Foundations Of Sequential Programming

Learning Objectives

1. Understand the concept of ~~HLL~~ High Level Language computer
2. Explore the justifications for using High Level Languages
3. Evaluate arguments ~~for and~~ against High Level Language Computers
4. Gain insights into High Level Language Computer Systems and Architecture
5. Establish the relationship between High Level Languages and computer Architecture
6. Examine basic machine Architecture
7. Learn about the Specifications and translations of program Languague (P/L) blocks
8. Discuss Structured Languages and parameter passing mechanisms

Introduction

High Level Language Computers, also known as high level programming languages have played a ~~vital~~ pivotal role in the evolution of modern computing. These Languages are a bridge between the complex hardware of a computer and the human readable code that programmers use to instruct the ~~comp~~ machine. A high level Language Computer is a type of computer systems that allows programmers to write program in high Level programming languages. High Level programming Languages are designed to be more human readable and abstract making it easier for programmers to express their ideas in a way that is close to natural Language. These computers use compilers or interpreters to translate high Level code into machine code that the computer can execute.

HISTORY

The development of high level Languages can be traced back to the mid 20th century

Formula translation
FOTRAN was one of the first high level Languages created in the 1950's for scientific and engineering applications. It allowed programmers to write mathematical equation more naturally without worrying about the intrixes of the computer hardware. Over the years many other High level languages emerged with its own strength and purpose. COBOL (Common Business Oriented Language) was designed for business application. While Languages like C, C++ and Java became versatile choices for various domains including system programming, web development and game design

KEY CHARACTERISTICS AND JUSTIFICATION FOR HIGH LEVEL LANGUAGE COMPUTERS

High Level Languages possess several key characteristics:
① Abstraction: They abstract away low level hardware details such as memory management and CPU instructions simplifying

the programming process

② Portability: Code written in high level Languages is generally more portable because it can be compiled or interpreted on different ~~levels~~ platforms without significant modific-ations

③ Readability: High level code is more human readable and easier to understand, fostering collaborations among programmers

④ Productivity: Programmers can develop softwares more quickly and efficiently in high Level Languages, as they can focus on solving problems rather than dealing low-level technicalities

Arguments for and against High Level Language Computers

High level programming Languages have become an integral part of modern software

development, but they are not without criticism. Briefly we will explore the arguments both for and against High Level Language Computers

Arguments for (Advantages)

① Accessibility: One of the primary arguments in favour of High Level Language computers is their accessibility. These languages are designed to be more human readable and closer to natural language making programming more approachable for a wider audience. This accessibility has led to a more diverse and inclusive programming community.

2) Productivity: High Level Languages are known for their ability to boosts programmers productivity. They offer built in functions and libraries that simplifies complex tasks allowing developers to focus on solving higher level problems rather than getting bugged down in low level details like memory management or hardwar especification instruction

3) Portability: Code written in High Level Language

Mebelua 9/ 11/39

tend to be more portable. Programmers can write codeonce and run it on multiple platform with minimal modification, thanks to compilers and interpreters that translate High level code into machine specific instructions. This portability reduces development time and effort

4) Maintenance: High level codes are generally easy to maintain. It's readability and abstraction makes it simpler for multiple programmers to collaborate on projects, debug codes and make updates. This reduces the risk of error and facilitates long term software sustainability

5) Innovation: High level languages encourages innovation by enabling programmers to explore new ideas and develop software more ~~more~~ rapidly. They provide the flexibility to experiment with different approaches fostering creativity in software development

Examples of LLL : Assembly, C

Arguments against (challenges)

Peformance Overhead: Critics argue that high Level anguages introduced performance overhead compared to Low Level Languages use assembly language or C. The abstraction layers and additional operations can result in slower execution time and rate memory consumption, which can be critical in some applications like real time systems or high performance computing.

2) Lack of Control: High Level Languages abstract away many low level details of hardware which some developers view as a disadvantage. They argue that it's abstraction limits their control over the system and can make it challenging to optimize code for specific hardware architechure

3) Learning Curve: While High Level languages aim to be more accessible, critics argue that there is still a learning curve especially for beginners. Learning the syntax and semantics of a specific High Level Language can be

challenging and understanding the underlining concept of programming may require additional effort.