Fonksiyonlar

Problem çözmenin temel ilkelerinden biri de, problemi mümkün olduğu kadar çok parçalara ayırmaktır. Parçalara ayrılan kısımlar sanki birbirinden bağımsızmış gibi düşünülerek çeşitli çözümler elde edilebilir. Ayrıca bu parçaları istediğimiz kadar, tekrar tekrar kullanabiliriz. İşte C programlama dilinde bu parçalama işlemini fonksiyon kullanarak yaparız. Kısaca işlev(fonksiyon); belli bir işi gerçekleştiren program deyimlerinin, karmaşık programları düzenlemek ve basitleştirmek için, programın bir birimi olarak gruplandırılmasıdır diyebiliriz. Örneğin bir program yazdığımızda mazı matematiksel işlemleri defalarca kullanacağımızı düşünelim. Bunun için her seferinde bu matematiksel kodları yazmak yerine, kullanacağımız kodları bir fonksiyon olarak tanıtırız. Bu işlemlere her ihtiyacımız olduğunda fonksiyonu çağırarak işlemlerin gerçekleşmesini sağlayabiliriz.

İşlev kullanmanın temel nedeni; böl ve yönet metodu ile kolaylık sağlanması, ayrıca yazılımın tekrar tekrar kullanılabilmesidir.

Şimdi basit bir ev yapacak olursak:

```
#include
void main() {
    printf("    /\\   \n");
    printf("   /~~\\  \n");
    printf("  /~~~~\\ \n");
    printf(" /~~~~~~\\\n");
    printf("==========\n");
    printf("|########|\n");
    printf("|#  ##  #|\n");
    printf("|########|\n");
    printf("|########|\n");
    printf("==========\n");
}
```

void main() fonksiyonunu açıklayarak devam edelim. Void kısmı geriye dönen değer tipini belirtir. Void yerine int yazarsak main fonksiyonu geriye dönüş için bir tam sayı bekler. Sonuna return 0; yazarak int main de bu dönüşü tamamlamış oluyoruz. Void ise herhangi bir değer dönmeyeceği zaman kullanacağımız bir tanımdır.

main() fonksiyonu özel bir yapıdır. Hazırladığımız yazılım main() fonksiyonu ile çalışmaya başlar. main() fonksiyon bloğu içerisinde yer almayan kodlar çalışmaz.

Yukarıdaki evi fonksiyon kullanarak çizmeye çalışalım. Evin her bir bölümü problemimizin parçaları olsun. Evin çatısı, katları ve katları ayıran zemin parçasını ayrı ayrı tanıtalım.

ALGORİTMA VE PROGRAMLAMA

Evin çatısı için;

```
void cati() {
    printf("    /\\   \n");
    printf("   /~~\\  \n");
    printf("  /~~~~\\ \n");
    printf(" /~~~~~~\\\n");
}
```

Evin katları için;

```
void kat() {
    printf("|########|\n");
    printf("|#  ##  #|\n");
    printf("|########|\n");
    printf("|########|\n");
}
```

Ve evin katlarını ayıran zemin içinse;

```
void zemin() {
    printf("==========\n");
}
```

Şimdi ise 3 katlı bir bina çizelim

```
#include
void cati() {
    printf("    /\\   \n");
    printf("   /~~\\  \n");
    printf("  /~~~~\\ \n");
    printf(" /~~~~~~\\\n");
}

void kat() {
    printf("|########|\n");
    printf("|#  ##  #|\n");
    printf("|########|\n");
    printf("|########|\n");
}

void zemin() {
    printf("==========\n");
}

int main() {
    cati();
    zemin();
    kat();
    zemin();
    kat();
    zemin();
    kat();
    zemin();
    return 0;
}
```

Görüldüğü gibi eğer bunları fonksiyon tanıtmadan çizmek isteseydik, int main() bloğu içerisinde bir çok çizim yapmamız gerekecekti. Oysa evimizi parçalara ayırarak, kullanmak istediğimiz kısımları işlev olarak tanıttık. Ne kadar işlev kullanmak istiyorsak, int main bloğu içerisinde onu çağırmamız yeterli olacaktır.

Örnek: Bir tamsayının küpünü veren fonksiyonu hazırlayalım.

```c
#include <stdio.h>
main() {
    int sayi;
    int kub(int);
    printf("sayiyi gir ");
    scanf_s("%d", &sayi);
    printf("Kubu = %d\n", kub(sayi));
}

int kub(int i) {
    int islem;
    islem = i * i*i;
    return islem;
}
```

İşlevin Tanımlanma Biçimi

```
dönüş_tipi işlev_adı(parametreler)
{
yerel tanımlamalar
deyimler
}
```

dönüş_tipi: Eğer fonksiyon bir değer geri gönderecek ise değerin tipini belirtir. Belirtilmez ise int kabul edilir. Eğer fonksiyon değer göndermeyecek ise dönüş_tipi yerine void yazılır.

işlev_adı: Fonksiyon çağırılırken kullanılacak ad (belirleyici).

parametreler: Fonksiyon için gerekli değerleri içerir. Her parametre değişken tanımlar gibi tanımlanır. Herbirinin arasında ‘,‘ (virgül) kullanmak gerekir.

yerel tanımlamalar: Bu fonksiyona özgü(değişken,sabit) tanımlamalar.

return değer; Eğer fonksiyon bir değer gönderecek ise bu return deyimi ile yapılır.

Fonksiyonun (işlevin) Prototipi:

Tanımlana bir fonksiyonun ana modül içerisinde prototipinin yazılması gerekir. Prototip ile fonksiyonun dönüş değeri ve aldığı parametrelerin tipleri tanımlanır. Bu bilgiye göre C derleyecisi fonksiyonu çağırdığında değerlerin uygun olduğunu sınar.

Yerel (local) ve Genel (global) Değişkenler

Genel değişken (global): Her yerde (main ve diğer işlevler) geçerlidir, değerine erişilebilir. Programın en başında, main işlevinin dışında tanımlanır.

Yerel değişken (local): Sadece tanımlandığı modülde geçerlidir, değerine erişilebilir.Modül içinde tanımlanır.

```
int a; /*tüm işlevlerden değerine erişilebilir, değeri değiştirilebilir. (global) */

main()

{

int b; /* sadece main işlevi içerisinden erişilebilir (local ) */

...

}

fonksiyon1(...);

{

int b; /* sadece fonksiyon1 işlevi içerisinden erişilebilir. main işlevindeki değişkenden bağımsızdır (local ) */

int c; /* sadece fonksiyon1 işlevi içerisinden erişilebilir (local ) */

...

}
```

Örnek: 1 den N e kadar olan sayıların toplamını fonksiyon ile bulan program

```
1   #include <stdio.h>
2   sayi(int sayi_1) {
3       int i, sonuc = 0;
4       for (i = 0; i <= sayi_1; i++) {
5           sonuc += i;
6       }
7       printf(" Toplam = %d", sonuc);
8   }
9   int main() {
10      int a, toplam;
11      printf("Sayiyi girin > ");
12      scanf_s("%d", &a);
13      sayi(a);
14      return 0;
15  }
```

Örnek: Klavyeden girilen 2 sayının çarpımını, küplerini, faktöriyelini, büyük sayıyı fonksiyon kullanarak bulan program.

```c
#include <stdio.h>

int cizgi() {
    int i;
    for (i = 0; i < 30; i++) {
        printf("_");
    }
    printf("\n");
}

/**********************/
carpim(int a, int b) {
    int carp;
    carp = a*b;
}

/**********************/
kup(int a, int b) {
    int kup_a, kup_b;
    kup_a = a * a*a;
    kup_b = b * b *b;
    printf("\n%d sayisinin kupu = %d", a, kup_a);
    printf("\n%d sayisinin kupu = %d", b, kup_b);
}

/**********************/
faktor(int a, int b) {
    int i, j, sonuc_a = 1, sonuc_b = 1;
    for (i = 1; i <= a; i++) {
        sonuc_a *= i;
    }
    printf("\n%d sayisinin faktoriyeli = %d", a, sonuc_a);
    for (j = 1; j <= b; j++) {
        sonuc_b *= j;
    }
    printf("\n%d sayisinin faktoriyeli = %d", b, sonuc_b);
}
/**********************/
```

```
39  buyuk(int a, int b) {
40      int buyuk_a, buyuk_b;
41      if (a > b) {
42          printf("%d > %d", a, b);
43      }
44      else if (a < b) {
45          printf("%d > %d", b, a);
46      }
47      else if (a == b) {
48          printf("%d = %d", a, b);
49      }
50      else
51          printf("Hatali giris yaptiniz.");
52  }
53
54  /*********************/
55  int main() {
56      int sayi_1, sayi_2;
57      printf("Lutfen ilk sayiyi giriniz > ");
58      scanf_s("%d", &sayi_1);
59      printf("Lutfen ikinci sayiyi giriniz > ");
60      scanf_s("%d", &sayi_2);
61      cizgi();
62      printf("\n");
63      printf("\n%d ve %d sayilarinin carpimi = %d\n", sayi_1, sayi_2, carpim(sayi_1, sayi_2));
64      printf("\n");
65      cizgi();
66      printf("\n");
67      kup(sayi_1, sayi_2);
68      printf("\n");
69      cizgi();
70      printf("\n");
71      faktor(sayi_1, sayi_2);
72      printf("\n");
73      cizgi();
74      printf("\n");
75      buyuk(sayi_1, sayi_2);
76      printf("\n");
77      cizgi();
78      printf("\n");
79      return 0;
80  }
```