BÖLÜM 8:

Anahtar Kavramlar

Kullanıcı-tanımlı Fonksiyon
Fonksiyon Aşırı Yükleme
Referanslar
Fonksiyon Şablonları
Özyinelemeli Fonksiyon
Rasgele Sayı Üretimi
Varsayılan Argüman

Öğrenme Hedefleri

1. Kullanıcı-tanımlı fonksiyon yazabilir.
2. Matematik kütüphanesi fonksiyonlarını kullanabilir.
3. Rasgele sayı üretebilir.
4. Kapsam kurallarını uygulayabilir.
5. Referans parametrelerini kullanabilir.
6. Varsayılan argüman kullanabilir.
7. Fonksiyon aşırı yükleme gerçekleştirebilir.
8. Fonksiyon şablonları oluşturabilir.
9. Özyinelemeli fonksiyon yazabilir ve kullanabilir.

1 GİRİŞ

Uygulamaları hayata geçirmek için yazılan programlar büyüklük açısından farklılıklar gösterir. Büyük programları oluşturmak için daha küçük kod parçaları yazmak ve bu parçaları birleştirerek çözüme ulaşmak gerekebilir. Bu bölümde program yazarken kullandığımız fonksiyonlar konusu anlatılacaktır. Öncelikli olarak C++'da program parçalarından bahsedilecek, daha sonra C++ standart kütüphanesinde yer alan matematik fonksiyonları gösterilecektir. Sonraki bölümlerde fonksiyon prototipleri ve nasıl kullanıldıkları örnekler ile anlatılacaktır. Rasgele sayı üretimi devam eden kısımlarda gösterilecek bir diğer konudur. Oyun programlarında veya simülasyon içeren programlarda şans faktörünü ifade etmek için rasgele sayı üretimi, bölümde yer alacak diğer bir alt başlık olacaktır. Kapsam kurallarında, farklı bloklarda tanımlanan değişkenlerin hangi öncelikler çerçevesinde erişilebileceği gösterilecektir. Global ve yerel değişkenlerle beraber statik değişkenler de bu konunun bir parçasıdır. Fonksiyonlar bölümünde değinilecek bir başka konu da fonksiyon aşırı yükleme olayıdır. Bu şekilde aynı isim ile birden fazla fonksiyon tanımlamasının ne şekilde yapılacağı açıklanacaktır. İlgili bir konuda fonksiyon şablonlarıdır. Tek bir tanım ile farklı veri türleri ile kullanılacak fonksiyonlar oluşturulacaktır. Bir problemi daha iyi bir şekilde çözüme kavuşturmak için kendi kendini çağıran fonksiyonlara ihtiyaç duyulabilir. Bu fonksiyonlara özyinelemeli fonksiyonlar ismi verilir ve bu konu bölümün son konusu olacaktır.

C++'da Program Parçaları

Program yazmamızın temel amacı karşılaştığımız problemlere çözüm oluşturmaktır. Bazı problemleri çözmek için birden fazla alt probleme çözüm getirmek gerekebilir. Her bir alt problem için farklı bir kısmi kod oluşturmak mümkündür. Bazen bir alt problem için yazdığımız çözüm, programımızın farklı noktalarında birden fazla kez kullanılmak durumunda olabilir. Aynı kod parçasını programın farklı noktalarına tekrar tekrar yerleştirmek kod büyüklüğünü gereksiz yere artıracaktır. Aynı zamanda programımızda yer alan gereksiz kod miktarı hata yapma ihtimalini artırmakta, dayanıklı bir program oluşturmayı önlemektedir. Dolayısı ile bir alt probleme getirilen çözüm sadece bir kez yazılıp, programda ihtiyaç olan noktalarda kullanılabilir. Bu kullanım için yazılan kod parçasının çağrılması gerekir. Alt probleme getirilen çözüm bir fonksiyon olarak oluşturulur. Fonksiyonun sadece bir kez tanımlanması yeterlidir. Programın içerisinden defalarca çağrılarak kullanılabilir.

Temel olarak fonksiyonlar bir programın gerçekleştirmesi gereken işlemin belirli parçalarını yapan bağımsız birimler olarak tanımlanırlar. C++ kütüphanelerinde tanımlanan kullanıma hazır fonksiyonlar mevcuttur. Ancak kullanıcılar isterlerse kendi fonksiyon tanımlamalarını gerçekleştirip programlarında kullanabilirler.

Math Kütüphanesi Fonksiyonları

C++ programlama dilinde yer alan matematik kütüphanesi bazı genel matematiksel hesaplamalar için kullanılan fonksiyonları içerir. Geliştiricilerin bu fonksiyonları kullanabilmeleri için <***cmath***> kütüphanesini programlarına eklemeleri gerekir. Örneğin, xy hesaplamasını gerçekleştirmek için

pow(x^y)

fonksiyonu kullanılır. Eğer bu fonksiyonu ***pow***(2,3) şeklinde çağırırsak sonuç 8 olarak döner. Aşağıda verilen Tablo 8.1'de matematik kütüphanesinde yer alan bazı önemli fonksiyonlar ve kullanımları verilmiştir.

Fonksiyon	Tanım	Örnek
pow(x, y)	x'in y. kuvvetini hesaplar (xy)	pow(2, 5) Sonuç:32
sqrt(x)	x'in karekökünü hesaplar	sqrt(25.0) Sonuç:5
log(x)	x'in doğal logoritması (e tabanında)	log(15) Sonuç: 2.70805
log10(x)	x'in logoritması (10 tabanında)	log10(100) Sonuç: 2
fmod(x,y)	x/y işleminin kalan değerini bulur	fmod(2.48,1.2) Sonuç: 0.08
ceil(x)	x'in değerini yukarıya yuvarlar	ceil(2.7) Sonuç: 3
floor(x)	x'in değerini aşağıya yuvarlar	floor(2.7) Sonuç: 2
exp(x)	üstel fonksiyon ex	exp(4) Sonuç: 54.5982
fabs(x)	x'in mutlak değerini hesaplar	fabs(-2.45) Sonuç: 2.45

Tablo 8.1 Matematik Kütüphanesi Fonksiyonları.

Fonksiyon Prototipleri

Program geliştiriciler kullanıcı tanımlı fonksiyonlar oluşturabilir ve bu fonksiyonu programlarında kullanabilirler. Aşağıdaki programda kullanıcı tanımlı ***maksimum*** fonksiyonu tanımlanmış ve kullanılmıştır. ***maksimum*** fonksiyonu 3 tam sayı parametre alır. Bu sayılardan en büyük değere sahip olanı bulunur ve çevrilir. Bir fonksiyonun prototipinde, döndürülen değerin veri türü, fonksiyonun ismi ve parantez içinde argümanların türleri ve isimleri sırasıyla yazılır. Eğer fonksiyon herhangi bir değer döndürmüyorsa döndürülen değerin türü ***void*** olarak yazılır. Bir fonksiyonun prototipi derleyiciye fonksiyonun ismini, dönüş türünü ve parametrelerin sıralı olarak türlerini söyler. Fonksiyonun prototipi aynı zamanda fonksiyonun tanımının ilk satırıyla birebir aynıdır. Tek fark prototipin noktalı virgül ile bitmesidir.

int maksimum(int sayi1, int sayi2, int sayi3);

Fonksiyonun tanımında, beklenen görevi tamamlamak için gerekli tüm işlemler gerçekleştirilir. Bu sayısal bir hesaplama olabileceği gibi sadece ekrana bilgi yazdıran bir görev de olabilir. Fonksiyonun tanımı, fonksiyon başlığından sonra kıvrımlı parantezler içerisine yerleştirilir.

```
//Program8.1.cpp
#include <iostream>
#include <iomanip>
using namespace std;
int maksimum(int sayi1, int sayi2, int sayi3);
int main()
```

```
{
        int i1, i2, i3;
        cout<<"3 tam sayi giriniz:  ";
        cin>>i1>>i2>>i3;
        cout<<"Maksimum sayi: "<<maksimum(i1,i2,i3)<<endl;
        return 0;
}
int maksimum (int sayi1, int sayi2, int sayi3)
{
        int mSayi;
        if(sayi1>sayi2 && sayi1>sayi3)
                mSayi=sayi1;
        if(sayi2>sayi1 && sayi2>sayi3)
                mSayi=sayi2;
        if(sayi3>sayi1 && sayi3>sayi2)
                mSayi=sayi3;
        return mSayi;
}
```

Program8.1 Çıktısı:

```
3 tam sayi giriniz:  32 45 28
Maksimum sayi: 45
```

Program8.1.cpp'de ***main*** fonksiyonda 3 tam sayı değişkeni tanımlanmış ve kullanıcıdan bu değişkenler için değer girmesi istenmiştir. Daha sonra kullanıcının girdiği değerler argüman olarak kullanılmış ve ***maksimum*** fonksiyonu çağrılmıştır. Fonksiyonun çevirdiği değer ***cout*** ile ekrana yazdırılmıştır.

maksimum fonksiyonunun tanımında ise parametre olarak gelen 3 tam sayı içerisinden en büyük değer hesaplanmış ve ***main*** fonksiyona döndürülmüştür. En büyük değeri hesaplamak için farklı yöntemler kullanılabilir. Burada 3 farklı ***if*** ifadesi ile en büyük değer bulunmuştur. Her bir ***if*** ifadesi bir tam sayı değişkeni diğer 2 değişken ile karşılaştırmış ve söz konusu değişken diğer 2 değişkenden de büyükse en büyük değer olarak belirlenmiştir. Bu fonksiyon açısından tüm tam sayı değerlerin birbirinden farklı olduğunu kabul edebiliriz. Herhangi 2 değer birbirine eşit olmayacaktır şeklinde ön kabulümüz olabilir.

DİKKAT EDELİM

Bir fonksiyonun prototipinde, tanımlanmasında ve çağrılmasında kullanılan parametrelerin/argümanların sayısı, türleri ve sıraları birbirleri ile uyumlu olmalıdır.

Rasgele Sayı Üretimi

Bazı uygulamalar için yazılan programlarda rasgele sayılar üretmek gerekir. Simülasyon içeren veya bazı oyun programlarında şans faktörü uygulamanın bir parçasıdır. C++ programlama dilinde rasgele tamsayılar üretmek için rand() fonksiyonu kullanılır.

i=rand();

rand() fonksiyonu 0 ile RAND_MAX arasında bir tamsayı çevirir. RAND_MAX ***<cstdlib>*** kütüphanesi içerisinde tanımlanmıştır. Farklı programlar farklı rastgele sayı aralıklarına ihtiyaç duyarlar. Örneğin yazı-tura simülasyonu sadece 2 farklı rastgele sayıya ihtiyaç duyarken, zar atma simülasyonu 6 farklı değer içerisinden rasgele bir seçim gerektirir. Program8.2.cpp'de 15 defa zar atma işlemi gerçekleştirilmiş ve elde edilen değerler çıktı olarak verilmiştir. Program8.3.cpp'de ise 24000 kez zar atılmış ve bir zar yüzünün kaç kez tekrarladığı ekrana yazdırılmıştır.

```
//Program8.2.cpp
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

int main()
{
int i;
for(i=1; i<=15; i++)
{
      cout<<setw(5)<<(1+rand()%6);
      if(i%3 == 0)
            cout<<endl;
}
return 0;
}
```

Program8.2.cpp Çıktısı

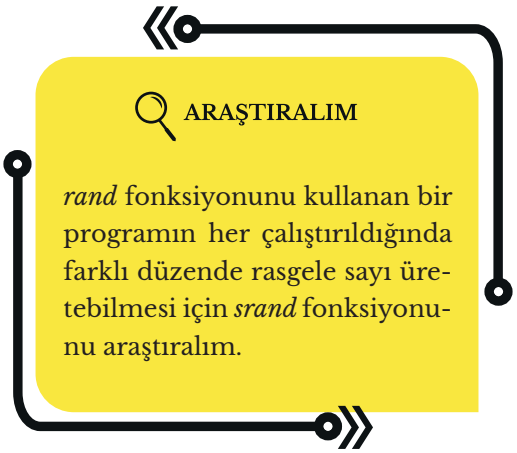
```
  6   4   3
  6   1   1
  5   1   3
  2   4   6
  4   6   1
```

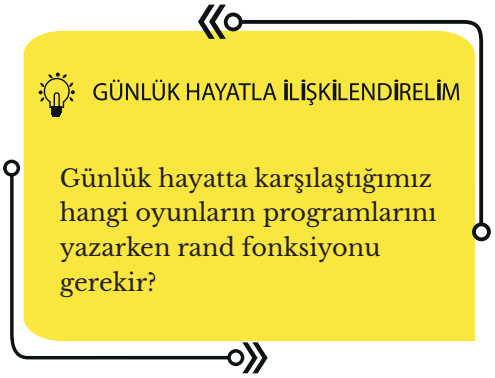
```
//Program8.3.cpp
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

int main()
{
unsigned int frekans1=0;
unsigned int frekans2=0;
unsigned int frekans3=0;
```

```
unsigned int frekans4=0;
unsigned int frekans5=0;
unsigned int frekans6=0;
int face;

for (unsigned int roll=1; roll <= 24000; ++roll)
{
face = 1 + rand() % 6;
switch (face) {
case 1:
++frekans1;
break;
case 2:
++frekans2;
break;
case 3:
++frekans3;
break;
case 4:
++frekans4;
break;
case 5:
++frekans5;
break;
case 6:
++frekans6;
break;
default:
cout << "Bu komut asla calismaz!";
}
}
cout << "Yuz" << setw(13) << "Frekans" << endl;
cout << " 1" << setw(15) << frekans1
<< "\n 2" << setw(15) << frekans2
<< "\n 3" << setw(15) << frekans3
<< "\n 4" << setw(15) << frekans4
<< "\n 5" << setw(15) << frekans5
<< "\n 6" << setw(15) << frekans6 << endl;
return 0;



}
```

Program8.3.cpp Çıktısı

```
Yuz   Frekans
1      3983
2      4082
3      3899
4      4013
5      3976
6      4047
```





KAPSAM KURALLARI

Bir fonksiyonda değişkenlere erişim kuralları, değişkenlerin tanımlandığı konuma göre belirlenir. Değişkenler sadece tanımlandıkları fonksiyon veya blok içerisinde doğrudan kullanılabilirler. Bu değişkenlere yerel değişkenler denir. İlgili fonksiyon veya blok çalıştığında bu değişkenler için bellekte yer ayrılır. Fonksiyon veya blok sonlandığında ayrılan bu alan iptal olur ve bellekteki değişken silinir. Aynı fonksiyon veya aynı blok daha sonra tekrar çağrılsa dahi değişkenler daha önceki değerlerine sahip olamazlar. ***main*** fonksiyonda veya diğer fonksiyonlarda tanımlanan tüm değişkenler yerel değişken olarak değerlendirilirler.

Eğer tanımlanan bir değişkenin programda yer alan tüm fonksiyonlarda kullanılabilmesi amaçlanıyorsa bu değişkenler global değişken olarak tanımlanır. Tanımlanma işlemi fonksiyonların dışında gerçekleştirilir.

```
//Program8.4.cpp
#include <iostream>
using namespace std;
void local();
void staticLocal();
void global();
int i=1;
int main()
{
	cout<<"main fonksiyonda global degisken i: "<<i<<endl;
	int i=5;
	cout<<"main fonksiyonda yerel degisken i: "<<i<< endl;
	{
		int i=7;
		cout<<"main fonksiyonda ic blokda yerel degisken i: "<<i<<endl;
	}
	cout<<"main fonksiyonda yerel degisken i: "<<i<<endl;
	local();
	staticLocal();
	global();
	local();
	staticLocal();
	global();
	cout<<"main fonksiyonda yerel degisken i: "<<i<<endl;
	return 0;
}
void local()
{
	int i=10;
	cout<<"\nyerel degisken i "<<i<< " local fonksiyonu baslarken"<<endl;
	i++;
	cout<<"yerel degisken i "<<i<< " local fonksiyonu biterken"<<endl;
}
void staticLocal()
{
	static int i=100;
	cout<<"\nyerel statik degisken i "<<i<<" staticLocal fonksiyonu baslarken "<<endl;
	i++;
	cout<<"yerel statik degisken i "<<i<<" staticLocal fonksiyonu biterken"<<endl;
}
void global()
{
	cout<<"\nglobal degisken i "<<i<<" global fonksiyonu baslarken"<<endl;
```

```
    i *= 20;
    cout<<"global degisken i "<<i<<" global fonksiyonu biterken"<<endl;
}
```

Program8.4.cpp Çıktısı

```
main fonksiyonda global degisken i: 1
main fonksiyonda yerel degisken i: 5
main fonksiyonda ic blokda yerel degisken i: 7
main fonksiyonda yerel degisken i: 5

yerel degisken i 10 local fonksiyonu baslarken
yerel degisken i 11 local fonksiyonu biterken

yerel statik degisken i 100 staticLocal fonksiyonu baslarken
yerel statik degisken i 101 staticLocal fonksiyonu biterken

global degisken i 1 global fonksiyonu baslarken
global degisken i 20 global fonksiyonu biterken

yerel degisken i 10 local fonksiyonu baslarken
yerel degisken i 11 local fonksiyonu biterken

yerel statik degisken i 101 staticLocal fonksiyonu baslarken
yerel statik degisken i 102 staticLocal fonksiyonu biterken

global degisken i 20 global fonksiyonu baslarken
global degisken i 400 global fonksiyonu biterken
main fonksiyonda yerel degisken i: 5
```

DİKKAT EDELİM

Kapsam kurallarına dikkat edilmeden gerçekleştirilen değişken isimlendirmeleri, programın doğru olmayan sonuçlar üretmesine sebep olabilir.

REFERANSLAR VE REFERANS PARAMETRELERİ

Birçok programa dilinde parametreler fonksiyonlara iki farklı yöntem ile gönderilir. Birinci yöntem değer ile gönderim, ikinci metot ise referans ile gönderimdir. Bu bölümde ilk olarak gösterdiğimiz yaklaşım değer ile gönderimdir. Fonksiyona iletilmek istenen parametrenin bellekte bir kopyası yapılır ve fonksiyon içerisinde bu kopya kullanılır. Dolayısı ile bu parametre için fonksiyonun içerisinde gerçekleştirilecek işlemlerden/değişikliklerden fonksiyonu çağıran diğer fonksiyondaki (genellikle main program/fonksiyon) orijinal değişken etkilenmez. İkinci yaklaşım olan referans ile gönderimde çağrılan fonksiyona gönderilen orijinal parametrenin bir kopyası oluşturulmaz. Fonksiyon doğrudan orijinal değişkene erişip, işlem yapma ve değiştirme hakkına sahip olur. Bu durumu gerçekleştirebilmek için referans parametreler kullanılır. Fonksiyon ve bu fonksiyonu çağıran fonksiyon ilgili parametreye doğrudan erişim hakkına sahiptir. Aslında bu değişken bellekte sadece bir defa yer almaktadır ve bu bellek alanı orijinal değişken ismi ile çağıran fonksiyonda, referans parametre ile çağrılan fonksiyonda erişilebilmektedir.

C++ referans ile gönderim için referans parametrelerine sahiptir. Referans ile gönderim ile çağırılan fonksiyona çağıran fonksiyonun verisine doğrudan erişim ve değiştirme hakkı verilir.

Referans parametresi bir fonksiyon çağrıldığında karşılık gelen argümana bir takma ad olarak değerlendirilebilir. Bu durumu göstermek için, farklı bir ifade ile parametrenin referans ile gönderim metoduyla aktarıldığını ifade etmek için '***&***' sembolü kullanılır.

int &sayi;

Bu ifade; ***sayi*** değişkeni bir tam sayı değere referanstır şeklinde okunur. Fonksiyon çağrılırken orijinal değişken sadece ismi ile kullanılır. Çağrılan fonksiyonda sayi değişkeni referans olarak tanımlandığı için çağıran fonksiyondaki orijinal değişkene doğrudan erişim sağlayabilir.

Program8.5.cpp değer ve referans ile gönderim yaklaşımları arasındaki farkı göstermek için yazılmıştır.

```
//Program8.5.cpp
#include <iostream>
using namespace std;
void fonksiyon1(int);
void fonksiyon2(int&);
int main()
{
      int n;
      cout<<"Bir tam sayi giriniz:\n";
      cin>>n;
      fonksiyon1(n);
      cout<<n<<endl;
      fonksiyon2(n);
      cout<<n<<endl;
      return 0;
}
void fonksiyon1(int sayi)
{
      sayi = sayi + 5;
}
void fonksiyon2(int& sayi)
```

```
{
    sayi = sayi * 4;
}
```

Program8.5.cpp Çıktısı

```
Bir tam sayi giriniz:
10
10
40
```

DİKKAT EDELİM

Referans ile bir fonksiyona gönderilen bir parametrenin değerinde yapılan değişiklik, orijinal değişkende de gerçekleşecektir.

ARAŞTIRALIM

Referans ile gönderim yaklaşımında gerçekleşebilecek istenmeyen değişiklikleri önlemek amacıyla const kullanımını araştıralım.

VARSAYILAN ARGÜMANLAR

Bazı fonksiyonlar, aynı argüman için aynı değerle defalarca çağrılabilirler. Bu durumlarda varsayılan argüman belirlenebilir ve kullanılabilir. Varsayılan argüman fonksiyon çağrılırken tekrar yazılmak zorunda değildir. Ancak ilgili argüman için farklı bir değer kullanılmak istenirse, fonksiyon çağrılırken yeni değer parametre olarak kullanılabilir. Bu durumda varsayılan argüman değil, gönderilen parametre fonksiyonun içerisinde kullanılacaktır. Varsayılan argüman atama operatörü (0) ile varsayılan değerine sahip olur. Program8.6.cpp'de varsayılan argümanları içeren bir fonksiyonun kullanıldığı program yer almaktadır.

```
//Program8.6.cpp
#include <iostream>
```

```
using namespace std;
int alanDiktortgen(int a=1, int b=1)
{
	return a*b;
}
int main()
{
	cout<<"Varsayilan Argumanlar ile Diktortgen Alani:"<<alanDiktortgen()<<endl;
	cout<<"Kenarlardan bir tanesinin verildigi Diktortgen Alani:"<<alanDiktortgen(5)<<endl;
	cout<<"Her iki kenarin belirlendigi Diktortgen Alani:"<<alanDiktortgen(5,10)<<endl;

	return 0;
}
```

Program8.6.cpp Çıktısı:

```
Varsayilan Argumanlar ile Diktortgen Alani:1
Kenarlardan bir tanesinin verildigi Diktortgen Alani:5
Her iki kenarin belirlendigi Diktortgen Alani:50
```

TEKLİ KAPSAM ÇÖZÜMLEME OPERATÖRÜ

Bir kapsamda aynı isimde yerel bir değişken olduğu durumda, global değişkene erişmek amacıyla Tekli Kapsam Çözümleme Operatörü (::) kullanılır. Bu operatöre sadece yerel değişken ve global değişke aynı isimde ise ve global değişken kullanılmak isteniyorsa ihtiyaç vardır. Eğer global değişkenin ismi kapsamda yer alan yerel değişkenlerden farklı ise global değişken ilgili operatör olmadan doğrudan erişilebilir ve kullanılabilir. Tekli Kapsam Çözümleme Operatörü sadece global değişkene erişmek için kullanılır, bir dış kapsamdaki yerel değişkene erişmek için kullanılmaz. Program8.7.cpp isimli programda bu operatörün kullanımı gösterilmektedir.

```
//Program8.7.cpp
#include <iostream>
using namespace std;
int i=1;
int main()
{
	int i=5;
	cout<<"Yerel degisken: "<<i<<endl;
	cout<<"Global degisken: "<<::i<<endl;
	return 0;
}
```

Program8.7.cpp Çıktısı

```
Yerel degisken: 5
Global degisken: 1
```

FONKSİYON AŞIRI YÜKLEME

Bazı benzer işlemler için, farklı fonksiyonlar tanımlamak gerekebilir. Bu durumlarda program geliştirici aynı fonksiyon ismini kullanmak isteyebilir. Farklı ancak benzer işlemler gerçekleştiren fonksiyonlar aynı isimde tanımlanabilirler. Bu tanımlamayı sağlayabilmek için fonksiyonların farklı imzalara sahip olması gerekir. Bu kavram fonksiyon aşırı yükleme olarak adlandırılır. Bir fonksiyonun imzası, fonksiyonun ismi ile beraber argümanların sayısı, türü ve sırası bilgilerini içerir. C++ derleyici uygun fonksiyonu, bu bilgileri kullanarak seçer. Matematik kütüphanesindeki fonksiyonlar için aşırı yükleme uygulanmıştır. Bu fonksiyonlar ***float, double*** ve ***long double*** türleri için tanımlanmışlardır.

Program8.8.cpp'de aşırı yükleme gerçekleştirilen topla fonksiyonları sırasıyla 2 tam sayı ve 2 kesirli sayıyı toplamak için kullanılmaktadır. ***main*** fonksiyonda ilk çağrılan ***topla*** fonksiyonu ***int*** argüman aldığı için, parametreleri int olan fonksiyon kullanılır. İkinci olarak çağrılan fonksiyondaki argümanlar ***double*** olduğu için, ***double*** parametrelere sahip olan fonksiyon tanımı kullanılır.

```
//Program8.8.cpp
#include <iostream>
using namespace std;
int topla(int sayi1, int sayi2)
{
    cout<<"Iki tam sayinin toplami: ";
    return sayi1 + sayi2;
}
double topla(double sayi1, double sayi2)
{
    cout<<"Iki kesirli sayinin toplami: ";
    return sayi1 + sayi2;
}
int main()
{
    int a=5, b=8;
    double c=7.5, d=9.2;
    cout<<topla(a,b)<<endl;
    cout<<topla(c,d);
    return 0;
}
```

Program8.8.cpp Çıktısı

```
Iki tam sayinin toplami: 13
Iki kesirli sayinin toplami: 16.7
```

FONKSİYON ŞABLONLARI

Aşırı yükleme fonksiyonlar farklı veri türleri üzerinde, farklı program mantığı kullanan benzer işlemler içerirler. Program mantığı ve işlemler farklı veri türleri için aynı ise aşırı yükleme işlemi yerine fonksiyon şablonları kullanılabilir. Aslında fonksiyon şablonları, fonksiyon aşırı yüklemenin daha kompakt bir şekilde yazılmasıdır. Bir başka ifade ile sadece bir fonksiyon tanımlayarak bir aşırı yükleme fonksiyon ailesini tanımlamış oluruz.

Aşağıda verilen kod, 3 parametreden en büyük değeri çeviren bir fonksiyon şablonu için yazılmıştır. Bütün fonksiyon şablonlarının tanımlanması template anahtar kelimesi ile gerçekleştirilir. Devamında

üçgen parantezler içerisinde şablon parametre listesi sıralanır. Bu listede yer alan parametrelerin başında ***typename*** veya ***class*** anahtar kelimeleri yer alır. Burada yer alan tür parametreleri temel veri türleri veya kullanıcı tanımlı veri türleri için bir yer tutucu olarak değerlendirilebilirler. Genellikle büyük harf kullanılarak tanımlanırlar. Bizim örneğimizde T harfi kullanılmıştır. Tür parametreleri fonksiyonun parametrelerini belirtmede, fonksiyonun dönüş değerini ifade etmede ve fonksiyonun içerisinde yerel değişkenleri tanımlamada kullanılabilirler. Şablon fonksiyonu normal fonksiyonlar gibi tanımlanırlar, ancak tür parametrelerini gerçek veri türlerini temsil etmek için kullanırlar.

```
template <typename T>
T maksimum (T v1, T v2, T v3)
{
        T m = v1;
        if(v2>m)
                m= v2;
        if(v3>m)
                m = v3;
        return m;
}
```

Bu örnekte şablon sadece bir tür parametresi tanımlıyor. Parametre listesinde sadece farklı bir tür parametresi var. Derleyici programın kaynak kodunda maksimum fonksiyonunun çağrıldığını gördüğünde tür parametresini ilgili belirlenen tür ile değiştirir. Buradaki örnekte tüm tür parametreleri aynı olduğu için (T), tüm parametreler aynı tür olmalıdır. Program8.9.cpp'de yer alan ***main*** fonksiyon şablon fonksiyonunu 3 farklı veri türü ile çağırmaktadır ***(int, double ve char).***

```
//Program8.9.cpp
#include <iostream>
using namespace std;
template <typename T>
T maksimum (T v1, T v2, T v3)
{
        T m = v1;
        if(v2>m)
                m= v2;
        if(v3>m)
                m = v3;
        return m;
}
int main()
{
        int i1, i2, i3;
        double d1, d2, d3;
        char c1, c2, c3;

        cout<<"3 tam sayi giriniz:\n";
        cin>>i1>>i2>>i3;
```

```
    cout<<"3 ondalikli sayi giriniz:\n";
    cin>>d1>>d2>>d3;

    cout<<"3 karakter giriniz:\n";
    cin>>c1>>c2>>c3;

    cout<<"Tam sayilarin maksimum olan degeri: "<<maksimum(i1,i2,i3)<<endl;
    cout<<"Kesirli sayilarin maksimum olan degeri: "<<maksimum(d1,d2,d3)<<endl;
    cout<<"Karakterlerin maksimum olan degeri: "<<maksimum(c1,c2,c3)<<endl;

    return 0;
}
```

Program8.9.cpp Çıktısı

```
3 tam sayi giriniz:
4 2 7
3 kesirli sayi giriniz:
2.3 5.4 1.2
3 karakter giriniz:
c a b
Tam sayilarin maksimum olan degeri: 7
Kesirli sayilarin maksimum olan degeri: 5.4
Karakterlerin maksimum olan degeri: c
```

ÖZYİNELEMELİ FONKSİYONLAR

Bazı problemlerde fonksiyonların kendilerini çağırmaları daha kullanışlı olabilir. Özyinelemeli fonksiyonlar kendilerini ya doğrudan ya başka bir fonksiyon üzerinden çağırırlar. Sadece ***main*** fonksiyon/program başka bir fonksiyon içerisinden çağrılamaz. Öncelikle özyineleme kavramı üzerinde tartışalım. Özyinelemeli problem çözme yaklaşımı bazı temel ortak ögelere sahiptir. Bir özyinelemeli fonksiyon çağrıldığında bu fonksiyon sadece temel durumu çözmeyi bilir. Eğer fonksiyon bu temel durumu çözmek için çağrılmışsa, cevabı çevirir. Eğer daha karmaşık bir durum için çağrılmışsa, özyinelemeli fonksiyon problemi iki parçaya ayırır. Birinci kısım fonksiyonun çözümünü bildiği kısım, ikinci kısım ise fonksiyonun problemin büyüklüğünü azaltıp kendisini çağırdığı parçadır. İkinci kısım aslında orijinal problemin aynısı veya çok benzeridir. Sadece bazı ölçütler açısından daha küçük bir problemdir. Bu kısım için fonksiyonun tekrar çağrılması özyineleme adımı olarak değerlendirilir. Özyineleme adımı çalışırken, orijinal fonksiyon açıktır. Başka bir ifade ile çalışmasını tamamlamamıştır. Bu durumda dikkat edilecek bir nokta da özyineleme adımlarının sayısının birden fazla olabileceğidir. Dolayısı ile özyineleme fonksiyon çağrılmasında birçok açık, tamamlanmamış fonksiyon söz konusu olabilir. Bu özyineleme fonksiyon çağrılma sürecinde, orijinal problem giderek küçülür ve temel probleme ulaşır. Temel problemin çözümü de fonksiyon tarafından bilindiği için doğrudan çevrilir. Tabii ki bu çevrilme en son özyineleme fonksiyonunu çağıran fonksiyon içine olur. Döndürülen bu değer çözümü bilinen birinci kısım ile birleştirilerek bir önceki fonksiyona döndürülür. Bir örnek olarak faktöriyel probleminin çözümünü tartışalım.

Bir pozitif tamsayının faktöriyeli aşağıdaki şekilde tanımlanır:

n! = n * (n-1) * (n-2) ... 1

n! = n * (n-1)!

Burada görüldüğü gibi orijinal problemimizin büyüklüğü ***n*** değerinden ***n-1*** e düşmüştür. Eğer ***(n-1)!*** değerini bilirsek bu çözümü ***n*** ile çarparak ***n!*** değerini hesaplayabiliriz. Sayısal bir örnek aşağıdaki gibi verilebilir.

5! = 5 * 4 * 3 * 2 *1

5! = 5 * 4!

Program8.10.cpp özyinelemeli fonksiyon kullanarak, kullanıcıdan alınan bir değerin faktöriyelini hesaplar. Hesaplanan değer ekrana yazdırılmaktadır.

//Program8.10.cpp

```
#include <iostream>
#include <iomanip>
using namespace std;
unsigned long factorial(unsigned long);
int main()
{
	int n;
	cout<<"Bir tam sayi giriniz:\n";
	cin>>n;
	cout<<factorial(n)<<endl;
	return 0;
}
unsigned long factorial (unsigned long number)
{
	if(number==0 || number==1)
		return 1;
	else
		return number * factorial(number-1);
}
```

Program8.10.cpp Çıktısı

```
Bir tam sayi giriniz:
7
5040
```

DİKKAT EDELİM

Temel problemi doğru tanımlanmamış özyinelemeli fonksiyonlar sonsuz döngüye girip asla sonlanmayabilirler.

BÖLÜM ÖZETİ

Fonksiyonlar bir programda problemin çözümüne yönelik gerçekleştirilen işlemlerin yer aldığı kod parçalarıdır. Bazen bu çözümler program içerisinde birden fazla kez çağrılabilir. Küçük problemlere getirilen bu çözümler, fonksiyon olarak sadece bir kez kodlanırlar ve program içerisinde gerekli noktalarda defalarca çağrılabilirler.

<***cmath***> matematik kütüphanesi, bazı temel matematiksel işlemler için yazılmış kullanıma hazır fonksiyonları içerir. Örneğin; ***sqrt*** fonksiyonu fonksiyona gönderilen argümanın karekökünü hesaplar. C++ program geliştiricileri hazır kütüphane fonksiyonlarını kullanabilecekleri gibi, kendi fonksiyonlarını da tanımlayıp kullanabilirler. Eğer geliştiricinin tanımladığı fonksiyon ***main*** fonksiyondan sonra programa eklenirse, ***main***'den önce derleyiciyi bilgilendirmek amacıyla fonksiyonun prototipinin verilmesi gerekir. Fonksiyonun prototipi, fonksiyonun ismi, döndürülen değerin türü, fonksiyonun aldığı argümanların türleri ve sırasından oluşur.

Oyun veya simülasyon içeren uygulamalarda şans faktörünü programımıza eklemek gerekir. ***rand()*** fonksiyonu 0 ile RAND_MAX arasında bir sayı üretir. Mod operatörü *'%'* ve ***rand()*** kullanarak istenilen aralıkta sayı üretimi gerçekleştirilebilir. Örneğin; ***1*** + ***rand()*** % 6 işlemi ***1*** ile ***6*** arasında rasgele sayılar üretir.

Bir fonksiyon veya blok içerisinde tanımlanan bir değişken, yerel değişken olarak kabul edilir ve sadece tanımlandığı alanda erişilip kullanılabilir. Fonksiyonların dışında tanımlanan global değişkenler, programda tüm fonksiyonlarda erişilip kullanılabilirler.

Bir argüman fonksiyona referans ***(&)*** ile gönderilebilir. Bu durumda fonksiyon ilgili değişkenin bellekteki orijinal tanımlandığı alana erişebilir. Fonksiyonda değişken üzerinde gerçekleştirilen işlemler doğrudan orijinal değişken üzerinde gerçekleşir. Bu durum referans ile gönderim olarak adlandırılır. Ancak bir argüman referans ile gönderilmezse, fonksiyon içerisinde ilgili değişkenin bir kopyası oluşturulur ve bellekte farklı bir yere yerleştirilir. Bu değişken üzerinde yapılan değişiklikler orijinal değişkeni etkilemez ve durum değer ile gönderim olarak isimlendirilir.

Bir fonksiyon tanımlanırken, bazı argümanlar için varsayılan değerler atanabilir. Fonksiyon çağrılırken ilgili argüman için bir değer gönderilmezse varsayılan değer kullanılır.

Bir fonksiyon içerisinde, yerel değişkenler ile oluşabilecek bir karmaşıklığı önlemek amacıyla, global değişkene ulaşmak için tekli kapsam çözümleme operatörü ***(::)*** kullanılır.

Benzer işlemler gerçekleştiren ancak farklı imzalara sahip fonksiyonlar aynı isimde tanımlanabilirler. Derleyici hangi fonksiyonun çağrıldığına, gönderilen parametrelerin sayısına, türlerine ve sıralarına bakarak karar verir. Bu duruma fonksiyon aşırı yükleme adı verilir. Eğer bu fonksiyonlarda kullanılan program mantığı aynı ve fonksiyonlar sadece argümanların veri türleri açısından farklılık gösteriyorsa fonksiyon şablonları kullanılabilir.

Bazı uygulamalarda problemin çözümü için özyinelemeli fonksiyonlar kullanılır. Özyinelemeli fonksiyonlar, fonksiyonların problemi daha küçük bir problem haline getirerek kendilerini çağırdıkları yapılardır. Problemin bir çözüme ulaşması için temel problemin doğru bir şekilde tanımlanması gerekir.

GÖZDEN GEÇİRELİM

1. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
int fonksiyon1(int b)
{
	b=b+5;
	return b+7;
}
int main()
{
	int a=5;
	a=fonksiyon1(a);
	cout<<a<<endl;
	return 0;
}
```

a. 5
b. 10
c. 12
d. 17
e. 22

2. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
int fonksiyon1(int b)
{
	b=b+5;
	return b+7;
}
int main()
{
	int a=5;
	fonksiyon1(a);
	cout<<a<<endl;
	return 0;
}
```

a. 5
b. 10
c. 12
d. 17
e. 22

3. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
int fonksiyon1(int &b)
{
	b=b+5;
	return b+7;
}
int main()
{
	int a=5;
	fonksiyon1(a);
	cout<<a<<endl;
	return 0;
}
```

a. 5
b. 10
c. 12
d. 17
e. 22

4. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
void print(int a)
{
	if(a==0)
	{
		cout<<endl;
		return;
	}
	else
	{
		cout<<a%10<<" ";
		return print(a/10);
	}
}

int main()
{
	int i=4562;
	print(i);
	return 0;
}
```

a. 2 6 5 4
b. 4 5 6 2
c. 5 6 4 2
d. 2 4 5 6
e. 6 5 4 2

5. Aşağıdakilerden hangisinde fonksiyon prototipi yanlış tanımlanmıştır?

a. void yazdir();
b. int sorular();
c. void soru yazdir(int c);
d. int yazdir(int a);
e. void yazdir(int a, int b);

6. Aşağıdakilerden hangisi C++ programlama dilinde int hesaplama(int a, int b) prototipine sahip bir fonksiyonun aşırı yükleme eşlerinden birisi olabilir?

a. int ortalamaHesaplama(int a, int b)
b. double hesaplama1(double a, double b)
c. int hesaplama(int a, int b)
d. int hesaplama1(int a, int b)
e. double hesaplama(double a, double b)

7. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

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

int main()
{
    int a=25;
    cout<<pow(sqrt(a),3)<<endl;
    return 0;
}
```

a. 5 b. 25
c. 125 d. 250
e. 625

8. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
int i=1;
int main()
{
    cout<<i<<" ";
    i=2;
    cout<<i<<" ";
    {
        int i=3;
        cout<<i<<" ";
    }
    cout<<i<<endl;
    return 0;
}
```

a. 2 1 2 3
b. 1 2 3 2
c. 3 1 2 2
d. 1 3 2 2
e. 2 1 2 3

9. Aşağıdakilerden hangisinde C++ programlama dilinde double hesaplama(int a, int b) prototipine sahip bir fonksiyon doğru bir şekilde çağrılmamıştır?

a. int d=heseplama(4,5);
b. double w=hesaplama(3,2);
c. cout<<hesaplama(5,8)<<endl;
d. double k=hesaplama(1,2) + 4.0;
e. cout<<5.7 + hesaplama(9,3)<<endl;

10. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
void fonksiyon1(int &a)
{
    a=a+10;
}
int fonksiyon2(int a)
{
    int b=a;
    fonksiyon1(b);
    return b;
}
int main()
{
    int i=5;
    cout<<fonksiyon2(i)<<endl;
    return 0;
}
```

a. 5 b. 10
c. 15 d. 20
e. 25

Yanıt Anahtarı
1. d 2. a 3. b 4. a 5. c
6. e 7. c 8. b 9. a 10. c

KAYNAKÇA

Deitel, P., Deitel, H. (2017). C++ How to Program. 10. Baskı. Pearson. Essex, İngiltere.

Dikici, M. (2017). C++ Programlama Dili: Yapısal Programlama, Nesnelerle Programlama ve Jenerik Programlama. Seçkin Yayıncılık. İstanbul, Türkiye.

Çobanoğlu, B. (2021). C/C++ Programlama Eğitim Kitabı. Kodlab. İstanbul, Türkiye.

Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . . Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.

Kaleli, C., Bilge, A., Ergin, S., & Şora Günal, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.

FAYDALI KAYNAKLAR VE İNTERNET KAYNAKLARI

https://en.cppreference.com/w/

https://www.w3schools.com/cpp/default.asp