Karadeniz Teknik Üniversitesi
Bilgisayar Mühendisliği Bölümü
Öğr. Gör. Selçuk Cevher

YDM 214 Mesleki İngilizce I
1. Arasınav, 1 Nisan 2011, 10:00
Süre : 90 dakika

NAME: **STUDENT ID:**

1. UNIX was originally written in **assembly** language. The assembler is simply named **A**. It was found by assembly programmers that certain things were done **over and over, repeatedly, more than once, etc** and to speed things up a bit, the language **B** was designed to automate some repetitive tasks.

2. C is **almost** context-free. The only exceptions are the characters **less than**, **greater than** and **colon**. What they mean depends on **where, location, context, etc** they are used.

3. There is no **prototype, declaration, etc** for main function in the header files. It's always declared with a return type of an **int**. Another way to define the function is with **two** arguments. **argc** is a count of the number of arguments and **argv** is an array of **pointers to strings**. No matter what, the pointer in the array at the offset of **argc** is a **null pointer**.

4. When the logic flow of your program comes to a **sequence point**, all operations are completed before it goes on. Any two operations performed on **opposite** sides of a sequence point are **safe** from the **side effects** of each other.

5. Please list the sequence points used in C, and specify the rule which should be followed while writing C statements to prevent negative impacts of any possible side effect.

   ; (semicolon)
   , (comma)
   || (logical or)
   && (logical and)
   ?: (conditional operator)
   function arguments

   Rule: If any value is being modified inside a statement, don ‘t use that same variable for anything else in that same statement.

6. Please explain the possible problems, if any, regarding the order of evaluation of side effects in the following C statements:

   Statement 1: `add( i + 1, i = j + 2 );`

   The arguments of a function call can be evaluated in any order. The expression `i + 1` may be evaluated before `i = j + 2`, or `i = j + 2` may be evaluated before `i + 1`. The result is different in each case.

   Statement 2: `myproc( getc(), getc() );`

   Likewise, it is not possible to guarantee what characters are actually passed to the `myproc` since the arguments of a function call can be evaluated in any order.

   Statement 3: `x[i] = i++;`

   The value of `x` that is modified is unpredictable. The value of the subscript could be either the new or the old value of `i`. The result can vary under different compilers or different optimization levels.

7. Please translate the following paragraph into Turkish language:

   “You should never write code that compares two floating point numbers for equality unless they got that way by one being assigned to the other. If you calculate either of the values, you'll need to decide how close they need to be to be considered equal. Define a constant of that size and test for the difference being between them no larger than that constant.”

Birbirlerine atanmadıkları sürece iki kayan noktalı sayıyı eşitlik açısından kıyaslayan kodu asla yazmamanız gerekir. Kıyaslanacak değerlerden birini hesaplarsan, eşit kabul edilebilmeleri için birbirlerine ne kadar yakın olmaları gerektiğine karar vermen gerekecektir. Bu boyutta bir sabit tanımlayın ve aralarındaki farkın bu sabitten daha büyük olup olmadığını test edin.

8. Please translate the following paragraph in English language:

"Tamsayı veri tipleri kullanılmadan önce boyutları kontrol edilmelidir. C dili ilk ortaya çıktığında yalnızca 8 tane tamsayı veri tipini içeriyordu. Günümüzde ise, bir tamsayı bildirimi yapmanın 15 farklı yolu mevcuttur."

The sizes of integer data types should be checked before being used. When C appeared for the first time, it had 8 integer data types. However, today, there are 15 different ways to declare an integer.