Feb 28

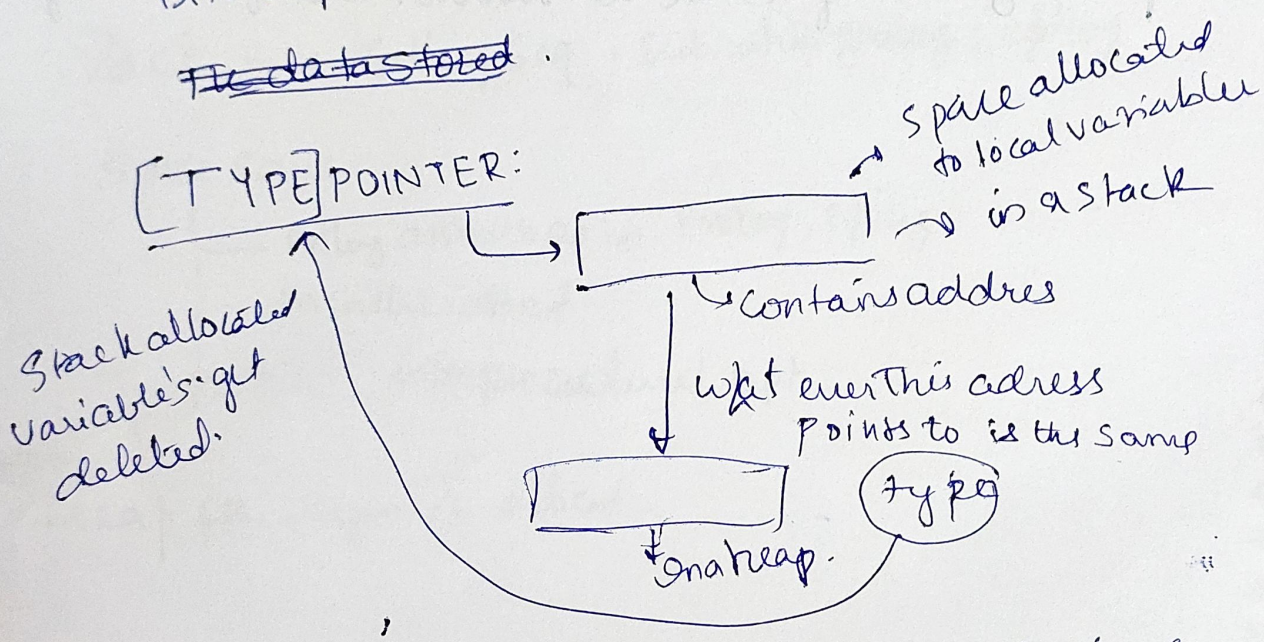
pointers are used, not accessed by user,

↳ they have ~~spar~~ occupy space ....

Pointer type is all same
(ie) char pointer, integer pointer ...., pointer
is effectively same.

→ given type of ~~void~~ pointer decides where those Bits represent, a number, character ... etc.

~~It's data stored~~.



Rust has pointers to creat program at System Level.

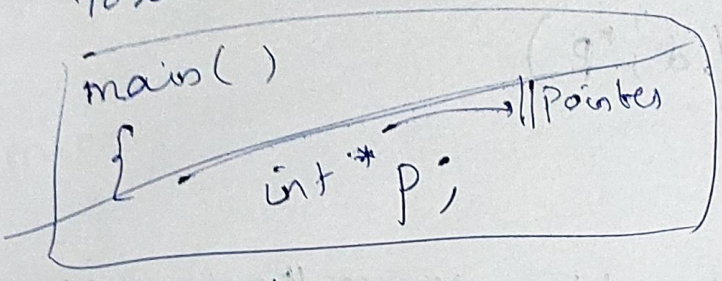
facebook app can use pointers to access ram and what ever app's in ram can be exploited.

bit boundary are created, using Pointers dynamical allocate memory from heap.

OS will load a program at same part, then malware is spread to get remote access

* they randamization of load of diffirent blocks takes - ing place to prevent this.

→ Pointers being used as reference to some memory locations.



~~int *p();~~

int *p;

foo(). { int n=3; p=&n; } // binding of p, and 3 is allocated binging add is stored in p.

main()

{

foo();

print("%d", *p) // n is deleted, p has add of n ∵ p is out of foo.

}

Stack frame of a program

gooda ns: segmentation fault

bad ans: 3 // Illigal access to a portion of memory that is supposed to be free.

```
main( )
{ int * p = malloc (sizeof(int));
    *p = 3;
    free(p);
    printf(" %d", *p)
}
```

These are called dangling reference, if referencing a address ~~with~~ this which is invalid, but pointer is ~~alive~~ alive).

```
foo( )
{ int *p = malloc(sizeof(int))
    *p = 3;
    printf("%d", *p);

main( )
{ foo( );
}
```

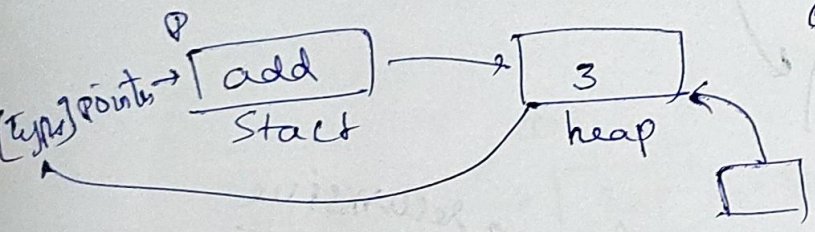
P is local variable

when program called multiple times heap doesn't get deleted, so, after much time, memory leak occurs.

~~If some lang~~ Dangling Reference (less treath)?

memory leak → 'C' uses free, (ie) it delete element in heap and then in stack using destructor.

Not only in pointers but also in references (in some language)



Ex: L = [1,2,3] → garbage collector

L = [4,5,6]

↓

list is created and referred to L twice but initial reference space is not destroy

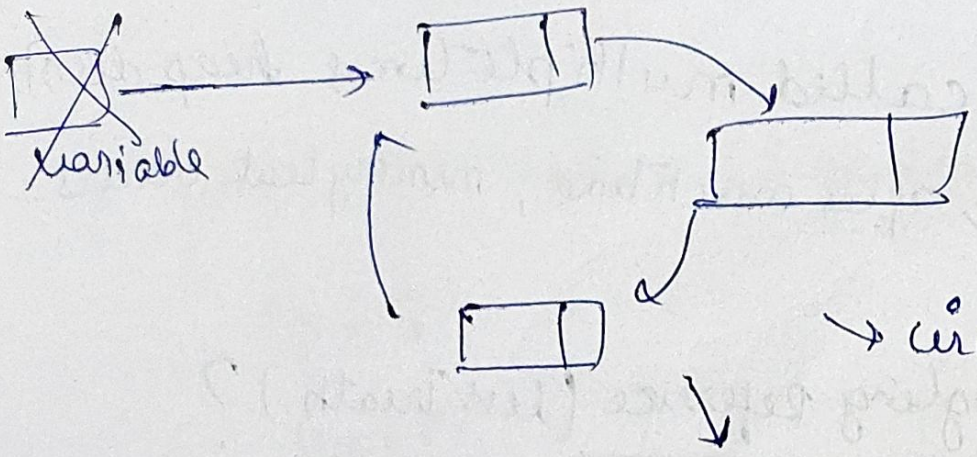
if L = [1,2,3] creates again third creates again.

2 variables ⇒ count = 2 access

If block has count = 0

↳ no reference, can be deleted by garbage collector.

If refrence count ≠ 0 (is it being used?)



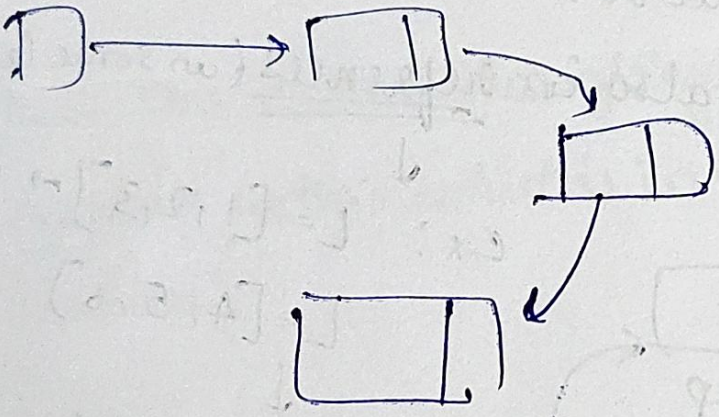
reference cycle

→ circular ~~ties~~ ~~pointers~~ referencing.

↓

Non zero count ~~for~~ of references.

then garbage collector don't deleted, (Perl)

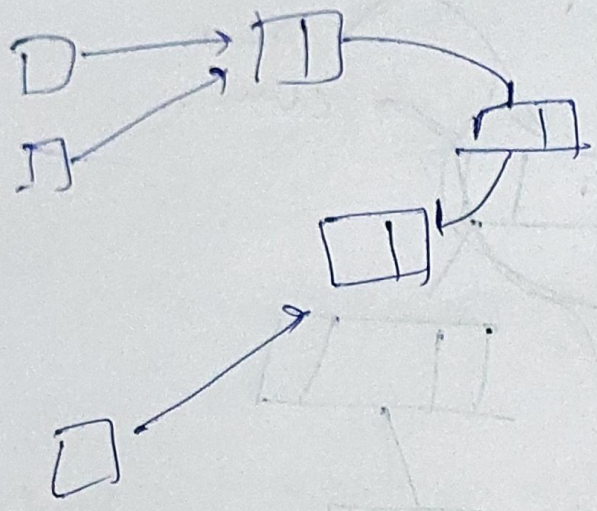


Mark & sweep algo → recursive

accessable blocks are ~~ob~~ mark as useful

but unaccessable blocks are unmarked as useless

→ then useless are sweeped.



garbage collector runs when space runs out / garbage collector crash because, sufficient space is not there for it run.

// how tho remove recursive nature of mark & sweep.

garbage collector can run in diferent core.

↓

not a single operation. (∵ heap

| concurrent running gc has problem.

Program → access heap ⟷ GC (✓) (accepted)

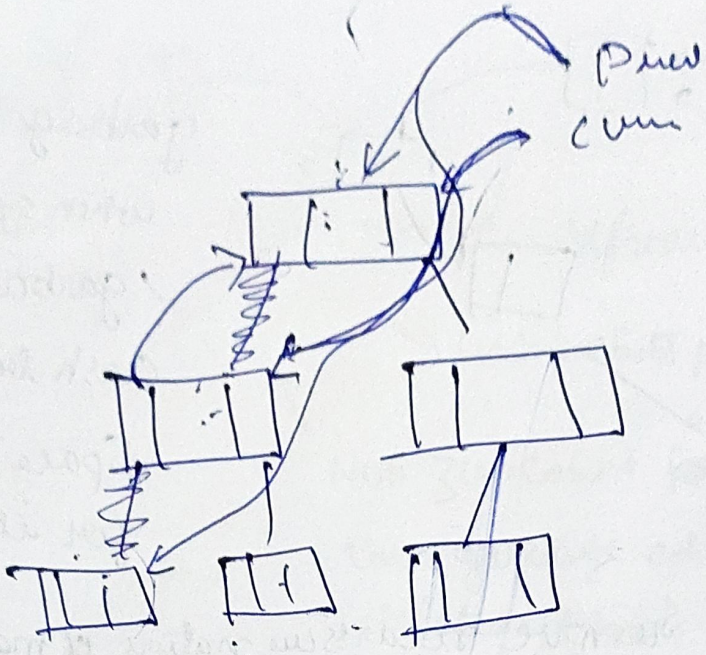
Program → heap ← GC (✓) (accepted)

program → heap ← GC (X) → not accepted

↳ referents / pointer

↓ Can be removed easily.

↓ Can be remove



Mar 1] ~~data type~~:

List → can defined in recursive manner.

List → object, List

↓ → NULL

Contains diff combination of objects.

* In functional programing Language List is important. (Lisp, ~~used~~ exploits recursive manner of List, program is also consider as a list.

List → Executable
↳ takes data.

Ex: Python:

L = [1, 2, 3] → needs to be evaluated (by grammar, parse tree)

* but List cannot be determined wether executable or not just looking at parse tree.

⇒ New Languages added "List comprehension" in to the syntax (ex: Python)

example

A = [2 * x for x in l]

Write expression to evaluates/equal to a list (shorthand way of writing)
↓
efficient than for loops... etc

List comprehension ≈ "mapreduce"
generalised
↓
apply function & grouping.

Datatype: File I/O

* All programs ~~above~~ ok with file I/O getting I/P from keboard, screen → O/P.

* dependent on API of Operating system.

* Os ~~access~~ file, optimize, caching a file, ~~data~~
  It provide ~~opsts~~ abstraction called API.

* if text file 1bit of data (reading) → charac
  if image file 1bit of data (") → some thing deff

∴ file ~~ty~~ type is imp for Os.

Open op in ~~(file)~~ file I/O : (read mode, no writing)
(write mode, creation when not existing)

↳ Read, write, seek, close.

⇓

Read line, Reads bit by bit & (forming list of string)
give O/P combines

Read lines, Read diff lines

file → Sequential structure, no random access
(ex: 35th bite of file, I want (X))
↳ like tape.

seek → (35th bite of file, it is not random, not constant time

close: OS ensure that a particular file is not access by any one.
↳ book keeping
close unless closed, write function don't goto disk instead they are store in cache waiting to go hard disk in 1 go.
/ not written persistantly.

LinkedList

A operation on type: Assignment, equality

Taking value of right hand side and assigning to left hand side

Not same.

boolean check for left & write hand side

Assigning from 1 pointer to another
→ shallow copy (address is copied, not data)
→ deep copy (even data is copied).

~~2 Int. pointer & float pointer are~~

In equality: (check b/w reference) → check value than type

* Int type points & float type pointers.

being

rust → references / not pointers

↳ So, no dangling reference

rust reference

↳ values compared

raw Pointers → address are compared

march 4|

function → takes input, gives output (const func → no input)
↳ Parameters.

→ maintain modularity of programming language.
↳ not a feature of Program language.

→ libraries (calling library func)
↳ used by other programs to complete a operation.
(modular programs)

⟹ For a imperative language (func not important).
" " functional language (" are ").

func definition →

func return → C, C++ returns only 1 value.

Python, Java. returns multiple values

Functions / Procedures :

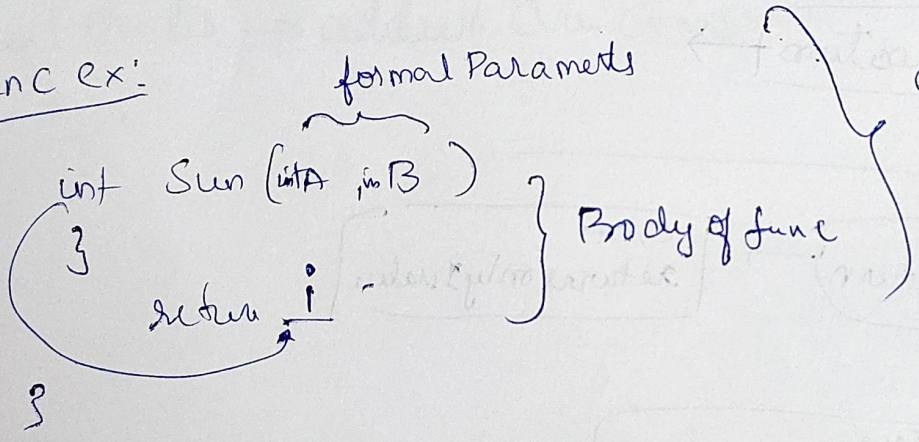
- returns a value
- don't return value (only does some work / Procedure which may or may not have effect on program)

→ Block of Codes

↳ Can be referred to same block of code by refering name of func

If has Name → func / ~~else : procedure~~

func ex:



func Call ← Sum (5, 3) // expression

Actual parameters

declaration

If Sum ([Array], A)

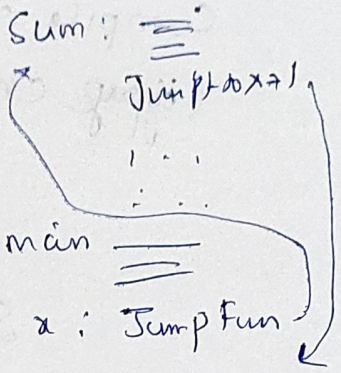
If Sum ([Array], B)

↳ can be any expression.

assemble level language ⇒ Goto, Jump ✓ used.

~~Inting~~

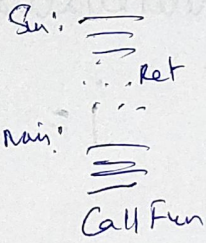
Inlinning (taking func & writing it where ever it is needed).



Call Fun, RET } does Book keeping, so know which line to Comeback

Return.

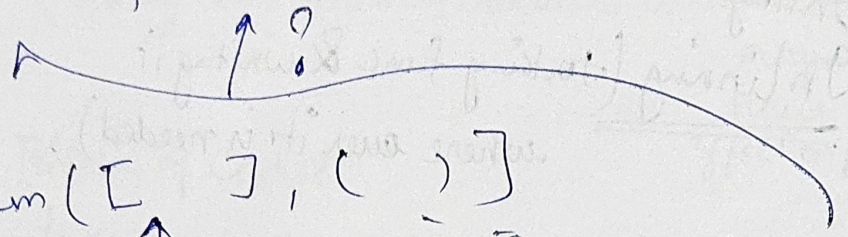
Stack of Program: except for statically allocated memory it is empty.



There is variable part (which accomodate variables of diff size).

Compile-time (typeg checking) | Run time

Process evaluation (complete type)



↑ what value to be passed, when is determined.

↓ evaluation? (*)

↓ run time

Parameter evaluation: C++:

j = Sum(i++ , ++i)

↳ bad in C

↳ variables are (deactived (i)) as out of stope, they are in stack pointer

(a, b) ↳ statically allocated into stack (becomes TOS ~~stack~~ is stope). (comes to next step)

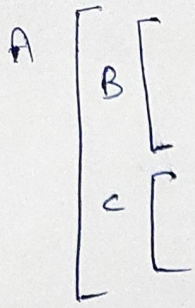
after execution of fun (a, b) goes away from ~~over~~ ever it is called (add where it is called is also in stack) ↓ (comes to next step)

assign to j & returning to Sum(i++, ++i)

↓ & is diff step

the next step after return to sum(i++, ++i)

Nested func:



If C is called instead of A, Context of A should also be kept / or you will lose axiss to C.

Static links ? dynamic links ?

book keeping kept by compiler

Pascal / Ada: func needs to be declare fust than defination it self.

rust :

Header file: → Contains delaration of C/C++ file where in defination reside.

↳ ? In library func usage (Compiled code)

~~ques~~ · how dose compiles knows a deklerations is correct ?
(No problem in compile time ; but while linking ~~it try~~

Compiler checks with Compiled library Code to ensure every thing is defined or not).

Inlining:

Inline int Sum(_, _)

↓ ↳ Inline decrease compile/run time.

(compiler decides whether a func should be inlined or not)

Mar 5

Parameter passing:

int Sum(int A, int B)

↳ formal parameters.

Calling func → Sum(i, 5)

↳ actual parameters

pre-order way of calling a func (in every language)
except normal operators can be overloaded.

int +(. . . . . .

Operand before operator
↳ preorder → int +(

Operator b/w operand
↳ Inorder → 5 + 6

2 ways of Calling func:

1st { i is copied to formal para A
5 " " " " " B } Call by value

2nd. If addr is copied instead of value
then it is Call by reference

a, i are references of pointing to same variable.
have

2 variable ~~pointing~~ to same value. ✱

Ex 1:

```
int x
fun (int y)          // (int * y)
{                          (*y = 3)
    y = 3
    print ("%d", x);
}
main ()
{
    x = 2;
    fun (x);          // &x
    prn ("%d", x);
}
```

If calling by value → costly func (: copies data)
* value of x don't change.

So, ans: 2, 2 → readonly

(2) By refernce, x, y same vari; → write
* value of x change, y don't change. ans: 3, 3
* x, y point ag 3

other way of calling func :

Call by value / return.

(*) ~~Similar to call by reference~~ :

ans : 2, 3
↓ ↳ main
fun

→ In array in C

(i.e) ~~int~~ main ( )
{
x = [2, 3, 4]
fun(x)
print ("%d", x)
}

If 1st element add = reference of Array

∴ In Array case only call by reference works.

call by reference → less costly (Copying reference)
(de-referencing?)
↳ costly (*)

(*) ~~of variable~~

Value can be change if call by refern

" dont " " " " " value.

→ C++ ; absence of refrence (*)

↳ we should use de-referencing

pointers (**)

fun (int & y) → refrence // similar to pointers.

y = 3 ;

→ Java // C, → Call by value (default)

// fortran → " " refrence

↳ there is L value is needed.
(addres)

// fun(x ≠3) → exp // not having add.

// ~~but~~ fortran stors exp & sends add.

↳ hybrid model., int, float, char data type

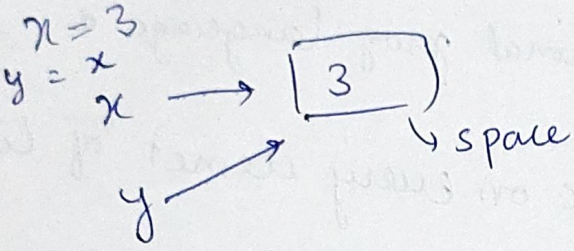
Call by value

if using class, obj ... → call by ref

↳ ~~user defn~~

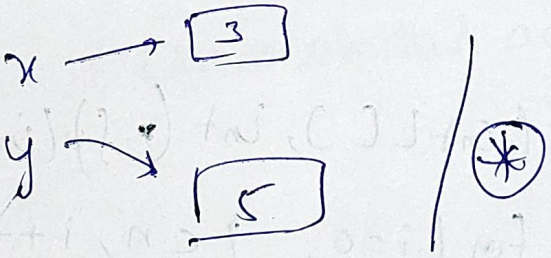
user defined data type
basic defined " "

→ Python:



only reference. So only call by refrence.

If x = 3 // immutable not like in C
y = x
y = 5



⊛ faster → call by refrence. but not change value.
(read only)

fun (const int y), value stored in y is constant.

Rust:

Call by reference but reference can be immutable (*)

passing func as parameter:

* lmp in functional prog language

Applying func on every elemet of list L

↳ apply [L, fun)
f [l [i]]

In C, C++, nesting of Sub routines is not allowed.

apply (int L[ ], int (* f)(int)) → // (& in python)
{ for (i=0; i<n, i++)
{ f (l[i]) (*) no need of de-referencing
}
}

On imperative programming lang less seen

```
{ in a,b;
  fun (x)
    {
      y:x+a+b;
    }
}
```

referencing this func

// a, b are not defined in func
So a,b not be referenced

So, Surrounding Scope cannot be accessed in C, C++ (Closures)

March 6

default parameter // can be omitted actual parameter list.

Python:

func being called without

Ex:

```
def sum(i, j=5):
    return i+j
```

default parameter (∵ when no value is given 'j=5' is considered)

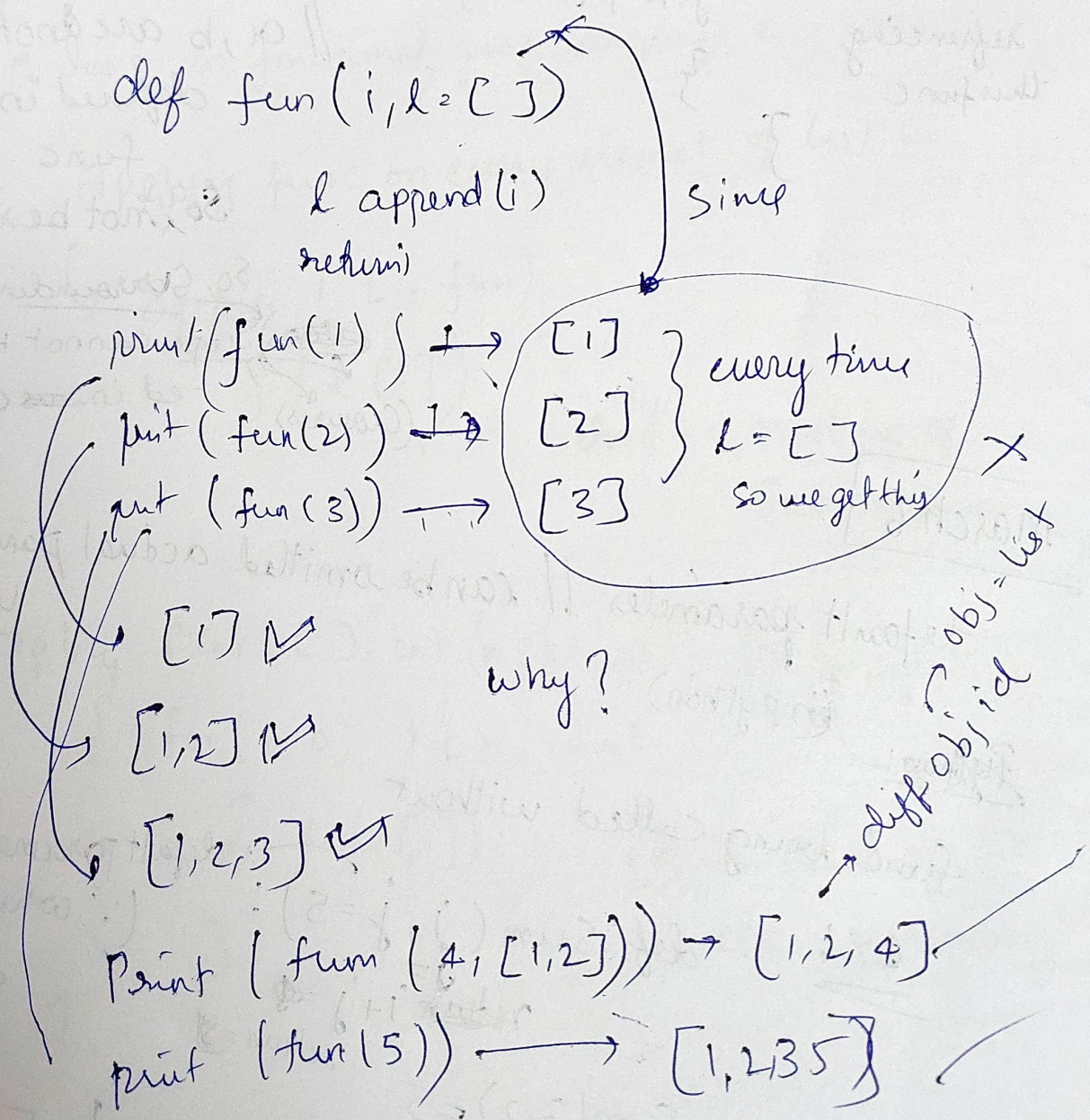
∵ 5 is immutable object, so, not 14.

Sum(3,2) = 5
Sum(4) = 9
Sum(5) = 10 // why not 14

Python →

all, non-default parameters should be at begening
and all default " " " end

Variable based language :- It works the same
(ex: ?)



id()

→ every obj has id, so id() returns id.

So, def fun(i, l=[])
    print(id(l))
    l.append(i)
    return l

l, default variable called again when no default parameter mentioned.

print(fun(1)) — [1] id: x
print(fun(2)) — [1,2] id: x
print(fun(3)) — [1,2,3] id: x

→ So [1,2,4] id: y

[1,2,3,5] id: r

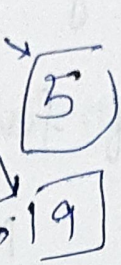
obj by convention followed by python
Once object created it will use same reference over & over again if default para is mentioned. if other wise

def sum(i, j=5) → positional parameters

print(id())

y = i+j

return y



new object created ∴ diff id.

sum(4) = 9 id: x

sum(5) = 10 id: y // ∵ not 14.

∴ check if it is mutable or immutable object.

Can skip default parameters.

ex: f(dp, p, dp) → 2nd position then what.

then you should use name parameter

In some case how many parameters be passed.

cannot be determined.

ex: printf, scanf in C

("", i, f) ... used

strings, char, int...

rust println

Can accept multiple parameter but don't know how many are

Types, no

int printf(char* format, ...);

what are these

... ellipses

* there are 2 lists

Va-list params;

↓ based on compiler.

→ list of all parameters

printf (char* format, ...)

Va-start (Params, format)

↳ last known parameter

i = Va_arg (parms, int)

↳ expected parameter type from eclipse

f = Va_arg (Params, double)

In runtime if Va_arg doesn't match then error occur. // overloading → of no. of parameters are fixed.

func

printf ("%d.%f", 1, 0.5);

printf ("%f, %d", 0.5, i)

if d → int

if f → double.

python is of dynamic type.

↳ even is possible

type coarsion ?

Statically typed

defining, checking is compile time

Return

return i;

↳ returns value of i // type check by return

return type of a func

(i.e) int fun

⇒ In python

→ fun = i

→ return ; // fortran do at end of subroutine

// return variable is defined with type. (ex: rtn : int)

↳ variable

then put rtn = i

In python:

return 1,2

(i,j = foo())

Rust: specify exp ⇒ return type } same

11/3/24

generic functions

```
fn Sum(){
        Println!("Inside function");
}
```

```
fn main(){
    Sum();
}
```

Output: Inside function

```
fn main(){
    fun(1);
}
fun Sum(){
    Println!("Inside fn")
}
```

Out:

Type inference

fn

Im-1

output: error

Im-2

output = 5

~~last~~

Im-3 (inside func)

output = 11, 22

Exp Vs Statment

valid sentence, gives a value

↳ sentence ~~(or)~~ in syntax of a language, complete in itself. do give value

for fun last exp is going to be return value

↳ ∴ dont put ; , if you put ; at return then it becomes statment.

If you put ; at end ⇒ error.
() → implicit unit type for return value. if not returning anything.

∴ "tail exp" ⇒ return exp

Sum(x) {
  return x
}
↓ return refrence

I m 4
output: 5, 8

once gone back variable will be refreshed & hence mutable.

multiple values returning:

variable → mutable
sending reference → all refrence

~~output~~ output
I m 5
output: (3, 5)

copying value of refrence into y & x

Alway call by value!!

& In → J = 4

*y → refrence.

once liftime over refrence add where obj is there dies.

& refrences.

Static lifetime

∴ Rust → memory safe language.

Mar 13

Generic functions

OS → queue
- ↳ Scheduler
- r/w blocks
- stages of process writting

to remove same implementation repetedly we use generics on subroutine / class

↓

doesn't depend on type of argument

polymorphysim / Generics

(diff funcs written with same name, when name called Compilies take decision what to be called.

↳ no 1 func - 1 name

diff type of data goes to this func

↳ data type

↓

generic of data type

```
Sort ( int arr[ ], int len )
  {
    .
    .
    .
  }
```

↳ type is ignored ⇒ any type can be taken (T)

↳ Whatever type

Sort [T arr[ ], int len] → func^s with generic nature.

↳ float point array
int array

- T is deffined during giving input.

Generic subroutine with diff data type's

↳ C++, Java, C# ...

In C++ ↓

Calls them Template

↓

```
template <T>
Sort (T arr[], int a)
 {
   :
 }
```

Ex: Char st[100] ↳ string of length 100

sort[st, 100]

Then compiles makes 2 copies of subroutine with data type T and data type ~~sort~~ char

parallelism to polymorphism

* If T has codes of ~~with~~ related int instead of char, compiles do typecheck.

Java:

does'nt do different instances of a func(s)

→

```
publicStatic < T extends compariable<T> >void Sort(T arr[])
{
}
```

↳ Sub routine generic in nature

All the call's to sub routine ~~are~~ maintains in same

Code no parllelism of polymorphisms

↳ does type checking, type safe (∵ we put extensions)

Errors

Char Lib[2][4] = {"ABC", "DEF"}

Sort(Lib, 2) Sort them

have copy of characters / strings.
pointers

Compares pointes of the string but not the value.

↳ but answer may not be expected.

logical error not noticed by compiles.

---

func returns a refrence to Type T

↓

compares refrences not value here.

&[T] → refrence to array T

checking type safty, elements of T be of spcial property.

Exceptions handling

(Aborting execution, why)

↳ call special func's

↳ returns main calling fun

How to raise exception, handle them?

↳ runtime env handle left out expection

unhandled exception ↳ when it arises.

[→ depth of nesting depth.

]

python:

→ exception handling

Try: → func, ops...

= } any of content where we are expecting exception

(responsible for exception handling)

except:

↳ can undo actions which cause exceptions.

(a bookkeeping

Stack trace

Exception handling : more like Error handling

↳ 2 returns

↙ unsuccessfull.

↘ successfull.

C!

Error value : 0 ⟶ no error (perfect execution)

otherwise ⟶ whatever it returned is not correct value.

2 3rd level -, 5th level, - Liberaries

OpenSSL

(Passing a func)

EH:

given: ~~name of a func~~ a func which can do cleanup activity

error in ~~call~~ func, ~~when~~ then pass error handling func

→ Stop execution

→ kill or reexicate - - -

may loss the track of nesting depth. (try & catch)

(check photos)

Ex:

(PLY) lang

↳ On Overflow (exception)

Begin

=

=

=

END

} binded dynamically as a func

where ever Overflow occer this is executed, may loss track of nesting depth.

C: EH

try {
:
:
} any exception occurred here are only handled here.

}

Catch {

}

→ any exception here is not handled here by catch.

Overflow exp
end of file exp ... many exp

↓

all are handled here (only here)

after 1 try block you can have multiple catch block.

Catch {(i.o error e)}

Catch ((end of file e))

i.o → derived

If E.O.F is inherited by i.o E from Parent then

If E.O.F error occurs then i.o error is also flagged.

→ Inheritance of exceptions.

(main thread)
(current func)

→ unhandle exception - propagated to main func
if not like (10 is not handled, then it returns a exception)
catch { } → Handler, if not handled in main thread, then it crashes & handled at Runtime.

→ catch - all - exception (undo's all op's / generic cleaning)

python: finally: -
C : Catch :

→ some languages we need to specify which errors to be handled.

→ Python/~~Java~~: no need to specify

C++ : → all exception's handled by user, exception occur / program stopes.

Java : checked, unchecked class of handling

checked → handled in Runtime at worse / ~~search~~ for handler (may crash)

unchecked → program has to stop. (Panic occurs) (no - search for handler) (crashes always).

Exception handling:

Rust:

Panic	Result
kill program with some message trace (see stack where it killed)	Enumerated types. OK(T) ERROR(E) T, E are generic type, depends on context.

Im I : Panic.

Im II : compiler is undecided where bound is defined or not. ∴ not done in compilation time, by done at runtime.

Im III : Recover it with some result

No processing on output so, no error.
(don't panic, ∵ recoverable.

Im IV : If OK file pointer is created ok(file) ⇒ file.
If error ⇒ panic! ↯
↳ Problem opening file.

If ~~hello.txt exists~~ hello.txt exist

ok (file) ⇒ file

Apr 3

Object oriented (OO) → to reuse data (arrays, structures ...)

- OO features.
- group of data & operations are required to a particular action
  - data items
  - function's

modules → combine data & functions to get single entity. (many instances ⇒ many entities)
(there are restrictions on access patterns)

Ex: Student → name, disiple, department (is visibile but not accessable from outside)

~~Student~~ "object" → To offer above facilities.

OOL → C++, Java, C#, python ...

changed to machine language (dont have objects)

restrictions are only at language level.

Class → Templete which has blue print ...

Ex: →

```
Class student {
    int ID;
    int prog;
    int desiplain;
    int dept;
    calc-cgpa(){ -- ID}.
    cal-sgpa();
}
```

member → class variable.

object

Student S1

S1.ID
S1.prog
S1.calc-cgpa()
S1.calc-sgpa()

Instance of class that is created to contain the state is called object.

Java: used for types with same implementation

- Objects class they are for abstraction.

C++: → 3 ways of restriction of members

→ public → member can be accessed by anyone (not class variable) → object variable

→ private

→ protected

In public → member → obj var not class var
& obj should be accessable (In scope)
then only obj var can be accessable.

private → not accessable by any one, only some functions can accessable. in the scope of class not outside the scope

like as Cale_Cgpa use ID is the class.

* we use public function to let give access to member variables which are private.

→ getter() → gets value of member variable from inside → to a particular member's only.
setter(val) → set " " " " to some other value

→ androidestudio: custome class provides a suggent to add getter & setter functions to class

C# → get( ) → It is implictly calls when we use
{
return ID
}

i = S1.ID;

class with repeated code: then we use "derivade class"

derived class:

derived class → Class ug Student : Student ← Base Class

{ → restrictions on members

}

does restriction in Base class are inherited into derived class?

C++ → gives option wheter our new derived class will have public or private

If class is derived public with private mem then.

C++: you cannot reduce restriction you can only increase restrictions ∴ private in Base class will remain same.

- protected member are accessible to derived class
  private " " only " " Base class not to derived class.

↳ reduce work by not writing common part.

Java: protected is a bit diff
any func in same module can use the protected mem.

var → mem, copy of membervariable
manage, copy func to derived class.
↓
no-sences
∴ reference to func

S1.calc_cgpa()
↓
in S1 only uses reference here.

→ new obj, what if variable need to be initialize.
↳ create a constructor (has same name as class)

ugstudent()
{ } routines for initialization
}

C, C++ (copies)

~~int ID;~~

Python : int ID

↘ object created and refrenced not copied.
(explicit creation of object) ~~int ID~~ for variable

x : S2, S1

↘ have seperate mem location for ID

Apr 5

feature OOP's:

Stack, queue ⟷ (storage) array, list, linklist

Push
Pop

abstracted out (Storage) ↘ (implimentation details)

Sub / inner class:

```
Class outer {
    class inner{

    }
}
inner(i);
```

* visibility of inner is available only in outer if 'i' is public then only it in the outer class,
* for inner class (IC) outer class (OC) is outside, if 'i' private outer class can't access.

Java: IC can access OC var

C#:

Constructors:

for intializing

- called automatically.
- constructor has same name as a class
- many constuctors is possible for 1 class. the diff is sign of constructor (input)

```
Class stack{
    stack()
    {

    }
    stack (int n)
    {
    }
}
```

Stack S;
↳ based on this defination the following construct is called.

Java, Python: Constructor is called as

stack S = stack( ); → object is created

∴ they uses refferces as variable

↳ ∴ creates reference not object

destructor:

C++: ~stack(int n) → release of intzalition

{ → delete those memory (garbage collector)

}

Py, Jave: no destructor

as out of scope of S then it is deleted

Class stack : Private adt

stack ( )

{ super( );

~sta

* ~~Cons~~ Call base class constructor then called derivid class constructor.

→ Initialize base first then derived.

use Super() to acces base class constru so, inorder to change the order.

Constr

~~oop class polymorphism~~ overloading; same name of ^ used differently by complier.

ex:

```
Class Student { }
  Class ugstudent : student
  class pgstudent : student

Student s;   s. calc sgpa
ugstu u;     u. calc-sgpa // not in stor pg st
pgstu p;     P. calc_sgpa // not in ug or stu
```

```
Student * sp;
sp = &u;
sp = &p;
```

} created pointer for base class assigned addr of derived classes into ~~inviddof~~ pointer variable of base class.

sp → calc_sgpa(); → calls from ug stu if it store addr of ug stu

Sorting func → ~~takes~~ takes Object O → superclass

any class reffernce in the variable "O"
and compare with other classes.

```
for ( ... )
{ sp = S[i]
  sp → calc-sgpa();
}
```

dynamic binding at runtime

we don't know if it calls ug / pg's.

we can overwritte mem's of base class in derived class.

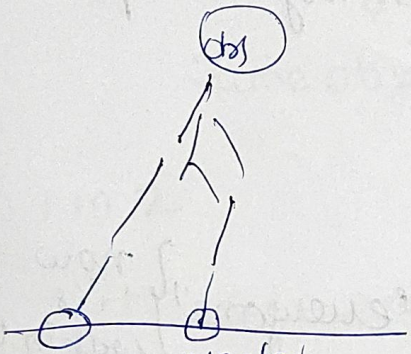
if ug has calc-sgpa but pg don't have.
then there is chance to flag a waring in compile time that dynamic binding don't work at runtime.

~~class~~ Class Node:

```
Self i = 0
add-node(i):
```

Abstraction,
inheritance, overloading, polymorphism
↳ In python.

→ to do → Abs, inheritance in python complete.



↳ derived class.

Apr 6

Python:

Class <name>: statments

stat1 ⟷ like i = 1234

def Print()

problem occured are: →

```
Class Test:
    i = 1234
    def printTest(self):
        print("Hello")
t = Test()
t.PrintTest()
```

output: Hello.

'i' → mem vari (public by default)

t → object of class T

p "self" → current objects scope

```
Class Test:
    i = "world"   # not defined
    def printTest(self)
        print("Hello" + i)
```

→ i = "everyone" } now 'i' is with in the scope

```
t = Test()
t.printTest()
```

→ Hello everyone

→ interpeter give implicitly a argument "self")

```
Class Test:
    :
    print("Hello" + self.i)
    :
```

but when we go this if this nothing () then error comes.

→ Hello world.

im1

t.i → refers to this String → i = "world"

& j also refers to

Same object another refrence.

im 2

T.i & j refer to same object.

im 3

x is not in class but created & as soon as we started using it.

u.x has no variable but on specific to object 't'

when Test is created it var 'i' and refrence of print