Ödev:4 - ComputeDate

İbrahim Fevzi Kayan

Ödev:

- *İçinde int mYear, int mMonth, int mDay şeklinde değişkenler olan CDate adında bir struct'ımız olsun.*
- *Yukarıda tanımlı CDate struct'ımızı girdi olarak alan, yine aynı formatta çıktı veren iki fonksiyon yazacağız.*
- *İlk fonksiyonumuz ComputeNextDate().*
- *İkinci fonksiyonumuz ComputePreviousDate().*
- *İsimlerinden de anlaşılacağı üzere bu iki fonksiyonun her biri, verilen bir tarih için önceki ve sonraki günün tarihlerini hesaplayıp döndürüyor olacak.*
- *Bu kapsamda tabi ki artık yıl, 30-31 günlük günler vb çetrefilli bir takım hususları ele almak gerekecek. Mesela IsLeapYear(int Year) şeklinde bool bir fonksiyon yazıp bunu kullanabilirsiniz.*
- *Ayrıca gün, ay ve yıl değerlerinin belirli birtakım aralıklarda olmasına ilişkin girdi kontrolleri yapmak gerekecek. Örneğin IsValidDate() şeklinde bool bir fonksiyon yazıp bunu diğer fonksiyonların içinden çağırıp kullanabilirsiniz.*
- *Buralarda hangi isimlerle yardımcı kaç adet fonksiyon yazacağınız size kalmış durumda.*
- *Bu ödev kapsamında bir diğer adım da şu: Yazdığınız fonksiyonların birim testlerini yapmak için hangi girdileri vermeyi düşünüyorsunuz? Bir başka deyişle, hangi test case'leriniz ve toplamda kaç adet test case'iniz olur? Bu test case'lerin gerekli olduğunu neden düşündünüz? Bu test case'lerin yeterli olduğunu neden düşündünüz? Bu hususları tartıştığınız bir essay yazmanızı da ayrıca rica edeceğim.*

Çözüm:

```
#include <stdio.h>
#include <stdbool.h>

typedef struct {
    int mYear;
    int mMonth;
    int mDay;
} CDate;

bool IsValidDate(int year, int month, int day) {
    if (year < 0 || month < 1 || month > 12 || day < 1)
        return false;
```

```
    int daysInMonth[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (year % 4 == 0) {
        if (year % 100 != 0 || year % 400 == 0)
            daysInMonth[2] = 29;
    }

    if (day > daysInMonth[month])
        return false;

    return true;
}

bool IsLeapYear(int year) {
    if (year % 4 == 0) {
        if (year % 100 != 0 || year % 400 == 0)
            return true;
    }
    return false;
}

CDate ComputeNextDate(CDate date) {
    CDate nextDate = date;

    nextDate.mDay++;

    int daysInMonth[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (IsLeapYear(nextDate.mYear))
        daysInMonth[2] = 29;

    if (nextDate.mDay > daysInMonth[nextDate.mMonth]) {
        nextDate.mDay = 1;
        nextDate.mMonth++;

        if (nextDate.mMonth > 12) {
            nextDate.mMonth = 1;
            nextDate.mYear++;
        }
    }

    return nextDate;
}

CDate ComputePreviousDate(CDate date) {
    CDate previousDate = date;

    previousDate.mDay--;

    int daysInMonth[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (IsLeapYear(previousDate.mYear))
        daysInMonth[2] = 29;

    if (previousDate.mDay == 0) {
        previousDate.mMonth--;

        if (previousDate.mMonth == 0) {
            previousDate.mMonth = 12;
            previousDate.mYear--;
        }

        previousDate.mDay = daysInMonth[previousDate.mMonth];
    }

    return previousDate;
```

```
}

int main() {
    CDate date;

    printf("Yili giriniz: ");
    scanf("%d", &date.mYear);
    printf("Ayi giriniz: ");
    scanf("%d", &date.mMonth);
    printf("Gunu giriniz: ");
    scanf("%d", &date.mDay);

    if (IsValidDate(date.mYear, date.mMonth, date.mDay)) {
        CDate nextDate = ComputeNextDate(date);
        CDate prevDate = ComputePreviousDate(date);

        printf("Girilen Tarih: %d-%02d-%02d\n",  date.mDay, date.mMonth, date.mYear);
        printf("Sonraki Tarih: %d-%02d-%02d\n", nextDate.mDay, nextDate.mMonth, nextDate.mYear );
        printf("Onceki Tarih: %d-%02d-%02d\n", prevDate.mDay, prevDate.mMonth, prevDate.mYear);
    } else {
        printf("Hatali Tarih!\n");
    }

    return 0;
}
```

Kod çıktısı

```
Yili giriniz: 1988
Ayi giriniz: 5
Günü giriniz: 29
Girilen Tarih: 29-05-1988
Sonraki Tarih: 30-05-1988
Onceki Tarih: 28-05-1988
```

TEST CASES

Test case'leri oluştururken, aşağıdaki senaryoları dikkate alabiliriz:

1. Geçerli tarihleri test etmek: Örneğin, 2023-08-06 gibi geçerli bir tarihi girdi olarak kullanarak, `ComputeNextDate` ve `ComputePreviousDate` fonksiyonlarından beklenen sonuçları almayı sağlamalıyız.
2. Geçersiz tarihleri test etmek: Örneğin, 30-02-2023 veya 01-13-2023 gibi geçersiz bir tarihi girdi olarak kullanarak, fonksiyonların doğru bir şekilde hata mesajı vermesini sağlamalıyız.
3. Artık yılları test etmek: Örneğin, 28-02-2020gibi bir artık yılı girdi olarak kullanarak, `ComputeNextDate` ve `ComputePreviousDate` fonksiyonlarından beklenen sonuçları almayı sağlamalıyız.

4. Yılın son günü ve ilk günü test etmek: Örneğin, 31-12-2023 ve 01-01-2023 gibi yılın son günü ve ilk gününü girdi olarak kullanarak, fonksiyonların doğru bir şekilde yıl ve ay değişikliklerini işlemesini sağlamalıyız.
5. Geçerli tarih aralıklarını test etmek: Tarih aralıklarının sınırlarını (örneğin, gün 1-31, ay 1-12, yıl 0 ve sonrası gibi) test ederek fonksiyonların doğru çalışmasını sağlamalıyız.
6. Hızlı bir şekilde birden çok artık yıl ve normal yılın ardışık tarihlerini test etmek için otomatik testler de oluşturulabilir.

Geçerli bir CDate tarihini alarak, ComputeNextDate ve ComputePreviousDate fonksiyonları, bir sonraki ve bir önceki tarihi hesaplamak için kullanılır. Bu fonksiyonlar, tarih hesaplama işlemleri sırasında artık yıl, 30 ve 31 gün gibi çetrefilli durumları ele alır ve hatalı girişleri engellemek için IsValidDate fonksiyonunu kullanır.

IsValidDate fonksiyonu, verilen yıl, ay ve gün değerlerinin geçerli bir tarih olup olmadığını kontrol eder. Geçersiz tarihler, yılın negatif olması, ayın 1-12 aralığında olmaması veya günün ayın gün sayısını aşması gibi durumları içerir. Ayrıca, IsLeapYear fonksiyonu, bir yılın artık yıl olup olmadığını belirlemek için kullanılır ve şubat ayının gün sayısını günceller.

ComputeNextDate fonksiyonu, verilen tarihin bir gün sonraki tarihini hesaplar. Tarih hesaplama işlemi sırasında, gün değeri gün sayısını aştığında, ay ve yıl değerleri de güncellenir. Ayrıca, ComputePreviousDate fonksiyonu, verilen tarihin bir gün önceki tarihini hesaplar ve aynı şekilde tarih hesaplama işlemi sırasında geçerli tarih aralığını kontrol eder.

Test case'lerinin kurgulanmasında amacımız, fonksiyonların tüm olası senaryolarda doğru çalışmasını sağlamaktır. Bu nedenle, test case'lerinin çeşitliliği ve kapsamlılığı önemlidir. Öncelikle, geçerli tarihleri test etmeliyiz, bu sayede fonksiyonların beklenen sonuçları ürettiğinden emin olabiliriz. Ayrıca, geçersiz tarihleri test ederek fonksiyonların hatalı girişlere nasıl tepki verdiğini gözlemlemeliyiz.

Artık yılları test etmek, tarih hesaplama işlemleri sırasında şubat ayının gün sayısının doğru bir şekilde güncellendiğini doğrulamamıza yardımcı olur. Yılın son günü ve ilk günü testleri, yıl ve ay değişikliklerini ele alıp almadığını kontrol eder. Geçerli tarih aralıklarını test etmek, tarih değerlerinin belirli sınırlar içinde olduğunda doğru sonuçları ürettiğini doğrulamak için önemlidir.

Otomatik testler, hızlı bir şekilde birden çok tarih kombinasyonunu test etmemize yardımcı olur ve daha kapsamlı test coverage elde etmemizi sağlar. Test case'lerinin yeterli olduğunu düşünmemizin nedeni, tüm önemli senaryoların kapsandığından emin olmamızdır. Her bir test case, kodun farklı kısımlarını ve farklı senaryoları test etmeli ve bu sayede olası hataları tespit etmeli ve düzeltmeliyiz.

Sonuç olarak, CDate struct'ını kullanan ComputeNextDate ve ComputePreviousDate fonksiyonları, tarih hesaplama işlemlerini doğru bir şekilde gerçekleştiren ve geçerli tarih aralıklarını kontrol eden kodlar olarak uygulanmalıdır. Test case'leri, fonksiyonların doğru çalıştığından ve çeşitli senaryolara uygun olduğundan emin olmak için kapsamlı ve çeşitlilik

gösteren bir şekilde kurgulanmalıdır. Bu, kodun güvenilir ve sağlam bir şekilde çalışmasını sağlamak için önemlidir.