration1 = std::chrono::duration_cast<std::chrono::microseconds>(end - start);

    start = std::chrono::high_resolution_clock::now();

    // Cached  retim
    auto cachedGen = generator.getCachedGenerator(1, 100);
    for (int i = 0; i < iterations; ++i) {
        volatile int result = cachedGen();
        (void)result;
    }

    end = std::chrono::high_resolution_clock::now();
    auto duration2 = std::chrono::duration_cast<std::chrono::microseconds>(end - start);

    std::cout << "Normal  retim : " << duration1.count() << "  s " << std::endl;
    std::cout << "Cached  retim : " << duration2.count() << "  s " << std::endl;
    std::cout << "H zlanma: " << static_cast<double>(duration1.count()) / duration2.count() << "x" << std::endl;
}

int main() {
    OptimizedRangeGenerator generator;

    std::cout << "=== Performansl  Aral k  retimi  ===" << std::endl;

    // Batch  retim
    auto batch = generator.generateBatch(1, 100, 10);
    std::cout << "Batch (10 say ): ";
    for (int num : batch) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    // Unique say lar
    auto unique = generator.generateUnique(1, 20, 8);
    std::cout << "Unique say lar: ";
    for (int num : unique) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    // Boolean array
    auto bools = generator.generateBoolArray(15, 0.3);
    std::cout << "Boolean array: ";
    for (bool b : bools) {
        std::cout << (b ? "T" : "F") << " ";
    }
    std::cout << std::endl;

    // Performance test
    std::cout << "\\n=== Performance Kar la t rmas ===" <<
       std::endl;
    performanceTest();

    return 0;
}
```

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)

```
#include <random>
#include <iostream>
#include <vector>
#include <array>

class ArrayRandomSelector {
private:
    std::mt19937 gen;

public:
    ArrayRandomSelector() : gen(std::random_device{}()) {}

    // Vector'den tekrarl se im
    template<typename T>
    std::vector<T> selectWithReplacement(const std::vector<T>&
       source, size_t count) {
        if (source.empty()) return {};

        std::uniform_int_distribution<> dist(0, source.size() - 1);
        std::vector<T> result;
        result.reserve(count);
```

```cpp

        for (size_t i = 0; i < count; ++i) {
            result.push_back(source[dist(gen)]);
        }

        return result;
    }

    // Array'den tekrarl  se im
    template<typename T, size_t N>
    std::vector<T> selectWithReplacement(const std::array<T, N>& source, size_t count) {
        if (N == 0) return {};

        std::uniform_int_distribution<> dist(0, N - 1);
        std::vector<T> result;
        result.reserve(count);

        for (size_t i = 0; i < count; ++i) {
            result.push_back(source[dist(gen)]);
        }

        return result;
    }

    // C-style array'den tekrarl  se im
    template<typename T>
    std::vector<T> selectWithReplacement(const T* source, size_t sourceSize, size_t count) {
        if (sourceSize == 0) return {};

        std::uniform_int_distribution<> dist(0, sourceSize - 1);
        std::vector<T> result;
        result.reserve(count);

        for (size_t i = 0; i < count; ++i) {
            result.push_back(source[dist(gen)]);
        }

        return result;
    }
};

int main() {
    ArrayRandomSelector selector;

    // Vector  rnei
    std::vector<std::string> colors = {"K rm z ", "Mavi", "Ye il", "Sar ", "Mor", "Turuncu"};
    auto selectedColors = selector.selectWithReplacement(colors, 4);

    std::cout << "Vector'den 4 renk se imi (tekrarl ): ";
    for (const auto& color : selectedColors) {
        std::cout << color << " ";
    }
    std::cout << std::endl;

    // Array  rnei
    std::array<int, 8> numbers = {10, 20, 30, 40, 50, 60, 70, 80};
    auto selectedNumbers = selector.selectWithReplacement(numbers,
        5);

    std::cout << "Array'den 5 say  se imi (tekrarl ): ";
    for (int num : selectedNumbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    // C-style array  rnei
    const char* fruits[] = {"Elma", "Armut", "Muz", "Portakal", "
         zm  "};
    auto selectedFruits = selector.selectWithReplacement(fruits, 5,
         3);

    std::cout << "C-array'den 3 meyve se imi (tekrarl ): ";
    for (const auto& fruit : selectedFruits) {
        std::cout << fruit << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Listing 24: Array'den tekrarlı random seçim

9.6.2 Tekrarsız Seçim (Without Replacement)

```
#include <random>
#include <iostream>
#include <vector>
#include <algorithm>
#include <set>

class UniqueArraySelector {
private:
    std::mt19937 gen;

public:
    UniqueArraySelector() : gen(std::random_device{}()) {}

    // Shuffle y ntemi (h zl , b y k diziler i in ideal)
    template<typename T>
    std::vector<T> selectUniqueShuffle(std::vector<T> source,
        size_t count) {
        if (count >= source.size()) return source;

        std::shuffle(source.begin(), source.end(), gen);
        source.resize(count);
        return source;
    }

    // Sample y ntemi (C++17, k   k se imler i in ideal)
    template<typename T>
    std::vector<T> selectUniqueSample(const std::vector<T>& source, size_t count) {
        if (count >= source.size()) return source;

        std::vector<T> result;
        result.reserve(count);
        std::sample(source.begin(), source.end(), std::back_inserter(result), count, gen);
        return result;
    }

    // Index-based se im (orjinal diziyi korur)
    template<typename T>
    std::vector<T> selectUniqueByIndex(const std::vector<T>& source, size_t count) {
        if (count >= source.size()) return source;

        std::set<size_t> selectedIndices;
        std::uniform_int_distribution<> dist(0, source.size() - 1);

        // Unique indeksler se
        while (selectedIndices.size() < count) {
            selectedIndices.insert(dist(gen));
        }

        // Se ilen elementleri topla
        std::vector<T> result;
        result.reserve(count);
        for (size_t index : selectedIndices) {
            result.push_back(source[index]);
        }

        return result;
    }

    // Reservoir sampling (b y k diziler i in memory-efficient)
    template<typename T>
    std::vector<T> selectUniqueReservoir(const std::vector<T>& source, size_t count) {
        if (count >= source.size()) return source;

        std::vector<T> reservoir;
        reservoir.reserve(count);

        //  lk  k elementi al
        for (size_t i = 0; i < count && i < source.size(); ++i) {
            reservoir.push_back(source[i]);
        }

        // Geri kalan elementler i in reservoir sampling
        std::uniform_int_distribution<> dist;
        for (size_t i = count; i < source.size(); ++i) {
            dist.param(std::uniform_int_distribution<>::param_type(0, i));
            size_t j = dist(gen);

            if (j < count) {
                reservoir[j] = source[i];
            }
        }

        return reservoir;
    }

    // A   rl  kl   se im
    template<typename T>
    std::vector<T> selectWeighted(const std::vector<T>& source,
                                  const std::vector<double>& weights,
                                  size_t count) {
        if (source.size() != weights.size() || source.empty()) return {};

        std::discrete_distribution<> dist(weights.begin(), weights.end());
        std::vector<T> result;
        result.reserve(count);

        for (size_t i = 0; i < count; ++i) {
            result.push_back(source[dist(gen)]);
        }

        return result;
    }
};

int main() {
    UniqueArraySelector selector;

    std::vector<std::string> students = {
        "Ali", "Ay e", "Mehmet", "Fatma", "Ahmet",
        "Zeynep", "Mustafa", "Elif", " mer ", "Seda"
    };

    std::cout << "=== Tekrars z Se im Y ntemleri ===" << std::endl;

    // Shuffle y ntemi
    auto shuffleResult = selector.selectUniqueShuffle(students, 4);
    std::cout << "Shuffle y ntemi (4   renci  ): ";
    for (const auto& student : shuffleResult) {
        std::cout << student << " ";
    }
    std::cout << std::endl;

    // Sample y ntemi
    auto sampleResult = selector.selectUniqueSample(students, 3);
    std::cout << "Sample y ntemi (3   renci  ): ";
    for (const auto& student : sampleResult) {
        std::cout << student << " ";
    }
    std::cout << std::endl;

    // Index-based y ntem
    auto indexResult = selector.selectUniqueByIndex(students, 5);
    std::cout << "Index-based (5   renci  ): ";
    for (const auto& student : indexResult) {
        std::cout << student << " ";
    }
    std::cout << std::endl;

    // Reservoir sampling
    auto reservoirResult = selector.selectUniqueReservoir(students,
        3);
    std::cout << "Reservoir sampling (3   renci  ): ";
    for (const auto& student : reservoirResult) {
        std::cout << student << " ";
    }
    std::cout << std::endl;

    // A  rl kl  se im
    std::vector<double> weights = {0.1, 0.2, 0.1, 0.3, 0.1, 0.2,
       0.1, 0.4, 0.1, 0.2};
    auto weightedResult = selector.selectWeighted(students, weights
        , 4);
    std::cout << "A  rl kl  se im (4   renci  ): ";
    for (const auto& student : weightedResult) {
        std::cout << student << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Listing 25: Array'den tekrarsız random seçim

9.6.3 Performance Karşılaştırması

```
#include <random>
#include <iostream>
#include <vector>
#include <chrono>
#include <algorithm>

class ArraySelectionBenchmark {
private:
    std::mt19937 gen;

public:
    ArraySelectionBenchmark() : gen(std::random_device{}()) {}

    // Benchmark fonksiyonu
    template<typename Func>
    double measureTime(Func&& func, int iterations = 1000) {
        auto start = std::chrono::high_resolution_clock::now();

        for (int i = 0; i < iterations; ++i) {
            func();
        }

        auto end = std::chrono::high_resolution_clock::now();
        auto duration = std::chrono::duration_cast<std::chrono::
           microseconds>(end - start);
```

```cpp

        return duration.count() / static_cast<double>(iterations);
    }

    void runBenchmarks() {
        // Test verisi olu tur
        std::vector<int> largeArray(10000);
        std::iota(largeArray.begin(), largeArray.end(), 1);

        const size_t selectCount = 100;

        std::cout << "=== Performance Kar  la t rmas  ===" <<
            std::endl;
        std::cout << "Array boyutu: " << largeArray.size() << std::
            endl;
        std::cout << "Se ilecek eleman say s : " << selectCount
            << std::endl;
        std::cout << "Her test 1000 kez tekrarland \\n" << std::
            endl;

        // Shuffle y ntemi
        auto shuffleTime = measureTime([&]() {
            auto copy = largeArray;
            std::shuffle(copy.begin(), copy.end(), gen);
            copy.resize(selectCount);
        });

        // Sample y ntemi
        auto sampleTime = measureTime([&]() {
            std::vector<int> result;
            result.reserve(selectCount);
            std::sample(largeArray.begin(), largeArray.end(),
                        std::back_inserter(result), selectCount, gen
                            );
        });

        // Index-based y ntem
        auto indexTime = measureTime([&]() {
            std::set<size_t> indices;
            std::uniform_int_distribution<> dist(0, largeArray.size
                () - 1);

            while (indices.size() < selectCount) {
                indices.insert(dist(gen));
            }

            std::vector<int> result;
            for (size_t idx : indices) {
                result.push_back(largeArray[idx]);
            }
        });

        // Sonu lar  yazd r
        std::cout << "Shuffle y ntemi:     " << shuffleTime << "
             s " << std::endl;
        std::cout << "Sample y ntemi:      " << sampleTime << "
             s " << std::endl;
        std::cout << "Index-based y ntem:  " << indexTime << "  s" << std::endl;

        std::cout << "\\n===  neriler  ===" << std::endl;
        std::cout << "   K    k se imler i in: std::sample" << std::endl;
        std::cout << "   B y k se imler i in: shuffle" << std::endl;
        std::cout << "   Memory k s tl : index-based" << std::endl;
        std::cout << "    ok  b y k diziler: reservoir sampling" << std::endl;
    }
};

int main() {
    ArraySelectionBenchmark benchmark;
    benchmark.runBenchmarks();

    return 0;
}
```

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.

```cpp
#include <random>
#include <iostream>
#include <thread>
#include <chrono>

class RandomNumberGenerator {
private:
    static thread_local std::mt19937 generator;
    static bool seeded;

public:
    // Thread-safe singleton pattern
    static std::mt19937& getGenerator() {
        if (!seeded) {
            auto seed = std::chrono::high_resolution_clock::now()
                           .time_since_epoch().count();
            generator.seed(static_cast<unsigned>(seed));
            seeded = true;
        }
        return generator;
    }

    // Utility fonksiyonlar
    static int randInt(int min, int max) {
        std::uniform_int_distribution<> dist(min, max);
        return dist(getGenerator());
    }

    static double randReal(double min = 0.0, double max = 1.0) {
        std::uniform_real_distribution<> dist(min, max);
        return dist(getGenerator());
    }

    static bool randBool(double probability = 0.5) {
        std::bernoulli_distribution dist(probability);
        return dist(getGenerator());
    }
};

// Static member tan mlar
thread_local std::mt19937 RandomNumberGenerator::generator;
bool RandomNumberGenerator::seeded = false;

int main() {
    // G venli ve kolay kullan m
    std::cout << "Random int (1-10): " << RandomNumberGenerator::randInt(1, 10) << std::endl;
    std::cout << "Random double: " << RandomNumberGenerator::randReal() << std::endl;
    std::cout << "Random bool: " << RandomNumberGenerator::randBool(0.7) << std::endl;

    return 0;
}
```

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.

```cpp
#include <cmath>
#include <iostream>
#include <chrono>
#include <vector>

// Yava  versiyon - her seferinde hesaplama
double slowCalculation(const std::vector<double>& data) {
    double sum = 0.0;
    for (size_t i = 0; i < data.size(); ++i) {
        sum += std::pow(data[i], 2) / std::sqrt(data.size()); // Her seferinde sqrt hesaplan r
    }
    return sum;
}

// H zl  versiyon -  nceden  hesaplama
double fastCalculation(const std::vector<double>& data) {
    double sum = 0.0;
    double sqrtSize = std::sqrt(data.size()); // Bir kez hesapla
    for (size_t i = 0; i < data.size(); ++i) {
        sum += (data[i] * data[i]) / sqrtSize; // pow yerine  arpma
    }
    return sum;
}
```

Listing 28: Performans optimizasyonu örneği

11 Hata Yönetimi ve Debugging

11.1 Matematiksel Hatalar

```cpp
#include <cmath>
#include <iostream>
#include <cerrno>
#include <cfenv>

void checkMathError(const std::string& operation) {
    if (errno == EDOM) {
        std::cout << "Domain error in " << operation << std::endl;
        errno = 0;
    } else if (errno == ERANGE) {
        std::cout << "Range error in " << operation << std::endl;
        errno = 0;
    }
}

int main() {
    // Domain error  rnei
    double result1 = std::sqrt(-1.0);
    checkMathError("sqrt(-1)");
    if (std::isnan(result1)) {
        std::cout << "sqrt(-1) returned NaN" << std::endl;
    }

    // Range error   rnei
    double result2 = std::exp(1000.0);
    checkMathError("exp(1000)");
    if (std::isinf(result2)) {
        std::cout << "exp(1000) returned infinity" << std::endl;
    }

    return 0;
}
```

Listing 29: Hata yönetimi

12 C++17 ve Sonrası Yeni Özellikler

12.1 Özel Matematik Fonksiyonları

C++17 ile birlikte gelen özel matematik fonksiyonları:

```cpp
#include <cmath>
#include <iostream>

#if __cplusplus >= 201703L
int main() {
    double x = 2.0;
    int n = 3;

    // Bessel fonksiyonlar
    std::cout << "Bessel J0(" << x << ") = " << std::cyl_bessel_j
        (0, x) << std::endl;
    std::cout << "Bessel Y0(" << x << ") = " << std::cyl_neumann(0,
         x) << std::endl;

    // Legendre polinomlar
    std::cout << "Legendre P" << n << "(" << 0.5 << ") = "
              << std::legendre(n, 0.5) << std::endl;

    // Laguerre polinomlar
    std::cout << "Laguerre L" << n << "(" << x << ") = "
              << std::laguerre(n, x) << std::endl;

    return 0;
}
#else
int main() {
    std::cout << "C++17 required for special math functions" << std
        ::endl;
    return 0;
}
#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)