CIS 343 - Structure of Programming Languages

Nathan Bowman

Slides by: Ira Woodring

Architecture and History (Sebesta Text)

Before we start, let's get a few definitions out of the way.

Imperative Programming: a programming model whereby we use statements to change state.

Procedural Programming: a programming model makes use of subroutines. Is also imperative, as these subroutines rely upon changing state with statements.

Object-oriented Programming: a programming paradigm that incorporates structure and function (state and behavior) of data structures into a single entity (often called a class).

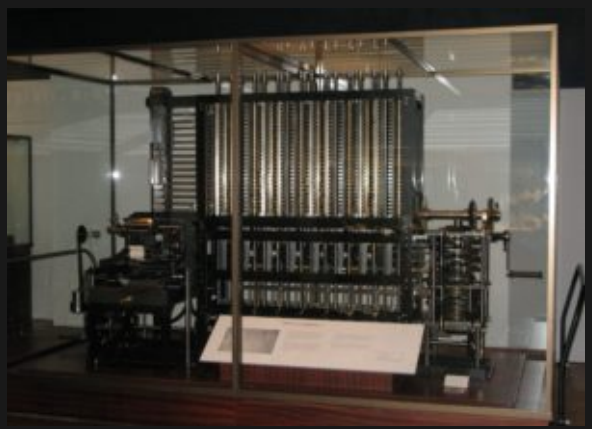
Functional Programming: a programming model that relies upon programs being built as the output of mathematical functions.

The history of computing goes back much farther than the actual times during which computers existed.

Mathematicians and others were laying the foundations for programming languages far before we had the hardware.

Charles Babbage worked on his difference engine - a mechanical device used to calculate polynomial functions.

Ada Lovelave is widely credited with writing the first computer "program", an algorithm for Babbage's machine.



Analytical Engine

https://www.youtube.com/embed/XSkGY6LchJs

Konrad Zuse's Plankalkül

Pronounced "Tsoo-zuh".

He was a German scientist working on electromechanical relays.

By 1945 most of his work had been destroyed by Allied bombing.

Konrad Zuse's Plankalkül

Built a computer called the Z4. He moved it to a Bavarian village and worked alone.

Plankalkül means "Program Calculus". He developed it for his Ph.D. dissertation. It was completed in 1945 but not published until 1972.

Konrad Zuse's Plankalkül

Simplest data type was the bit.

Other types, both integer and floating point, were built from bits.

Included arrays and record types, included nested records!

It had an iterative statement much like a `for`.

Konrad Zuse's Plankalkül

He even wrote 49 pages of algorithms to play chess, even though he was not a chess expert!

Konrad Zuse's Plankalkül

Had (what we would consider) a weird syntax.

```
  | A + 1 => A          // This is an assignment statement.
V | 4        5          // V line is for array subscripts
S | 1.n      1.n        // S line is for the data type,
                        // here an 'int' of 'n' bits.
```

This statement group would be written more like

```
a[5] = a[4] + 1;
```

today.

Pseudocodes

Not pseudocode as we think of it today.

Rather, was a group of languages from the 40s and 50s.

Pseudocodes

Short Code - Developed for the BINAC computer then used on UNIVAC I (the first commercial computer in the U.S.).

Memory had 72 bits grouped as 12 six-bit bytes.

Coded mathematical expressions.

Short Code Codes

Byte code	Operation	Byte Code	Operation	Byte Code	Operat
01	-	06	abs value	1n	(n+2)n power
02	)	07	+	2n	(n+2)n root
03	=	08	pause	4n	if <= n
04	/	09	(	58	print a tab

Variables were byte-pair Codes

```
X0 = SQRT(ABS(Y0))
```

Becomes

```
00 X0 03 20 06 Y0
```

Pseudocodes

Speedcoding

Created by John Backus (we will hear more about him later this semester).

Extended the IBM 701 machine to allow for floating-point calculations. Added square root, sine, arc tangent, exponent, and logarithm statements as well as conditional and unconditional branches.

INCREDIBLY slow by today's standards. Quicker than a human though.

An add instruction took 4.2 milliseconds.

Was interpreted; only 700 words of memory left for programs after the interpreter started running.

Pseudocodes

The UNIVAC "Compiling" System

Led by Grace Hopper at UNIVAC.

Expanded pseudocode into machine code. Made programs MUCH shorter.

** Grace Hopper**

A hero of Computer Science. Also a U.S. Navy Rear Admiral.

Was a pioneer of compilers and machine-independent programming languages. Prior to this time different languages were written for every machine.

She believed that programmers should be able to use English words that a compiler would translate into machine code.

** Grace Hopper**

Often called "Amazing Grace".

Served as a technical consultant on the creation of COBOL, a language based on her prior FLOW-MATIC language.

Has a navy missile destroy named after her, as well as a supercomputer.

Received 40 honorary degrees from universities around the world, as well as the National Medal of Technology and the Presidential Medal of Freedom.

Fortran and the IBM 704

During the 40s and early 50s there was no floating point hardware; people had to use interpreted systems to do floating point operations. These were very slow.

The IBM 704 was the first machine that included hardware for both integer and floating point operations. Fortran statements were very similar to the hardware statements of the 704.

Fortran and the IBM 704

Was **the** computer at the time.

Led to the creation of LISP, a musical language called MUSIC, inspired the voice for HAL in *2001: A Space Odyssey* when John Larry Kelly at Bell Labs demonstrated a voice synthesizer program singing "Bicycle Built for Two" to Arthur C. Clarke who was visiting a friend.

Fortran and the IBM 704

Fortran is often credited with being the first compiled high level language (though there are some questions about that). A lot of folks were working on this issue at that time.

Released first in 1956 (at least the manual for it; it had been around a bit longer).

By 1958, according to surveys, almost half of the code written for the 704 was written in Fortran.

Fortran and the IBM 704

Used no data typing statements at first; if a variable began with an `I`, `J`, `K`, `L`, `M`, or `N` it was in integer. Otherwise it was floating point.

Could not compile subroutines separately; any change in code required the entire program to be re-compiled.

All of these issues were fixed with later releases.

Fortran and the IBM 704

Continued growing new capabilities over the decades. Is still developed.

Hovers around #35 on the Tiobe Index. Similarly used languages (as of Summer 2017) are Lisp and Lua.

```
! sum.f90
! Performs summations using in a loop using EXIT statement
! Saves input information and the summation in a data file

program summation
implicit none
integer :: sum, a

print*, "This program performs summations. Enter 0 to stop."
open(unit=10, file="SumData.DAT")

sum = 0

do
 print*, "Add:"
```

Sample Fortran 90/95 code from https://en.wikibooks.org/wiki/Fortran/Fortran_example

This code can be compiled on EOS with the command

```
gfortran -o OUTPUT_FILENAME SOURCE_FILENAME(s)
```

So for instance, if you had named the file `source.f90` and want the executable to be called `sum` you would use

```
gfortran -o sum source.f90
```

LISP

Was the first functional programming language.

Function here refers to the concept of a mathematical function, not a "subroutine" or process abstraction.

Stands for **LIS**t **P**rocessor.

LISP

Written for artificial intelligence. Many people were interested in natural language processing and modeling the human brain.

Was preceded by a great deal of work at RAND, IBM, and other research houses.

LISP

Only has two data structures - atoms and lists.

Atoms are either symbols (identifiers) or numbers.

For instance:

```
(A B C D)                  // A list of 4 atoms
(A (B C) D (E (F G)))      // A list of 4 elements (some being li
```

LISP

As a functional programming language, all computation is accomplished by applying functions to arguments (which may be other functions).

Does not have assignment statements.

Makes heavy use of recursion; loops are unneeded.

LISP

Completely dominated artificial intelligence for over a quarter of a century.

Code can be compiled (now).

Today, functional languages are taking off (for good reason!).

```
;  LISP Example function
;  The following code defines a LISP predicate function
;  that  takes two lists as arguments and returns True
;  if the two lists are equal, and NIL (false) otherwise
   (DEFUN equal_lists (lis1 lis2)
    (COND
      ((ATOM lis1) (EQ lis1 lis2))
      ((ATOM lis2) NIL)
      ((equal_lists (CAR lis1) (CAR lis2))
(T NIL) )
)
```

LISP sample code from our textbook.

LISP

Today programmers use either Scheme or Common LISP (clisp).

Many other functional languages are being used today such as Clojure and Haskell.

ALGOL

One of the single most important languages ever created. Many programmers have never heard of it.

Was created to be a universal programming language for science.

Generalized many of Fortran's features, but also added new features and constructs.

ALGOL

Formalized the idea of the data type.

Syntax is still commonly used in academic literature when defining algorithms.

You may have noticed it before without knowing, if you saw an algorithm that included assignment statements as such:

```
y := 0;
```

Instead of the near ubiquitous

```
y = 0;
```

For instance:

```
for j=2 to A.length
  key = A[j]
  // Insert A[j] into the sorted
      sequence A[1..j-1].
  i = j - 1
  while i > 0 and A[i] > key
    A[i + 1] = A[i]
    i = i - 1
  A[i + 1] = key
```

Insertion sort pseudocode from the 3rd edition of *Introduction to Algorithms* by Cormen, Lieserson, Rivest, and Stein.

Or, if you've looked at ALGOL code before:

```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;
comment The absolute greatest element of the matrix a, of size
    is transferred to y, and the subscripts of this element to
begin
    integer p, q;
    y := 0; i := k := 1;
    for p := 1 step 1 until n do
        for q := 1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
```

From https://en.wikipedia.org/wiki/ALGOL_60

ALGOL

Over the years different versions of ALGOL added many interesting new features.

- it was the first imperative language to allow recursion
- it allowed passing parameters to subprograms by value or by name
- in introduced block structure (scoping rules)

ALGOL

It never achieved widespread use in the U.S. (did in Europe).

However, many concepts it introduced are still seen in programming languages today.

ALGOL

If you want to play with it, we don't have a compmiler on EOS. However, the website

https://www.tutorialspoint.com/execute_algol_online.p

Will let you explore it and many other languages.

COBOL

Common **B**usiness **O**riented **L**anguage

Came out of a meeting sponsored by the Department of Defense on designing a common language for business applications.

COBOL

According to our text COBOL has been used more than any other language, but oddly never affected the programming world the way ALGOL did.

So in some ways, they are very opposite of one another.

COBOL

Designers wanted to use English statements as much as possible, wanted it to be easy to use (even if that meant it was less powerful), and thought it should be designed in a way to remove as much implementation restrictions as possible.

COBOL

Provided the first implementation of "record" types (though Plankalkül had them).

Very good language for working with data. Process abstraction not as good.

Was the first programming language mandated for use by the Department of Defense, even though early compilers had poor performance.

```
$ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  Multiplier.
AUTHOR.  Michael Coughlan.
* Example program using ACCEPT, DISPLAY and MULTIPLY to
* get two single digit numbers from the user and multiply them

DATA DIVISION.

WORKING-STORAGE SECTION.
01  Num1                                PIC 9  VALUE ZEROS.
01  Num2                                PIC 9  VALUE ZEROS.
01  Result                              PIC 99 VALUE ZEROS.

PROCEDURE DIVISION.
```

From

http://www.csis.ul.ie/cobol/examples/Accept/Multiplier.

(A longer example is in your text).

BASIC

Beginners **A**ll **P**urpose **I**nstruction **C**ode

Created at Dartmouth. Technical students were able to use Fortran and Algol but the rest of the liberal arts student body had trouble.

They wanted the language to be usable on terminals for time-sharing.

BASIC

The language needed to

- be easy for non-tech students to use and learn
- be pleasant and friendly
- provide fast turnaround for homework
- allow free and private access
- consider user time more important than computer time

BASIC

Was the first widely used language that was used remotely.

Widely criticized for poor program structure.

Newer versions (such as VB.NET) allow full object-oriented programming.

```
REM  BASIC Example Program
REM  Input:  An integer, listlen, where listlen is less
REM          than 100, followed by listlen-integer values
REM  Output: The number of input values that are greater
REM          than the average of all input values
  DIM intlist(99)
  result = 0
  sum = 0
  INPUT listlen
  IF listlen > 0 AND listlen < 100 THEN
REM  Read input into an array and compute the sum
    FOR counter = 1 TO listlen
      INPUT intlist(counter)
      sum = sum + intlist(counter)
    NEXT counter
```

From our textbook.

PL/I

Was another IBM invention.

By the early 60s languages were written either for science or business.

IBM wanted to have a general purpose "language for everyone".

PL/I

Included best of:

- ALGOL 60 with recursion, block structure
- Fortran IV with separate compilation, global data
- COBOL 60 - data structures, I/O, and report generation, and new constructs.

PL/I

In addition,

PL/I

Was a little over ambitious. Included too many features to be useful for programmers.

Also had many poorly designed constructs (although it was the first for some which should be considered).

Was used somewhat in the 70s then sort of died. Was still in the 50-100 languages section from TIOBE index as of Fall 2017.

```
/* PL/I PROGRAM EXAMPLE
INPUT:  AN INTEGER, LISTLEN, WHERE LISTLEN IS LESS THAN
        100, FOLLOWED BY LISTLEN-INTEGER VALUES
OUTPUT: THE NUMBER OF INPUT VALUES THAT ARE GREATER THAN
        THE AVERAGE OF ALL INPUT VALUES   */
PLIEX: PROCEDURE OPTIONS (MAIN);
 DECLARE INTLIST (1:99) FIXED.
 DECLARE (LISTLEN, COUNTER, SUM, AVERAGE, RESULT) FIXED;
 SUM = 0;
 RESULT = 0;
 GET LIST (LISTLEN);
IF (LISTLEN > 0) & (LISTLEN < 100) THEN
DO;
/* READ INPUT DATA INTO AN ARRAY AND COMPUTE THE SUM */
    DO COUNTER = 1 TO LISTLEN;
```

Early Dynamic Languages

These languages weren't that popular, but they provided dynamic typing and storage allocation. Meaning?

- Variables were given a type when assigned a value
- Storage was only allocated when a value was assigned

Early Dynamic Languages

APL also from IBM. Not originally made to be a programming language; instead was meant to describe computer architecture.

Had a large number of symbols that required an odd character set. Made some things easy; a single operator could do a matrix transpose. But was hard to read.

https://en.wikipedia.org/wiki/APL_syntax_and_symbols

Early Dynamic Languages

Still used, though not widely. Oddly enough it hasn't changed much in its 50 years.

Can try it here!

http://tryapl.org/

Early Dynamic Languages

SNOBOL was a Bell Labs invention. Made for text processing. Still around though not widely used.

Stri**N**g **O**riented and sym**BO**lic **L**anguage

```
*     WORDSIZE.SNO
*
*     Program to read a file and display the number of words of
*     various word lengths.  To make the program more interesti
*     we shall only consider word lengths between 3 and 9.  Thi
*     us to demonstrate the use of an array with subscripts off
*     1, as well as array failure.
*
*     The file being scanned is read from standard input.  For
*     to scan the file TEXT.IN, type:
*
*         SNOBOL4 WORDSIZE <text.in *="" trim="" trailing="" bl

*     Upon end of file, print the values in the array.  Print h
*
```

From

http://groups.engin.umd.umich.edu/CIS/course.des/cis40

SIMULA 67

Written in Norway at the Norwegian Computing Center between '62 and '64. Was written for system simulation but quickly grew to be general purpose.

Was the first language to provide data abstraction. Used the class construct. Provided the foundation of OO programming.

ALGOL 68 and ALGOL Descendants

Many other languages based upon these early ones began to pop up, with different design focuses.

ALGOL 68 stressed orthogonality.

```
PROC gcd = (INT a, b) INT: (
  IF a = 0 THEN
    b
  ELIF b = 0 THEN
    a
  ELIF a > b  THEN
    gcd(b, a MOD b)
  ELSE
    gcd(a, b MOD a)
  FI
);
test:(
  INT a = 33, b = 77;
  printf(($x"The gcd of"g" and "g" is "gl$,a,b,gcd(a,b)));
  INT c = 49865, d = 69811;
```

Pascal, designed for teaching programming, stressed simplicity and expressivity.

```
function gcd_iterative(u, v: longint): longint;
  var
    t: longint;
  begin
    while v <> 0 do
    begin
      t := u;
      u := v;
      v := t mod v;
    end;
    gcd_iterative := abs(u);
  end;
```

C stressed flexibility and became the *lingua franca* of Computer Science.

```
int
gcd_iter(int u, int v) {
  if (u < 0) u = -u;
  if (v < 0) v = -v;
  if (v) while ((u %= v) && (v %= u));
  return (u + v);
}
```

Other languages began to come along that stressed completely different paradigms from the imperative model.

Prolog

In the early '70s Prolog was created to allow programmers to program via formal logic.

Is nonprocedural; programs don't provide steps for completing a task. Instead, they provide the form and characteristics of the desired result.

Prolog

For instance, the programmer provides facts or rules:

```
mother(joanne, jake).
father(vern, joanne).
```

Prolog

And rules:

```
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

Prolog

Then the system may be queried with a goal statement:

```
father(bob, darcie).
```

This statement asks the system to determine if `bob` is the father of `darcie`. The system will respond with `true` if it can prove this, or `false` otherwise.

Prolog

At one time, some computer scientists thought logic programming would overtake other forms of programming as the dominant paradigm. This never happened, likely because

- logic programs are often highly inefficient compared to other programs.
- we have determined it is not an effective paradigm for certain domains (primarily A.I. and databases are the best for it).

Ada

Developed for the Department of Defense.

They had a great number of embedded systems, and the costs of programming them was high. At that point they were using more than 450 languages, as every contractor was using their own language.

Ada

Our book's author calls it "History's Largest Design Effort", because of all the work that went into its creation.

The "High-Order Language Working Group" was created first. They identified the needs and made recommendations.

Many groups submitted proposals then for languages.

Ada

Four groups were ultimately chosen. All four of the proposals were based on Pascal.

Name was given in 1979 by Jack Cooper. Named after Ada Lovelace, who is thought of as the first programmer.

Design was published in ACM SIGPLAN Notices and distributed to 10,000 people around the globe. Over 500 language reports and suggestions were suggested.

Ada

Major contributions included:

- Packages. Provided means for encapsulating data objects, specifications for data and types, and procedures. Allowed for data abstraction.
- Extensive exception handling.
- Generic program units. So that you, for instance, could write a sort routine once and use it on many types of data.
- Concurrent execution of tasks using a rendezvous mechanism (a method of intertask communication and synchronization).

Ada

The development of the compiler turned out to be somewhat difficult. Compilers didn't come out for another four years (around 1985!).

Was sometimes thought to be too large and complex.

Still used today by the DoD.

```
-- Ada Example Program
         -- Input:  An integer, List_Len, where List_Len is les
-- than 100, followed by List_Len-integer values -- Output: Th
with Ada.Text_IO, Ada.Integer.Text_IO;
use Ada.Text_IO, Ada.Integer.Text_IO;
procedure Ada_Ex is
type Int_List_Type is array (1..99) of Integer; Int_List : Int
List_Len, Sum, Average, Result : Integer;
begin
Result:= 0;
Sum := 0;
Get (List_Len);
if (List_Len > 0) and (List_Len < 100) then
-- Read input data into an array and compute the sum for Count
      Get (Int_List(Counter));
```

From our textbook.

Smalltalk

First programming language to fully support OO.

Originated in a Ph.D. dissertation of a guy named Alan Kay in 1969.

He had the crazy idea that non-tech folks would one day program computers. Additionally, he thought that we would eventually have graphical interfaces, and computers that could compute millions of instructions per second on megabytes of memory!

Smalltalk

He ended up at Xerox at Palo Alto (many important things can from here!).

Created first Smalltalk in 1972.

All computing in Smalltalk was done by sending a message to an object to invoke a method. Object sends a return reply with either data or an acknowledgment.

Smalltalk

Used classes just like SIMULA 67.

Was essential in GUI and OO programming. Almost all ways we use today to design interfaces grew out of Smalltalk.

```
"Smalltalk Example Program"
"The following is a class definition, instantiations of which
class name
superclass
instance variable names
numSides
sideLength
"Class methods"
  "Create an instance"
  new
^ super new getPen
  "Get a pen for drawing polygons"
  getPen
ourPen <- Pen new defaultNib: 2
  "Instance methods"
```

C++

Bjarne Stroustrup at Bell Labs wanted to modify C to make it more like SIMULA and Smalltalk. He wanted things like

- derived classes
- public/private access control on inherited components
- constructors and destructors
- friend classes

C++

Language began to grow rapidly adding many other capabilities over the years.

Supports both procedural and OO programming.

Allows operator overloading, templates, multiple inheritance, exception handling, and many other advanced features.

C++

Remains remarkably widespread in its use.

Is large and unruly at times though. Hard to learn, and inherited many of C's problems.

Objective-C

Designed in the early 80s. Was C plus the classes and message passing of Smalltalk.

Was popularized by Steve Jobs when he left Apple and founded NeXT. Brought it back to Apple when he returned and used it to write OS X. Was the primary language of iPhone until Swift.

Delphi

Another hybrid language approach; brought OO facilities to Pascal.

Because C was a somewhat unsafe language, the languages based on it were as well.

Pascal was a safer, more elegant language, and Delphi sought to build on those benefits.

Ended up being less complex than C++. Did provide nice set of GUI interfaces though.

Java

Sun Microsystems wanted a language for embedded consumer devices such as toaster, microwaves, tvs, etc.

Primary goal was reliability. Sending out a million microwaves with bad programming could be disastrous, they wanted to help fix that.

Somehow though, it began to be used for the Web.

In retrospect this this can be summarized best with a quote from *The Hitchhikers Guide to the Galaxy:*

"The story so far: In the beginning the Universe was created. This has made a lot of people very angry and been widely regarded as a bad move."

- Douglas Adams, The Restaurant at the End of the Universe

Java

Based on C++, but smaller and more reliable.

Has class and primitive types.

No pointers, but does have references.

Has `boolean`s.

Java

Only supports OO though. You can't write a stand-alone subprograms.

Did not allow for multiple inheritance (somewhat can now).

First interpreter was 10 times slower than compiled C code, but now is very close to the same.

As of this writing it is the world's most used language (probably greatly due to the proliferation of Android devices, but certainly not entirely).

```
// Java Example Program
// Input: An integer, listlen, where listlen is less
// than 100, followed by length-integer values
// Output: The number of input data that are greater than // t
import java.io.*;
class IntSort {
public static void main(String args[]) throws IOException {
DataInputStream in = new DataInputStream(System.in); int listl
      counter,
      sum = 0,
      average,
      result = 0;
int[] intlist = new int[99];
listlen = Integer.parseInt(in.readLine()); if ((listlen > 0) &
/* Read input into an array and compute the sum */ for (counte
```

Perl

Written by Larry Wall at NASA. Wanted a language to help write reports.

Often called a scripting language, but it is compiled so isn't really.

Variables are statically typed and implicitly declared. All scalar variables start with a $, all arrays with an @, and hashes with %.

Includes many implicit variables (often used for parameters).

Perl

Is still widely used as a UNIX system admin tool.

For awhile was a primary language for web CGI (Common Gateway Interface). Fortunately those days have largely passed.

```
# Perl Example Program
# Input:  An integer, $listlen, where $listlen is less
#
#
#
($sum, $result) = (0, 0);
$listlen = <stdin>;
than 100, followed by $listlen-integer values. Output: The num
the average of all input values.
if (($listlen > 0) && ($listlen < 100)) {
  # Read input into an array and compute the sum
  for ($counter = 0; $counter < $listlen; $counter++) { $intli
} #- end of for (counter ...
# Compute the average
$average = $sum / $listlen;
```

From Sebesta book.

Javascript

Created at Netscape.

The web needed dynamic content. Server-side dynamic content could be done with CGI. Javascript addressed the client-side.

Has nothing to do with Java.

Javascript

Most common used in browsers, but that is changing. Javascript interpreter nows exist outside of the browser, even on the command-line, on mobile devices, and on server-side applications.

This means that web/mobile applications can be programed completely in the same language.

```
// example.js
//   Input: An integer, listLen, where listLen is less
//          than 100, followed by listLen-numeric values
//  Output: The number of input values that are greater
//          than the average of all input values
var intList = new Array(99);
var listLen, counter, sum = 0, result = 0;
listLen = prompt (
        "Please type the length of the input list", "");
if ((listLen > 0) && (listLen < 100)) {
// Get the input and compute its sum
for (counter = 0; counter < listLen; counter++) {
      intList[counter] = prompt (
                        "Please type the next number", "");
      sum += parseInt(intList[counter]);
```

Other scripting languages

I will leave it to you to explore PHP, Python, Ruby, and Lua from the book.

They are all important (some even being in the top 10 of the TIOBE index), and are explored in Chapter 2.

Also in Chapter 2 are overviews of C# and meta-languages.

Architecture

Remember, computers today operate much like Turing Machines.

We solve problems by the changing of state in memory.

Even when we are using a programming language that is of a paradigm different than the imperative model, that is still happening. There is just some abstraction going on.

Turing machines:
https://www.youtube.com/embed/dNRDvLACg5Q

Physical Turing machine:

https://www.youtube.com/embed/E3keLeMwfHY

Architecture

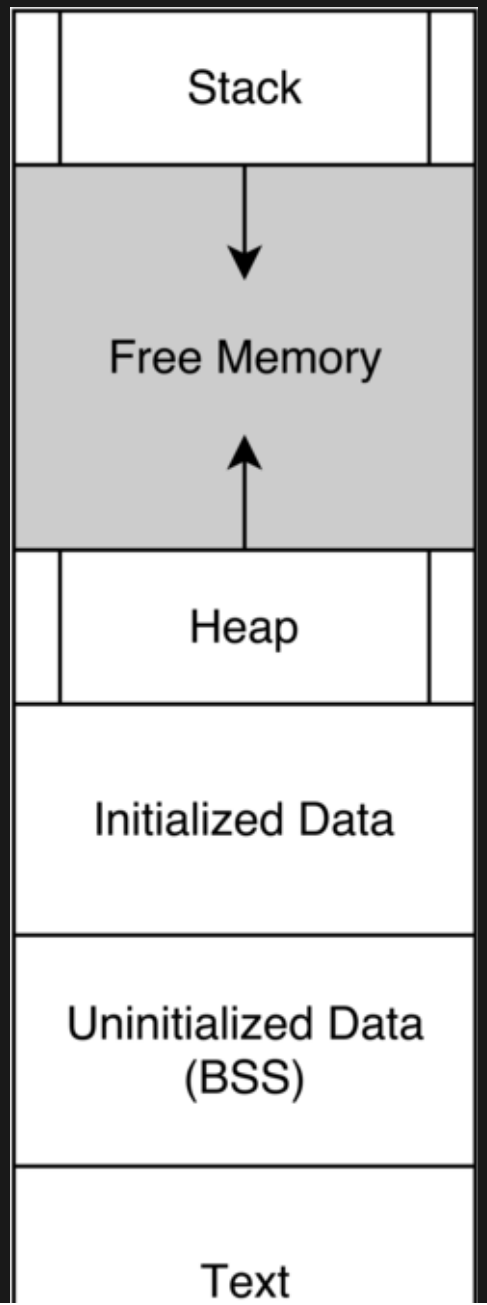
Additionally, most all computers we deal with are based on the Princeton (also called the Von Neumann) model of computation.

The Harvard model was the competing model - it never really took off wide-scale. This is often attributed to the fact that writing algorithms is typically considered to be simpler for the Princeton model.

Architecture

So we are using the Princeton model of computation, implying that memory holds data and instructions.

Let's examine our memory:


Stack
Free Memory
Heap
Initialized Data
Uninitialized Data
(BSS)
Text

Architecture

This becomes immensely important when we are programming in a language that allows us to manage memory (at least to manage it to some degree).

Consider C. Whenever we compile our code, the compiler creates static variables for us on the stack. If we need any kind of dynamically created variables (for example a storage area that could grow or shrink) then we must manage the memory ourselves.