Yapılandırılmış Program Geliştirme

Bir sorunu çözmek için bir program yazmadan önce, sorunu tam olarak anlamanız ve dikkatle planlanmış bir çözüm yaklaşımına sahip olmanız gerekir. 3. ve 4. Bölümler yapılandırılmış bilgisayar programları geliştirmeyi tartışıyor.

Algoritmalar

Herhangi bir bilgi işlem sorununun çözümü, bir dizi eylemi belirli bir sırayla gerçekleştirmeyi içerir. Algoritma, bir problemi çözmek için bir prosedürdür. Kısaca özetlersek;

1. yürütülecek eylemler ve

2. bu eylemlerin yürütülmesi gereken sıra.

Aşağıdaki örnek, eylemlerin yürütülmesi gereken sırayı doğru bir şekilde belirtmenin önemli olduğunu göstermektedir. Bir kişinin yataktan kalkıp işe gitmek için izlediği algoritmayı düşünün:

1. Yataktan kalkın,

2. pijamalarını çıkar,

3. duş al,

4. giyin,

5. kahvaltı yapın ve

6. işe giderken araba paylaşımı yap.

Bir bilgisayar programında da ifadelerin yürütüleceği sırayı belirlemeye program kontrolü denir. Bu ve bir sonraki bölümde, C'nin program kontrol yeteneklerini inceleyeceğiz.

Sözde kod (Pseudocode)

Sözde kod, günlük konuşma diline benzer. Yapılandırılmış C programlarına dönüştürmeden önce algoritmalar geliştirmenize yardımcı olan gayri resmi bir yapay dildir. Sözde kod kullanışlı ve kullanıcı dostudur. Bir programı bir programlama dilinde yazmadan önce "düşünmenize" yardımcı olur. Bilgisayarlar sözde kod yürütmez. Sözde kod tamamen karakterlerden oluşur, bu nedenle onu herhangi bir metin düzenleyicide yazabilirsiniz. Çoğunlukla, dikkatlice hazırlanmış sözde kodu C'ye dönüştürmek, bir sözde kod ifadesini C eşdeğeriyle değiştirmek kadar basittir. Sözde kod, sözde kodu C'ye dönüştürüp programı çalıştırdığınızda yürütülecek eylemleri ve kararları açıklar.

Tanımlar yürütülebilir ifadeler değildir; yalnızca derleyiciye gönderilen mesajlardır. Örneğin, tanım

```
int i = 0;
```

derleyici değişkeni i'nin türünü söyler, derleyiciye değişken için bellekte yer ayırmasını söyler ve onu “0” olarak başlatır. Ancak bu tanım, program yürütüldüğünde girdi, çıktı, hesaplama veya karşılaştırma gibi bir eylem gerçekleştirmez. Bu nedenle, bazı programcılar sözde kodlarına tanım eklemezler. Her bir değişkeni listelemeyi ve amacından kısaca bahsetmeyi seçer.

Kontrol Yapıları

Normalde, bir programdaki ifadeler, yazdığınız sırayla birbiri ardına yürütülür. Buna sıralı yürütme denir. Yakında göreceğiniz gibi, çeşitli C ifadeleri, yürütülecek bir sonraki ifadenin sıradaki bir sonraki ifadeden farklı olabileceğini belirlemenizi sağlar. Buna kontrolün devri denir. 1960'larda, kontrol transferlerinin ayrım gözetmeden kullanılmasının, yazılım geliştirme gruplarının yaşadığı birçok zorluğun temeli olduğunu gösterdi.

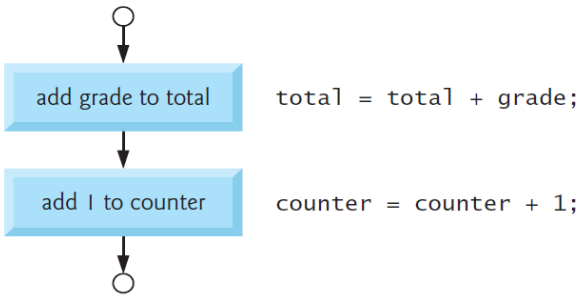
Bu problemlerin yaşanmasında “goto” deyiminin sorumlu olduğu gösterildi. Bir programdaki birçok olası hedeften birine kontrolün transferini belirtmenize izin veren “goto” deyiminin suçlu olduğuna işaret edildi. Sözde yapısal programlama kavramı, "goto eleme" ile neredeyse eşanlamlı hale geldi.

Böhm ve Jacopini'nin araştırması, programların herhangi bir “goto” deyimi olmadan yazılabileceğini gösterdi. Dönemin zorluğu, programcıların stillerini "goto'suz programlamaya" kaydırmalarıydı. Programlama mesleğinin yapılandırılmış programlamayı ciddiye almaya başlaması 1970'lere kadar sürdü. Yazılım geliştirme grupları, geliştirme sürelerinin azaldığını, sistemlerin daha sık zamanında teslim edildiğini ve yazılım projelerinin daha sık bütçe dahilinde tamamlandığını bildirdiğinden, sonuçlar etkileyiciydi. Yapılandırılmış tekniklerle üretilen programlar daha netti, hata ayıklaması ve değiştirmesi daha kolaydı ve ilk etapta hatasız olma olasılığı daha yüksekti. Böhm ve Jacopini'nin çalışması, tüm programların üç kontrol yapısı, yani sıra yapısı, seçim yapısı ve yineleme yapısı açısından (“sequence structure”, “selection structure” ve “iteration structure”) yazılabileceğini gösterdi. Sıra yapısı basittir—aksi belirtilmediği sürece, bilgisayar C ifadelerini yazıldıkları sırayla birbiri ardına yürütür.

Akış şemaları

Akış şeması, bir algoritmanın veya bir algoritmanın bir kısmının grafiksel bir temsilidir. Akış çizgileri adı verilen oklarla birbirine bağlanan dikdörtgenler, karolar, yuvarlatılmış dikdörtgenler ve küçük daireler gibi belirli özel amaçlı sembolleri kullanarak akış şemaları çizersiniz.

Çoğu programcı tarafından sözde kod tercih edilmesine rağmen, akış şemaları algoritmaları geliştirmenize ve temsil etmenize yardımcı olur. Akış şemaları, kontrol yapılarının nasıl çalıştığını açıkça gösterir. Bir sınavda sınıf ortalamasını hesaplayan bir algoritmanın bir bölümündeki bir sıra yapısı için aşağıdaki akış şemasını göz önünde bulundurun:



Dikdörtgen (veya eylem) sembolü, hesaplama, girdi veya çıktı gibi herhangi bir eylemi gösterir. Akış çizgileri, eylemlerin gerçekleştirileceği sırayı gösterir. Bu program bölümü önce toplama not ekler, ardından sayaca 1 ekler. Birazdan göreceğimiz gibi, bir programda herhangi bir yere tek bir eylem yerleştirilebilir, birkaç işlemi sırayla yerleştirebilirsiniz. Tam bir algoritma için bir akış şeması çizerken, ilk sembol “Başla” içeren yuvarlak bir dikdörtgen semboldür ve sonuncusu “Bitiş” içeren yuvarlak bir dikdörtgendir.

C'deki Seçim İfadeleri

C, ifadeler biçiminde üç tür seçim yapısı sağlar:

• "**if**" tek seçim ifadesi (Bölüm 3.5), yalnızca bir koşul doğruysa bir eylemi (veya eylemler grubunu) seçer (gerçekleştirir).

• "**if…else**" çift seçim ifadesi (Bölüm 3.6), bir koşul doğruysa bir eylemi (veya eylemler grubunu), koşul yanlışsa farklı bir eylemi (veya eylemler grubunu) gerçekleştirir.

• "**if…elseif**" Çoklu seçim değiştirme deyimi, bir ifadenin değerine bağlı olarak birçok farklı eylemden birini gerçekleştirir.

C'deki İterasyon İfadeleri

C, ifade biçiminde üç tür yineleme yapısı sağlar, yani "while" (Bölüm 3.7), "do…while" ve "for". Bu ifadeler görevleri tekrar tekrar gerçekleştirir. "do…while" ve "for"'u bir sonraki bölümde tartışacağız.

C yalnızca yedi kontrol ifadesine sahiptir: sıra (squence), üç tür seçim (selection) ve üç tür yineleme (iteration).

Her programı, programın uyguladığı algoritmaya uygun olduğu kadar çok sayıda kontrol ifadesi türünü birleştirerek oluşturursunuz. Her kontrol ifadesinin akış şeması gösteriminin, biri kontrol ifadesine giriş noktasında ve diğeri çıkış noktasında olmak üzere iki küçük daire sembolüne sahip olduğunu göreceğiz. Bu tek girişli/tek çıkışlı kontrol ifadeleri, net programlar oluşturmayı kolaylaştırır.

Birinin çıkış noktasını diğerinin giriş noktasına bağlayarak kontrol ifadesi akış şeması parçalarını birbirine bağlayabiliriz. Bu, yapı taşlarını istifleyen bir alt öğeye benzer, bu nedenle buna kontrol ifadesi yığınlaması diyoruz. Bu bölümün ilerleyen kısımlarında, kontrol deyimlerini birbirine bağlamanın tek yolunun iç içe yerleştirme olduğunu göreceksiniz. Bu nedenle, oluşturmamız gereken herhangi bir C programı, yalnızca iki yolla birleştirilmiş ve yalnızca yedi kontrol deyiminden oluşturulabilir. Bu sadeliğin özüdür.

"if" Seçim İfadesi

Seçim ifadeleri, alternatif eylem yolları arasından seçim yapar. Örneğin, bir sınavda geçme notunun 60 olduğunu varsayalım. Aşağıdaki sözde kod ifadesi "öğrencinin notu 60'tan büyük veya ona eşittir" koşulunun doğru mu yoksa yanlış mı olduğunu belirler:

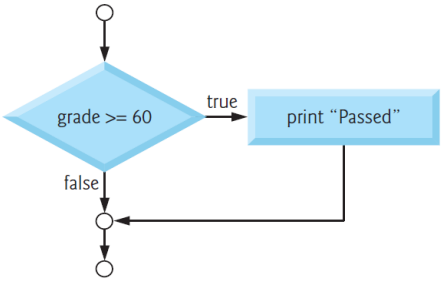
- Öğrencinin notu 60 ve üzeri ise
- "Geçti" yazdır

Koşul doğruysa, "Geçti" yazdırılır ve sıradaki sözde kod ifadesi "gerçekleştirilir". Sözde kodun gerçek bir programlama dili olmadığını unutmayın. Koşul yanlışsa, yazdırma yok sayılır ve sıradaki sözde kod deyimi gerçekleştirilir. Yukardaki sözde kod C'de şu şekilde yazılmıştır:

```
if (not >= 60) {
   puts("Gecti");
}
```

Tabii ki int değişken türünü de bildirmeniz gerekecek. C "if" deyimi kodu sözde koda çok yakındır. Bu, onu çok kullanışlı bir program geliştirme aracı yapan sözde kodun özelliklerinden biridir.

“if” ifadesinin ikinci satırındaki girinti isteğe bağlıdır ancak önemle tavsiye edilir. Yapılandırılmış programların doğal yapısını vurgular. Derleyici, girinti ve dikey boşluk için kullanılan boşluklar, sekmeler ve yeni satırlar gibi beyaz boşluk karakterlerini yok sayar. Aşağıdaki akış şeması bölümü, tek seçimli “if” deyimini göstermektedir:



Belki de en önemli akış şeması sembolünü içerir - bir kararın alınacağını gösteren üçgen (veya karar) sembolü. Karar sembolünün ifadesi tipik olarak doğru veya yanlış olabilen bir durumdur. Ondan çıkan iki akış çizgisi, ifade doğru veya yanlış olduğunda izlenecek yolları gösterir. Kararlar herhangi bir ifadenin değerine dayalı olabilir; sıfır yanlıştır ve sıfırdan farklı doğrudur. “if” ifadesi, tek girişli/tek çıkışlı bir ifadedir. Kalan kontrol yapıları için akış şeması segmentinin ayrıca gerçekleştirilecek eylemleri belirtmek için dikdörtgen semboller ve alınacak kararları belirtmek için üçgen semboller içerebileceğini yakında öğreneceğiz. Bu, vurguladığımız programlamanın eylem/karar modelidir.

if…else Seçim İfadesi

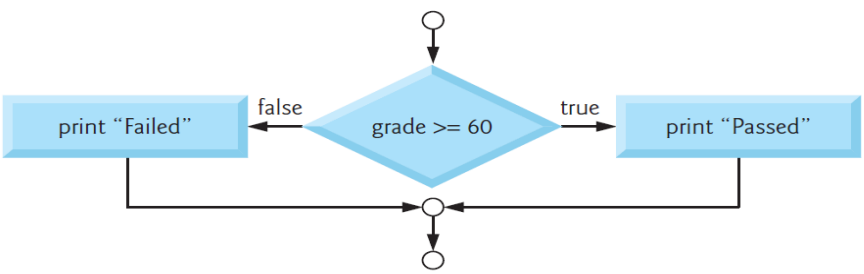
“if…else” seçim ifadesi, koşul doğru veya yanlış olduğunda gerçekleştirilecek farklı eylemleri belirtir. Örneğin, sözde kod ifadesi şöyle olsun;

Öğrencinin notu 60 ve üzeri ise
"Geçti" yazdır
Diğer durumlarda
"Başarısız" yazdır

öğrencinin notu 60'a eşit veya daha büyükse "Geçti" yazdırır; aksi halde "Başarısız" yazdırır. Her iki durumda da, yazdırmadan sonra sıradaki sözde kod deyimi "yürütülür". Bir programda birkaç girinti düzeyi varsa, her biri aynı ek boşluk miktarında girintilenmelidir. Yukardaki sözde kod C'de şu şekilde yazılabilir:

```
if (not >= 60) {
        puts("Gecti");
}
else {
        puts("Kaldi");
}
```

Aşağıdaki akış şeması, “if…else” ifadesinin kontrol akışını göstermektedir:



Koşul operatörü (?:), **if…else** deyimiyle yakından ilişkilidir. Bu operatör, C'nin tek üçlü operatörüdür, yani üç işlenen alır. Koşullu bir operatör ve onun üç işleneni bir koşullu ifade oluşturur. İlk işlenen bir koşuldur. İkincisi, koşul doğruysa koşullu ifadenin değeridir. Üçüncüsü, koşul yanlışsa koşullu ifadenin değeridir. Örneğin, aşağıdaki koşullu ifade bağımsız değişkeni, eğer grade >= 60 koşulu doğruysa, puts deyimi "Geçti" dizesini değerlendirir; aksi takdirde, "Başarısız" dizesi olarak değerlendirilir:

```
puts((grade >= 60) ? "Passed" : "Failed");
```

Koşullu operatörler, if…else deyimlerinin kullanılamadığı yerlerde, fonksiyonlara yönelik ifadeler ve bağımsız değişkenler (printf gibi) dahil olmak üzere kullanılabilir. Hassas hatalardan kaçınmak için koşullu operatörün (?:) ikinci ve üçüncü işlenenleri için aynı türden ifadeler kullanılması gerekir.

İç içe "if…else" ifadeleri ile birden fazla koşul test edilebilir.

Örneğin, aşağıdaki sözde kod ifadesine göre: 90'a eşit veya daha büyük notlar için AA, 80'e eşit veya daha büyük (ancak 90'dan küçük) notlar için BA, 70'e eşit veya daha büyük (ancak 80'den küçük) notlar için BB , 60 veya daha büyük (ancak 70'ten küçük) notlar için CB ve diğer tüm notlar için F yazdırılır.

Öğrencinin notu 90 ve üzerinde ise "AA" yazdır
else
Öğrencinin notu 80 ve üzerinde ise "BA" yazdır
else
Öğrencinin notu 70 ve üzerinde ise "BB" yazdır
else
Öğrencinin notu 60 ve üzeri ise "CB" yazdır
else
"F" yazdır
Bu sözde kod C'de şu şekilde yazılabilir:

```
if (grade >= 90) {
   puts("AA");
}
else {
        if (grade >= 80) {
                puts("BA");
        }
        else {
                if (grade >= 70) {
                        puts("BB");
                }
                else {
                        if (grade >= 60) {
                                puts("CB");
                        }
                        else {
                                puts("F");
                        } // end else
                } // end else
        } // end else
} // end else
```

Değişken değeri 90'a eşit veya daha büyükse, dört koşulun tümü doğrudur, ancak yalnızca ilk test yürütüldükten sonra puts ifadesi çalışır. Ardından, "dış" if…else ifadesinin else kısmı atlanır ve iç içe geçmiş if…else ifadesinin geri kalanı çalışır. Çoğu programcı önceki if ifadesini şu şekilde yazar:

```
if (grade >= 90) {
        puts("AA");
} // end if
else if (grade >= 80) {
        puts("BA");
} // end else if
else if (grade >= 70) {
        puts("BB");
} // end else if
else if (grade >= 60) {
        puts("CC");
} // end else if
else {
        puts("F");
} // end else
```

Her iki form da eşdeğerdir. İkinci biçimin okunması daha kolaydır. Programın okunabilirliğini azaltan ve bazen satırların kaymasına neden olan sağdaki derin girintiyi önler. Bir if'nin gövdesine birkaç ifade eklemek için, ifadeleri parantez ({ ve }) içine almalısınız. Bir çift parantez içinde yer alan bir dizi ifadeye bileşik ifade veya blok denir. Bileşik bir ifade, bir programda tek bir ifadenin yerleştirilebileceği herhangi bir yere yerleştirilebilir.

Aşağıdaki if…else deyiminin else kısmı, koşul yanlışsa yürütülecek iki deyimi içeren bileşik bir deyim içerir:

```
if (grade >= 60) {
   puts("Passed.");
} // end if
else {
   puts("Failed.");
   puts("You must take this course again.");
} // end else
```

Derleyici tarafından bir sözdizimi hatası ("else" yazım hatası gibi) yakalanır. Bir mantık hatası, yürütme zamanında etkisini gösterir. Önemli bir mantık hatası, bir programın başarısız olmasına ve erken sonlandırılmasına neden olur. Ölümcül (fatal) olmayan bir mantık hatası, bir programın çalışmaya devam etmesine ancak yanlış sonuçlar üretmesine neden olur.

Tek veya bileşik bir ifadenin yerleştirilebildiği her yerde, noktalı virgülle (;) gösterilen boş bir ifade yerleştirmek mümkündür. Aşağıdaki ifadede olduğu gibi bir if koşulundan sonra noktalı virgül konulması, tek seçimli if ifadelerinde mantık hatasına, çift seçim ve iç içe if…else ifadelerinde sözdizimi hatasına yol açar.

```
if (grade >= 60);
```

Parantezlerin (ayraçların) içine tek tek ifadeleri yazmadan önce bileşik ifadelerin her iki parantezini de yazın. Bu, ayraçlardan birinin veya her ikisinin atlanmasını önlemeye yardımcı olur, sözdizimi hatalarını ve mantık hatalarını önler. Birçok entegre geliştirme ortamı ve kod düzenleyicisi, siz açılış ayracı yazar yazmaz sizin için kapanış parantezini ekler.

Yineleme (İterasyon) İfadesi - while

Yineleme ifadesi (aynı zamanda iterasyon ifadesi veya döngüsü olarak da adlandırılır), bazı koşullar doğru kalırken bir eylemi tekrarlar. Örnek sözde kod deyimi şöyle olabilir;

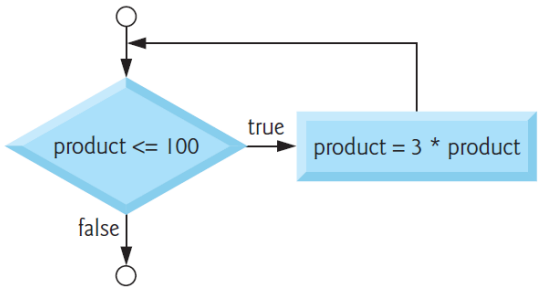
Alışveriş listemde daha fazla ürün varken
Sonraki öğeyi satın al ve listemden çıkar

bir alışveriş gezisi sırasında meydana gelen yinelemeyi açıklar. “Alışveriş listemde daha çok ürün var” koşulu doğru ya da yanlış olabilir. Eğer bu doğruysa, müşteri koşul doğru kalırken "Sonraki ürünü satın al ve listemden sil" eylemini tekrar tekrar gerçekleştirir. Sonunda, yani alışveriş listesindeki son öğe satın alındığında ve listeden çıkarıldığında koşul yanlış olacaktır. Bu noktada, yineleme sona erer ve yineleme ifadesinden sonraki ilk sözde kod ifadesi "çalışır".

Bir while ifadesi örneği olarak, 3'ün ilk kuvvetini 100'den büyük bulan bir program segmentini ele alalım. Tamsayı değişkeni çarpım 3 olarak başlatılır. Aşağıdaki kod parçası çalışmayı bitirdiğinde, ürün istenen yanıtı içerecektir:

```
int product = 3;
while (product <= 100) {
   product = 3 * product;
}
```

Döngü art arda çarpımı 3 ile çarpar, böylece art arda 9, 27 ve 81 değerlerini alır. product 243 olduğunda, product <= 100 koşulu false olur ve yinelemeyi sonlandırır—product'ın nihai değeri 243'tür. Yürütme, while'dan sonra bir sonraki ifade ile devam eder. While ifadesinin gövdesindeki bir eylem, sonunda koşulun yanlış olmasına neden olmalıdır; aksi takdirde, döngü asla sona ermez; bu, sonsuz döngü adı verilen bir mantık hatasıdır. Bir while yineleme deyiminde yer alan deyim(ler), tek bir deyim veya bileşik deyim olabilir. Aşağıdaki akış şeması segmenti, önceki while yineleme ifadesini göstermektedir:



Akış şeması, yinelemeyi açıkça gösterir; dikdörtgenden çıkan akış çizgisi, karara giren akış çizgisine geri döner. Döngü, koşul sonunda yanlış olana kadar her yinelemede karar kutusundaki koşulu test eder. Bu noktada “while” deyiminden çıkılır ve kontrol sıradaki bir sonraki deyimle devam eder.

Vaka Çalışması 1: Sayaç Kontrollü Yineleme

Algoritmaların nasıl geliştirildiğini göstermek için, bu bölümde ve sonraki bölümde bir sınıf ortalaması alma probleminin iki varyasyonunu çözeceğiz. Aşağıdaki problemi göz önünde bulunduralım:

On kişilik bir sınıf bir sınava girmiş olsun. Bu sınavın notları (0 ile 100 arasındaki tamsayılar) veriliyor. Sınavda sınıf ortalamasını belirleyin.

Sınıf ortalaması, notların toplamının öğrenci sayısına bölümüdür. Bu problemi çözecek algoritma notları girmeli, ardından sınıf ortalamasını hesaplamalı ve göstermelidir.

Yürütülecek eylemleri listelemek ve yürütülmeleri gereken sırayı belirlemek için sözde kod kullanalım. Notları birer birer girmek için sayaç kontrollü iterasyon kullanıyoruz. Bu teknik, bir dizi deyimin kaç kez çalıştırılacağını belirtmek için sayaç adı verilen bir değişken kullanır. Bu örnekte, 10 öğrencinin sınava girdiğini biliyoruz, dolayısıyla 10 not girmemiz gerekiyor. Yineleme, sayaç 10'u aştığında sona erer. Bu örnek olay incelemesinde, son sözde kod algoritmasını (Şekil 3.1) ve karşılık gelen C programı (Şekil 3.2) de veriliyor.

1 Toplamı sıfıra ayarla
2 Not sayacını bir olarak ayarlayın
4 Not sayacı ondan küçük veya ona eşitken
5 Sonraki notu girin
6 Notu toplama ekleyin
7 Not sayacına bir ekleyin
9 Sınıf ortalamasını toplam/10 olarak hesaplayın
10 Sınıf ortalamasını yazdırın

Şekil 3.1 | Sınıf ortalaması problemini çözmek için sayaç kontrollü yineleme kullanan sözde kod algoritması.

```c
// fig03_02.c
// Class average program with counter-controlled iteration.
#include <stdio.h>

// function main begins program execution
int main(void) {
   // initialization phase
   int total = 0; // initialize total of grades to 0
   int counter = 1; // number of the grade to be entered next

   // processing phase
   while (counter <= 10) { // loop 10 times
      printf("%s", "Enter grade: "); // prompt for input
      int grade = 0; // grade value
      scanf("%d", &grade); // read grade from user
      total = total + grade; // add grade to total
      counter = counter + 1; // increment counter
   } // end while

   // termination phase
   int average = total / 10; // integer division
   printf("Class average is %d\n", average); // display result
} // end function main
```

Şekil 3.2 | Sayaç kontrollü yineleme ile sınıf ortalama problemi.

Total, bir dizi değerin toplamını toplamak için kullanılan bir değişkendir (satır 8). counter, bu durumda girilen notların sayısını saymak için kullanılan bir değişkendir (satır 9). Toplamlar için değişkenler sıfır olarak başlatılmalıdır; aksi takdirde toplam, toplamın bellek konumunda saklanan önceki değeri içerecektir. Başlatılmamış bir değişken, bir "çöp" değeri içerir; bu değişken için ayrılan bellek konumunda en son saklanan değer. Bir sayaç veya toplam başlatılmamışsa, programınızın sonuçları muhtemelen yanlış olacaktır. Bunlar mantık hatası örnekleridir. Önceki örnek uygulamada notların toplamı 817 olsun. Ortalama 81 olur, ancak girdiğimiz notların toplamı 817'ydi. Elbette, 817'nin 10'a bölünmesi 81,7'yi, yani ondalık noktalı bir sayıyı vermelidir. Bir sonraki bölümde bu konuya bakalım.

Adım Adım İyileştirmeyle Formüle Etme Vaka Çalışması 2:

Gözcü Kontrollü Yineleme Sınıf ortalama problemini genelleştirelim. Aşağıdaki problemi göz önünde bulundurun:

Program her çalıştırıldığında isteğe bağlı sayıda notu işleyecek bir sınıf ortalaması alma programı geliştirin.

Birinci sınıf ortalama örneğinde, önceden 10 not olduğunu biliyorduk. Bu örnekte, kullanıcının kaç not girebileceğine dair bir gösterge verilmemiştir. Program, isteğe bağlı sayıda notu işlemelidir. Program, not girmeyi ne zaman durduracağını nasıl belirleyebilir? Sınıf ortalamasını ne zaman hesaplayacağını ve yazdıracağını nasıl bilecek?

Bunun bir yolu, "veri girişinin sonunu" belirtmek için bir koruyucu değer kullanmaktır. Bir koruyucu değer aynı zamanda sinyal değeri, kukla değer veya bayrak değeri olarak da adlandırılır. Kullanıcı, tüm geçerli notlar girilene kadar notları yazar. Kullanıcı daha sonra "son notun girildiğini" belirtmek için koruyucu değeri girer. Bu tür yinelemeye genellikle belirsiz yineleme denir çünkü döngü yürütülmeye başlamadan önce yinelemelerin sayısı bilinmez.

Kabul edilebilir bir giriş değeriyle karıştırılmayacak bir koruyucu değer seçmelisiniz. Bir kısa sınavdaki notlar negatif olmayan tam sayılardır, dolayısıyla –1 bu problem için kabul edilebilir bir koruyucu değerdir. Bu nedenle, sınıf ortalamalı programın çalıştırılması 95, 96, 75, 74, 89 ve -1 gibi bir girdi akışını işleyebilir. Program daha sonra sınıflar için sınıf ortalamasını hesaplar ve yazdırır. Koruyucu değeri -1, ortalama hesaplamasına girmemelidir. İkinci iyileştirme, Şekil 3.3'te gösterilmiştir. Okunabilirlik için sözde koda bazı boş satırlar ekledik.

1 Toplamı sıfırla
2 Sayacı sıfırla
3
4 Birinci notu girin (muhtemelen koruyucu değer)
5 Kullanıcı henüz koruyucu değer girmemişken
6 Bu notu toplam nota ekleyin
7 Not sayacına bir ekleyin
8 Sonraki notu girin (muhtemelen koruyucu değer)
9
10 Sayaç sıfıra eşit değilse
11 Ortalamayı sayaca bölünen toplam olarak ayarlayın
12 Ortalamayı yazdır
13 toplam sıfırsa
14 "Not girilmedi" Yazdır

Şekil 3.3 | Sınıf ortalama problemini çözmek için gözcü kontrollü yinelemeyi kullanan sözde kod algoritması.

Birçok program mantıksal olarak üç aşamaya ayrılabilir:

• program değişkenlerini başlatan bir başlatma aşaması,

• veri değerlerini giren ve program değişkenlerini buna göre ayarlayan bir işleme aşaması ve

• nihai sonuçları hesaplayan ve yazdıran bir sonlandırma aşaması.

Şekil 3.3'teki sözde kod algoritması daha genel sınıf ortalama problemini çözer. Bu algoritma, yalnızca iki iyileştirme düzeyinden sonra geliştirilmiştir. Bazen daha fazla seviye gereklidir. Bilgisayarda bir sorunu çözmenin en zor kısmı, çözüm için algoritma geliştirmektir. Doğru bir algoritma belirlendikten sonra, çalışan bir C programı üretmek genellikle basittir.

Birçok programcı, sözde kod gibi program geliştirme araçlarını hiç kullanmadan programlar yazar. Nihai hedeflerinin sorunu çözmek olduğunu ve sözde kod yazmanın yalnızca nihai çıktıların üretilmesini geciktirdiğini düşünüyorlar. Bu, kendi kullanımınız için geliştirdiğiniz küçük programlar için işe yarayabilir. Ancak endüstride muhtemelen üzerinde çalışacağınız önemli programlar ve yazılım sistemleri için resmi bir geliştirme süreci gereklidir.

Şekil 3.4, C programını göstermektedir. Yalnızca tamsayı notları girilmiş olsa da, ortalama hesaplaması muhtemelen ondalık noktalı bir sayı üretecektir. **int** türü böyle bir sayıyı temsil edemez. Dolayısıyla bu program, ondalık basamaklı sayıları, yani kayan noktalı sayıları işlemek için **double** veri türünü kullanmıştır.

```c
// fig03_04.c
// Class-average program with sentinel-controlled iteration.
#include <stdio.h>

// function main begins program execution
int main(void) {
   // initialization phase
   int total = 0; // initialize total
   int counter = 0; // initialize loop counter

   // processing phase
   // get first grade from user
   printf("%s", "Enter grade, -1 to end: "); // prompt for input
   int grade = 0; // grade value
   scanf("%d", &grade); // read grade from user

   // loop while sentinel value not yet read from user
   while (grade != -1) {
      total = total + grade; // add grade to total
      counter = counter + 1; // increment counter

      // get next grade from user
      printf("%s", "Enter grade, -1 to end: "); // prompt for input
      scanf("%d", &grade); // read next grade
   } // end while

   // termination phase
   // if user entered at least one grade
   if (counter != 0) {

      // calculate average of all grades entered
      double average = (double) total / counter; // avoid truncation

      // display average with two digits of precision
      printf("Class average is %.2f\n", average);
   } // end if
   else { // if no grades were entered, output message
      puts("No grades were entered");
   } // end else
} // end function main
```

3.4 | Gözcü kontrollü yinelemeli sınıf ortalamalı program.

Ortalamalar genellikle 7.2 veya 93.5 gibi kesirli kısımlar içeren değerlerdir. Bu kayan noktalı sayılar, **double** veri türü ile temsil edilebilir. Satır 32, hesaplamamızın kesirli sonucunu yakalamak için ortalama değişkenini **double** tipi olarak tanımlar. Normalde, total / counter (satır 32) hesaplamasının sonucu bir tamsayıdır çünkü total ve counter int değişkenleridir. İki tamsayı bölmek, tamsayı bölünmesiyle sonuçlanır; hesaplamanın herhangi bir kesirli kısmı kesilir (yani kaybolur). Önce geçici kayan noktalı sayılar oluşturarak tamsayı değerlerle bir kayan nokta hesaplaması üretebilirsiniz. C, bu görevi gerçekleştirmek için unary cast operatörünü sağlar. Satır 32

```
double average = (double) total / counter;
```

işleneni olan total'in geçici bir kayan noktalı kopyasını oluşturmak için atama operatörünü (**double**) kullanır. Toplamda saklanan değer hala bir tamsayıdır. Bir atama operatörünün bu şekilde kullanılmasına açık dönüştürme denir. Hesaplama şimdi, sayaçta saklanan int değerine bölünen bir kayan noktalı değerden (toplamın geçici double versiyonu) oluşur.

C, aritmetik ifadelerdeki işlenen veri türlerinin yalnızca aynı olmasını gerektirir. Karışık tür ifadelerde, derleyici, aynı türde olduklarından emin olmak için seçilen işlenenler üzerinde örtük dönüştürme adı verilen bir işlem gerçekleştirir. Örneğin, **int** ve **double** veri türlerini içeren bir ifadede, **int** işlenenlerin kopyaları yapılır ve dolaylı olarak **double** türüne dönüştürülür. Toplamı açıkça bir **double**'a dönüştürdükten sonra, derleyici dolaylı olarak sayacın **double** kopyasını yapar, ardından kayan noktalı bölme işlemini gerçekleştirir ve kayan noktalı sonucu ortalamaya atar. Bölüm 5'te, C'nin farklı türdeki işlenenleri dönüştürme kuralları verilecektir.

Atama operatörleri, bir tür adının etrafına parantez konularak oluşturulur. Atma, yalnızca bir işlenen alan tekli bir operatördür. C ayrıca artı (+) ve eksi (-) operatörlerin tekli sürümlerini de destekler, böylece -7 veya +5 gibi ifadeler yazabilirsiniz. Atama operatörleri sağdan sola gruplanır ve tekli + ve tekli - gibi diğer tekli operatörlerle aynı önceliğe sahiptir. Bu öncelik *, / ve % çarpım operatörlerinden bir düzey daha yüksektir.

Şekil 3.4, ortalamanın değerini biçimlendirmek için printf dönüştürme özelliği %.2f'yi (satır 35) kullanır. f, bir kayan noktalı değerin yazdırılacağını belirtir. .2 kesinliktir—değerin ondalık virgülün sağında iki (2) hanesi olacaktır. Kesinlik belirtilmeden %f dönüştürme belirtimi kullanılırsa, varsayılan kesinlik, %.6f dönüştürme belirtimi kullanılmış gibi ondalık noktanın sağında 6 basamaktır. Kayan noktalı değerler hassasiyetle yazdırıldığında, yazdırılan değer belirtilen ondalık basamak sayısına yuvarlanır. Hafızadaki değer değişmez. Aşağıdaki ifadeler sırasıyla 3,45 ve 3,4 değerlerini gösterir:

```
printf("%.2f\n", 3.446); // displays 3.45
printf("%.1f\n", 3.446); // displays 3.4
```

Kayan noktalı sayılar her zaman "%100 kesin" olmasa da, çok sayıda uygulamaları vardır. Kayan noktalı sayılar genellikle bölme yoluyla oluşur. 10'u 3'e böldüğümüzde sonuç 3.3333333… 3'lerin dizisi sonsuz tekrar ediyor. Bilgisayar, böyle bir değeri tutmak için yalnızca sabit miktarda alan ayırır, bu nedenle depolanan kayan nokta değeri yalnızca yaklaşık bir değer olabilir. Kayan noktalı sayılar çoğu bilgisayar tarafından yalnızca yaklaşık olarak temsil edilir.

Vaka Çalışması 3: İç İçe Kontrol İfadeleri

Algoritmayı sözde kod ve adım adım iyileştirme kullanarak formüle edelim ve karşılık gelen bir C programı yazalım. Tıpkı bir çocuğun yapı taşlarını istiflemesi gibi, kontrol ifadelerinin birbiri üzerine (sırayla) istiflenebileceğini gördük. Bu vaka incelemesinde, bir kontrol ifadesini diğerinin içine yerleştirerek göreceğiz. Aşağıdaki sorun bildirimini göz önünde bulunduralım:

Bir kolej, öğrencileri devlet lisanslama sınavına hazırlayan bir kurs veriyor. Geçen yıl bu kursu tamamlayan öğrencilerden 10'u lisans sınavına girmiş olsun. Doğal olarak kolej, öğrencilerinin sınavda ne kadar başarılı olduğunu bilmek ister. Sonuçları özetlemek için bir program yazmanız istendi. Size bu 10 öğrencinin bir listesi verildi. Her ismin yanında, öğrenci sınavı geçerse 1, başarısız olursa 2 eklenecek. Programınız sınav sonuçlarını aşağıdaki gibi analiz etmelidir:

1. Her bir test sonucunu girin (yani, 1 veya 2). Program başka bir test sonucu istediğinde "Sonucu girin" uyarı mesajını görüntüleyin.
2. Her tipteki test sonuçlarının sayısını sayın.
3. Geçen ve başarısız olan öğrenci sayısını gösteren test sonuçlarının bir özetini görüntüleyin.
4. Sınavı sekizden fazla öğrenci geçtiyse, "Eğitmene ikramiye!" mesajını yazdırın.

Sorun bildirimini dikkatlice okuduktan sonra aşağıdaki gözlemleri yapıyoruz:

1. Program 10 test sonucunu işlemelidir. Sayaç kontrollü bir döngü kullanacağız.
2. Her test sonucu bir sayıdır—ya 1 ya da 2. Program bir test sonucunu her okuduğunda, sonucun 1 mi yoksa 2 mi olduğunu belirlemesi gerekir. Algoritmamızda 1'i test edeceğiz. Sayı 1 değilse, 2 olduğunu varsayacağız. Alıştırma sizden her test sonucunun 1 veya 2 olduğundan emin olmanızı istiyor.
3. Biri sınavı geçen öğrencilerin sayısını, diğeri ise sınavda başarısız olan öğrencilerin sayısını saymak için iki sayaç kullanılır.
4. Program tüm sonuçları işledikten sonra, sınavı 8'den fazla öğrencinin geçip geçmediğine karar vermeli ve geçtiyse "Eğitmene Bonus!" yazdırmalıdır.

Şekil 3.5, Bu problemin çözümü için sözde kod'u (Buna sahte kod da denir) vermektedir. Okunabilirlik için boş satırlar kullanıyoruz.

```
1 geçen değişkenini sıfırla
2 Başarısız değişkenini sıfırla
3 Öğrenci değişkenini bir olarak başlat
4
5 Öğrenci sayacı 10'dan küçük veya 10'a eşit olduğu sürece
6 Bir sonraki sınav sonucunu girin
7
8 Öğrenci geçerse
9 Geçenlere bir ekleyin
10 Diğer durumlarda (else)
11 Başarısızlara bir ekleyin
12
13 Öğrenci sayacına bir ekleyin
14
15 Geçiş sayısını yazdırın
16 Başarısız sayısını yazdırın
17 Sekizden fazla öğrenci geçerse
18 "Öğretmene ikramiye!" Yazdır
```

3.5 | Kontrol problemi için sözde kod.

Bu sözde kod artık C'ye dönüştürmek için yeterince düzenlenmiştir. Şekil 3.6, C programını ve iki örnek yürütmeyi göstermektedir.

```c
// fig03_06.c
// Analysis of examination results.
#include <stdio.h>

// function main begins program execution
int main(void) {
   // initialize variables in definitions
   int passes = 0;
   int failures = 0;
   int student = 1;

   // process 10 students using counter-controlled loop
   while (student <= 10) {
      // prompt user for input and obtain value from user
      printf("%s", "Enter result (1=pass,2=fail): ");
      int result = 0; // one exam result
      scanf("%d", &result);

      // if result 1, increment passes
      if (result == 1) {
         passes = passes + 1;
      } // end if
      else { // otherwise, increment failures
         failures = failures + 1;
      } // end else

      student = student + 1; // increment student counter
   } // end while

   // termination phase; display number of passes and failures
   printf("Passed %d\n", passes);
   printf("Failed %d\n", failures);

   // if more than eight students passed, print "Bonus to instructor!"
   if (passes > 8) {
      puts("Bonus to instructor!");
   } // end if
} // end function main
```

Atama Operatörleri

C, atama ifadelerini kısaltmak için çeşitli atama operatörleri sağlar. Örneğin, ifade

c = c + 3;

toplama atama operatörü += ile şu şekilde kısaltılabilir:

c+= 3;

+= operatörü, operatörün sağındaki ifadenin değerini operatörün solundaki değişkenin değerine ekler ve sonucu soldaki değişkende saklar. Böylece, c += 3 ataması, c'nin mevcut değerine 3 ekler. Aşağıdaki tabloda aritmetik atama operatörleri, bu operatörleri kullanan örnek ifadeler ve açıklamalar gösterilmektedir:

Atama operatörü	Örnek ifade	Açıklama	Atamalar
int c = 3, d = 5, e = 4, f = 6, g = 12; Olduğunu varsayalım:			
+=	c += 7	c = c + 7	c'ye 10
-=	d -= 4	d = d - 4	d'ye 1
*=	e *= 5	e = e * 5	e'ye 20
/=	f /= 3	f = f / 3	f'ye 2
%=	g%= 9	g = g % 9	g'ye 3

Arttırma ve Azaltma Operatörleri

Tekli artırma işleci (++) ve tekli azaltma işleci (--) sırasıyla bir tamsayı değişkenine bir ekler ve bir çıkarır. Aşağıdaki tabloda, her bir işlecin iki versiyonu özetlenmektedir:

Operatör	Örnek ifade	Açıklama
++	++a	a'yı 1 artırır ve a'nın içinde bulunduğu ifadede a'nın yeni değerini kullanır.
++	a++	a'nın bulunduğu ifadede a'nın geçerli değerini kullanır, sonra a'yı 1 artırır.
--	--b	b'yi 1 azaltır ve b'nin bulunduğu ifadede b'nin yeni değerini kullanır.
--	b--	b'nin bulunduğu ifadede b'nin geçerli değerini kullanır sonra b'yi 1 azaltır.

c değişkenini 1 artırmak için c = c + 1 veya c += 1 ifadeleri yerine ++ operatörünü kullanabilirsiniz. Ön artırma veya ön azaltma operatörleri olarak adlandırılır. Bir değişkenin arkasına ++ veya -- koyarsanız, bunlar artma operatörleri olarak anılır. Geleneksel olarak, tekli operatörler işlenenlerinin yanına boşluk bırakmaksızın yerleştirilmelidir.

Şekil 3.7, ++ operatörünün artırma öncesi ve artırma sonrası sürümleri arasındaki farkı göstermektedir. c değişkeninin art arda eklenmesi, printf deyiminde kullanıldıktan sonra artmasına neden olur. c değişkeninin önceden artırılması, printf deyiminde kullanılmadan önce artırılmasına neden olur. Program, ++ kullanmadan önce ve sonra c'nin değerini görüntüler. Azaltma operatörü (--) benzer şekilde çalışır.

```c
// fig03_07.c
// Preincrementing and postincrementing.
#include <stdio.h>

// function main begins program execution
int main(void) {
// demonstrate postincrement
        int c = 5; // assign 5 to c
        printf("%d\n", c); // print 5
        printf("%d\n", c++); // print 5 then postincrement
        printf("%d\n\n", c); // print 6

// demonstrate preincrement
        c = 5; // assign 5 to c
        printf("%d\n", c); // print 5
        printf("%d\n", ++c); // preincrement then print 6
        printf("%d\n", c); // print 6
} // end function main
```

3.7 | Ön artırma ve sonra artırma.

Şekil 3.6'daki üç atama ifadesini dikkate alalım;

```c
passes = passes + 1;
failures = failures + 1;
student = student + 1;
```

Bunlar atama operatörleri ile daha kısa ve öz olarak şu şekilde yazılabilir:

```c
passes += 1;
failures += 1;
student += 1;
```

Ayrıca ön artırma operatörleri ile şöyle yazılabilir;

```c
++passes;
++failures;
++student;
```

veya sonra artırma operatörleri ile;

```c
passes++;
failures++;
student++;
```

Bir ifadedeki bir değişkeni kendi başına artırırken veya azaltırken, artırma öncesi ve artırma sonrası formları aynı etkiye sahiptir. Yalnızca bir değişken, daha büyük bir ifade bağlamında göründüğünde, ön

artırma ve son artırmanın farklı etkileri vardır (ve benzer şekilde, ön azaltma ve son azaltma için). ++ veya -- operatörünün işleneni olarak yalnızca basit bir değişken adı kullanılabilir. Arttırma veya azaltma operatörünü basit bir değişken adı dışında bir ifadede kullanmaya çalışmak bir sözdizimi hatasıdır; örneğin, ++(x + 1).

C genellikle bir operatörün işlenenlerinin değerlendirileceği sırayı belirtmez. Bir sonraki bölümde birkaç operatör için bunun istisnalarını göreceğiz. Hassas (gizli-çözümü zor) hatalardan kaçınmak istiyorsak, ++ ve -- operatörleri yalnızca tam olarak bir değişkeni değiştiren ifadelerde kullanılmalıdır. Aşağıdaki tablo, şimdiye kadar gösterilen operatörleri azalan öncelik sırasına göre listeler.

Operatörler	Gruplama	Tip
++ *(postfix)* -- *(postfix)*	sağdan sola	sonek
+ - (*type*) ++ *(prefix)* -- *(prefix)*	sağdan sola	tekli
* / %	soldan sağa	çarpımsal
+ -	soldan sağa	toplamsal
< <= > >=	soldan sağa	İlişkisel
== !=	soldan sağa	eşitlik
?:	sağdan sola	koşullu
= += -= *= /= %=	sağdan sola	atama

Üçüncü sütun, çeşitli operatör gruplarını adlandırır. Koşullu operatör (?:), tekli operatörler artırma (++), eksiltme (--), artı (+), eksi (-) ve atama operatörleri ile =, +=, -=, * atama operatörlerinin olduğuna dikkat edin.