\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lecture 2: Introduction to Problem Solving and Programming Concepts

English Summary

This lecture establishes the foundational concepts of problem-solving within the software engineering discipline. It defines a problem as a situation requiring a solution under specific constraints. For software engineers, problem-solving is the mental activity of writing an original program to perform tasks while meeting all stated constraints. The lecture introduces essential terminology, including Computer , Hardware , Software , and a Program, which is an implementation of an algorithm in a programming language.

A core concept is the Algorithm, defined as a finite and ordered set of instructions to perform a task. Algorithms are language-independent plans that can be expressed through Pseudocode (a mix of natural language and code-like statements) or visualized with a Flowchart (a graphical representation using geometric shapes). Programming languages are classified by their level of abstraction: Low-level languages (like machine and assembly language) are hardware-dependent , while High-level languages are people-oriented and machine-independent. A Compiler is the tool that translates high-level source code into low-level machine code that the hardware can execute.

The lecture also introduces Programming Paradigms, which are the preferred approaches to programming. The two main classifications are Imperative (specifying how to solve a problem) and Declarative (specifying what problem to solve). This course focuses on Structured Programming, an imperative paradigm based on three fundamental control structures that eliminate chaotic program flow:

Sequence: Instructions are executed one after another in the order they are written.

Selection: A choice is made between two paths based on a condition (e.g., if/then/else).

Repetition / Iteration: A block of instructions is repeated as long as a condition is true (e.g., while, for).

The lecture concludes by justifying the choice of C/C++ for its high performance and the choice of structured programming as a good starting point for learning more advanced concepts like Object-Oriented Programming (OOP).

Türkçe Özet

Bu ders, yazılım mühendisliği disiplini içindeki problem çözmenin temel kavramlarını oluşturur. Bir problem, belirli kısıtlamalar altında bir çözüm gerektiren bir durum olarak tanımlanır. Yazılım mühendisleri için problem çözme, belirtilen tüm kısıtlamaları karşılarken görevleri yerine getirmek için özgün bir program yazmanın zihinsel faaliyetidir. Ders, bir programlama dilinde bir algoritmanın uygulanması olan Program , Yazılım , Donanım ve Bilgisayar gibi temel terminolojiyi tanıtır.

Temel bir kavram, bir görevi gerçekleştirmek için sonlu ve sıralı bir talimatlar kümesi olarak tanımlanan Algoritma'dır. Algoritmalar, Sözde Kod (Pseudocode) (doğal dil ve kod benzeri ifadelerin bir karışımı) aracılığıyla metinsel olarak ifade edilebilen veya bir Akış Şeması (Flowchart) (geometrik şekiller kullanılarak yapılan bir grafiksel temsil) ile görselleştirilebilen dilden bağımsız planlardır. Programlama dilleri soyutlama düzeylerine göre sınıflandırılır: Düşük seviyeli diller (makine ve assembly dili gibi) donanıma bağımlıyken , Yüksek seviyeli diller insan odaklı ve makineden bağımsızdır. Bir Derleyici, yüksek seviyeli kaynak kodunu donanımın yürütebileceği düşük seviyeli makine koduna çeviren bir araçtır.

Ders ayrıca, programlamaya tercih edilen yaklaşımlar olan Programlama Paradigmalarını da tanıtır. İki ana sınıflandırma, programcının bir problemi nasıl çözeceğini belirttiği İmperatif ve programcının sadece ne problemi çözeceğini belirttiği Deklaratif'tir. Bu ders, kaotik program akışını ortadan kaldıran üç temel kontrol yapısına dayanan imperatif bir paradigma olan Yapısal Programlama'ya odaklanır:

Sıralama (Sequence): Talimatlar yazıldıkları sırayla birbiri ardına yürütülür.

Seçim (Selection): Bir koşula göre iki yol arasında bir seçim yapılır (örneğin, if/then/else).

Tekrarlama / Yineleme (Repetition / Iteration): Bir talimat bloğu bir koşul doğru olduğu sürece tekrarlanır (örneğin, while, for).

Ders, yüksek performansı nedeniyle C/C++ seçimini ve Nesne Yönelimli Programlama (OOP) gibi daha ileri konuları öğrenmek için iyi bir başlangıç noktası olarak yapısal programlama seçimini gerekçelendirerek sona erer.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lecture 3: Software Development Life Cycle (SDLC)

English Summary

This lecture focuses on the Software Development Life Cycle (SDLC), a structured framework for creating high-quality software. It begins by defining the qualities of good software, which include efficiency (speed and resource usage), ease of maintenance, robustness (how it's affected by errors), and security. The SDLC is presented as a cyclical process composed of five essential phases:

Requirement Analysis: This is the initial and most critical phase, where the problem is fully understood, and a detailed list of what is expected from the software is created.

Modelling & Design: An abstract model of the solution is developed using tools like UML (Unified Modeling Language), flowcharts, or pseudocode. A good design is crucial as it directly impacts the final software quality.

Development: The design model is translated into source code using a chosen programming language.

Testing: The program is thoroughly checked for errors. This phase identifies and corrects Syntax Errors (violations of the programming language's rules) , Logical Errors (where the program runs but produces incorrect results) , and Run-time Errors (bugs that occur during execution).

Deployment & Maintenance: The final software is released to users, and crucial documentation explaining the work done at every stage is prepared. This phase also includes ongoing support and updates.

The lecture applies this framework to a practical example: building an Automated Teller Machine (ATM) system. The Requirement Analysis for the ATM is detailed, specifying functionalities like viewing balance, withdrawing cash, and depositing funds, as well as the user interface components (screen, keypad, cash dispenser, deposit slot). In the Modelling & Design phase, UML diagrams are introduced. A Use Case Diagram is used to show the interactions between the user (actor) and the system's main functions (e.g., Withdraw Cash). An Activity Diagram is used to model the workflow or sequence of events for a specific process, such as the steps involved in a cash withdrawal transaction.

Türkçe Özet

Bu ders, yüksek kaliteli yazılım oluşturmak için yapılandırılmış bir çerçeve olan Yazılım Geliştirme Yaşam Döngüsü (SDLC) üzerine odaklanmaktadır. İyi yazılımın niteliklerini tanımlayarak başlar; bu nitelikler arasında verimlilik (hız ve kaynak kullanımı), bakım kolaylığı, sağlamlık (hatalardan nasıl etkilendiği) ve güvenlik bulunur. SDLC, beş temel aşamadan oluşan döngüsel bir süreç olarak sunulur:

Gereksinim Analizi: Bu, problemin tam olarak anlaşıldığı ve yazılımdan beklenen her şeyin ayrıntılı bir listesinin oluşturulduğu ilk ve en kritik aşamadır.

Modelleme ve Tasarım: UML (Birleşik Modelleme Dili), akış şemaları veya sözde kod gibi araçlar kullanılarak çözümün soyut bir modeli geliştirilir. İyi bir tasarım, nihai yazılım kalitesini doğrudan etkilediği için hayati önem taşır.

Geliştirme: Tasarım modeli, seçilen bir programlama dili kullanılarak kaynak koduna çevrilir.

Test Etme: Program hatalar için kapsamlı bir şekilde kontrol edilir. Bu aşama, Sözdizimsel Hataları (programlama dilinin kurallarının ihlali) , Mantıksal Hataları (programın çalıştığı ancak yanlış sonuçlar ürettiği durumlar) ve Çalışma Zamanı Hatalarını (yürütme sırasında ortaya çıkan hatalar) tanımlar ve düzeltir.

Dağıtım ve Bakım: Nihai yazılım kullanıcılara sunulur ve projenin her aşamasında yapılan işi açıklayan önemli belgeler hazırlanır. Bu aşama ayrıca sürekli destek ve güncellemeleri de içerir.

Ders, bu çerçeveyi pratik bir örneğe uygular: bir Otomatik Vezne Makinesi (ATM) sistemi oluşturmak. ATM için Gereksinim Analizi, bakiye görüntüleme, para çekme ve para yatırma gibi işlevlerin yanı sıra kullanıcı arayüzü bileşenlerini (ekran, tuş takımı, para dağıtıcı, para yatırma yuvası) belirterek ayrıntılı olarak ele alınır. Modelleme ve Tasarım aşamasında, UML diyagramları tanıtılır. Bir Kullanım Senaryosu Diyagramı, kullanıcı (aktör) ile sistemin ana işlevleri (örneğin, Para Çekme) arasındaki etkileşimleri göstermek için kullanılır. Bir Etkinlik Diyagramı, nakit çekme işlemi gibi belirli bir sürecin iş akışını veya olaylar dizisini modellemek için kullanılır.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lecture 4: Algorithm, Pseudocode & Flowchart Details

English Summary

This lecture provides an in-depth exploration of the fundamental components of algorithms. It defines key terms essential for algorithm design:

Identifier: A special name defined by the programmer for variables, constants, functions, etc.. Rules for creating identifiers include using letters, numbers, and underscores, but not starting with a number.

Variable: A symbolic name given to a memory location where data can be stored and changed during program execution.

Constant: An identifier whose value is required to remain unchanged throughout the program.

Assignment Operation: The process of writing data to a variable, typically using the = symbol. For example, count = count + 1 increments the value of count.

Counter Variable: A specific type of variable used to track the number of times an operation is performed.

The lecture also details the various operators used in algorithms:

Arithmetic Operators: Used for mathematical calculations. The priority of these operations is crucial, with parentheses having the highest priority, followed by exponentiation, then multiplication/division, and finally addition/subtraction.

Comparison Operators: Used to compare two values, such as equals (==), not equals (!=), greater than (>), and less than (<).

Logical Operators: Used to combine multiple conditions. The basic operators are AND, OR, and NOT.

Finally, the lecture covers the core control structures:

Decision/Comparison Structure: Implemented using IF statements to alter the program's flow based on a condition.

Loop Structure: Used to repeat a block of operations a certain number of times or until a condition is met. Creating a loop involves setting an initial value for a loop variable, defining the step amount (increment/decrement), and specifying the end condition.

Türkçe Özet

Bu ders, algoritmaların temel bileşenlerini derinlemesine bir şekilde inceler. Algoritma tasarımı için gerekli olan anahtar terimleri tanımlar:

Tanımlayıcı (Identifier): Değişkenler, sabitler, fonksiyonlar vb. için programcı tarafından tanımlanan özel bir isimdir. Tanımlayıcı oluşturma kuralları arasında harfleri, sayıları ve alt çizgiyi kullanmak, ancak bir sayıyla başlamamak yer alır.

Değişken (Variable): Program yürütme sırasında verilerin saklanabildiği ve değiştirilebildiği bir bellek konumuna verilen sembolik bir addır.

Sabit (Constant): Değerinin program boyunca değişmeden kalması gereken bir tanımlayıcıdır.

Atama İşlemi (Assignment Operation): Genellikle = sembolü kullanılarak bir değişkene veri yazma işlemidir. Örneğin, sayac = sayac + 1 ifadesi sayac değerini artırır.

Sayıcı Değişkeni (Counter Variable): Bir işlemin kaç kez gerçekleştirildiğini izlemek için kullanılan özel bir değişken türüdür.

Ders ayrıca algoritmalarda kullanılan çeşitli operatörleri detaylandırır:

Aritmetik Operatörler: Matematiksel hesaplamalar için kullanılır. Bu işlemlerin önceliği hayati önem taşır; parantezler en yüksek önceliğe sahiptir, ardından üs alma, sonra çarpma/bölme ve son olarak toplama/çıkarma gelir.

Karşılaştırma Operatörleri: Eşittir (==), eşit değildir (!=), büyüktür (>) ve küçüktür (<) gibi iki değeri karşılaştırmak için kullanılır.

Mantıksal Operatörler: Birden fazla koşulu birleştirmek için kullanılır. Temel operatörler VE (AND), VEYA (OR) ve DEĞİL (NOT)'dir.

Son olarak, ders temel kontrol yapılarını kapsar:

Karar/Karşılaştırma Yapısı: Bir koşula göre programın akışını değiştirmek için IF ifadeleri kullanılarak uygulanır.

Döngü Yapısı: Bir işlem bloğunu belirli sayıda veya bir koşul karşılanana kadar tekrarlamak için kullanılır. Bir döngü oluşturmak, bir döngü değişkeni için bir başlangıç değeri belirlemeyi, adım miktarını (artırma/azaltma) tanımlamayı ve bitiş koşulunu belirtmeyi içerir.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lecture 5: Algorithm, Pseudocode and Flowchart Examples

English Summary

This lecture is entirely practical, designed to solidify the theoretical concepts of algorithm and flowchart design through a series of examples that increase in complexity. Each problem is presented with both its algorithm (written in steps) and its corresponding flowchart, demonstrating the direct translation from a logical plan to a visual representation.

The lecture begins with basic examples such as writing 'Hello World' to the screen, summing two predefined numbers, and then summing two numbers entered from the keyboard. It then moves to a slightly more complex problem involving a simple algebraic calculation (x2+y). A key concept introduced is the conditional loop, shown in an example where the program continues to accept numbers from the user and adds them to a sum variable until the total is greater than 100. This illustrates how a decision structure (IF sum <= 100 THEN goto STEP 3) can be used to control the repetition of a process.

A significant portion of the lecture is dedicated to solving the problem of factorial calculation, for which four different solutions are presented. This approach highlights that a single problem can have multiple valid algorithmic solutions. The examples demonstrate different loop structures: one using an incrementing counter (counter = counter + 1), another using a decrementing counter (counter = counter - 1), and two others implementing for loops with different start and end conditions.

The final, most advanced example involves estimating the square root of a number using an iterative method. This problem introduces a more complex algorithm that requires an initial guess (x) and an acceptable error value (ε). The algorithm is based on the mathematical approximation where a new, more accurate estimate (y) is calculated from the previous one (x) using the formula y=x+b, where b≅(a−x2)/2x. The process repeats, with the new estimate becoming the old one in the next iteration, until the absolute difference (b) is smaller than the acceptable error.

Türkçe Özet

Bu ders, algoritma ve akış şeması tasarımının teorik kavramlarını, karmaşıklığı artan bir dizi örnekle pekiştirmek amacıyla tamamen pratiktir. Her problem, hem adımlar halinde yazılmış algoritması hem de buna karşılık gelen akış şeması ile sunulur; bu, mantıksal bir plandan görsel bir temsile doğrudan çeviriyi gösterir.

Ders, ekrana 'Merhaba Dünya' yazdırmak, önceden tanımlanmış iki sayıyı toplamak ve ardından klavyeden girilen iki sayıyı toplamak gibi temel örneklerle başlar. Daha sonra, basit bir cebirsel hesaplama (x2+y) içeren biraz daha karmaşık bir probleme geçer. Tanıtılan önemli bir kavram, programın kullanıcıdan sayıları kabul etmeye devam ettiği ve toplam 100'den büyük olana kadar bir toplam değişkenine eklediği bir örnekte gösterilen koşullu döngüdür. Bu, bir karar yapısının (EĞER toplam <= 100 İSE ADIM 3'e git) bir sürecin tekrarını nasıl kontrol edebileceğini gösterir.

Dersin önemli bir bölümü, dört farklı çözümün sunulduğu faktöriyel hesaplama problemini çözmeye ayrılmıştır. Bu yaklaşım, tek bir problemin birden fazla geçerli algoritmik çözümü olabileceğini vurgular. Örnekler farklı döngü yapılarını sergiler: biri artan bir sayaç (sayac = sayac + 1), diğeri azalan bir sayaç (sayac = sayac - 1) ve diğer ikisi farklı başlangıç ve bitiş koşullarına sahip for döngülerini uygular.

Son ve en gelişmiş örnek, bir sayının karekökünü yinelemeli bir yöntemle tahmin etmeyi içerir. Bu problem, bir başlangıç tahmini (x) ve kabul edilebilir bir hata değeri (ε) gerektiren daha karmaşık bir algoritma sunar. Algoritma, yeni, daha doğru bir tahminin (y) bir önceki tahminden (x) y=x+b formülü kullanılarak hesaplandığı matematiksel bir yaklaşıma dayanır; burada b≅(a−x2)/2x'dir. Süreç, mutlak fark (b) kabul edilebilir hatadan daha küçük olana kadar, yeni tahminin bir sonraki yinelemede eski tahmin haline gelmesiyle tekrarlanır.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lecture 6: C++ Programming Language Concepts

English Summary

This lecture provides a comprehensive introduction to the C++ programming language. It begins with its history, noting that C++ is a continuation of the C programming language and was developed by Bjarne Stroustrup in 1979. A core concept is the build process of a C++ program, which involves several stages: the source code is written in an Editor, processed by a Preprocessor, translated into object code by a Compiler, linked with libraries by a Linker, and finally placed into memory by a Loader for the CPU to execute.

The general structure of a C++ program is introduced, highlighting the use of preprocessor directives like #include <iostream> to add library files, the using namespace std; declaration, and the main() function, which serves as the program's starting point. Several fundamental programming concepts are detailed:

Variables: A variable is a symbolic name for a memory location where data is stored. Variable names must adhere to specific rules, such as being composed of letters, numbers, and underscores, and not beginning with a digit. Key data types like int (for integers) and char (for single characters) are introduced.

Constants: These are identifiers whose values cannot be changed during program execution. They are defined either with the #define preprocessor directive or the const keyword.

Operators: C++ provides a rich set of operators. Arithmetic operators include +, -, \*, /, %, and special assignment operators like += and ++. Relational operators (==, !=, >, <) are used for comparisons , while Logical operators (&& for AND, || for OR) are used to combine conditions. The concept of Operation Priority is crucial, as it determines the order in which operators are evaluated in a complex expression.

Namespaces: This is a key concept for managing larger programs. A namespace provides a scope to identifiers (names of variables, functions, etc.) to avoid name conflicts that can occur when different parts of a program or different libraries use the same identifier names.

Library/Header Files: These files contain declarations for pre-written functions, allowing programmers to reuse code. For example, the <iostream> header is needed for input/output commands like cin and cout, <iomanip> is needed for formatting functions like setw() , and <cmath> is required for mathematical functions like sqrt().

Türkçe Özet

Bu ders, C++ programlama diline kapsamlı bir giriş yapar. Tarihçesiyle başlar, C++'ın C programlama dilinin bir devamı olduğunu ve 1979'da Bjarne Stroustrup tarafından geliştirildiğini belirtir. Temel bir kavram, bir C++ programının derleme sürecidir; bu süreç birkaç aşamadan oluşur: kaynak kod bir Düzenleyici'de yazılır, bir Ön İşlemci tarafından işlenir, bir Derleyici tarafından nesne koduna çevrilir, bir Bağlayıcı ile kütüphanelerle birleştirilir ve son olarak CPU'nun yürütmesi için bir Yükleyici tarafından belleğe yerleştirilir.

Bir C++ programının genel yapısı, kütüphane dosyalarını eklemek için #include <iostream> gibi ön işlemci direktiflerinin, using namespace std; bildiriminin ve programın başlangıç noktası olarak hizmet eden main() fonksiyonunun kullanımını vurgulayarak tanıtılır. Birkaç temel programlama kavramı detaylandırılmıştır:

Değişkenler: Bir değişken, verilerin saklandığı bir bellek konumu için sembolik bir addır. Değişken adları, harflerden, sayılardan ve alt çizgilerden oluşmak ve bir rakamla başlamamak gibi belirli kurallara uymalıdır. int (tamsayılar için) ve char (tek karakterler için) gibi anahtar veri türleri tanıtılır.

Sabitler: Bunlar, program yürütme sırasında değerleri değiştirilemeyen tanımlayıcılardır. Ya #define ön işlemci direktifi ya da const anahtar kelimesi ile tanımlanırlar.

Operatörler: C++ zengin bir operatör seti sunar. Aritmetik operatörler +, -, \*, /, % ve += ve ++ gibi özel atama operatörlerini içerir. İlişkisel operatörler (==, !=, >, <) karşılaştırmalar için kullanılırken , Mantıksal operatörler (&& VE için, || VEYA için) koşulları birleştirmek için kullanılır. İşlem Önceliği kavramı, operatörlerin karmaşık bir ifadede hangi sırayla değerlendirileceğini belirlediği için hayati önem taşır.

İsim Alanları (Namespaces): Bu, daha büyük programları yönetmek için önemli bir kavramdır. Bir isim alanı, bir programın farklı bölümleri veya farklı kütüphaneler aynı tanımlayıcı adlarını kullandığında ortaya çıkabilecek ad çakışmalarını önlemek için tanımlayıcılara (değişkenlerin, fonksiyonların adları vb.) bir kapsam sağlar.

Kütüphane/Başlık Dosyaları: Bu dosyalar, programcıların kodu yeniden kullanmasına olanak tanıyan önceden yazılmış fonksiyonlar için bildirimler içerir. Örneğin, <iostream> başlığı cin ve cout gibi girdi/çıktı komutları için, <iomanip> başlığı setw() gibi biçimlendirme fonksiyonları için ve <cmath> başlığı sqrt() gibi matematiksel fonksiyonlar için gereklidir