As I stated before, our ultimate objective in this course is to provide you with the knowledge and skills necessary to create a new programming language.

Let’s say I asked you to do that right now. You might ask yourself:
How can I express the rules of my language?
Should it be object-oriented, procedural, or are there better options?
How do I create a compiler for the language? Should it even be a compiled language?
Semicolons or not?
COURSE TOPICS

- **Language concepts**
  - Classification of programming languages
  - **Common language constructs**: sequencing, loops, conditions, etc
  - **Names, Scopes, and Bindings**: How and when bindings for local names are defined in languages with scoping rules
  - **Control Flow**: How programming constructs define control flow and how the choice of constructs can affect programming style
  - **Subroutines and Parameter Passing**: How the subroutine calling mechanism is implemented and how and when parameters are passed and evaluated
  - **Exception Handling**: How to improve the robustness of programs.
COURSE TOPICS

• **Language implementations**
  • Common techniques used in compilers and interpreters.
  • **Lexical analysis**: Identifying correct words in a program.
  • **Syntax analysis**: Identifying syntactically correct program structures.
  • **Semantics analysis**: Identifying meaningful programs.

• **Alternative programming**
  • **Functional Programming**: Programming with Scheme
  • **Logic Programming**: Programming with Prolog
HISTORY OF PROGRAMMING LANGUAGES

• 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.
MACHINE LANGUAGE PROGRAMS

- Many early machines required programs to be expressed in machine language.
  
  55 89 e5 53  ➞ Could you tell me what’s going on here?
  83 ec 04 83

- Programs written for the ENIAC (1946) were written in the language of the machine itself.
  - A program for adding two numbers essentially outlined instructions for configuring the ENIAC’s many large plugboards and switchboards.

- This is tedious and difficult to create and maintain.
ASSEMBLY LANGUAGE PROGRAMS

- Assembly Language Programs were developed so that machine operations could be expressed in mnemonic abbreviations.
  - Enables larger, reusable, and relocatable programs.
  - Machine code is produced by assembler, not by programmer.

- Here is an example punched card for the IBM 709x Assembly Language.
ASSEMBLY LANGUAGE EXAMPLE

• You’ve all seen MIPS before.

• Here’s an example MIPS assembly program to compute GCD.

• Easier to understand, but it might take you quite a bit of time to tell me what’s going on here.

```assembly
addiu sp, sp, -32
sw ra, 20(sp)
jal getint
nop
jal getint
sw v0, 28(sp)
lw a0, 28(sp)
move v1, v0
beq a0, v0, D
slt at, v1, a0
A: beq at, zero, B
    nop
    b
    subu a0, a0, v1
B: subu v1, v1, a0
C: bne a0, v1, A
    slt at, v1, a0
D: jal putint
    nop
    lw ra, 20(sp)
    addiu sp, sp, 32
jr ra
move v0, zero
```
Example MIPS R4000 machine code of the assembly program:

27bdff0d afd0014 0c1002a8 00000000 0c1002a8 afd0014 8fa4001c 
00401825 10820008 0064082a 10200003 00000000 10000002 00832023 
00641823 1483fffa 0064082a 0c1002b2 00000000 8fb0014 27bd0020 
03e00008 00001025
HIGH-LEVEL PROGRAMMING LANGUAGES

• Mid 1950’s: Development of FORTRAN (FORmula TRANslator), the first higher-level programming language.
  • Mainly developed for solving numerical problems.

• Other high-level languages soon followed.
  • Algol-58, COBOL, Lisp, BASIC, C

• Important result: programming is now a machine-independent task.
  • High-level Source Code → Intermediate Representation → Machine Code
FORTRAN 77

- FORTRAN is still widely used for scientific, engineering, and numerical problems.
- FORTRAN 77 includes:
  - Subroutines, if-then-else, do-loops.
  - Types (primitive and arrays).
  - Variable names are upper case and limited to 6 chars.
  - No recursion.
  - No structs/classes, unions.
  - No dynamic allocation.
  - No case-statements and no while-loops.

```fortran
PROGRAM GCD

C variable names that start with I,J,K,L,N,M are integers
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
ELSE
   WRITE (*, *) 'GCD =', I
END
```

```c
C variable names that start with I,J,K,L,N,M are integers
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
ELSE
   WRITE (*, *) 'GCD =', I
END
```
OBJECT-ORIENTED PROGRAMMING

- Developed in the 1980’s.
- An important innovation for software development.
  - Encapsulation and inheritance.
  - Dynamic binding.
GENEALOGY OF PROGRAMMING LANGUAGES
• Until now, you’ve likely only seen C++, C, and perhaps Java. Some of you may have experience with Python, PHP, JavaScript, etc.

• There are A LOT of languages out there — some of them may be completely different than anything you’ve seen before.

• Let’s begin our whirlwind introduction to the wide world of programming languages…
• FORTRAN had a dramatic impact on computing in the early days.

• Still used for numerical computation.

```fortran
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 90, 95, HPF

- Major revisions
  - Recursion, Pointers, Records
- New control constructs
  - while-loop
- Extensive set of array operations.
  - \( A[1:N] = B[1:N] \times 1000.0 \)
- HPF (High-Performance Fortran) includes constructs for parallel computation.

```fortran
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
```
Lisp

- 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).

```scheme
(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))
  )
)
```
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.
- 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
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

• BASIC (Beginner’s All-Purpose Symbolic Instruction Code).

• Intended for interactive use (interpreted) and easy for "beginners".

• Structure of early basic dialects were similar to Fortran.
PL/I

- 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.

```pli
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;
END AVEX;
```
ADA AND ADA95

- Originally intended to be the standard language for all software commissioned by the US Department of Defense.
- 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.

```ada
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;
```
SMALLTALK-80

- Developed by XEROX.
- The first full implementation of an object-oriented language.

```smalltalk
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: intlist 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
```
PROLOG

• The most widely used logic programming language.

• Declarative: states what you want, not how to get it.

• Based on formal logic.

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.

% 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, []).
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.
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.

```
sum []    = 0
sum (a:x) = a + sum x

avex []   = []
avex (a:x) = [n | n <- a:x, n > sum (a:x) / length (a:x)]
```
C (ANSI C, K&R C)

• 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.

```c
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");
    }
    else
        printf("Error in input list length\n");
}
```
C++

- 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.

```c++
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;
}
```
• 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).

```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");
    }
  }
}
```
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.

• 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
  • 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

- Declarative: Implicit solution. What should the computer do?
  - Functional
    - Lisp, Scheme, ML, Haskell
  - Logic
    - Prolog
  - Dataflow
    - Simulink, Scala
- Imperative: Explicit solution. How should the computer do it?
  - Procedural
    - Fortran, C
  - Object-Oriented
    - Smalltalk, C++, Java

Note that these classifications aren’t entirely rigid. Languages can have multiple classifications.
CONTRASTING EXAMPLES

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

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

Logical (Prolog):
```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).
```
We will discuss compiling and interpretation.

- Compilation and interpretation.
- Virtual machines.
- Static linking and dynamic linking.
- Compiler in action (g++).
- Integrated development environments.