Course Objectives

- Improve the background for choosing appropriate programming languages
- Be able to program in *procedural*, *object-oriented*, *functional*, and *logical* programming languages
- Understand the significance of the design of a programming language and its implementation in a *compiler* or *interpreter*
- Enhance the ability to learn new programming languages
- Increase the capacity to express general programming concepts and to choose among alternative ways to express things in a particular programming language
- Simulate useful features in languages that lack them
- Understand how programs are parsed and translated by a compiler
- Be able, in principle, to design a new programming language
Course Outline

1. **Introduction**: History, overview, and classification of programming languages
2. **Functional Programming**: Programming with Scheme and Haskell
3. **Logic Programming**: Programming with Prolog
4. **Compilers and Interpreters**: How programs are translated into machine code
5. **Syntax**: How syntax is defined and how syntax can impact ease-of-use
6. **Semantics**: How the meaning and behavior of programming constructs can be defined and interpreted
7. **Axiomatic Semantics**: How programs can be analyzed and proven correct
8. **Names, Scopes, and Bindings**: How and when bindings for local names are defined in languages with scoping rules
9. **Control Flow**: How programming constructs define control flow and how the choice of constructs can affect programming style
10. **Subroutines and Parameter Passing**: How the subroutine calling mechanism is implemented and how and when parameters are passed and evaluated
11. **Exception Handling**: How to improve the robustness of programs
Important Events in Programming Language History

- 1940s: The first electronic computers were monstrous contraptions
  - Programmed in binary machine code by hand via switches and later by card readers and paper tape readers
  - Code is not reusable or relocatable
  - Computation and machine maintenance were difficult: machines had short mean-time to failure (MTTF) because vacuum tubes regularly burned out
  - The term “bug” originated from a bug that reportedly roamed around in a machine causing short circuits
Assembly Languages

- **Assembly languages** were invented to allow machine operations to be expressed in mnemonic abbreviations
  - Enables larger, reusable, and relocatable programs
  - Actual machine code is produced by an assembler
  - Early assemblers had a one-to-one correspondence between assembly and machine instructions

- “**Speedcoding**”: expansion of macros into multiple machine instructions to achieve a form of higher-level programming
Assembly Language Example

Example MIPS assembly program to compute GCD

Example MIPS R4000 machine code of the assembly program

Actual MIPS R4400 IC
The First High-Level Programming Language

- Mid 1950s: development of **FORTRAN** (FORmula TRANslator), the arguably first higher-level language
  - Finally, programs could be developed that were machine independent!

- Main computing activity in the 50s: solve numerical problems in science and engineering

- Other high-level languages soon followed:
  - **Algol 58** was an improvement compared to Fortran
  - **COBOL** for business computing
  - **Lisp** for symbolic computing and artificial intelligence
  - **BASIC** for "beginners"
  - **C** for systems programming
FORTRAN 77 Example

PROGRAM GCD

C variable names that start with
C I, J, K, L, N, M are integers
C
C read the parameters
READ (*, *) I, J
C loop while I!=J
10 IF I .NE. J THEN
    IF I .GT. J THEN
        I = I - J
    ELSE
        J = J - I
    ENDIF
    GOTO 10
ENDIF

C write result
WRITE (*, *) 'GCD =', I
END
Important Events in Programming Language History

- 1980s: Object-oriented programming
  - Important innovation for software development
    - Encapsulation and inheritance
    - Dynamic binding
  - The concept of a “class” is based on the notion of an “abstract data type” (ADT) in Simula 67, a language for discrete event simulation that has class-like types but no inheritance
Genealogy of Programming Languages

- Fortran I
- Fortran II
- Fortran IV
- Basic
- PL/I
- Algol 58
- Algol 60
- Algol W
- Pascal
- Modula-2
- Modula-3
- Ada
- Ada 95
- Java
- Smalltalk 80
- C
- C++
- Eiffel
- Scheme
- Simula
- Simula 67
- BCPL
- Lisp
- Lisp
- ML
- Common Lisp
- CLOS
- Miranda
- Haskell
Overview: FORTRAN I,II,IV,77

PROGRAM AVEX
INTEGER INTLST(99)
ISUM = 0
C read the length of the list
READ (*, *) LSTLEN
IF ((LSTLEN .GT. 0) .AND. (LSTLEN .LT. 100)) THEN
C read the input in an array
DO 100 ICTR = 1, LSTLEN
READ (*, *) INTLST(ICTR)
ISUM = ISUM + INTLST(ICTR)
100 CONTINUE
C compute the average
IAVE = ISUM / LSTLEN
C write the input values > average
DO 110 ICTR = 1, LSTLEN
IF (INTLST(ICTR) .GT. IAVE) THEN
WRITE (*, *) INTLST(ICTR)
END IF
110 CONTINUE
ELSE
WRITE (*, *) 'ERROR IN LIST LENGTH'
END IF
END

- FORTRAN had a dramatic impact on computing in early days
- Still used for numerical computation
FORTRAN 90, 95, HPF

PROGRAM AVEX
INTEGER INT_LIST(1:99)
INTEGER LIST_LEN, COUNTER, AVERAGE
  C read the length of the list
  READ (*, *) LISTLEN
  IF ((LIST_LEN > 0) .AND. (LIST_LEN < 100)) THEN
    C read the input in an array
    DO COUNTER = 1, LIST_LEN
      READ (*, *) INT_LIST(COUNTER)
    END DO
    C compute the average
    AVERAGE = SUM(INT_LIST(1:LIST_LEN)) / LIST_LEN
    C write the input values > average
    DO COUNTER = 1, LIST_LEN
      IF (INT_LIST(COUNTER) > AVERAGE) THEN
        WRITE (*, *) INT_LIST(COUNTER)
      END IF
    END DO
  ELSE
    WRITE (*, *) 'ERROR IN LIST LENGTH'
  END IF
END

- Major revisions
  - Recursion
  - Pointers
  - Records
- New control constructs
  - while-loop
- Extensive set of array operations
- HPF (High-Performance Fortran) includes constructs for parallel computation
Lisp

(DEFINE (avex lis)
    (filtergreater lis (/ (sum lis) (length lis)))
)

(DEFINE (sum lis)
    (COND
      ((NULL? lis) 0)
      (ELSE (+ (CAR lis) (sum (CDR lis)))))
)

(DEFINE (filtergreater lis num)
    (COND
      ((NULL? lis) '())
      ((> (CAR lis) num) (CONS (CAR lis)
        (filtergreater (CDR lis) num))
      (ELSE (filtergreater (CDR lis) num)
    )
  )
)

- Lisp (LIst Processing)
- The original functional language developed by McCarthy as a realization of Church's lambda calculus
- Many dialects exist, including Common Lisp and Scheme
- Very powerful for symbolic computation with lists
- Implicit memory management with garbage collection
- Influenced functional programming languages (ML, Miranda, Haskell)
Algol 60

The original block-structured language
- Local variables in a statement block
- First use of Backus-Naur Form (BNF) to formally define language grammar
- All subsequent imperative programming languages are based on it
- No I/O and no character set
- Not widely used in the US
- Unsuccessful successor Algol 68 is large and relatively complex

comment avex program
begin
    integer array intlist [1:99];
    integer listlen, counter, sum, average;
    sum := 0;
    comment read the length of the input list
    readint (listlen);
    if (listlen > 0) L (listlen < 100) then
        begin
            comment read the input into an array
            for counter := 1 step 1 until listlen do
                begin
                    readint (intlist[counter]);
                    sum := sum + intlist[counter]
                end;
            comment compute the average
            average := sum / listlen;
            comment write the input values > average
            for counter := 1 step 1 until listlen do
                if intlist[counter] > average then
                    printint (intlist[counter])
            end
        else
            printstring ("Error in input list length")
        end
end
COBOL

IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 FAHR PICTURE 999.
77 CENT PICTURE 999.

PROCEDURE DIVISION.
DISPLAY 'Enter Fahrenheit' UPON CONSOLE.
ACCEPT FAHR FROM CONSOLE.
DISPLAY 'Celsius is ' CENT UPON CONSOLE.
GOBACK.

- Originally developed by the Department of Defense
- Intended for business data processing
- Extensive numerical formatting features and decimal number storage
- Introduced the concept of records and nested selection statement
- Programs organized in divisions: IDENTIFICATION: Program identification
  ENVIRONMENT: Types of computers used
  DATA: Buffers, constants, work areas
  PROCEDURE: The processing parts (program logic).
BASIC

REM avex program
DIM intlist(99)
sum = 0
REM read the length of the input list
INPUT listlen
IF listlen > 0 AND listlen < 100 THEN
REM read the input into an array
FOR counter = 1 TO listlen
   INPUT intlist(counter)
   sum = sum + intlist(counter)
NEXT counter
REM compute the average
average = sum / listlen
REM write the input values > average
FOR counter = 1 TO listlen
   IF intlist(counter) > average THEN
      PRINT intlist(counter);
   NEXT counter
ELSE
   PRINT "Error in input list length"
END IF
END
AVEX: PROCEDURE OPTIONS (MAIN);
DECLARE INTLIST (1:99) FIXED;
DECLARE (LISTLEN, COUNTER, SUM, AVERAGE) FIXED;
SUM = 0;
/* read the input list length */
GET LIST (LISTLEN);
IF (LISTLEN > 0) & (LISTLEN < 100) THEN
DO;
/* read the input into an array */
DO COUNTER = 1 TO LISTLEN;
GET LIST (INTLIST(COUNTER));
SUM = SUM + INTLIST(COUNTER);
END;
/* compute the average */
AVERAGE = SUM / LISTLEN;
/* write the input values > average */
DO COUNTER = 1 TO LISTLEN;
IF INTLIST(COUNTER) > AVERAGE THEN
   PUT LIST (INTLIST(COUNTER));
END;
ELSE
   PUT SKIP LIST ("ERROR IN INPUT LIST LENGTH");
END AVEX;

- Developed by IBM
  - Intended to replace FORTRAN, COBOL, and Algol
- Introduced exception handling
- First language with pointer data type
- Poorly designed, too large, too complex
Ada and Ada95

with TEXT_IO;
use TEXT_IO;

procedure AVEX is
  package INT_IO is new INTEGER_IO (INTEGER);
  use INT_IO;
  type INT_LIST_TYPE is array (1..99) of INTEGER;
  INT_LIST : INT_LIST_TYPE;
  LIST_LEN, SUM, AVERAGE : INTEGER;
begin
  SUM := 0;
  -- read the length of the input list
  GET (LIST_LEN);
  if (LIST_LEN > 0) and (LIST_LEN < 100) then
    -- read the input into an array
    for COUNTER := 1 .. LIST_LEN loop
      GET (INT_LIST(COUNTER));
      SUM := SUM + INT_LIST(COUNTER);
    end loop;
    -- compute the average
    AVERAGE := SUM / LIST_LEN;
    -- write the input values > average
    for counter := 1 .. LIST_LEN loop
      if (INT_LIST(COUNTER) > AVERAGE) then
        PUT (INT_LIST(COUNTER));
        NEW_LINE;
      end if
    end loop;
  else
    PUT_LINE ("Error in input list length");
  end if;
end AVEX;

- Originally intended to be the standard language for all software commissioned by the US Department of Defense
- Very large
- Elaborate support for packages, exception handling, generic program units, concurrency
- Ada 95 is a revision developed under government contract by a team at Intermetrics, Inc.
  - Adds objects, shared-memory synchronization, and several other features
Smalltalk-80

class name           Avex
superclass           Object
instance variable names  intlist

"Class methods"
"Create an instance"
  new
    ^ super new

"Instance methods"
"Initialize"
initialize
  intlist <- Array new: 0

"Add int to list"
  add: n | oldintlist |
    oldintlist <- intlist.
    intlist <- Array new: intlist size + 1.
    intlist replaceFrom: 1 to: oldintlist size with: oldintlist.
    ^ intlist at: intlist size put: n

"Calculate average"
  average | sum |
    sum <- 0.
    1 to: intlist size do:
      [:index | sum <- sum + intlist at: index].
    ^ sum // intlist size

"Filter greater than average"
  filtergreater: n | oldintlist i |
    oldintlist <- intlist.
    i <- 1.
    1 to: oldintlist size do:
      [:index | (oldintlist at: index) > n
        ifTrue: [oldintlist at: i put: (oldintlist at: index)]]
    intlist <- Array new: oldintlist size.
    intlist replaceFrom: 1 to: oldintlist size with: oldintlist

- Developed by XEROX PARC: first IDE with windows-based graphical user interfaces (GUIs)
- The first full implementation of an object-oriented language
- Example run:

```smalltalk
av <- Avex new
av initialize
av add: 1
1
av add: 2
2
av add: 3
3
av filtergreater: av average
av at: 1
3
```
Prolog

avex(IntList, GreaterThanAveList) :-
    sum(IntList, Sum),
    length(IntList, ListLen),
    Average is Sum / ListLen,
    filtergreater(IntList, Average, GreaterThanAveList).

% sum(+IntList, -Sum)
% recursively sums integers of IntList
sum([Int | IntList], Sum) :-
    sum(IntList, ListSum),
    Sum is Int + ListSum.
sum([], 0).

% filtergreater(+IntList, +Int, -GreaterThanIntList)
% recursively remove all integers <= Int from IntList
filtergreater([AnInt | IntList], Int, [AnInt | GreaterThanIntList]) :-
    AnInt > Int, !,
    filtergreater(IntList, Int, GreaterThanIntList).
filtergreater([AnInt | IntList], Int, GreaterThanIntList) :-
    filtergreater(IntList, Int, GreaterThanIntList).
filtergreater([], Int, []).

- The most widely used logic programming language
- Declarative: states what you want, not how to get it
- Based on formal logic
Pascal

program avex(input, output);

    type
        intlisttype = array [1..99] of integer;
    var
        intlist : intlisttype;
        listlen, counter, sum, average : integer;

begin
    sum := 0;
    (* read the length of the input list *)
    readln(listlen);
    if ((listlen > 0) and (listlen < 100)) then
    begin
        (* read the input into an array *)
        for counter := 1 to listlen do
            begin
                readln(intlist[counter]);
                sum := sum + intlist[counter]
            end;
        (* compute the average *)
        average := sum / listlen;
        (* write the input values > average *)
        for counter := 1 to listlen do
            if (intlist[counter] > average) then
                writeln(intlist[counter])
        end
    else
        writeln('Error in input list length')
    end.

- Designed by Swiss professor Niklaus Wirth
- Designed for teaching "structured programming"
- Small and simple
- Had a strong influence on subsequent high-level languages Ada, ML, Modula
Haskell

- The leading purely functional language, based on Miranda
- Includes curried functions, higher-order functions, non-strict semantics, static polymorphic typing, pattern matching, list comprehensions, modules, monadic I/O, and layout (indentation)-based syntactic grouping

\[
\begin{align*}
\text{sum} \ [x] & \quad = \quad 0 \\
\text{sum} \ (a:x) & \quad = \quad a + \text{sum} \ x
\end{align*}
\]

\[
\begin{align*}
\text{avex} \ [x] & \quad = \quad [] \\
\text{avex} \ (a:x) & \quad = \quad [n \mid n \leftarrow a:x, n > \text{sum} \ (a:x) / \text{length} \ (a:x)]
\end{align*}
\]
main()
{   int intlist[99], listlen, counter, sum, average;
   sum = 0;
   /* read the length of the list */
   scanf("%d", &listlen);
   if (listlen > 0 && listlen < 100)
   {   /* read the input into an array */
      for (counter = 0; counter < listlen; counter++)
      {   scanf("%d", &intlist[counter]);
          sum += intlist[counter];
      }
      /* compute the average */
      average = sum / listlen;
      /* write the input values > average */
      for (counter = 0; counter < listlen; counter++)
      {   if (intlist[counter] > average)
          printf("%d\n", intlist[counter]);
      }
   }
   else
   {   printf("Error in input list length\n");
   }
}

- One of the most successful programming languages
- Primarily designed for systems programming but more broadly used
- Powerful set of operators, but weak type checking and no dynamic semantic checks
main()
{
    std::vector<int> intlist;
    int listlen;
    /* read the length of the list */
    std::cin >> listlen;
    if (listlen > 0 && listlen < 100)
    {
        int sum = 0;
        /* read the input into an STL vector */
        for (int counter = 0; counter < listlen; counter++)
        {
            int value;
            std::cin >> value;
            intlist.push_back(value);
            sum += value;
        }
        /* compute the average */
        int average = sum / listlen;
        /* write the input values > average */
        for (std::vector<int>::const_iterator it = intlist.begin();
             it != intlist.end(); ++it)
        {
            if ((*it) > average)
                std::cout << (*it) << std::endl;
        } else
            std::cerr << "Error in input list length" << std::endl;
    }
    else
        std::cerr << "Error in input list length" << std::endl;
}

- The most successful of several object-oriented successors of C
- Evolved from C and Simula 67
- Large and complex, partly because it supports both procedural and object-oriented programming
Java

```
import java.io;
class Avex
{
    public static void main(String args[]) throws IOException
    {
        DataInputStream in = new DataInputStream(System.in);
        int listlen, counter, sum = 0, average;
        int [] intlist = int[100];
        // read the length of the list
        listlen = Integer.parseInt(in.readLine());
        if (listlen > 0 && listlen < 100)
        {
            // read the input into an array
            for (counter = 0; counter < listlen; counter++)
            {
                intlist[counter] =
                Integer.valueOf(in.readline()).intValue();
                sum += intlist[counter];
            }
            // compute the average
            average = sum / listlen;
            // write the input values > average
            for (counter = 0; counter < listlen; counter++)
            {
                if (intlist[counter] > average)
                {
                    System.out.println(intlist[counter] + "\n");
                }
            }
        }
        else
        {
            System.out.println("Error in input length\n");
        }
    }
}
```

- Developed by Sun Microsystems
- Based on C++, but significantly simplified
- Supports only object-oriented programming
- Safe language (e.g. no pointers but references, strongly typed, and implicit garbage collection)
- Portable and machine-independent with Java virtual machine (JVM)
Other Notable Languages

- **C#**
  - Similar to Java, but platform dependent (MS .NET)
  - Common Language Runtime (CLR) manages objects that can be shared among the different languages in .NET

- **Simula 67**
  - Based on Algol 60
  - Primarily designed for discrete-event simulation
  - Introduced concept of coroutines and the class concept for data abstraction

- **APL**
  - Intended for interactive use ("throw-away" programming)
  - Highly expressive functional language makes programs short, but hard to read

- **Scripting languages**
  - Perl, Python, Ruby, …
Why are There so Many Programming Languages?

- **Evolution**
  - Design considerations: What is a good or bad programming construct?
  - Early 70s: structured programming in which goto-based control flow was replaced by high-level constructs (e.g. while loops and case statements)
  - Late 80s: nested block structure gave way to object-oriented structures

- **Special Purposes**
  - Many languages were designed for a specific problem domain, e.g:
    - Scientific applications
    - Business applications
    - Artificial intelligence
    - Systems programming
    - Internet programming

- **Personal Preference**
  - The strength and variety of personal preference makes it unlikely that anyone will ever develop a universally accepted programming language
What Makes a Programming Language Successful?

- **Expressive Power**
  - Theoretically, all languages are equally powerful (Turing complete)
  - Language features have a huge impact on the programmer's ability to read, write, maintain, and analyze programs
  - Abstraction facilities enhance expressive power

- **Ease of Use for Novice**
  - Low learning curve and often interpreted, e.g. Basic and Logo

- **Ease of Implementation**
  - Runs on virtually everything, e.g. Basic, Pascal, and Java

- **Open Source**
  - Freely available, e.g. Java

- **Excellent Compilers and Tools**
  - Fortran has extremely good compilers
  - Supporting tools to help the programmer manage very large projects

- **Economics, Patronage, and Inertia**
  - Powerful sponsor: Cobol, PL/I, Ada
  - Some languages remain widely used long after "better" alternatives
Classification of Programming Languages
## Classification of Programming Languages

<table>
<thead>
<tr>
<th>Declarative</th>
<th>Functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit solution</td>
<td>(Lisp, Scheme, ML, Haskell)</td>
</tr>
<tr>
<td>&quot;What the computer should do&quot;</td>
<td>Logical</td>
</tr>
<tr>
<td></td>
<td>(Prolog)</td>
</tr>
<tr>
<td></td>
<td>Dataflow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Imperative</th>
<th>Procedural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit solution</td>
<td>&quot;von Neumann&quot; (Fortran, C)</td>
</tr>
<tr>
<td>&quot;How the computer should do it&quot;</td>
<td>Object-oriented</td>
</tr>
<tr>
<td></td>
<td>(Smalltalk, C++, Java)</td>
</tr>
</tbody>
</table>
Contrasting Examples

Procedural (C):
int gcd(int a, int b)
{ while (a != b)
    if (a > b) a = a-b; else b = b-a;
    return a;
}

Functional (Haskell):
gcd a b
| a == b = a
| a > b = gcd (a-b) b
| a < b = gcd a (b-a)

Logical (Prolog):
gcd(A, A, A).
gcd(A, B, G) :- A > B, N is A-B, gcd(N, B, G).
gcd(A, B, G) :- A < B, N is B-A, gcd(A, N, G).