History of Programming Languages.

Von Neumann provided us two important concepts :

a) Shared-program technique

b) Conditional Control transfer

Shared-program technique :

✓ Actual computer hardware should be simple and not need to be rewired everytime to perform the specified task

Instead,

✓ Complex program/instructions should be used to control the simple hardware.

↳ This allows faster reprogramming.

Conditional Control Transfer :

✓ This provides the notion of subroutine, or blocks of code that could be jumped to in any order.

(jumped to → instead of a single set of chronologically ordered steps)

✓ Also suggests that code should be able to branch based on logical statements.

- → if, else
- → For loop

For instance,

If (expression)
THEN.

Loop such as with FOR

In general, conditional control transfer provides the idea of libraries

↳ block code that can be reused again and again.

→ Generally, pre-compiled, where actual functionality is implemented.

In C

scanf, printf } stdio.h → headerfile

sqrt, pow } Math.h

→ But, actual implementation of these functions are done in .lib file
Say, MATH.lib

So, library —

- a set of code that can be reused over and over
- Pre-compiled
- Available in standard form to be used in other code.

Programming Languages ...

Short code :

- Appeared at around 1949
- First computer language to instruct electronic devices.
- Programmer needed to change the statements into 1's and 0's

→ First step towards the complex language of today.

In 1951, American scientist — Grace Hopper wrote the first compiler A (A-0)

First version

Assembly Language :

- Mnemonic symbols are used for instructions codes and memory locations
- A program called an assembler translates symbolic assembly language code to binary machine code.

Mnemonics & symbols :

Generally, mnemonics are something similar to abbreviation that helps to remember something.

For instance,

MOV stands for moving data between registers & memory

LD → Load data to given location.

ADD Add the numbers

Only one .ORIG per program

Where to start placing things in memory.

```
.ORIG x3000 ; Address of the first instruction
LD  R1, FIRST   (→ memory location)
LD  R2, SECOND
ADD R3, R2, R1 ; Add the number in R2 and R1, and
               place the sum in R3
ST  R3, SUM ; Copy the number in R3 to memory
            location SUM.
```

... continues

```
HALT ; Halt the program
FIRST  .FILL #5
SECOND .FILL #6
SUM    .BLKW #
```

↳ Declare a group of characters in memory.

.END Tells where the program source ends.

FORTRAN :

- FORmula TRANslating System, generally popular among the mathematicians.

Shortcoming of Assembly languages :

- Lacks in abstraction capability of mathematical notation

↳ a notation or way of expressing ideas that makes them concise, simple, and easy for human mind to grasp.

```
int first=5
int second=6
int sum= first + second
```

sample C program to add two numbers

- The other shortcomings of assembly language is —
  - it is often hardware specific; particular computer hardware architecture has its own machine language set.
    - Hence, customized dialect of assembly language is necessary.

FORTRAN (FORmula TRANSlating System)

- Designed and Developped by John Backus from IBM.
- Intended for Mathematicians and scientists
- Early versions were close to Assembly language
  - Later versions have undergone numerous revisions.
- It is still in use, and the versions are being upadated.
  - in physical systems
  - Astrophysics

FORTRAN ⟶ I ⟶ ... IV ⟶ 66 ⟶ 77 ⟶ 90

LISP (List Processing Language)

- Second-oldest high-level programming, widely used in artificial intelligence (AI).
- Designed by John McCarthy in 1958
- Basic and only data type was list; other data types were added later.
- LISP programs are written a set of lists
  - → easy to modify and hence, grow on its own.
  - → Formatting syntax is different than standard boolean logic.

OR (x, y) parenthesized prefixed used in LISP

x OR y standard Boolean logic

Data structure:

Pure lisp has two kind of data structure — i) Atoms ii) lists.

Atoms :

- Either symbols
  - ↦ identifiers
  - ↳ Numerical symbols/literals

(A B C D) — A → atom

Simple list, in which elements are restricted to atoms.

Lists :

- Specified by delimiting their elements parentheses.
- Nested list structures are also specified by parentheses.

(A (B C) D (E (F G))) — elements numbered 1 (A), 2 ((B C)), 3 (D), 4 ((E (F G))) (expression of the nested list)

Here,

1. atom A
2. sublist (B, C)
3. atom D
4. Sublist (E (F G)) → second element is another sublist.

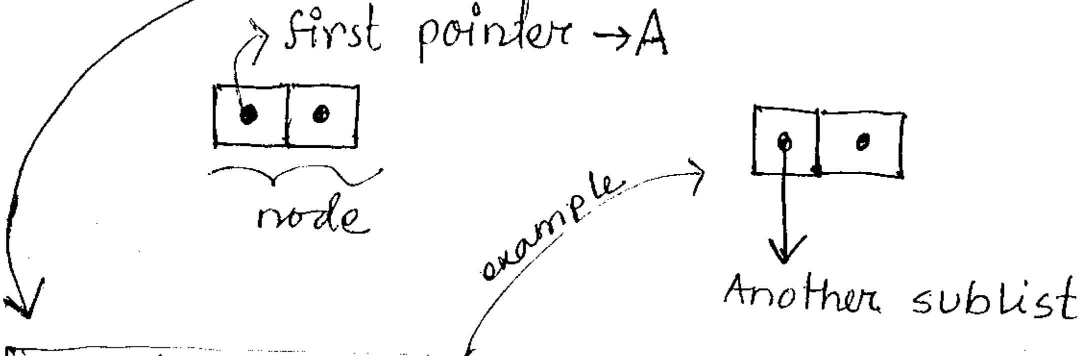
Lists are stored as single-linked list structures → has nodes

- ↦ each node has two pointers
- ↳ represents a list node.

✓ Node containing an atom has its first pointer pointing towards some representation, of the atom.
- Symbol
- Numeric value

OR

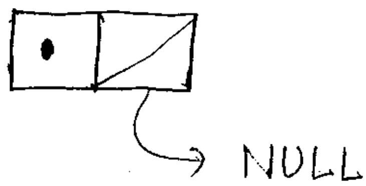
the first pointer may point toward a sublist



A node for sublist/element has its first pointer pointing towards the first node of the sublist

✓ In both cases, the second pointer of a node points to the next element in the list.

✓ Last element of a list doesn't have any successor — so, its link is Null



▣ Example : Internal representation of two Lisp lists.

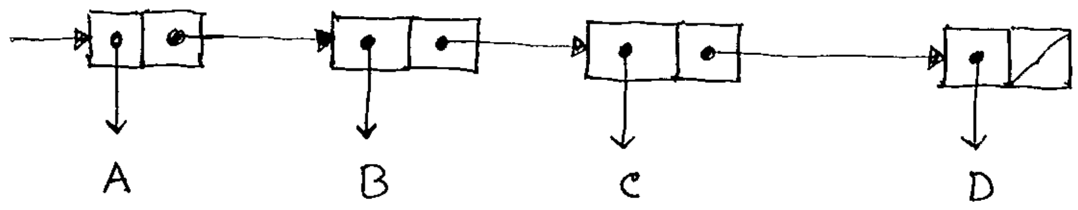


Fig. Representing List (A B C D)

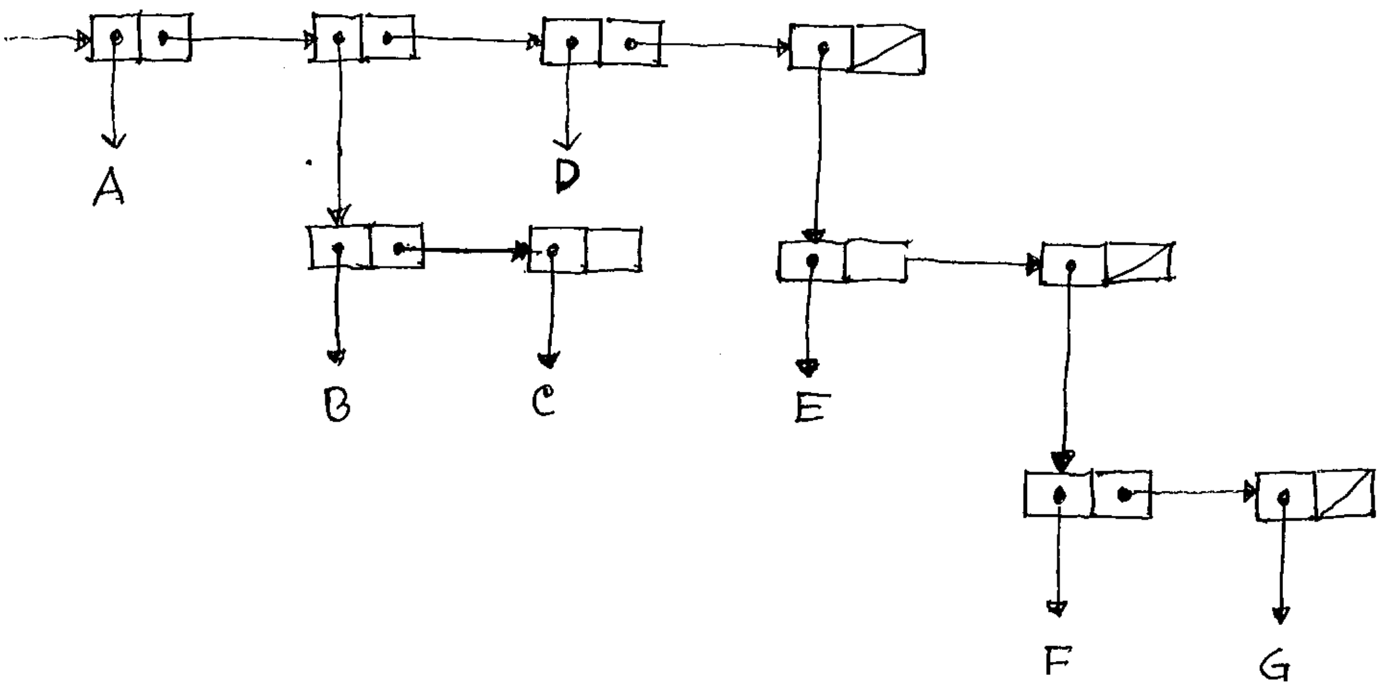


Fig. Representing list

(A (B C) D (E (F G)))

More on LISP

- It is purely a functional programming
  - computations are performed by applying functions to arguments.
  - Assignment statements and variables are not necessary in functional language program.
- Still the dominant language for AI
- Common LISP and Scheme are two main version.
- Symbolic Programming
  - Symbols : +, −, 1, 2, 3, 4 etc.
  - Symbolic Expression : (+ 1 2), (+ (* 3 4) 2)
  - Generally, it manipulate symbolic expressions.

Descendants of LISP

Common LISP and Scheme are the two most widely used dialects.

SCHEME :

- Emerged from MIT in the mid 70s
- Small size
- Extensive use of static scoping

Also known as lexical scoping. Scoping provides the range of functionality of a variable

Because of this functionality range, the variable can be referenced from within the small block of code where the variable is defined.

Scope is determined during compilation.

Example:

```
int x= 6
main()
{1 x=3;
{2 int x;
{ x = 2; }
printf ("%d\n", x) }2
printf ("%d\n", x) }1
return 0;
```

Output:

2
3

Static scoping

- Definition of a variable is set or resolved by
  - Looking at its containing block
  - or, by looking at its function

  ↓ if fails

  variable definition is resolved by searching the outer containing block

  ↓

  and so on.

Example:

```
int a= 10, b= 20;
int main ()
{
    int a = 5;
    {
        int c;
        c = b/a;
        printf ("%d", c);
    }
}
```

→ Containing block for variables a, b.

ALGOL 58, came in 1958

- It was, in some sense, a decendant of Fortran
- Generalized many Fortran features and added several new constructs and concepts.
- Concept of data types was formalized
- Added the concept of compound statement (begin ... end)
  → Other subsequent programming languages inherited this.
- Identifiers could be of <u>any length</u>.
  → Fortran I's restricted this to 6 or fewer characters
- <u>Any dimensional array</u> was allowed.
  → Fortran I's limitation was ≤ 3 dimensions.
- <u>Nested selection</u> of statements were possible
  → If had an else-if clause was not in Fortran.

SCOPING

- Scoping is necessary to allow reuse of variables.
- An acceptable norm (and widely practiced as well) is to use shorter variable names. But in program that has millions of lines of code, shorter variable names may fall short.
  - So, we must reuse variable names and scoping just allows it.

For instance,

```
int fnc 1 ( )
  { int a = 5; }

int fnc 2 ( )
  { int a = 10; }
```

Here, variable "a" is reused in fnc 2. → because of scoping

scope of "a" within fnc 1

"a" is redefined here. → Scope of "a" is within fnc

Scoping Types

1. Static scoping
   C, C++, Java etc.

2. Dynamic scoping
   Perl (both static and dynamic)

Dynamic scoping

Let's consider a pseudocode :

```
{
  printf ("%d", x)
              ↳ for integer
}
```

```
{
  printf (x)
}
```

Free variable
- Value not known
- data type not defined.

We can use scoping concept to obtain the necessary information needed.

Assume,

```
Test ()
{ print (x)
}
```

```
Sample ()
{ x = 20;
  Test ()
}
```

value of x is fixed at point of invocation

Point of invocation.

So, dynami scoping obtains $x = 20$ and print accordingly

Whereas, for the above case, static scoping takes values from main ()

suppose, in main (), we have
main () { x= 10; }
it will print x = 10;

* Main fn^c is often known as ancestor block

Example question on static and dynamic scoping :

Pseudocode :

```
x : real number value

Sample1 ( )
{ print (x); }

Sample2
{ x : real number value
  x = 5.5 ;
  Sample1 ( )
}

Main ( )
{ x = 0.25
  Sample 1 ( )
  Sample 2 ( )
}
```

value of x from main() will be printed here.

point of invocation is sample2

execution starts from here

(1) Control goes to sample 1

(2) Control goes here

(3)

Static		Dynamic	
	0.25		0.25
	0.25		5.5

☐ Example question : Hypothetical language.

```
int i
program main ()
{ i= 10;
  call fn^c ();
}
procedure fn^c ()
{
int i =20;   point of invocation
call g();
}
procedure g()
{
  print (i);
}
```

Static : x = 10 (↑ global value)

Dynamic : y = 20

ALGORITHMIC LANGUAGE ALGOL 60

- ALGOL60 was the outcome of a combined quest for the search of an universal programming language.

  ↓ In fact, it is the

  First step toward sophistication

Motivation:

- FORTRAN arrived, but it was for IBM
- Other languages were developped, but they were for specific machines.
- No portable language.
  - ↳ All languages developped till then were machine specific.
- Absence of a universal programming language for communicating algorithms.

Goals of the language:

- Close to mathematical notation (syntax of the language)
- Should be good for algorithms
- It should be translatable to machine code.