Principles of Programming Languages II (CSC 432)
LECTURE NOTES 1

A programming language is an artificial language designed for expressing algorithms on a computer.

A programming language is a **computer language** that is used by **programmers (developers) to communicate with computers**. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.

A programming language is mainly used to **develop desktop applications, websites, and mobile applications**.

The benefits of programming languages as a course

REASONS FOR STUDYING CONCEPTS OF PROGRAMMING LANGUAGES

1. Increased ability to express ideas/algorithms

In Natural language, the depth at which people think is influenced by the expressive power of the language they use. In programming language, the complexity of the algorithms that people Implement is influenced by the set of constructs available in the programming language

2. Improved background for choosing appropriate Languages

Many programmers use the language with which they are most familiar, even though poorly suited for their new project. It is ideal to use the most appropriate language.

3. Increased ability to learn new languages

For instance, knowing the concept s of object oriented programming OOP makes learning Java significantly easier and also, knowing the grammar of one's native language makes it easier to learn another language.

4. Better Understanding of Significance of implementation

5. Better use of languages that are already known

6. The overall advancement of computing

APPLICATION DOMAINS

1. Scientific Applications
2. Data processing Applications
3. Text processing Applications
4. Artificial intelligence Applications
5. Systems Programming Applications
6. Web software

SCIENTIFIC APPLICATIONS can be characterized as those which predominantly manipulate numbers and arrays of numbers, using mathematical and statistical principles as a basis for the algorithms. These algorithms encompass such problem as statistical significance test, linear programming, regression analysis and numerical approximations for the solution of differential and integral equations. FORTRAN, Pascal, Meth lab are programming languages that can be used here.

DATA PROCESSING APPLICATIONS can be characterized as those programming problems whose predominant interest is in the creation, maintenance, extraction and summarization of data in records and files.
COBOL is a programming language that can be used for data processing applications.

TEXT PROCESSING APPLICATIONS are characterized as those whose principal activity involves the manipulation of natural language text, rather than numbers as their data. SNOBOL and C language have strong text processing capabilities

ARTIFICIAL INTELLIGENCE APPLICATIONS are characterized as those programs which are designed principally to emulate intelligent behavior. They include game playing algorithms such as chess, natural language understanding programs, computer vision, robotics and expert systems. LISP has been the predominant AI programming language, and also PROLOG usingthe principle of "Logic programming" Lately AI applications are written in Java, c++ and python.

SYSTEMS PROGRAMMING APPLICATIONS involve developing those programs that interface the c omputersystem (the hardware) with the programmer and the operator. These programs include compilers, assembles, interpreters, input-output routines, program management facilities and schedules for utilizing and serving the various resources that comprise the system. Ada and Modula – 2 are examples of programming languages used here. Also is C.

WEB SOFTWARE
This is a collection of languages which include:
- Markup (e.g. XHTML)
- Scripting for dynamic content under which we have the o Client side, using scripts embedded in the XHTML documents e.g. Javascript, PHP o Server side, using the common Gateway interface e.g. JSP, ASP, PHP
- General- purpose, executed on the web server through cGI e.g. Java, C++.

CRITERIA FOR LANGUAGE EVALUATION AND COMPARISION
1. Expressivity
2. Well –Definedness
3. Data types and structures
4. Modularity
5. Input-Output facilities
6. Portability
7. Efficiency
8. Pedagogy
9. Generality

Expressivity means the ability of a language to clearly reflect the meaning intended by the algorithm designer (the programmer). Thus an "expressive" language permits an utterance to be compactly stated, and encourages the use of statement forms associated with structured programming (usually "while "loops and "if – then – else" statements).

By "well-definiteness", we mean that the language's syntax and semantics are free of ambiguity, are internally consistent and complete. Thus the implementer of a well-defined language should have, within its definition a complete specification of all the language's expressive forms and their meanings. The programmer, by the same virtue should be able to predict exactly the behavior of each expression before it is actually executed.

By "Data types and Structures", we mean the ability of a language to support a variety of data values (integers, real, strings, pointers etc.) and non-elementary collect ions of these.

Modularity has two aspects: the language's support for sub-programming and the language's extensibility in the sense of allowing programmer – defined operators and data types. By sub programming, we mean the ability to define independent procedures and functions (subprograms), and communicate via parameters or global variables with the invoking program.

In evaluating a language's "Input-Output facilities" we are looking at its support for sequential, indexed, and random access files, as well as its support for database and information retrieval functions.

A language which has **"portability"** is one which is implemented on a variety of computers. That is, its design is relatively"machine – independent". Languages which are well- defined tend to be more portable than others.

An **"efficient"** language is one which permits fast compilation and execution on the machines where it is implemented. Traditionally, FORTRAN and COBOL have been relatively efficient languages in their respective application areas.

Some languages have better **"pedagogy"** than others. That is, they are intrinsically easier to teach and to learn, they have better textbooks; they are implemented in a better program development environment, they are widely known and used by the best programmers in an application area.

Generality: Means that a language is useful in a wide range of programming applications. For instance, APL has been used in mathematical applications involving matrix algebra and in business applications as well.

INFLUENCES ON LANGUAGE DESIGN

1. Computer Architecture: Languages are developed around the prevalent computer architecture, known as the Von Neumann architecture (the most prevalent computer architecture).The connection speed between a computer's memory and its processor determines the speed of that computer. Program instructions often can be executed much faster than the speed of the connection; the connection speed thus, results in a bottleneck (Von Neumann bottleneck). It is the primary limiting factor in the speed of computers.

2. **Programming Methodologies:** New software development methodologies (e.g. object Oriented Software Development) led to new paradigms in programming and by extension, to new programming languages.

LANGUAGE PARADIGMS (Developments in Programming Methodology)

1. Imperative

This is designed around the Von Neumann architecture. Computation is performed through statements that change a program's state. Central features are variables, assignment statements and ileration, sequency of commands, explicit state update via assignment. Examples of such languages are Fortran, Algol, Pascal, e/c++, Java, Perl, Javascript, Visual BASIC.NET.

2. Functional

Here, the main means of making computations is by applying functions to parameters. Examples are LISP, Scheme, ML, Haskell. It may also include OO (Object Oriented) concepts.

3. Logic

This is Rule-based (rules are specified in no particular order). Computations here are made through a logical inference process. Examples are PROLOG and CLIPS. This may also include OO concepts.

TRADE-OFFS IN LANGUAGE DESIGN

1. Reliability Vs. Cost of Execution: For example, Java demands that all references to array elements be checked for proper indexing, which leads to increased execution costs.
2. Readability vs. Writability: - APL provides many powerful operators land a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability.
3. Writability (Flexibility) vs. reliability: The pointers in c++ for instance are powerful and very flexible but are unreliable.

IMPLEMENTATION METHODS

1. Compilation – Programs are translated into machine Language & System calls
2. Interpretation – Programs are interpreted by another program (an interpreter)
3. Hybrid – Programs translated into an intermediate language for easy interpretation
4. Just –in-time – Hybrid implementation, then compile sub programs code the first time they are called.
5. Pure Interptretation

COMPILATION

- Translated high level program (source language) into machine code (machine language)
- Slow translation, fast execution
- Compilation process has several phases

    *Lexical analysis converts characters in the source program into

*lexical units (e.g. identifiers, operators, keywords).
*Syntactic analysis: transforms lexical units into parse trees whichrepresent *he syntactic structure of the program.
*Semantics analysis check for errors hard to detect during syntacticanalysis; generate intermediate code.
*Code generation – Machine code is generated

- **INTERPRETATION**
- Easier implementation of programs (run-time errors can easily andimmediately be displayed).
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more memory space and is now rare3 for traditional high level languages.
- Significant comeback with some Web scripting languages like PHP and JavaScript.
- Interpreters usually implement as a read-eval-print loop:
*Read expression in the input language (usually translating it in some internal form)
*Evaluates the internal forms of the expression
*Print the result of the evaluation
*Loops and reads the next input expression until exit
- Interpreters act as a virtual machine for the source language:
*Fetch execute cycle replaced by the read-eval-print loop
*Usually has a core component, called the interpreter "run-time"that is a compile program running on the native machine.

HYBRID IMPLEMENTAITON
- This involves a compromise between compilers and pure interpreters. A high level program is translated to an intermediate language that allows easy interpretation.
- Hybrid implementation is faster than pure interpretation. Examples of the implementation occur in Perl and Java.
*Perl programs are partially compiled to detect errors before interpretation.
*Initial implementat6ions of Java were hybrid. The intermediate form, byte code, provides portability to any machine that has a bytecode interpreter and a run time system (together, these are called Java Virtual Machine).

JUST-IN-TIME IMPLEMENTATION
This implementation initially translates programs to an intermediate language then compile the intermediate language of the subprograms into machine code when they are called.
- Machine code version is kept for subsequent calls. Just-in-time systems are widely used for Java programs. Also .NET languages are implemented with a JIT system.

Study Questions:
1. Why is it useful for a programmer to have some background in language design, even though he or she may never actually design a programming language?
2. What does it mean for a program to be reliable?

3. What is: Aliasing?; What is exceptional handling?
4. Why is readability important in writability?
5. What are the three fundamental features of an object-oriented language?
6. What role does symbol table play in compiler?
7. What does a linker do?
8. What are the advantages in implementing a language with pure interpreter?
9. What are the three general methods of implementing a programming language?
10. Which produces faster program execution, a compiler or a pure interpreter and how?

A BRIEF HISTORY OF PROGRAMMING LANGUAGES.
Assembly languages
IBM 704 and Fortran–FORmula TRANslation
LISP –LISt Processing
ALGOL 60 – International Algorithmic language
Simul 67 – First object oriented language
Ada – history's largest design effort
C++ - Combining Imperaive and Object – Oriented Features
Java – An Imperative – Based Object – Oriented language
Prolog– Logic Programming

ALGOL
ALGOL 68 (short for **ALGO**rithmic **L**anguage 19**68**) is an imperative computer programming language that was conceived as a successor to the ALGOL 60 programming language, designed with the goal of a much wider scope of application and more rigorously defined syntax and semantics. ALGOL 68 features include expression-based syntax, user-declared types and structures/tagged-unions, a reference model of variables and reference parameters, string and array and matrix slicing and also concurrency.
ALGOL 68 was designed by IFIP Working Group 2.1. On December 20, 1968 the language was formally adopted by Working Group 2.1 and subsequently approved for publication by the General Assembly of IFIP.
ALGOL 68 was defined using a two-level grammar formalism invented by Adriaanvan Wijngaarden. Van Wijngaarden grammars use a context-free grammar to generate an infinite set of productions that will recognize a particular ALGOL 68 program; notably, they are able to express the kind of requirements that in many other programming language standards are labelled.

Notable language elements
Bold symbols and reserved words
There are 61 such reserved words (some with "brief symbol" equivalents) in the standard sub-language: **mode**, **op**, **prio**, **proc**, **flex**, **heap**, **loc**, **long**, **ref**, **short**, **bits**, **bool**, **bytes**, **char**, **compl**, **int**, **real**, **sema**, **string**, **void**, **channel**, **file**, **format**, **struct**, **union**, **of**, **at** "@", **is** ":=:", **isnt**":/=:", ":≠:", **true**, **false**, **empty**, **nil** "○", **skip** "~", **co comment** "¢", **pr**, **pragmat**, **case in ouse*in* outesac**"(~ | ~ |: ~ | ~ | ~)", **for from to by while do od**,

if then elif*then* **else fi** "(~ | ~ |: ~ | ~ | ~)", **par begin end** "(~)", ***go to, **goto**, **exit** ".". **mode: Declarations**
The basic data types (called **modes** in *ALGOL 68* parlance) are **real**, **int**, **compl**(complex number), **bool**, **char**, **bits** and **bytes**. For example:
intn = 2;
con is a fixed constant of 2.**co**
intm := 3;
com is a newly created local variable whose value is initially set to 3.
This is short for **ref int**m = **locint**:= 3; **co**
Real avogadro = 6.0221415▯23; **co** Avogadro's number **co**
longlong real pi = 3.14159 26535 89793 23846 26433 83279 50288 41971 69399 37510;
Compl *square root of minus one* = 0 ⊥1
However, the declaration **real** *x;* is just syntactic sugar for **ref real** *x* = **loc real**;. That is, *x* is really the *constant identifier* for a *reference to* a newly generated local **real** variable.

Special characters
Most of Algol's "special" characters (×, ÷, ≤, ≥, ≠, ¬, ∨, ∧, ▯, →, ↓, ↑, □, ⌊, ⌈, ⎩, ⎧, ○, ⊥and ¢) can be found on the IBM 2741 keyboard with the APL "golf-ball" print head inserted, these became available in the mid 1960s while ALGOL 68 was being drafted. These characters are also part of the unicode standard and most of them are available in several popular fonts.

Transput: Input and output
Transput is the term used to refer to ALGOL 68's input and output facilities. There are pre-defined procedures for unformatted, formatted and binary transput. Files and other transput devices are handled in a consistent and machine-independent manner. The following example prints out some unformatted output to the standard output device: print ((newpage, "Title", newline, "Value of i is ", i, "and x[i] is ", x[i], newline))
Note the predefined procedures newpage and newline passed as arguments.

C++
C++ (pronounced "see plus plus") is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate levellanguage, as it comprises a combination of both high-level and low-level language features. It was developed by Bjarne Stroustrup starting in 1979 at Bell Labs as an enhancement to the C language and originally named C with Classes. It was renamed C++ in 1983.
C++ is one of the most popular programming languages and its application domains include systems software (such as Microsoft Windows), application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games.
C++ is sometimes called a hybrid language; it is possible to write object oriented or procedural code in the same program in C++. This has caused some concern that some

C++ programmers are still writing procedural code, but are under the impression that it is object orientated, simply because they are using C++. Often it is an amalgamation of the two. This usually causes most problems when the code is revisited or the task is taken over by another coder.
C++ continues to be used and is one of the preferred programming languages to develop professional applications.

Language features
C++ inherits most of C's syntax. The following is Bjarne Stroustrup's version of the Hello world program that uses the C++ standard library stream facility to write a message to standard output:

```
#include <iostream>
int main()
{
std::cout<< "Hello, world!\n";
}
```

Within functions that define a non-void return type, failure to return a value before control reaches the end of the function results in undefined behaviour (compilers typically provide the means to issue a diagnostic in such a case). The sole exception to this rule is the main function, which implicitly returns a value of zero.

Operators and operator overloading
C++ provides more than 35 operators, covering basic arithmetic, bit manipulation, indirection, comparisons, logical operations and others. Almost all operators can be overloaded for user-defined types, with a few notable exceptions such as member access (. and .*). The rich set of overloadable operators is central to using C++ as a domain-specific language. The overloadable operators are also an essential part of many advanced C++ programming techniques, such as smart pointers. Overloading an operator does not change the precedence of calculations involving the operator, nor does it change the number of operands that the operator uses (any operand may however be ignored by the operator, though it will be evaluated prior to execution).
Overloaded "&&" and "||" operators lose their short-circuit evaluation property.

C#
During the development of the .NET Framework, the class libraries were originally written using a managed code compiler system called Simple Managed C (SMC). In January 1999, Anders Hejlsberg formed a team to build a new language at the time called Cool, which stood for "C-like Object Oriented Language". Microsoft had considered keeping the name "Cool" as the final name of the language, but chose not to do so for trademark reasons. By the time the .NET project was publicly announced at the July 2000 Professional Developers Conference, the language had been renamed C#, and the class libraries and ASP.NET runtime had been ported to C#.
Some notable distinguishing features of C# are:

1. It has no global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.
2. Local variables cannot shadow variables of the enclosing block, unlike C and C++. Variable shadowing is often considered confusing by C++ texts.
3. C# supports a strict Boolean data type, bool. Statements that take conditions, such as while and if, require an expression of a type that implements the true operator, such as the boolean type. While C++ also has a boolean type, it can be freely converted to and from integers, and expressions such as if(a) require only that a is convertible to bool, allowing a to be an int, or a pointer. C# disallows this "integer meaning true or false" approach, on the grounds that forcing programmers to use expressions that return exactly bool can prevent certain types of common programming mistakes in C or C++ such as if (a = b) (use of assignment = instead of equality ==).
4. In addition to the try...catch construct to handle exceptions, C# has a try...finally construct to guarantee execution of the code in the finally block.
5. C# currently (as of version 4.0) has 77 reserved words.
6. Multiple inheritances are not supported, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication and simplify architectural requirements throughout CLI.

Common Type System (CTS)
C# has a unified type system. This unified type system is called Common Type System (CTS). A unified type system implies that all types, including primitives such as integers, are subclasses of the System. Object class. For example, every type inherits a ToString() method. For performance reasons, primitive types (and value types in general) are internally allocated on the stack.

Categories of data types
CTS separate data types into two categories:
1. **Value types:** they are plain aggregations of data. Instances of value types do not have referential identity nor a referential comparison semantics - equality and inequality comparisons for value types compare the actual data values within the instances, unless the corresponding operators are overloaded. Value types are derived from System. ValueType, always have a default value, and can always be created and copied. Some other limitations on value types are that they cannot derive from each other (but can implement interfaces) and cannot have an explicit default (parameterless) constructor. Examples of value types are all primitive types, such as int (a signed 32-bit integer), float (a 32-bit IEEE floating-point number), char (a 16-bit Unicode code unit), and System. DateTime (identifies a specific point in time with nanosecond precision). Other examples are enum (enumerations) and struct (user defined structures).

2. **Reference types:** they have the notion of referential identity - each instance of a reference type is inherently distinct from every other instance, even if the data within both instances is the same. This is reflected in default equality and inequality

comparisons for reference types, which test for referential rather than structural equality, unless the corresponding operators are overloaded (such as the case for System. String). In general, it is not always possible to create an instance of a reference type, nor to copy an existing instance, or perform a value comparison on two existing instances, though specific reference types can provide such services by exposing a public constructor or implementing a corresponding interface (such as ICloneable or IComparable). Examples of reference types are object (the ultimate base class for all other C# classes), System. String (a string of Unicode characters), and System. Array (a base class for all C# arrays). Both type categories are extensible with user-defined types.

Study Questions

1. Make an educated guess as to the most common syntax error in Lisp programs.
2. Describe in detail the three most important reasons, in your opinion, why ALGOL 60 did not become a very used language.
3. Do you think language design committee is a good idea? Support your opinion.
4. Give a brief general description of a markup/programming hybrid language
5. Why in your opinion, do new scripting languages appear more frequently than new compiled languages?
6. Write a program to implement N factorial in; Machine Language, Assembly Language, Scripting Language and any other five high level languages.