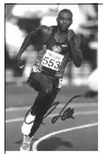
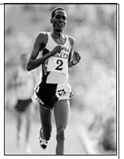
CS 330 Lecture 1

- Outline
  - Course administration
  - History of programming languages



Course administration



- Reading email is required
- Everything will be on the web page (if anything missing PLEASE email me)
- Emphasis on work not inspiration
  - Pace yourself (1-2 hr / lecture)
- Grading, copying
  - Oral presentation random sampling policy
  - Open book exam

A quote to think about

In CACM “The next 1000 yrs”, Vol 44 (3) with topics such as Digital Immortality, Virtual Beings, Cyborgs etc.

“Computing central challenge, “How not to make a mess of it” has not been met. On the contrary, most of our systems are much more complicated than can be considered healthy, and are too messy and chaotic to be used in comfort and confidence. The average customer of the computing industry has been served so poorly that he expects his system to crash all the time, and we witness a massive worlwide distribution of bug-ridden software for which we should be deeply ashamed.“
- E. Dijkstra

Programming
o
l
languages

C
C++
Java
Lisp
Perl
Python

Any others ?

The only way to learn a programming language is by writing programs in it. (B.Kernighan & D. Ritchie)

The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities. (E. Dijkstra)

Making the simple complicated is commonplace; making the complicated awsomely simple, that's creativity (Charles Mingus)

The more original a discovery the more obvious it seems afterwards (Arthur Koestler)

What is a Programming Language ?

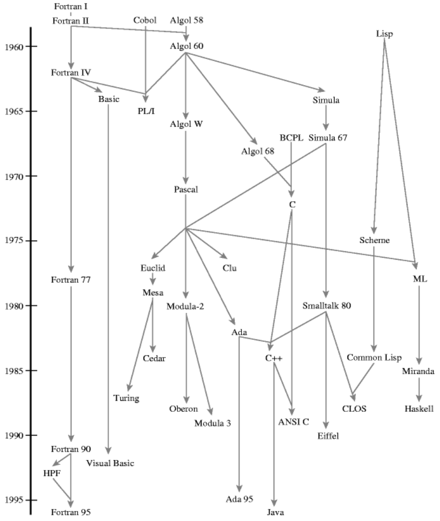
What is a Programming Language ?

A programming language is a system of notation for describing computations. A useful programming language must therefore be suited for both description(i.e., for human writers and readers of programs) and for computation (i.e., for efficient implementation on computers). But human beings and computers are so different that it is difficult to find notational devices that are well suited to the capabilities of both.
- R. Tennant (Principles of Programming Languages, Prentice Hall, 1981)

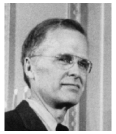
One doesn't really understand the bones of a language until one has tried to design one.
- J.R.R Tolkien when asked why spend years designing "High Elvish":

History

Fortran	50s
Cobol	50s
Algol	60s
Simula	60s
Lisp	50s



Fortran (FORmula TRANslator)



- John Backus & team in 1954
- Goals: scientific computing, efficiency
- Important concepts
  - High level programming language
  - Translator (what we call compiler)
  - Machine-independent programs
  - Floating-point numbers

We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler that could produce efficient programs. J. Backus

Fortran (95) Code example

Consistently separating words by spaces became a general custom about the tenth century A.D., and lasted until about 1957, when FORTRAN abandoned the practice — Sun FORTRAN Reference Manual .

```
PROGRAM sphere

c  This program determines the surface area and volume of a sphere,
given
c  its radius.

c  Variable declarations
   REAL  rad, area, pi

c  Definition of variables
c       rad = radius, area = surface area

c  Assign a value to the variable pi.
        pi = 3.141593

c  Input the value of the radius and echo the inputted value.
        PRINT *, 'Enter the radius of the sphere.'
        READ *, rad
        PRINT *, rad, ' is the value of the radius.'

c  Compute the surface area and volume of the sphere.
        area = 4.0 * pi * rad**2
        volume = (4.0/3.0) * pi * rad**3

c  Print the values of the radius (given in cm), the surface area
(sq cm),
c  and the volume (cubic cm).
        PRINT *,'In a sphere of radius', rad, ' , the surface area
is',
     +          area, ' and its volume is', volume, '.'
        STOP

        END
```

FORTRAN is not a flower but a weed — it is hardy, occasionally blooms, and grows in every computer." — A.J. Perlis.

Cobol

(COmmon Business Oriented Language)

- Grace Hopper 1950s designed Flowmatic which led to Cobol in 1959
- Business applications
  - Record structure
  - Separation of data structures from execution
  - Emphasis on readability but VERY wordy
  - Versatile formatting
- “I had a running compiler and nobody would touch it. They said computers could only do arithmetic”

Cobol code example

```
 $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  Michael Coughlan.
* Uses the ACCEPT and DISPLAY verbs to accept a student record
* from the user and display some of the fields.  Also shows how
* the ACCEPT may be used to get the system date and time.
* The YYYYMMDD in "ACCEPT  CurrentDate FROM DATE YYYYMMDD."
* is a format command that ensures that the date contains a
* 4 digit year.  If not used, the year supplied by the system will
* only contain two digits which may cause a problem in the year 2000.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentId       PIC 9(7).
   02  StudentName.
       03 Surname      PIC X(8).
       03 Initials     PIC XX.
   02  CourseCode      PIC X(4).
   02  Gender          PIC X.
* YYMMDD
01 CurrentDate.
   02  CurrentYear     PIC 9(4).
   02  CurrentMonth    PIC 99.
   02  CurrentDay      PIC 99.
* YYDDD
01 DayOfYear.
   02  FILLER          PIC 9(4).
   02  YearDay         PIC 9(3).
* HHMMSSss   s = S/100
01 CurrentTime.
   02  CurrentHour     PIC 99.
   02  CurrentMinute   PIC 99.
   02  FILLER          PIC 9(4).
PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter student details using template below".
    DISPLAY "Enter - ID,Surname,Initials,CourseCode,Gender"
    DISPLAY "SSSSSSSNNNNNNNNIICCCCG".
    ACCEPT  StudentDetails.
    ACCEPT  CurrentDate FROM DATE YYYYMMDD.
    ACCEPT  DayOfYear FROM DAY YYYYDDD.
    ACCEPT  CurrentTime FROM TIME.
    DISPLAY "Name is ", Initials SPACE Surname.
    DISPLAY "Date is " CurrentDay SPACE CurrentMonth SPACE CurrentYear.
    DISPLAY "Today is day " YearDay " of the year".
    DISPLAY "The time is " CurrentHour ":" CurrentMinute.
    STOP RUN.
```

The use of COBOL
cripples the mind;
it's teaching should,
therefore, be regarded
as a criminal offence
-E.Dijkstra

COBOL is for morons
-E.Dijkstra

Algol

- Designed by international committee as a universal language
- Nested structure of environments & control
- E-BNF (Backus Naur Form) syntax specification
- Never really used except for publishing algorithms
- Huge influence on future languages

Algol 68 minority report

We regard the current Report on Algorithmic Language ALGOL 68 as the fruit of an effort to apply a methodology for language definition to a newly designed programming language. We regard the effort as an experiment and professional honesty compels us to state that in our considered opinion we judge the experiment to be a failure in both respects.

The failure of the description methodology is most readily demonstrated by the sheer size of the Report in which, as stated on many occasions by the authors, "every word and every symbol matters" and by the extreme difficulty of achieving correctness. (Dijkstra, Hoare and others)

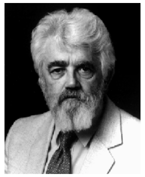
No proper program contains an indication which as an operator-applied occurrence identifies an operator-defining occurrence which as an indication-applied occurrence identifies an indication-defining occurrence different from the one identified by the given indication as an indication-applied occurrence.

- ALGOL 68 Report

Algol code example

```
// the main program (this is a comment)
begin
  integer N;
  Read Int(N);
  begin
    real array Data[1:N];
    real sum, avg;
    integer i;
    sum:=0;
    for i:=1 step 1 until N do
      begin real val;
        Read Real(val);
        Data[i]:=if val<0 then -val else
val
      end;
    for i:=1 step 1 until N do
      sum:=sum + Data[i];
    avg:=sum/N;
    Print Real(avg)
  end
end
```

Lisp
(List Processing)



- John McCarthy in 1960
- Code and data are S-Expressions (lists)
- Simple syntax – very flexible and powerful
- Garbage collection
- Lot's of silly parentheses
- No types
- :-) Lisp interpreter written in Lisp

Lisp code examples

```
(defun factorial (n)
          (cond ((= n 0) 1)
             (t  (* n (factorial (1- n))))
          )
       )
```

```
(defun average (numbers)
  (if (null numbers)
     (error "Average of the empty list is
undefined.")
     (/ (reduce #'+ numbers)
       (length numbers))))
```

```
(defun tax-bracket (income)
  "Determine what percent tax should be paid
for this income."
  (cond ((< income 10000.00) 0.00)
     ((< income 30000.00) 0.20)
     ((< income 50000.00) 0.25)
     ((< income 70000.00) 0.30)
     (t           0.35)))
```

Lisp interpreter in Lisp

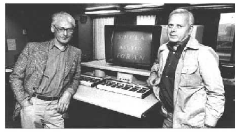
67 lines of code using only:
quote, atom, eq, cons,car, cdr, cond

```
(defun eval. (e a)
 (cond
  ((atom e) (assoc. e a))
  ((atom (car e))
   (cond
    ((eq (car e) 'quote) (cadr e))
    ((eq (car e) 'atom)  (atom   (eval. (cadr e) a)))
    ((eq (car e) 'eq)    (eq     (eval. (cadr e) a)
                                 (eval. (caddr e) a)))
    ((eq (car e) 'car)   (car    (eval. (cadr e) a)))
    ((eq (car e) 'cdr)   (cdr    (eval. (cadr e) a)))
    ((eq (car e) 'cons)  (cons   (eval. (cadr e) a)
                                 (eval. (caddr e) a)))
    ((eq (car e) 'cond)  (evcon. (cdr e) a))
    ('t (eval. (cons (assoc. (car e) a)
                     (cdr e))
               a))))
  ((eq (caar e) 'label)
   (eval. (cons (caddar e) (cdr e))
          (cons (list. (cadar e) (car e)) a)))
  ((eq (caar e) 'lambda)
   (eval. (caddar e)
          (append. (pair. (cadar e) (evlis. (cdr e) a))
                   a)))))
```

```
(defun evcon. (c a)
  (cond ((eval. (caar c) a)
         (eval. (cadar c) a))
        ('t (evcon. (cdr c) a))))

(defun evlis. (m a)
  (cond ((null. m) '())
        ('t (cons (eval.  (car m) a)
                  (evlis. (cdr m) a)))))
```

Simula



Dahl and Nygaard at the time of Simula's development

- Ole Johan Dahl & Karl Nygaard 1962-1967 in Olso, Norway
- Discrete event simulator
- First object-oriented language
- Classes, objects, inheritance, dynamic binding
- Influenced design of C++, Java etc

Simula code example

```
  class Shape(x, y); integer x; integer y;
   virtual: procedure draw;
   begin
     comment -- get the x & y components for the object --;
     integer procedure getX;
       getX := x;
     integer procedure getY;
       getY := y;
     comment -- set the x & y coordinates for the object --;
     integer procedure setX(newx); integer newx;
       x := newx;
     integer procedure setY(newy); integer newy;
       y := newy;
     comment -- move the x & y position of the object --;
     procedure moveTo(newx, newy); integer newx; integer newy;
       begin
         setX(newx);
         setY(newy);
       end moveTo;
     procedure rMoveTo(deltax, deltay); integer deltax; integer
deltay;
       begin
         setX(deltax + getX);
         setY(deltay + getY);
       end moveTo;
   end Shape;
```

Things to talk/think about

- Why study programming languages
- Persistance of established technology
- Interpreters, compilers, environments
- Compile-time, run time
- Writing, building, growing a program

Paradigms

- Imperative
  - Sequence of statements, variables
- Functional
  - Functions, expressions and bindings
- Logic
  - Symbolic logic
- “Object oriented”
  - Lot's of little computers (smalltalk vision)

Euclid's algorithm

- Greatest common divisor gcd(8,18) = ?, gcd(36, 15) = ?
- Take the remainder of dividing 36 by 15 = 6
- Take the remainder of dividing 15 by 6 = 3
- Take the remainder of dividing 6 by 3 = 0
- 3 is the gcd
- gcd(a,b) if b = 0 then a else gcd(b, a mod b)
- What about gcd(15,36) ?

Imperative GCD (Ada)

```
Procedure gcd(u, v: integer, x: out integer) is
y, t, z: integer;
begin
    z := u;
    y := v;
    loop
      exit when y =0;
      t := y;
      y:= z mod y;
      z := t;
    end loop;
    x:= z;
end gcd;
```

Object-oriented GCD

```
 public class IntWithGcd
{ private int value;
  public IntWithGcd(int val) {value = val;}
  public intValue() { return value;}
  public int gcd(int v)
  {
     int z = value;
     int y = v;
     while (y != 0)
      {
         int t = y;
         y = z % y;
         z = t;
       }
       return z;
}
```

Functional gcd (Scheme)

```
(define gcd (u v)
  (if (=v 0) u
      (gcd v (modulo u v))))
```

ogic programming gcd (Prolog)

```
gcd(U,V,U) :- V = 0.
gcd(U,V,X) :- not (V=0),
                Y is U mod V,
                gcd (V, Y, X)
```

More things to think about

- Efficiency of execution
- Efficiency of translation + complexity of compiler
- Programming efficiency
- Orthogonality, Generality, Simplicity , Uniformity

Next lecture:
Syntax & Semantics

- Form & meaning
- Check the course web page for additional readings and suggested exercises (not graded)