PROGRAMLAMA DİLLERİ FİNAL ÖDEVİ

1. How does a decimal value waste memory space?

   Ondalık değerler, ondalık basamaklar için ikili kodlar şeklinde, karakter dizilerine çok benzer depolanır. Bu gösterim, ikili kodlamış onluk sayılar (BCD) olarak adlandırılır. Bazı durumlarda, bayt başına bir basamak gerekirken diğerlerinde bayt başına iki basamak gereklidir. Her iki durumda da, ikili temsillerinden daha fazla depolama alanı kullanılır. Bir ondalık basamağı kodlamak için en az dört bit gerekir.

   Bu nedenle, altı basamaklı kodlanmış bir ondalık sayıyı saklamak için 24 bit bellek gerekir. Ancak, aynı sayıyı ikili olarak göstermek sadece 20 bit alır. Ondalık değerler, dört bitlik ikili eşdeğerleri kullanılarak kodlanır 0 (0000) ila 9 (1001). Bu, tek bir baytın 0 ile 99 arasındaki değerleri tutabileceği anlamına gelir. Ancak aynı baytı bir ikili değeri tutmak için kullanmak, 0 ile 255 (veya -128 ve +127) arasında daha fazla değerin kullanılabilmesini sağlar.

   Kısacası N ondalık basamak, ondalık biçimde hafızada tutulmak istenirse 4N bit gerektirecektir. Öte yandan, aynı N ondalık basamak bir ikili sayı olarak tutulacaksa log2(10N) bit, yani yaklaşık 3,32N bit gerektirmelidir. Bu nedenle, ondalık gösterimde yaklaşık %20 daha fazla bellek kullanır.

2. What significant justification is there for the -> operator in C and C++?

   “->” operatörü struct göstericilerine (pointer) özel olarak tasarlanmış bir işlevselliğe sahiptir. Struct göstericisinin işaretlediği yapının bellek adresindeki nesneyi direkt olarak elde edip üye elemanlarını getirir.

   Ayrıca struct göstericisinin işaretlediği yapının bellek adresindeki nesneyi elde etmek için kullanılabilecek 2 yol vardır. Bu yollardan p –> q yazmak, (*p).q yazmaktan biraz daha kolaydır.

   UYARI: Struct içerisinde değer tipte olmayan üye elemanlar, managed type (yönetilen tip) olmasından dolayı göstericiler tarafından işaret edilememektedir. Bu yüzden göstericiler ile işaretlerken struct içerisinde değer tipin dışında başka elemanların bulunmaması gerekmektedir.

   UYARI: “int”, “char”, “double”, “decimal”, “float” vs. gibi türler değer tipli türlerdir.

   UYARI: “->” operatörüyle structın elemanlarına ulaşırken elemanların erişim belirleyicilerinin public olmalıdır.

3. What are all of the differences between the enumeration types of C++ and those of Java?

   C/C++'da numaralandırmalar aşağı yukarı tam sayılardır (başlık altında bu şekilde işlenirler). Enumlara metod veya benzer bir şey verilemez, temel olarak yalnızca enum türünü değiştirebilir ve onlara listelediğiniz değerleri verebilirsiniz.

   C++'dan farklı olarak, Java'da enumlar tamsayı sabiti değildir, herhangi bir arabirime veya sınıfa benzer sınıflardır. Bu nedenle Java'daki enum, enum int modeliyle mümkün olmayan tür güvenliği ve derleme zamanı denetimini sağlar. Javada enumlar donanımlı bir türdür ve onu bir sınıf veya arayüz olarak kullanabilirsiniz.

4. Design a set of simple test programs to determine the type compatibility rules of a C compiler to which you have access. Write a report of your findings.

   Test Programı 1: Temel Tip Dönüşümü

```c
#include <stdio.h>

int main() {
    int sayi = 10;
    float sonuc = sayi / 3.0;
    printf("Sonuç: %f\n", sonuc);

    return 0;
}
```

   Beklenen Çıktı: Sonuç: 3.333333

   Bu program, bir int (tamsayı) değeri float'a (kayan noktalı sayı) bölerken C derleyicisinin otomatik olarak tür dönüştürme yapıp yapmadığını test eder. Derleyici doğru bir şekilde işlerse, sonuç float (bir kayan noktalı) sayı olacaktır.

Test Programı 2: Örtük Dönüşüm (Implicit Conversion)

```c
#include <stdio.h>

int main() {
    int sayi = 68;
    char karakter = sayi;
    printf("Karakter: %c\n", karakter);

    return 0;
}
```

Beklenen Çıktı: Karakter: D

Bu program, C derleyicisinin otomatik olarak bir tamsayıdan bir karaktere örtük tür dönüştürmesi gerçekleştirip gerçekleştirmediğini test eder. Derleyici doğru bir şekilde işlerse, karakter değişkeni 'D'nin ASCII değerini tutacaktır.

Test Programı 3: Açık Dönüşüm (Explicit Conversion)

```c
#include <stdio.h>

int main() {
    float sayi = 4.29;
    int yuvarlatilmisSayi = (int) sayi;
    printf("Yuvarlanan Sayı: %d\n",yuvarlatilmisSayi);

    return 0;
}
```

Beklenen Çıktı: Yuvarlanmış Sayı: 4

Bu program, C derleyicisinin, atama operatörünü kullanarak açık tür dönüştürmeyi doğru bir şekilde işleyip işlemediğini test eder. Float değeri bir int (tamsayıya) dönüştürülmeli ve çıktı yuvarlanan sayıyı göstermelidir.

Test Programı 4: Tip Promosyonu (Type Promotion)

```c
#include <stdio.h>

int main() {
    int a = 24;
    double b = 8.14;
    double sonuc = a + b;
    printf("Sonuç: %lf\n", sonuc);


    return 0;
}
```

Beklenen Çıktı: Sonuç: 32.140000

Bu program, C derleyicisinin bir int (tamsayı) ve bir double toplanırken tip yükseltme yapıp yapmadığını test eder. Tamsayı (int), toplama işleminden önce bir double'a yükseltilmeli ve sonuç bir double değer olmalıdır.

Test Programı 5: İşlemlerde Tip Uyumluluğu

```c
#include <stdio.h>

int main() {
    int a = 27;
    int b = 8;
    float sonuc = (float)a / b;
    printf("Sonuç: %f\n", sonuc);

    return 0;
}
```

Beklenen Çıktı: Sonuç: 3.375000

Bu program, C derleyicisinin artimetik işlemlerde tür uyumluluğunu işleyip işlemediğini test eder. Tamsayı bölme işlemi yapılmalı ve ardından ondalık kısmı saklamak için sonuç bir kayan noktalı sayıya (float) dönüştürülmelidir.

Yazılan ve derlenen test programları doğrultusunda C derleyicisi hakkında tespit edilen bilgiler:

- Tip dönüşümü otomatik olarak yapabilir. Test Programı 1'de tamsayı (int) olan sayi float 3.0'a bölündüğünde sonuç float (kayan noktalı sayı) bir sayı olduğundan dolayı C derleyicisi otomatik olarak tip dönüşümü yapmıştır. Ve sonuç "Sonuç: 3.333333" olmuştur.

- Bir tamsayıdan bir karaktere örtük tip dönüşümü otomatik olarak gerçekleştirebilir. Test Programı 2'nin çıktısı "Karakter: D" şeklindedir. Karakter değişkeni D'nin ASCII kodu olan a sayısını tuttuğu için C otomatik olarak tip dönüştürmesi yapmıştır.

- Atama operatörünü kullanarak açık tür dönüşümü yapabilir. Test Programı 3'ün çıktısı "Yuvarlanmış Sayı: 4" şeklindedir. Float değer bir int'e (tamsayıya) dönüştürülmüş ve çıktı yuvarlanan sayıyı göstermiştir.

- Bir int ve bir double sayı toplanırken tip yükseltme yapılır. Test Programı 4'ün çıktısı "Sonuç: 32.140000" şeklindedir. Tamsayı (int), toplama işleminden önce bir double'a yükseltilmiş ve sonuç bir double değer olmuştur.

- C aritmetik işlemlerde tür uyumluluğunu sağlar. Test Programı 5'in çıktısı "Sonuç: 3.375000" şeklindedir. Tamsayı bölme işlemi yapılmış ve ardından ondalık kısmı saklamak için sonuç bir kayan noktalı sayıya (float) dönüştürülmüştür.

5. Run the following code on some system that supports C to determine the values of sum1 and sum2. Explain the results.

main.c

```c
#include <stdio.h>

int fun(int *k)
{
    *k += 4;
    return 3 * (*k) - 1;
}

void main()
{
    int i = 10, j = 10, sum1, sum2;
    sum1 = (i/2) + fun(&i);
    sum2 = fun(&j) + (j/2);

    printf("Sum1: %d", sum1);
    printf("\n");
    printf("Sum2: %d", sum2);
}
```

Output

```
/tmp/zYiKKywrb4.o
Sum1: 46
Sum2: 48
```

C kodu derleyicide çalıştırıldığında sum1'in değerinin 46 ve sum2'nin değeri 48 olduğu görülmüştür.

Kodlar incelendiğinde "fun" fonksiyonu "sum2"de "sum1"den farklı olarak daha önce çağırılmıştır.

"sum1" değerinin 46 olmasının nedeni:

14. satırda verilen işlem sırasına göre yapıldığında (i/2)'den "5" ve "fun(&i)" den "41" elde edilir. Bu sayıların toplanmasıyla "sum1"in değeri "46" olmuş olur.

"sum2" değerinin 48 olmasının nedeni:

Başlangıçta 10 olan j'nin değeri fonksiyonun çağırılması sonucunda 7. satırdaki işlemin (*k += 4;) yapılmasıyla 14'e yükselmiştir. "sum2" hesaplanırken "fun(&j)" fonksiyonunun çıktısı "41" ve j/2'den "7" gelince sum2 toplamının sonucu 48 olmuştur.

SONUÇ:

"Sum1" ve "sum2" değerleri elde edilirken yapılan işlemlerin sırasının farklı olması sonucu değiştirmiştir.

6. Rewrite the previous program of in C++, Java, and C#, run them, and compare the results.

- C ++:

main.cpp

```cpp
#include <iostream>

int fun(int *k)
{
    *k += 4;
    return 3 * (*k) - 1;
}

int main()
{
    int i = 10, j = 10, sum1, sum2;
    sum1 = (i/2) + fun(&i);
    sum2 = fun(&j) + (j/2);

    std::cout << "Sum1: " << sum1 << std::endl;
    std::cout << "Sum2: " << sum2 << std::endl;

    return 0;

}
```

Output

```
/tmp/OGmsfiufGt.o
Sum1: 46
Sum2: 48
```

- Java:

Main.java

```java
import java.io.*;

class Main {
    public static int fun(int[] k) {
        k[0] += 4;
        return 3 * (k[0]) - 1;
    }

    public static void main(String[] args) {
        int i = 10, j = 10, sum1, sum2;
        sum1 = (i/2) + fun(new int[]{i});
        sum2 = fun(new int[]{j}) + (j/2);
        System.out.println("Sum1: " + sum1);
        System.out.println("Sum2: " + sum2);
    }
}

```

Output

```
java -cp /tmp/8sBCCMk3lv Main
Sum1: 46
Sum2: 46
```

- C#:

Main.cs

```
1  using System;
2
3  class Program
4  {
5      static int Fun(ref int k)
6      {
7          k += 4;
8          return 3 * k - 1;
9      }
10
11     static void Main()
12     {
13         int i = 10, j = 10, sum1, sum2;
14         sum1 = (i / 2) + Fun(ref i);
15         sum2 = Fun(ref j) + (j / 2);
16
17         Console.WriteLine("Sum1: " + sum1);
18         Console.WriteLine("Sum2: " + sum2);
19     }
20 }
```

Output

```
mono /tmp/EAcGcGwlCw.exe
Sum1: 46
Sum2: 48
```

C, C++ ve C# dillerinde sum1 ve sum2'nin değerleri aynı çıkmıştır. Fakat Java'da aynı durum söz konusu değildir.

"sum1" ve "sum2" nin değerleri "fun" fonksiyonunun farklı sıralarda çağrılmasında dolayı C, C++ ve C# dillerinde farklı çıkmıştır.

Java'da ise "sum1" ve "sum2" değerleri aynı çıkmıştır. Kısaca Java dilinde metodun önce ve sonra çağrılması durumu değiştirmemiştir. İşlem önceliği olarak önce parantez içindeki işlem yapılmış sonra metod çağrılmıştır.

Sevgi Nur ÖKSÜZ 21360859073