

# PRINCIPLES OF PROGRAMMING

CP 111

# COURSE OUTLINE

- An overview of programming languages
- Basic Concepts
- Names, Bindings, and Scopes
- Data Types
- Expressions and Assignment Statements
- Basic Program Structure and the Integrated Development Environment
- Algorithm Development Using Pseudocode
- Decision structures
- Loop Structures

# COURSE ACTIVITIES

- 3 Reading AssignmentS
- 3 TESTS (NOV, DEC & JAN)
- 3 QUIZZES
- Tutorials
- 9 Lecturers
- 1 University Examination

# OVERVIEW OF PROGRAMMING LANGUAGE

# What is programming language

- A programming language is a set of rules that provides a way of telling a computer what operations to perform.
- A programming language is a notational system for describing computation in a machine-readable and human-readable form.
- A programming language also has words, symbols and rules of grammar.

# What is programming language

- Tool for instructing machine
- Means for communicating between programmers
- Vehicle for expressing high level design
- It acts as a tool for experiment
- Controlling computerized devices

# Why study programming language ?

There are various reason regarding the study of programming languages  
but following are six main reasons

# Why study programming language

## 1. Increased capacity to express ideas.

- The depth at which people can think is heavily influenced by the expressive power of their language.
- Example use of drawings have helped people express themselves.

# Why study programming language

## 2. Improved background for choosing appropriate languages.

- Many professional programmers have a limited formal education in computer science, limited to a small number of programming languages.
- They are more likely to use languages with which they are most comfortable than the most suitable one for a particular job.
- If these programmers were familiar with a wider range of languages and language constructs, they would be better able to choose the language with the features that best address the problem.

# Why study programming language

## 3. Increased ability to learn new languages.

- Computer science is considered as a ever growing discipline especially most software technologies
- programming languages are not yet mature. Therefore, they are still evolving.
- The understanding of programming language design and implementation makes it easier to learn new languages.

# Why study programming language

## 4. Better understanding of the significance of implementation..

- It is often necessary to learn about language implementation; it can lead to a better understanding of why the language was designed the way that it was.
- Fixing some bugs requires an understanding of implementation issues.

# Why study programming language

## 5. Better use of languages that are already known.

- Some languages are better for some jobs than others.
  - (i) FORTRAN and APL for calculations, COBOL and RPG for report generation, LISP and PROLOG for AI, etc.
- By understanding how features are implemented, you can make more efficient use of them.

# Why study programming language

## 6. To make it easier to design a new language

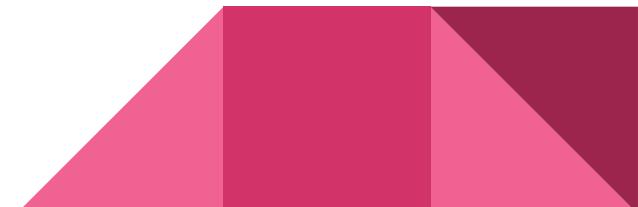
- Designing a new language require prior knowledge of previous one to make it effective, efficient and convenient to users.
- The previous knowledge as well as concepts are usual to design a new language irrespective of their work domains.

# Programming Domains

- Computers have been applied to different areas, from controlling nuclear power plants to providing video games in mobile phones.
- great diversity in computer use, programming languages with very different goals have been developed.

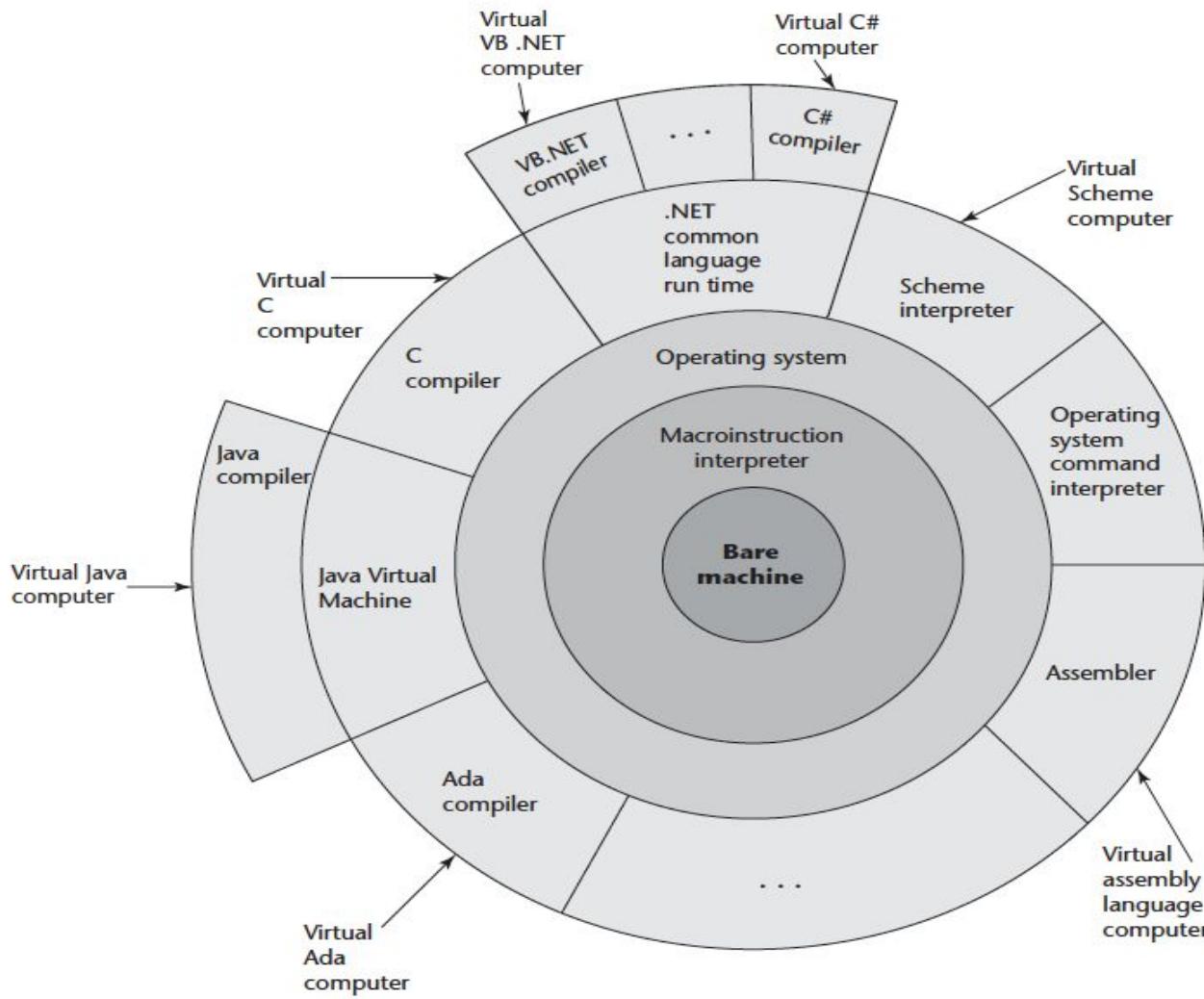
# Programming Domains

- Scientific Application (1940s and early 1950)
- Business Application (1950)
- Artificial Intelligence (1959)
- System Programming (1960s and 1970s)
- Web Software



# Compilation

- A programs can be translated into machine language, which can be executed directly on the computer
- Programming languages can be implemented by three general methods.
  - **Compiler implementation**
  - **Pure interpretation**
  - **Hybrid Implementation Systems**



# Compiler Interpretation

- Very fast program execution, once the translation process is complete
- The language that a compiler translates is called the **source language**
- The process of compilation and program execution takes place in **4 phases.**
  - **Lexical Analyzer**
  - **Syntax Analyzer**
  - **Intermediate code generator / semantic analyzer**
  - **Code Generator**

# Compiler interpretation phases

## 1. Lexical Analyzer

- Gathers the characters of the source program into lexical units.
- The lexical units of a program are identifiers, special words, operators, and punctuation symbols.

## 2. Syntax Analyzer

- Takes the lexical units from the lexical analyzer and uses them to construct hierarchical structures called parse trees.

# Compiler interpretation phases

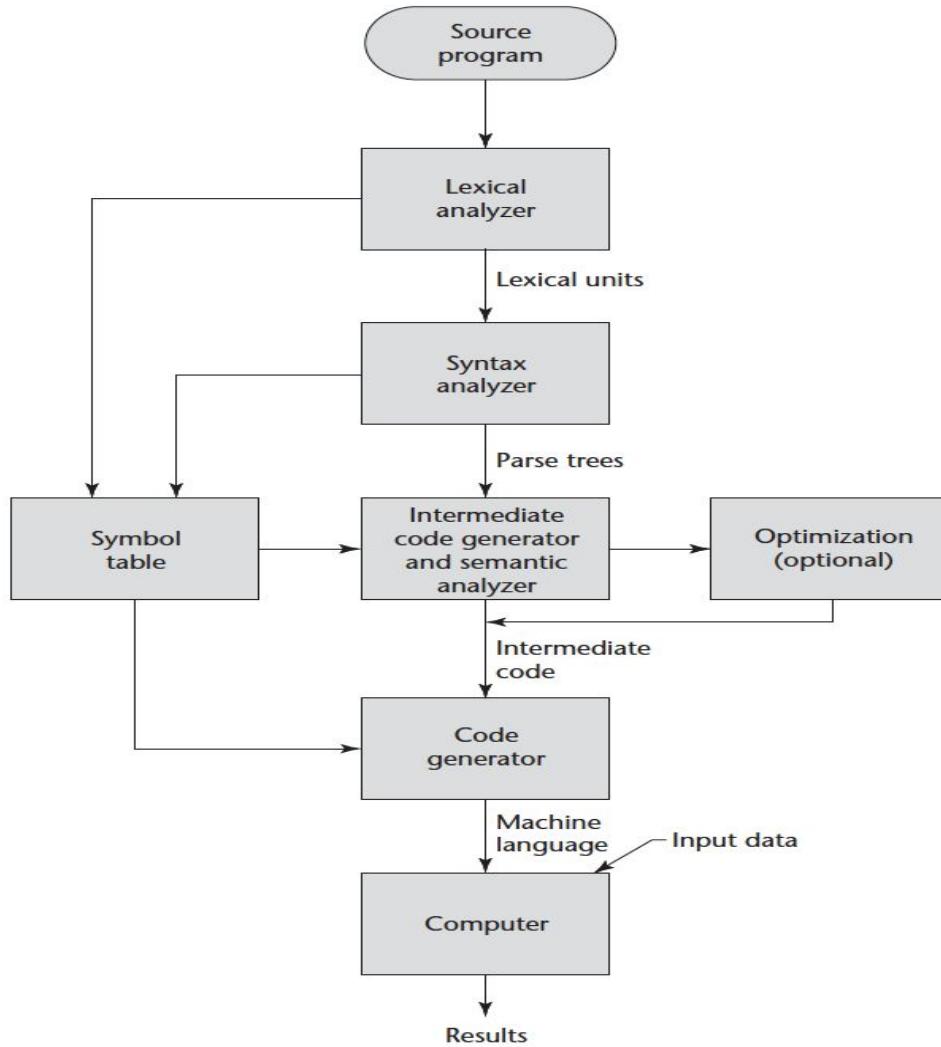
## 3. Intermediate code generator & semantic analyzer

- produces a program in a different language, at an intermediate level between the source program and the final output of the compiler: the machine language program.
- The semantic analyzer checks for errors, such as type errors, that are difficult, if not impossible, to detect during syntax analysis.

## 4. Code Generator

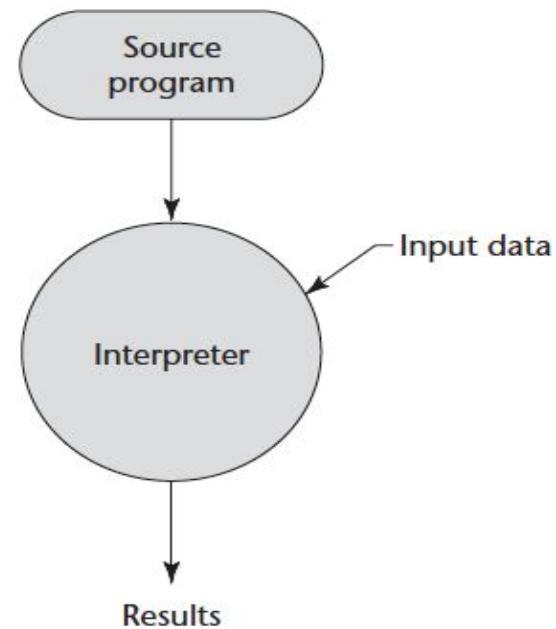
- translates the optimized intermediate code version of the program into an equivalent machine language program.

# The compilation process



# Pure Interpretation

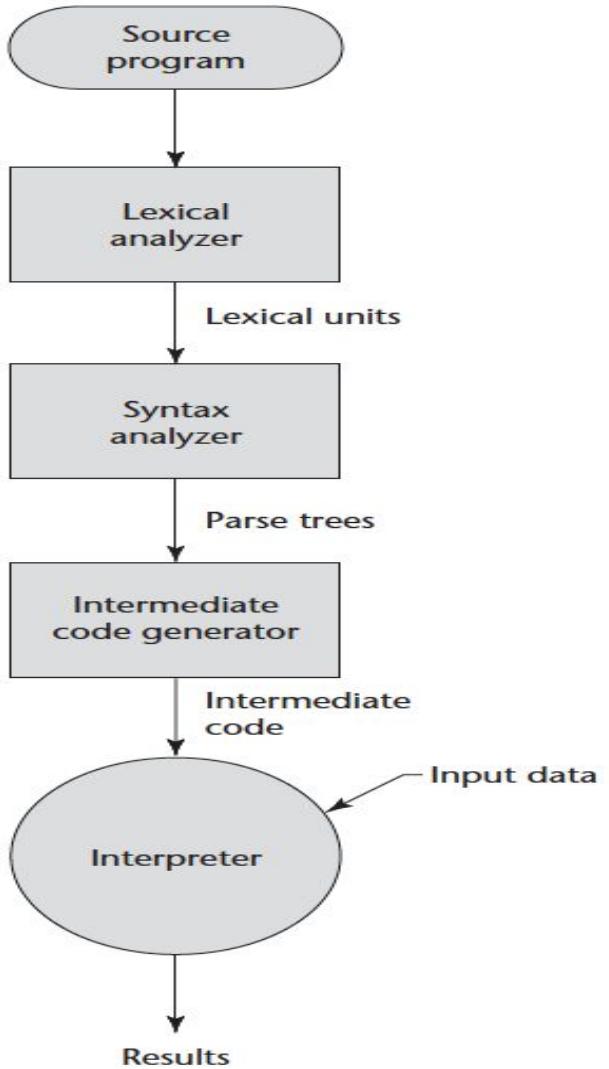
- A program is interpreted by another program called an interpreter, with no translation
- It allows easy implementation of many source-level debugging operations, because all run-time error messages can refer to source-level units.



# Hybrid implementation system

- It translates high-level language programs to an intermediate language designed to allow easy interpretation.
- This method is faster than pure interpretation because the source language statements are decoded only once.
- Instead of translating intermediate language code to machine code, it simply interprets the intermediate code

## Hybrid implementation system



# Preprocessors

- is a program that processes a program immediately before the program is compiled.
- Preprocessor instructions are embedded in programs.
- Preprocessor instructions are commonly used to specify that the code from another file is to be included.

# Levels of Programming language

**High-level program**

```
class Triangle {  
    ...  
    float surface()  
    return b*h/2;  
}
```

**Low-level program**

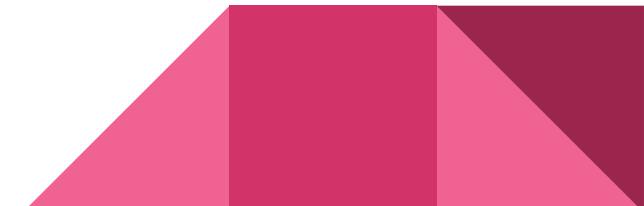
```
LOAD r1,b  
LOAD r2,h  
MUL r1,r2  
DIV r1,#2  
RET
```

**Executable Machine code**

```
0001001001000101  
0010010011101100  
10101101001...
```

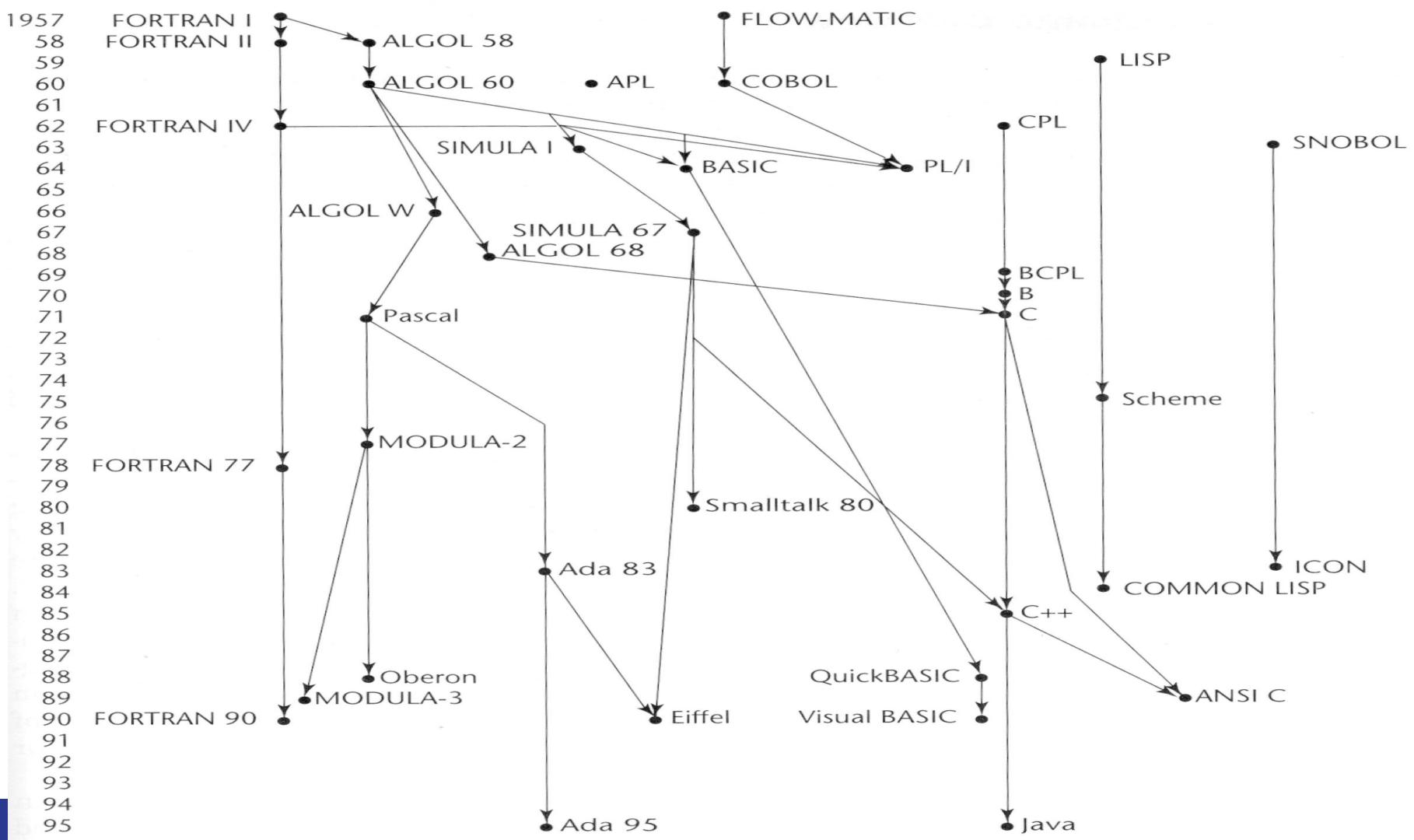
# Types of Programming Language

- First Generation Languages
- Second Generation Languages
- Third Generation Languages
- Fourth Generation Languages
- Fifth Generation Languages



# Reading assignment !

In each programming language generation, write its characteristics, limitations and at least three examples of programming language in that generation.



# Language Categories

Programming languages are often categorized into four bins:

- Imperative Programming (C)
- Object-Oriented Programming (C++)
- Logic/Declarative Programming (Prolog)
- Functional/Applicative Programming (Lisp)

# Imperative Programming

Imperative languages are command-driven or statement-oriented languages.

The basic concept is the machine state (the set of all values for all memory locations).

A program consists of a sequence of statements and the execution of each statement changes the machine state.

Programs take the form:

Statement1;

statement2;

# Object-Oriented Programming

It is based on classes of objects.

An object has variable components and is equipped with certain operations. Only these operations can access the object's variable components.

A class is a family of objects with similar variable components and operations.

Eg C++, Java

# Logic/Declarative Programming

It is based on a subset of predicate logic. Programs in logic programming languages are collections of facts and rules.

Sometimes called a Rule-based or declarative languages

It execute checking to see if a particular condition is true and if so, perform the appropriate actions.

Example

condition1 -> action 1

condition2 -> action 2

PROLOG was the ancestral logic language, and is still the most popular.

# Functional/Applicative Programming

An functional programming language looks at the function that the program represents rather than the state changes as each statement is executed.

The key question is: What function must be applied to our initial machine and our data to produce the final result?

The ancestral functional language was LISP, ML and HASKELL are modern functional languages.

it does support variables and assignment.

Statements take the form: function n(function1, function2, ... (data)) ... )

# Special Programming Languages

## Scripting Languages

- JavaScript and VBScript
- Php and ASP
- Perl and Python

## Command Languages

- sh, csh, bash

## Text processing Languages

- LaTex, PostScript

The background of the slide features a large, solid dark blue rectangle. In the top right corner, there is an abstract geometric pattern composed of several triangles. These triangles are primarily different shades of blue, creating a sense of depth and movement. The overall aesthetic is clean and modern.

# What determines a good language?

# Readability

The quality of a language that enables a programmer to understand and comprehend the nature of a computation easily and accurately.

## Simplicity:

- Multiplicity features - that is, having more than one way to accomplish a particular operation. (`count = count + 1`, `count += 1`, `count++`, `++count`)
- Operator overloading- a single operator symbol has more than one meaning.

## Orthogonality:

The quality of a language that features provided have as few restrictions as possible and be combinable in any meaningful way

# Readability

## Data Types:

The presence of adequate facilities for defining data types and data structures in a language is another significant aid to readability. Eg timeOut = 1 does not make sense as timeOut = true for a boolean case.

## Syntax Design:

Special words. **If** and **endif**, **loop** and **endloop**

# Writability

A measure of how easily a language can be used to create programs for a chosen problem domain.

## Simplicity:

If a language has a large number of different constructs, some programmers might not be familiar with all of them

## Orthogonality:

A smaller number of primitive constructs and a consistent set of rules for combining them (that is, orthogonality) is much better than simply having a large number of primitives.

# Writability

**Support for Abstraction:** the ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored.

**Expressivity:** it means that there are very powerful operators that allow a great deal of computation to be accomplished eg. Loops (for and while), increments (count++ easier than Count= Count + 1)

# Reliability

A program is said to be reliable if it performs to its specifications under all conditions

## Type Checking:

Simply testing for type errors in a given program, either by the compiler or during program execution.

- run-time type checking is expensive, compile-time type checking is more desirable.
- failure to type check, at either compile time or run time, has led to countless program errors eg C in variable mis-match.

# Reliability

**Exception Handling:** The ability of a program to intercept run-time errors (as well as other unusual conditions detectable by the program), take corrective measures, and then continue is an obvious aid to reliability

Eg. C++, Java, Ada, C# but practically nonexistent in many widely used languages, including C and Fortran.

**Aliasing:** having two or more distinct names that can be used to access the same memory cell. Eg two pointer pointing to one cell.

# Reliability

## Readability and Writability:

- Programs that are difficult to read are difficult both to write and to modify.
- The easier a program is to write, the more likely it is to be correct.

# Cost

- The cost of training programmers to use the language (simplicity and orthogonality)
- The cost of writing programs in the language (writability)
- The cost of compiling programs
- The cost of executing programs (run time type checking prohibits fast code execution)
- The cost of the language implementation system  
A language whose implementation system is either expensive or runs only on expensive hardware will have a much smaller chance of becoming widely used.
- The cost of maintaining programs

# Language evaluation criteria & the characteristics that affect them

Characteristic	CRITERIA		
	READABILITY	WRITABILITY	RELIABILITY
Simplicity	•	•	•
Orthogonality	•	•	•
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

# Programming Environment

- It refers to the environment in which programs are created, designed and tested.
- It consist of a set of support tools and command language for invoking them.

# Questions?