CSE 425/ L-01/ 10.07.2024/

* Class Performance ⇒ 5%.

  Project ⇒ 5%.

  HW / Assignment ⇒ 10%.

  Programming ⇒ 5%.

  Quiz ⇒ 2 out of 2 ⇒ 20%. [35 minute conceptual Question]

  Exam-1 (Mid) ⇒ 25%.

  Exam-2 (Final) ⇒ 30%. [Comprehensive Syllabus]

* Ebook ⇒ Sebesta
* Q-1 ⇒ L8-L9

  Mid ⇒ L-16

  Q-2 ⇒ L22-23

* Number Theory - Research Area

* Why the first programming language is formed as sequential?

⇒ Computer was task specific and we need to modify hardware to build new algorithm. So, its depends on hardware. And there is no other way.

* Programming language need to be user friendly. Because 90% of ~~the~~ time spend on debugging rather than coding.

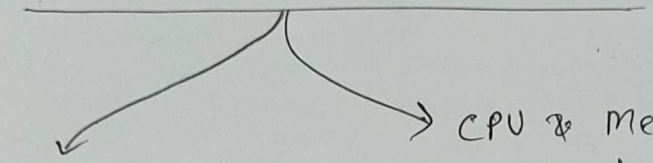
Programming Languages:

- Sequential C, R
- Functional Lisp
- Logical Prolog

- OOP Java, C++, R
- Non-OOP C, R
- Structural Fortran, R

} get some comparative idea

Von-Neumann Architecture



- CPU
- Memory
- I/O Devices

CPU & Memory are physically separated, and connected by BUS

- Address BUS
- Control BUS
- Data BUS

CPU: Central Processing Unit

⇒ Control Unit: Control order of the instruction to be executed. Also, controls the retrieval of desired operands.

⇒ ALU: Arithmetic Logic Unit

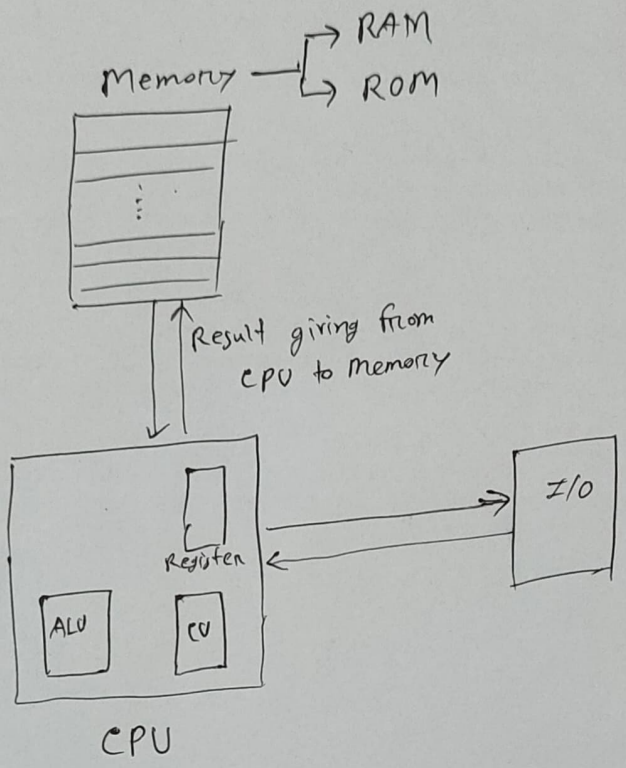
- Perform all the logical and mathematical operations.

⇒ Register: Temporary storage to store and transfer data upon fetching from main memory.

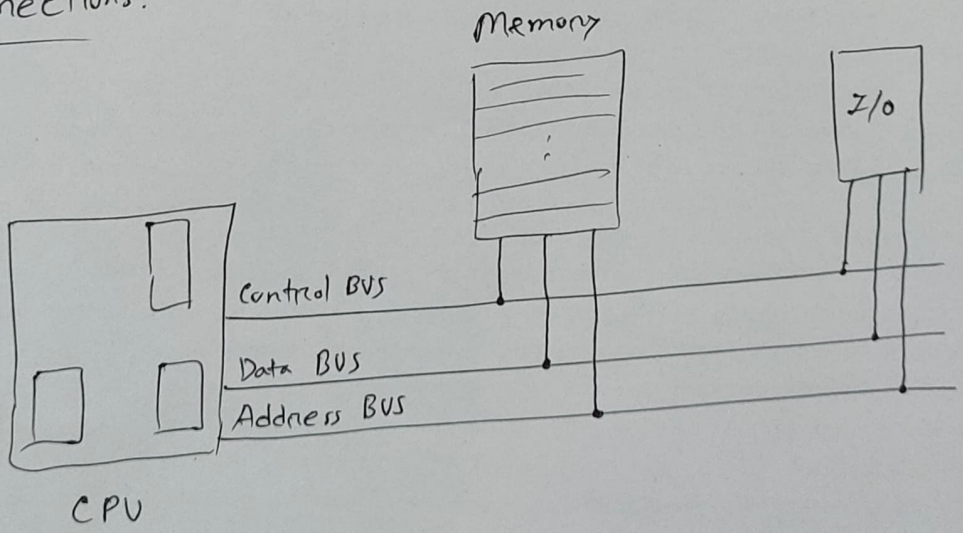
L-02 / 15.07.2024 /

Von Neumann Architecture:

- CPU and memory are physically separated.



BUS Connections:



- Von neumann bottleneck
  - Fetching, sometimes CPU stay on idle.

- CXL connection in data center

- At first programmer need to do code using instruction on short code. Then programming languages arrived.
  - Using 1 & 0
  - Low level

- Machine Code:
  - Sequence of 1's and 0's

Very compler ←

xxxx
4 bits for op. code

xxxx xxxx xxxx
12 bits for operand

⇒ To simplify the process, we got compiler in place
↳ 1951 (Grace Hopper)
Female scientist from US Navy

- Memory:
  - RAM ⇒ Volatile
  - ROM ⇒ Non-Volatile

Compilation:

⇒ Three ways with which a programming language can be implemented.

1. Compiler:
   - Program translated to machine code.
   - C/C++

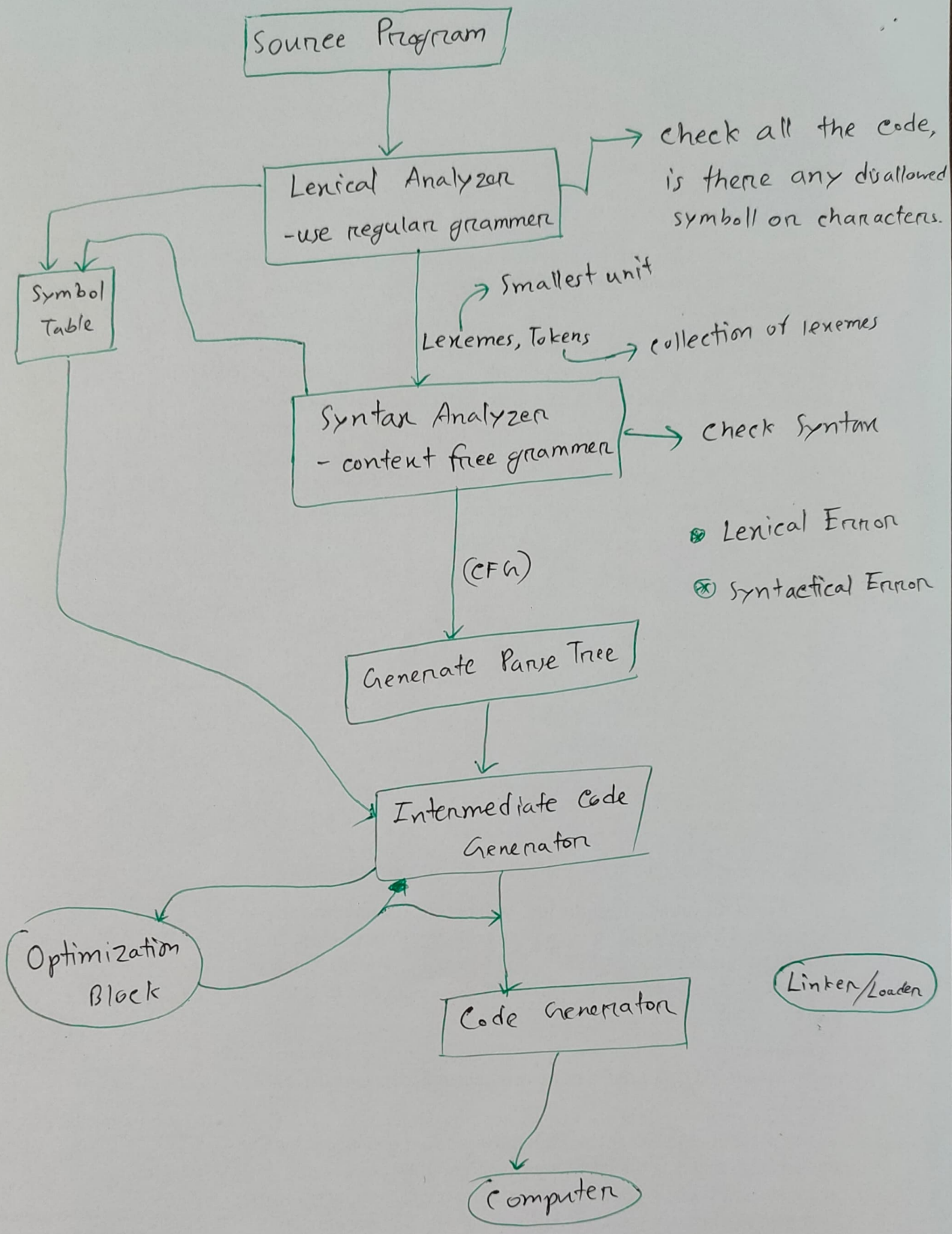
2. Interpreter: (by a software)
   - Program is interpreted not translated.
   - Python / Java Script

3. Hybrid Approach:
   - compromised approach, 1 * 2 combined.
   - Java

Compiler based implementation:

⇒ Translator, that translate high-level instruction directly to the machine code.

⇒ Done in two step:
   - generates an intermediate code, which structure is very similar to assembly language code.
   - a assembler is in place to do the final translation to machine code.



L-03/02.09.2024/

Restart

No content

L-04/04.02.2024/

- Language construction (Implementation): LLVM Compiler
  - Compiler based
  - Interpreter based
  - Hybrid
- We need to optimize our code in the intermediate code generator phase, because its harder to optimize the code in machine language formate.
- Intermediate Code Generation:

⇒ TAC - Three Address Code → operator on register

- Maximum 3, not more than 3 operands
- use output from syntax analyzer ⇒ AST
  - So, AST is the input of Intermediate code.

⇒ Very close to assembly language code.

$t_1 = t_1 + t_2$

operator → $+$

operand → t_2

destination → t_1

✳ a = b * c + b * d ;

⇒

$$\left.\begin{aligned} -t_1 &= b * c ; \\ -t_2 &= b * d ; \end{aligned}\right\} \text{ b used twich.}$$

$$-t_3 = -t_1 + -t_2 ;$$

$$a = -t_3 ;$$

✳ x = (-b + sqrt(b^2 - 4 * a * c)) / (2 * a) ;

⇒ TAC

$t_1 = b * b ;$

$t_2 = 4 * a ;$

$t_3 = t_2 * c ;$

$t_4 = t_1 - t_3 ;$

$t_5 = sqrt(t_4) ;$

$t_6 = 0 - b ;$

$t_7 = t_5 + t_6 ;$

$t_8 = 2 * a ;$

$t_9 = t_7 / t_8 ;$

$x = t_9 ;$

Flow control TAC is important for exam

✳ Variable assignment in TAC

variable = constant

var1 = var2

var1 = var2 OP var3

var1 = constant OP var2

✳ Permitted operator:

+, -, *, /, %.

✳ Boolean & others:

<, ==, ||, &&

y = -x ;

y = 0 - x ;

| y = -1 * x ;

* b = (x <= y);

⇒ TAC:

$t_0 = x < y$;

$t_1 = x = y$;

$t_2 = t_0 \;||\; t_1$;

* Flow Control:

⇒ two way to control the flow

1. GOTO "Label"
2. IFZ value GOTO "Label"

*

```
int x;
int y;
while (x < y){
    x = x * 2;
}
y = x;
```

⇒ TAC:

```
L0:
    t0 = x < y;
    IFZ t0 GOTO L1;
    x = x * 2;
    GOTO L0;
L1:
    y = x;
```

L-05 / 09.09.2024 /

- TAC - Three Address Code
  - Three operand at most
  - Flow control
    - GOTO Label
    - IF2 value GOTO Label

- Practice with array for exam

```
int x;
int y;
while (x<y){
    x=x*2;
}
y=x;
```

TAC

```
L0:
    t0 = x < y ;
    IF2 t0 GOTO L1 ;
    x = x * 2 ;
    GOTO L0 ;
L1:
    y = x ;
```

CS143: Compiler

```
void main(){
    int x,y;
    int i = x*x + y*y;
    while (i>5){
        i = i-x;
    }
}
```

TAC

```
main:
    BEGIN Function "Label";
    t0 = x * x;
    t1 = y * y;
    i = t0 + t1;
L0:
    t2 = 5 < i;
    IF2 i GOTO L1;
    i = i - x;
    GOTO L0;
L1:
    END FUNCTION;
```

- Practice more with for loop & array, nested if-else.

TAC is very high level assembly language.

TAC :

```
C = A + B ;

ADD A, B, C ;
ADD C, A, E ;
```

Assembly :

```
LD   A, R1 ;
LD   B, R2 ;
ADD  R1, R2, R3 ;
ST   R3, C ;
LD   C, R4 ;
LD   A, R5 ;
ADD  R4, R5, R6 ;
ST   R6, E ;
```

We need to optimize these redundant use.

Optimization Process :

- Address mode selection
  - helps to improve instruction
- Peephole optimization
  - Remove redundant expression
- Common sub-expression elimination
  - remove redundant computation
- Register allocation
  - reduce number of register allocation

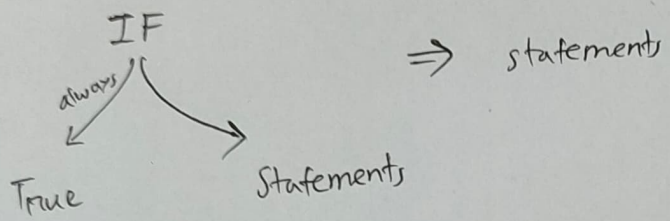
Graph theory, graph coloring for register allocation.

⇒ available in Kenneth Rosen's book.

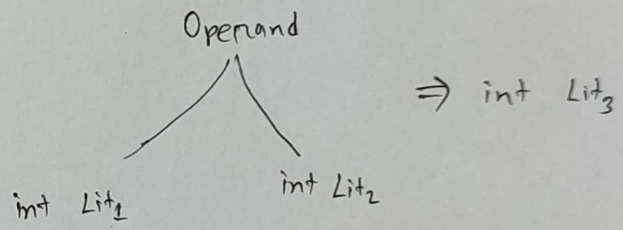
Peephole optimization:

- could be done on
  - the AST level
  - the IR level
  - the Generated code level

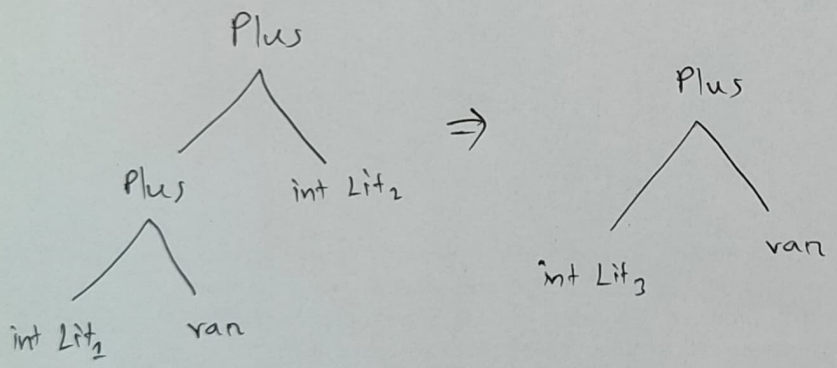
AST level:



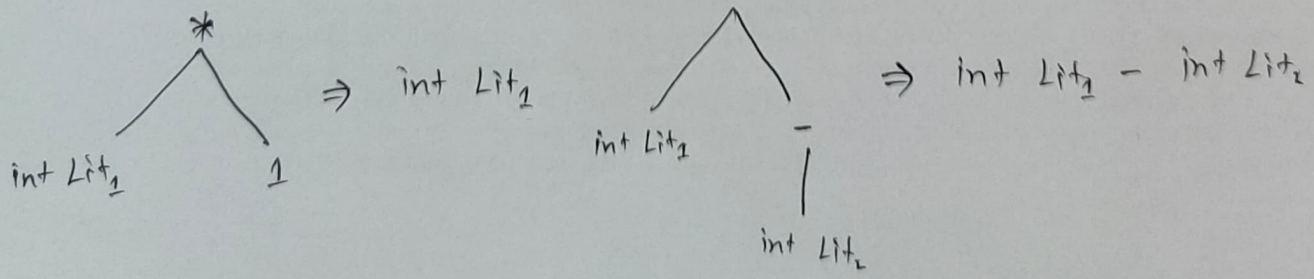
⇒ statements



⇒ int Lit_3



Constant folding:



Constant Propagration:

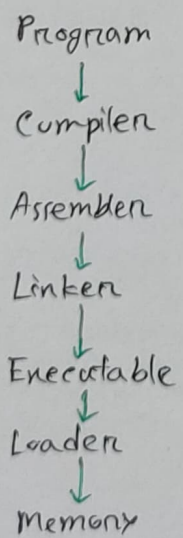
```
int a = 30;
int b = 9 - (a/5);
int c;
c = b * 4;
if (c > 10) {
    c = c - 10;
}
return c * (60/a);
```

```
int a = 30;
int b = 3;
int c = 12;
if (c > 10) {
    c = c - 10;
}
return c * 2;
```

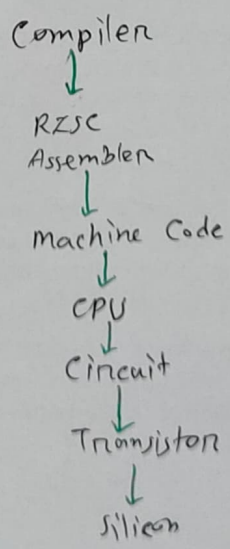
- if we know the calculation, we need to do it during compilation.

L-06 / 11.02.2024 /

- In C,



In C,

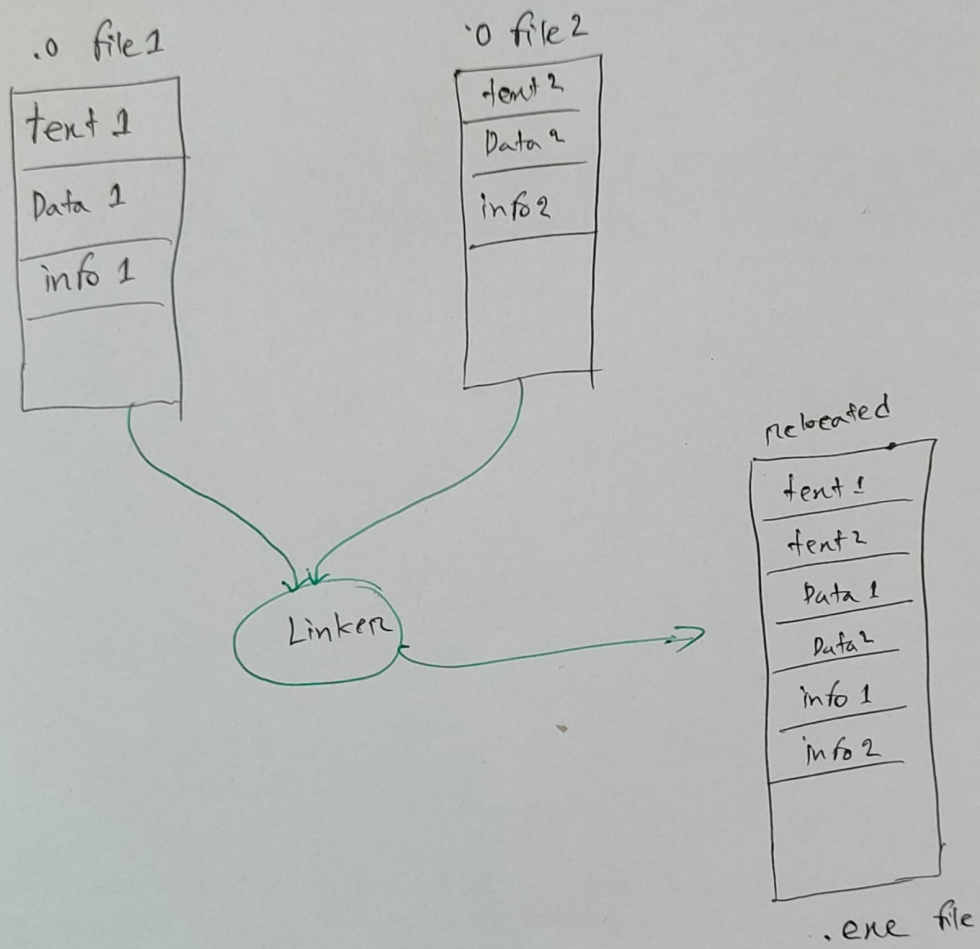


Object File

```
int x = 10;
x = 2*x/*15;
```

```
ADDI  x5, x0, 10
MULI  x5, x5, 2
ADDI  x5, x5, 15
```

```
0 0 1 0 0 0 1
1 0 0 1 1 0 1 1
1 0 0 0 1 0 0 1 0
```



Object file:

⇒ Text: Main code from the instruction

⇒ Data: Binary representation of the static data

⇒ Symbol Table: Contains list of labels of these files and static data that can be referenced.

Task Performed by the Linker:

- Take text segment of each .o files and include those in the executable files.
- Same for the data segment.
- Resolve References.

- Linker produces the executable files.
- Linker must solve,
  - 'printf' routine (which was unknown to assembler)
    - because its an external object
    - put 'zero' address
  - Linker does the memory re-arrangement
    - address resolution.

Language Design Criteria:

- availability (Price)
- Politics

"C" is popular due to availability and it is included in Union OS

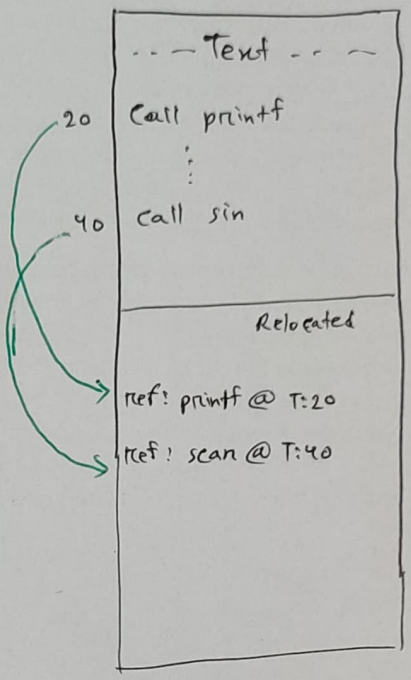
Advantage: Very fast

Disadvantage: Learning is difficult.

2-07 / 18.09.2024

- extern float, sin() → math.c

  extern printf(), scanf() → stdio.c




```
main(){
    printf(---);
    scanf(---);
    result = sin(--);
}
```

Efficiency:

⇒ System Level

⇒ Programmers Perspective

- 90%. time we spend in debugging the code.
  - maintainability
    - not too many surprises
- how well the features of the PL are integrated.
  - regularity

Pseudocode

```
if (x > 0) {
    num_solutions = 2;
    r1 = sqrt(x);
    r2 = - r1;
}
else if (x == 0) {
    num_solutions = 1;
    r1 = 0.0
}
else
    num_solutions = 0;
```

Python

```
if x > 0.0:
    num_solutions = 2.
    r1 = sqrt(x)
    r2 = -r1
elif x = 0.0:
    num_solutions = 1
    r1 = 0.0
else
    num_solutions = 0
```

Regularity:

i. Generality

ii. Orthogonality ***

iii. Uniformity

Uniformity:

- Refers to consistency in appearance
- similar things should behave in a similar way

```
int function (){
    ----
}
```

```
class name {

};
```

⇒ violation of uniformity.

Generality:

- Avoids special cases in the availability or use of language construct.
- Example: nested function not possible in 'C':
  - ⇒ in 'C'
    "==" is used to compare equality.
    However, two arrays can't be compare using "==" in 'C'.

Orthogonality:

- Language constructs don't behave differently in different contents.

⇒ C, C++: values of all data types can be returned from a function except arrays.

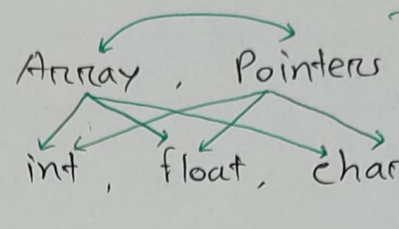
⇒ ALGOL - Algorithmic Language
- ⇒ first general purpose language
- ⇒ one of the most orthogonality
- ⇒ Also too much orthogonality

L-08/23.09.2024/

Orthogonality:

Type operator : Array, Pointers

Data type : int, float, char



all kind of combination should be available.

⇒ Any combination should be valid language. construct.

- too much is not good.

Array of pointers code:

```
#include <stdio.h>
int main() {
    int var1 = 10;
    int var2 = 20;
    int var3 = 30;
    int *ptr_array[3] = {&var1, &var2, &var3};
}
```

Define array of pointer and write an iterative code to show it. ***

ALGOL : Algorithmic Language - 1960

- conditional assignment

..... = ...

⇒ Changes abruptly at the LHS.
- causing enormous complexity.
- didn't get the popularity it expected.

Expressiveness:

- easy to write code
- complex process & structure to be implemented concisely.

```
while (*b != 0){
    *a = *b;
    a++;
    b++;
}
```

```
while (*a++ = *b++)
```

```
# include <stdio.h>
void main(){
    int a;
    float b, c, d;
    a = 750;
    b = a / 350;
    c = 750;
    d = c/350;
    printf("%.2f  %.2f , b, d);
}
```

⇒ What would be the output?

⇒ 2.00 2.14

Fix: float f = $\frac{\text{float}(a)}{\text{float}(b)} \equiv \frac{(\text{float})\,a}{(\text{float})\,b}$

Operator Overloading:

```
a = 10;
b = 5;
a+b = 15;
```

- `+` → addition
- `+` symbol may be used in string concatenation.

⇒ use of `&` → logical AND
→ unary operator
↳ has to do with address

= == !=	+ ++	- --

Types
Type checking } Security & Reliability

L-9 / 25.09.2024

Project Idea

1. Bengali Programming Language design for secondary level students.
   - Compiler base
   - conditional statement
   - Iterative (For loop)
   - Basic File handling
   - Array

2. Vectorized / Parallelization of Language construct.

3. Domain specific Programming Language.
   ↳ Graph based PL.

Reliability:

- if program perform as per the specification in all conditions.
- Type checking
- Exception Handling
- Aliasing
- Readability & Writability

Type checking:

- Done during compilation phase (faster) or during run time.
  ↳ prefered at the early phase of implementation.

=>
```
void test_function (int a){
    ....
    ....
}
```

```
int main (void){
    ....
    ....
    test_function (7001);
}
```

↳ Type Error

Exception handling:

- intercepting run-time errors, and take connective measures
- Divided by 'zero' issue
- C++, Java : Exception management ~~error~~ available.

~~Atom~~ Aliasing :

- Same memory location being accessed through more than one referencing.
- Allowed but very detrimented in ~~Ref~~ reliability. issue.

History

Von Neumann Architecture:

- shared-program architecture
- conditional control transfer (Branching - conditional statement)
  - sub-routine
  - looping

First programming language

⇒ Short Code - 1949

- Instruction were written as sequence of 1's and 0's.

- Then designed a compiler.

  ⇒ First compiler - 1951 - by Grace Hopper.

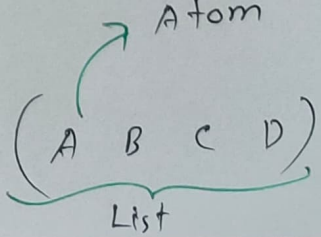
- Then came, Assembly Language.

  - mneumonic based language
    ↳ MOV, LD/LDZ. MUL/MULZ
  - close to machine code
  - assembler does the conversion to machine code.

- Then Fortan, and then LISP.

- LISP: List Processing Language
  - second best programming language.
  - 1958 - John McCarthy
  - basic & only data type was LIST
  - introduced symbolic programming
    ⇒ x or y | OR (x,y)
  - used in AI application
    decendent such as SCHEME

- LIST:
  - Lists
  - Atom ⇒ identifier / Numeric Letters.

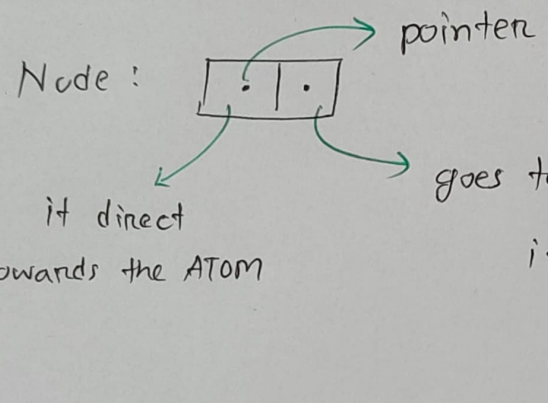


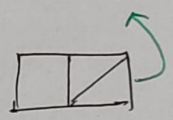
$(A\ (B\ C)\ D\ (E\ (F\ G)))$ ⇒ List within a List
- nested list

⇒ stored sparsely in memory
- two pointer for each atom.

⊛ How to represent:

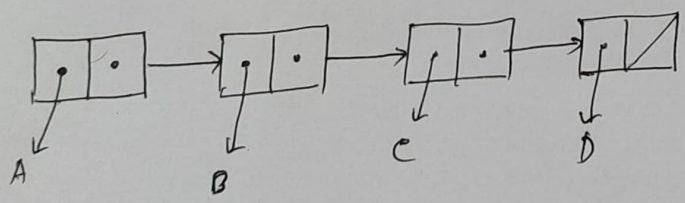
- Box pointers diagram



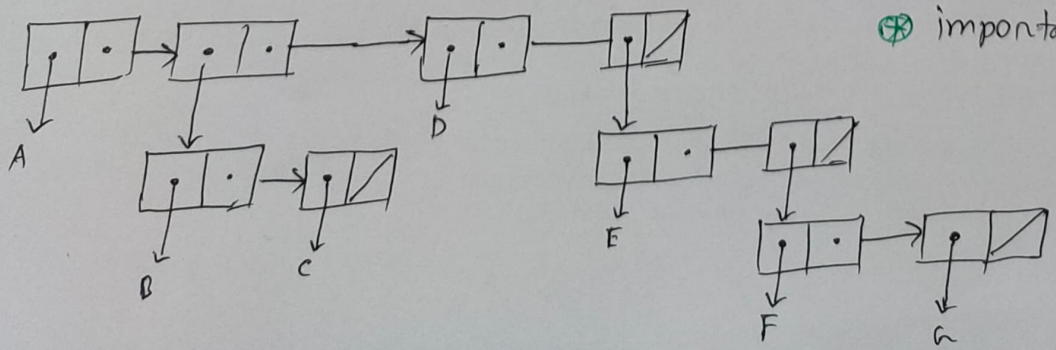


⊛ $(A\ B\ C\ D)$

⇒



⊛ $(A\ (B\ C)\ D\ (E\ (F\ G)))$



⊛ important for exam.