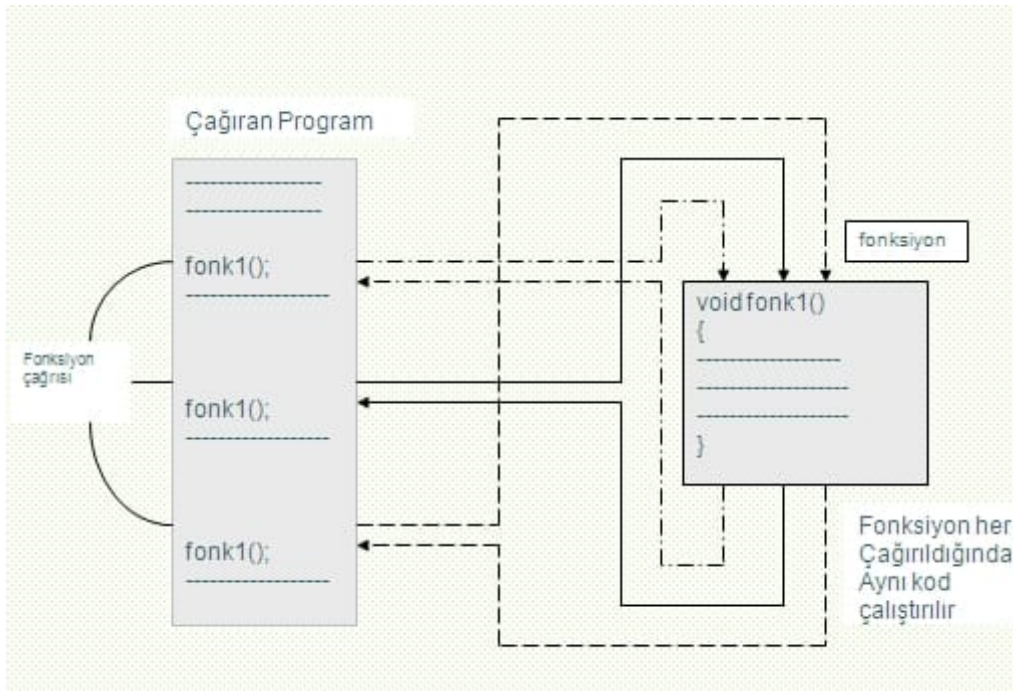


7. FONKSİYONLAR

Giriş

Bir fonksiyon, bir veya birkaç program ifadesini gruplar ve bu gruba bir isim verir. Bu program ifadeleri grubu, daha sonra programın başka bölümleri tarafından çağırılabilir. Fonksiyonların kullanılmasının ilk nedeni programın kavramsal olarak düzenlenmesi ihtiyacından ortaya çıkmıştır. Programların fonksiyonlar halinde yazılması yapısal programlamanın da temelidir.

Bundan başka, programları fonksiyonlar halinde yazılırsa daha kısa kod yazarak işlemlerinizi yürütebilirsiniz. Bir fonksiyon, programın içerisinde birkaç defa çağırılıp çalıştırılabilir. Dolayısıyla bu daha kısa, daha verimli program yazmamızı sağlar. Şekil 7.1’de bir programda, fonksiyonun değişik bölümler tarafından nasıl çağırıldığı gösterilmiştir.



7.1. Basit Fonksiyonlar

7.1.1. Program 7.1

İçerisinde ekrana 50 adet “=” karakteri basan bir fonksiyon bulunan bir C++ Programı yazınız.

```
#include<iostream>

using namespace std;

void cizgi(); //fonksiyon deklarasyonu(prototip)

int main()

{

cizgi();

cout<<"Veri Tipi"<<" "<<"boyut"<<endl;

cizgi();

cout<<"char -128 128"<<endl

<<"short -32 768 32 767" <<endl

<<"int sistemlere baęlı " <<endl

<<"long -2 147 483 648 2 147 483 647"<<endl;

cizgi();

return 0;

}

void cizgi() //fonksiyon tanımı

{

for(int i=0;i<=50;i++)

cout<<"=";

cout<<endl;

}

}
```

Program 7.1'in C++ kodu

```
=====
Veri Tipi          boyut
=====
char               -128          128
short              -32 768        32 767
int                sistemlere baęlı
long               -2 147 483 648  2 147 483 647
=====
Press any key to continue_
```

Program 7.1'in ekran ıktısı

Yukarıdaki program iki fonksiyondan oluşmaktadır: main() ve cizgi(). Şimdiye kadar programlarımızda yalnızca tek bir fonksiyon; main fonksiyonu yer alıyordu. Bu uygulamada iki fonksiyon kullanılmıştır. İkinci fonksiyonu; cizgi() fonksiyonunu programa dâhil etmek için üç temel bileşen kullanılmıştır. Bu bileşenler fonksiyonun deklare edilmesi, fonksiyonun çağırılması ve fonksiyonun tanımıdır.

7.1.2. Fonksiyonların Deklare edilmesi

Fonksiyonlarda değişkenler gibi önce derleyiciye bildirilir ve daha sonra kullanılabilir. Bunun iki yolu vardır. Birinci yol fonksiyonu önce deklare etmektir. Diğer yol ise fonksiyonu tanımlamaktır. Yukarıdaki örnekte birinci yol tercih edilmiş; çizgi fonksiyonu önce deklare edilmiştir.

void çizgi();

Bu deklarasyon derleyiciye, daha sonraki bir noktada çizgi isminde bir fonksiyon sunmayı planladığımızı anlatır. void anahtar kelimesi fonksiyonun döndürdüğü bir değer olmadığını, boş parantezler ise fonksiyonun argüman almadığını bildirir. Fonksiyon deklarasyonu noktalı virgül ile biter. Yani fonksiyon bildirimi bir ifadedir.

Ayrıca, fonksiyon deklarasyonu bir prototiptir. Çünkü fonksiyon deklarasyonu, fonksiyonun bir taslağını, modelini sunar.

7.1.3. Fonksiyonu Çağırarak

7.1 'de verilen programı incelediğimizde, çizgi() fonksiyonunun main() fonksiyonu içinden üç kez çağırıldığını görüyoruz. Bu üç çağırının her biri aşağıdaki şekilde görünür:

çizgi();

Çağırının söz dizimi, deklarasyonda kullanılan ifade ile aynıdır. Çağrı noktalı virgül ile sona erer. Çağrı ifadesini çalıştırmak fonksiyonu çalıştırmak anlamına gelir; yani bu durumda kontrol fonksiyona geçer. Fonksiyonun çalışması bittiğinde kontrol fonksiyon çağırısını takip eden ifadeye geçer.

7.1.4. Fonksiyonun Tanımı

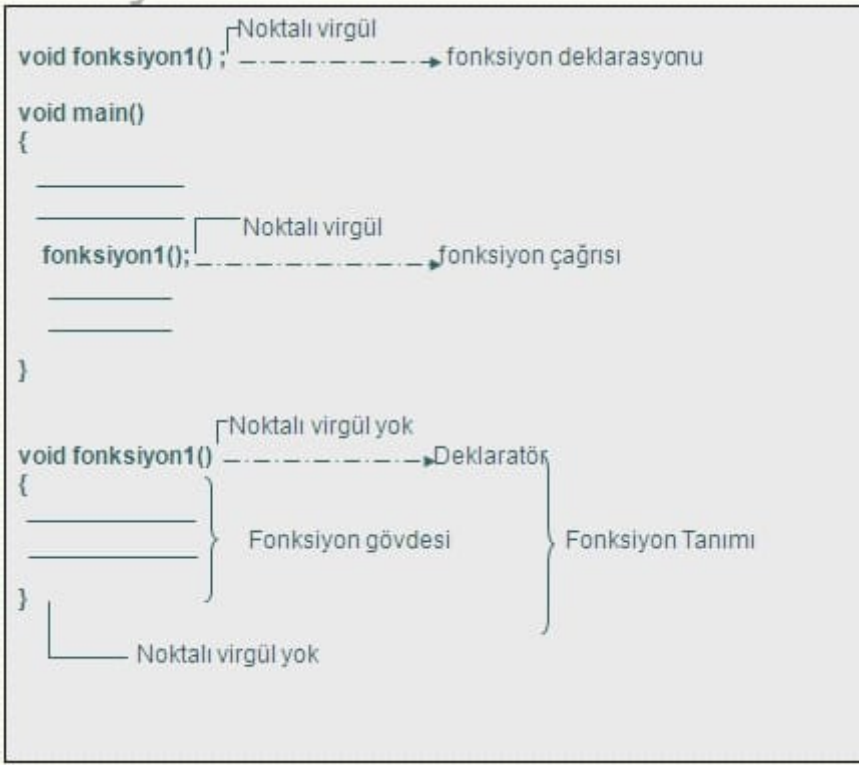
Fonksiyon tanımı, fonksiyonun asıl kodunu içerir. Çizgi() fonksiyonunun tanımı aşağıdaki gibidir.

```
void çizgi()
{
    for(int i=0;i<=50;i++)
        cout<<" ";
    cout<<endl;
}
```

Fonksiyon tanımı

Tanım deklarator olarak adlandırılan bir satırdan ve bunu takip eden fonksiyon gövdesinden ibarettir. Fonksiyon gövdesi, küme parantezleri ile sınırlandırılmış, fonksiyonu oluşturan ifadelerden meydana gelir.

Burada dikkat edilmesi gereken husus, deklaratorün deklarasyonla uyum sağlamak zorunluluğudur. Deklarator, aynı fonksiyon ismini kullanmalı, aynı tipte ve aynı sırada argüman içermeli(Eğer argüman kullanılmışsa) ve döndürdüğü değer aynı tipte olmalıdır. Aşağıda fonksiyon deklarasyonu, fonksiyon çağırısı ve fonksiyon tanımının söz dizimi verilmiştir.



Fonksiyon çağrıldığında kontrol, fonksiyonun gövdesindeki ilk ifadeye aktarılır. Sonra fonksiyon gövdesindeki diğer ifadeler çağırılır. Kapanan parantezle karşılaşılınca kontrol fonksiyonu çağırın programa geri döner.

7.1.5. Deklarasyondan Kurtulmak

Fonksiyon tanımını, program listesi içinde, fonksiyona yapılan ilk çağrıdan önce yazarak fonksiyon deklarasyonundan kurtulabiliriz. Bu yaklaşım kısa programlar için daha basit olabilir. Yukarıda verilen programın deklarasyon kaldırılarak yazılan versiyonu aşağıda verilmiştir.

```

#include<iostream>

using namespace std;

void cizgi()
{
    for(int i=0;i<=50;i++)
        cout<<"=";
    cout<<endl;
}

int main()
{
    cizgi();

    cout<<"Veri Tipi"<<" "<<"boyut"<<endl;

    cizgi();

    cout<<"char -128 128"<<endl
    <<"short -32 768 32 767" <<endl
    <<"int sistemlere bagli " <<endl
    <<"long -2 147 483 648 2 147 483 647"<<endl;

    cizgi();

    return 0;
}

```

Program 7.1'in deklarasyon kullanmadan yazılan versiyonu



The screenshot shows a Windows command prompt window with the title "C:\Users\arge\Desktop\D÷ng³ler_Kararlar\Debug\Cpp5.exe". The output of the program is as follows:

```

=====
Veri Tipi          boyut
=====
char               -128          128
short              -32 768        32 767
int                sistemlere bagli
long               -2 147 483 648  2 147 483 647
=====
Press any key to continue

```

Yukarıdaki programın ekran çıktısı

7.2. Fonksiyonlara Argüman Aktarmak

Argüman, bir programdan bir fonksiyona aktarılan veri parçasıdır. Argümanlar bir fonksiyonun değişik değerlerle çalışmasına imkân verir, hatta onu çağıran programın isteklerine bağlı olarak değişik işler bile

yapabilir.

7.2.1. Sabitleri Aktarmak

Yukarıdaki fonksiyonu her defasında 50 '=' basan bir fonksiyon yerine herhangi bir karakteri herhangi bir sayıda basacak bir fonksiyon haline getirebiliriz. Aşağıda, sabitleri argüman olarak aktaran bir program verilmiştir.

```
#include<iostream>

using namespace std;

void cizgi(char, int);

int main(){

    cizgi('+',50);

    cout<<"Veri Tipi"<<" "<<"boyut"<<endl;

    cizgi('-',25);

    cout<<"char -128 128"<<endl|

    <<"short -32 768 32 767" <<endl

    <<"int sistemlere bagli " <<endl

    <<"long -2 147 483 648 2 147 483 647"<<endl;

    cizgi('*',50);

    return 0;

}

void cizgi(char ch, int n)

{

    for(int i=1;i<=n; i++)

        cout<<ch;

    cout<<endl;

}
```

Fonksiyonlara sabitleri argüman olarak aktaran program

```

C:\Users\arge\Desktop\D+ngiler_Kararlar_Debug\Cpp4.exe
*****
Veri Tipi          boyut
-----
char               -128          128
short              -32 768        32 767
int                sistemlere bagli
long               -2 147 483 648  2 147 483 647
*****
Press any key to continue

```

Fonksiyonlara sabitleri argüman olarak aktaran programın ekran çıktısı

7.2.2. Değişkenleri Argüman Olarak Aktarmak

Aşağıdaki örnekte karakterler ve karakterlerin kaç kez tekrarlanacağını kullanıcının belirlemesine izin veren bir örnek verilmiştir. Argüman olarak kullanılan değişkenlerin veri tipleri, tıpkı sabitlerde olduğu gibi, fonksiyon deklarasyonunda ve tanımında belirtilen tiplerle eşleşmelidir.

```

#include<iostream>

using namespace std;

void cizgi(char,int);

int main()
{
    char ch; int n;

    cout<<"karakter giriniz.....: ";
    cin>>ch;

    cout<<"karakter sayisi giriniz...: ";
    cin>>n;

    cizgi(ch,n);

    return 0;
}

void cizgi(char fch, int fn)
{
    for(int i=1;i<=fn; i++)

        cout<<fch;

        cout<<endl;
}

```

Fonksiyonlara değişkenleri argüman olarak aktaran program

```
C:\Users\arge\Desktop\Denetimler_Kararlar_Debug\Cpp3.exe
karakter giriniz.....: &
karakter sayisi giriniz..: 4
&&&&
Press any key to continue
```

Fonksiyonlara sabitleri argüman olarak aktaran programın ekran çıktısı

7.3. Fonksiyonlardan Değer Döndürmek

Bir fonksiyon çalışmasını tamamladıktan sonra, kendisini çağıran programa tek bir değer döndürebilir. Genellikle döndürülen bu değer, fonksiyonun çözdüğü problemin cevabını içerir. Aşağıdaki programdaki fonksiyon değer döndüren bir fonksiyondur.

```
#include<iostream>

using namespace std;

float fcevre(float);

int main()
{
    float kenar, cevre;

    cout<<"karenin kenarini giriniz..: ";

    cin>>kenar;

    cevre=fcevre(kenar);

    cout<<endl<<"karenin çevresi = "<<cevre<<endl;

    return 0;
}

float fcevre(float a)
{
    float sonuc;

    sonuc=4*a;

    return sonuc;
}
```

Fonksiyonlardan değer döndüren program


```

C:\Users\arge\Desktop\D+ng³ler_Kararlar_Debug\Cpp2.exe
karenin kenarini giriniz...: 4
karenin evresi = 16
Press any key to continue

```

Fonksiyonlardan değ r d nd ren programın ekran  ıktısı

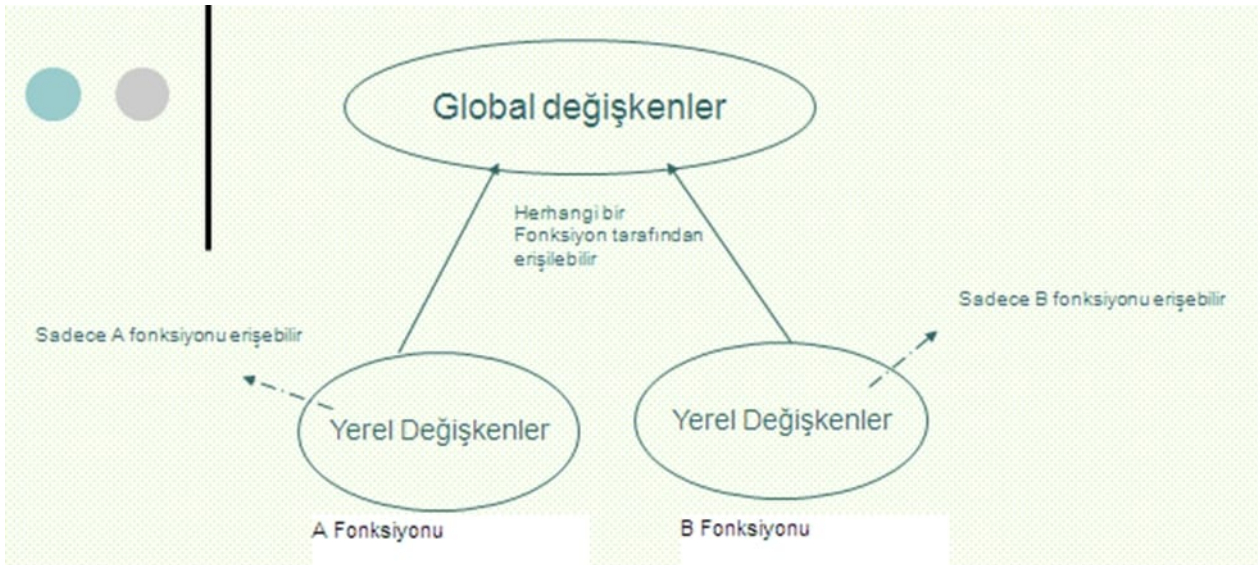
Bir fonksiyon bir değ r d nd recekse, bu değ rin veri tipi a ık a belirlenmelidir. Fonksiyon deklarasyonu bunu, veri tipini fonksiyon deklarasyonunda ve tanımında fonksiyon isminden  nce yerle tirerek ger ekle tirir.

7.3.1. return İfadesi

fcevre() fonksiyonuna bir arg man aktarılır ve bu değ r **a** parametresinde saklanır. Fonksiyon i erisindeki ifadeler  alı tırılır. Hesaplanan değ r **sonuc** deėi keninde saklanır. Bu deėi kenin değ ri daha sonra, fonksiyonu  aėıran main() fonksiyona return ifadesi ile d nd r l r.

return sonuc;

Hem main() fonksiyonunda hem de fcevre() fonksiyonunda cevre deėi kenini saklamak i in bir yerin mevcut olduėuna dikkat ediniz. fcevre fonksiyonunda sonuc deėi keni main fonksiyonunda da cevre deėi keni hesaplanan değ ri saklar. Fonksiyon d nd ė  zaman sonuc i indeki değ r cevre i ine kopyalanır. Fonksiyonu  aėıran program, fonksiyonun i indeki sonuc deėi kenine eri emez. Sadece deėi kenin değ ri d nd r l r. Bu i lem a aėıda g sterilmi tir.



Bir fonksiyona bir ok arg man aktarılırken, fonksiyondan sadece bir değ r d ner. Fonksiyonlardan birden fazla değ r d nd rmenin ba ka y ntemleri vardır. Ancak bu y ntemler ileri C++ konuları i inde yer alır.

7.3.2. Gereksiz Deėi kenlerden Kurtulmak

Yukarıdaki programda programın daha iyi anla ılması i in kullanılan, ancak gerekli olmayan bir ak tane deėi ken vardır. A aėıdaki programda gereksiz deėi kenler  ıkartılmı tır.

```

#include<iostream>

using namespace std;

float fcevre(float);

int main()
{
    float kenar;

    cout<<"karenin kenarini giriniz..: ";

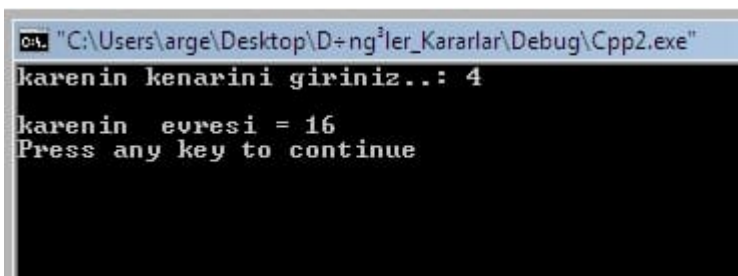
    cin>>kenar;

    cout<<endl<<"karenin çevresi = "<<fcevre(kenar)<<endl; //fcevre fonksiyonu cout'a
    doğrudan eklenir.

    return 0;
}

float fcevre(float a)
{
    return 4*a; // dogrudan deger döndürülür
}

```



```

C:\Users\arge\Desktop\D+ng³ler_Kararlar\Debug\Cpp2.exe
karenin kenarini giriniz..: 4
karenin çevresi = 16
Press any key to continue

```

Gereksiz değişken kullanılmadan yazılan programın ekran çıktısı

main içerisindeki çevre değişkeni çıkartılmış ve bunun yerine `cout<<endl<<"karenin çevresi = "<<fcevre(kenar)<<endl;` ifadesi yazılmıştır. `fcevre()` fonksiyonundan sonuç değişkeni çıkartılmış ve doğrudan `return 4*a;` ifadesi yazılmıştır. Bu programın çalıştırılması sonucunda yukarıdaki programın çalıştırılmasından elde edilen sonuçlar elde edilir.

7.4. Yerel(Inline) Fonksiyonlar

Kısa fonksiyonlar kullanıldığında, programın çalışmasını hızlandırmak için fonksiyon gövdesini doğrudan fonksiyonu çağıran programın kodunun içine yerel(inline) olarak yerleştirmek daha iyi olabilir. Bu şekilde daha hızlı çalışan programlar elde edebiliriz. Bunun için yalnızca yazılan fonksiyonun inline olduğunu belirtmek yeterlidir. Derleyici derleme sırasında inline fonksiyonu çağıran fonksiyonun içerisine yerleştirir.

```
#include<iostream>

using namespace std;

inline float ortalama(float vize, float final)
{
    return 0.4*vize+0.6*final;
}

int main()
{
    int vize, final;

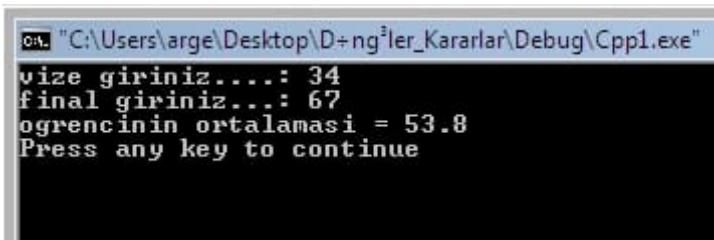
    cout<<"vize giriniz.....: ";cin>>vize;

    cout<<"final giriniz....: ";cin>>final;

    cout<<"öğrencinin ortalaması = "<<ortalama(vize,final)<<endl;

    return 0;
}
```

İçerisinde inline fonksiyon bulunan C++ programı

A screenshot of a Windows command prompt window. The title bar shows the file path: "C:\Users\arge\Desktop\D+ng³ler_Kararlar\Debug\Cpp1.exe". The command prompt shows the following text: "vize giriniz.....: 34", "final giriniz....: 67", "öğrencinin ortalaması = 53.8", and "Press any key to continue". The user has entered 34 for the exam score and 67 for the final score, and the program has calculated the average as 53.8.

```
CA. "C:\Users\arge\Desktop\D+ng³ler_Kararlar\Debug\Cpp1.exe"
vize giriniz.....: 34
final giriniz....: 67
öğrencinin ortalaması = 53.8
Press any key to continue
```

İçerisinde inline fonksiyon bulunan C++ programın ekran çıktısı

7.5. Fonksiyonların Aşırı Yüklenmesi

Aşırı yüklenen fonksiyonlar konusu fonksiyonların ilgi çeken konularından biridir. Fonksiyonların aşırı yüklenmesi, aynı isimdeki fonksiyonların farklı parametre sayısı veya farklı parametre tipinden yararlanılarak programların içerisine birden fazla yazılmasını sağlar. Programların içerisinde aşırı yüklenen fonksiyonların yazılması programların okunabilirliğini artırır. Aşağıda, içerisinde aşırı yüklenen fonksiyon bulunan bir program gösterilmiştir.

```
/* Fonksiyonlar aşırı yükleniyor */  
#include <iostream>  
using namespace std;  
void cizgiciz();  
void cizgiciz(char);  
void cizgiciz(char, int);  
int main()  
{  
    cizgiciz();  
    cizgiciz('-');  
    cizgiciz('=', 20);  
  
    return 0;  
}  
void cizgiciz()  
{  
    cizgiciz('*', 10);  
}  
void cizgiciz(char k)  
{  
    cizgiciz(k, 10);  
}  
void cizgiciz(char k, int s)  
{  
    for (int i = 0; i < s; i++) cout << k;  
    cout << endl;  
}
```

Fonksiyonların aşırı yüklenmesi

7.6. Kapsam(Scope) ve Depolama Sınıfı

Bir *değişkenin kapsamı*, değişkene programın hangi bölümlerinin erişebileceğini belirler. Değişkenin *depolama sınıfı* ise değişkenin ne kadar süre ile var olacağını gösterir. Aşağıda, programlarda kullanılan değişkenler bu açıdan ele alınmıştır.

7.6.1. Yerel Değişkenler

Bir fonksiyonun içinde tanımlanan değişkenlere yerel(local) değişkenler denir. Çünkü bu değişkenler yerel kapsama sahiptir. Şimdiye kadar yazdığımız programlarda kullanılan değişkenlerin tamamı kullanıldıkları fonksiyonun gövdesi içinde tanımlanmıştır. Bir yerel değişken, içinde tanımlandığı fonksiyon çağırılana kadar ortaya çıkmaz. Yani bu değişkenler için fonksiyon çağırılana kadar bellekte yer ayrılmaz. Kontrol, yerel değişkenin tanımlı olduğu fonksiyona geçtiğinde değişken ortaya çıkar; yani bellekte değişken için yer ayrılır. Kontrol tekrar fonksiyonu çağırarak programa geçtiğinde, değişkenler ortadan kaldırılır ve değerleri yitirilir. Bu değişkenlere *otomatik* ismi de verilir. Çünkü değişkenler fonksiyon çağırıldığında otomatik olarak oluşturulur ve fonksiyon geri döndüğünde otomatik olarak ortadan kaldırılır. Değişkenlerin oluşturulması ve yok edilmesi arasında geçen süreye yaşam süresi denir. Yerel bir değişkenin yaşam süresi, içinde tanımlı olduğu fonksiyonun çalışma süresi ile aynıdır. Değişkenlerin yaşam sürelerinin bu şekilde kısıtlanmış olması, bellek alanını daha verimli olarak kullanılması sonucunu doğurur.

Bir değişkenin kapsamı(Scope) değişkenin erişilebilirliği olarak adlandırılır. Bir fonksiyon içinde tanımlı değişkenler, sadece tanımlı oldukları fonksiyon içinde görülebilirler. Bu durum, bir fonksiyon içerisinde tanımlı değişkenin değerinin, başka bir fonksiyon tarafından değiştirilememesi sonucunu doğurur. Yani güvenliği artırır.

7.6.2. Global Değişkenler

Global değişkenler bir fonksiyonun dışında tanımlanan değişkenlerdir. Bir global değişkene dosya bir dosya içerisindeki bütün fonksiyonlar tarafından erişilebilir. Bu değişkenlere bazen harici(External) değişkenler de denir. Çünkü bu değişkenler herhangi bir fonksiyonun dışında tanımlanır. Aşağıda global değişken kullanan bir program verilmiştir.

```
#include<iostream>

using namespace std;

#include<conio.h>

char ch = 'a';

void oku();

void yaz();

int main(){

    while(ch != '\r') {

        oku();

        cout<<endl<<endl<<endl<<endl;

        yaz();

        cout<<endl<<endl;

    }

    return 0;

}

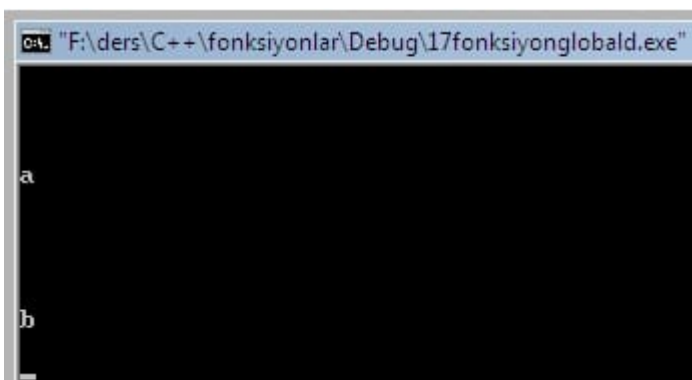
void oku() {

    ch=getch();

}

void yaz() {

    cout<<ch; }
```



İçerisinde global değişken tanımlanmış programın ekran çıktısı

Eğer bir global değişkene ilk değer atanmış ise:

```
int sayi=15;
```

değer atama işlemi, program ilk kez yüklendiğinde yapılır. Eğer global değişkene program içerisinde herhangi bir değer atanmamış ise bu durumda değişkenin değeri sıfır olur. Yerel değişkenlere değer atanmadığı zaman, değerlerinin rastgele, anlamsız birer değer olduğunu hatırlayınız.

Global değişkenler programın yaşam süresi boyunca varlıklarını sürdürürler. Program çalıştırıldığında global değişken için bellekte yer ayrılır ve bu yer, program çalıştığı sürece bu değişken için ayrılır. Yine bu değişkenler tanımlandığı dosya boyunca erişilebilirdir.

7.6.3. Statik Yerel Değişkenler

Statik yerel değişkenler, otomatik yerel değişkenler ile aynı erişilebilirliğe sahiptir. Ancak statik yerel değişkenlerin yaşam süresi global değişkenlerle aynıdır. Yaşam süresi açısından bakıldığında statik yerel değişkenlerin global değişkenlerden farkı global değişken program çalıştırıldıktan hemen sonra ortaya çıkar, statik yerel değişken ise tanımlı olduğu fonksiyon çağırılana kadar ortaya çıkmaz. Aşağıda statik yerel içerisinde statik yerel değişken tanımlı bir C++ programı verilmiştir.

```
#include<iostream>

using namespace std;

float ortalama(float);

int main() {

    float sayi=1,ort;

    while(sayi !=0) {

        cout<<"sayi giriniz...: "; cin>>sayi;

        if(sayi!=0){

            ort=ortalama(sayi);

            cout<<"yeni ortalalama= "<<ort<<endl; } }

    return 0;

}

float ortalama(float yenisayi) {

    static float toplam=0;

    static int sayac=0;

    sayac++;

    toplam+=yenisayi;

    return toplam/sayac;

}
```

```
cmd: "F:\ders\C++\fonksiyonlar\Debug\18fonksiyonstaticdegisken.exe"
sayi giriniz...: 15
yeni ortalalama= 15
sayi giriniz...: 34
yeni ortalalama= 24.5
sayi giriniz...: 5
yeni ortalalama= 18
sayi giriniz...: _
```

İçerisinde statik yerel değişken tanımlı programın ekran çıktısı

Bölüm Özeti

Bu bölümde fonksiyonların, yapısal programlamanın temel taşı olduğunu, programlar içerisinde yapılan tekrarlardan programcıları koruduğunu, fonksiyonlar halinde yazılan programların bilgisayarda çalışırken maliyetlerinin daha düşük olduğunu öğrendiniz.

Fonksiyon deklare etmeyi, tanımlamayı, çağırmayı, fonksiyonlara argüman aktarmayı, çağıran fonksiyona değer döndürmeyi örnekler çözerek öğrendiniz.

Ayrıca fonksiyonları aşırı yükleyerek daha okunabilir programlar yazma becerisi kazandınız. Yazacağınız fonksiyonların kısa olması durumunda bu fonksiyonları yerel olarak yazarak programlarınıza performans kazandırdınız.

Bunlara ilave olarak, bu bölümün sonunda, değişkenlerin program içerisinde tanımlandıkları yerlerin önemli olduğunu, yerel değişkenlere, sadece tanımlandıkları yerden ulaşılabilmesini, global değişkenlere ise bütün fonksiyonlardan ulaşılabilmesini gördük. Bunların hangilerini ne zaman tercih edeceğimiz konusunda bilgi edindik.