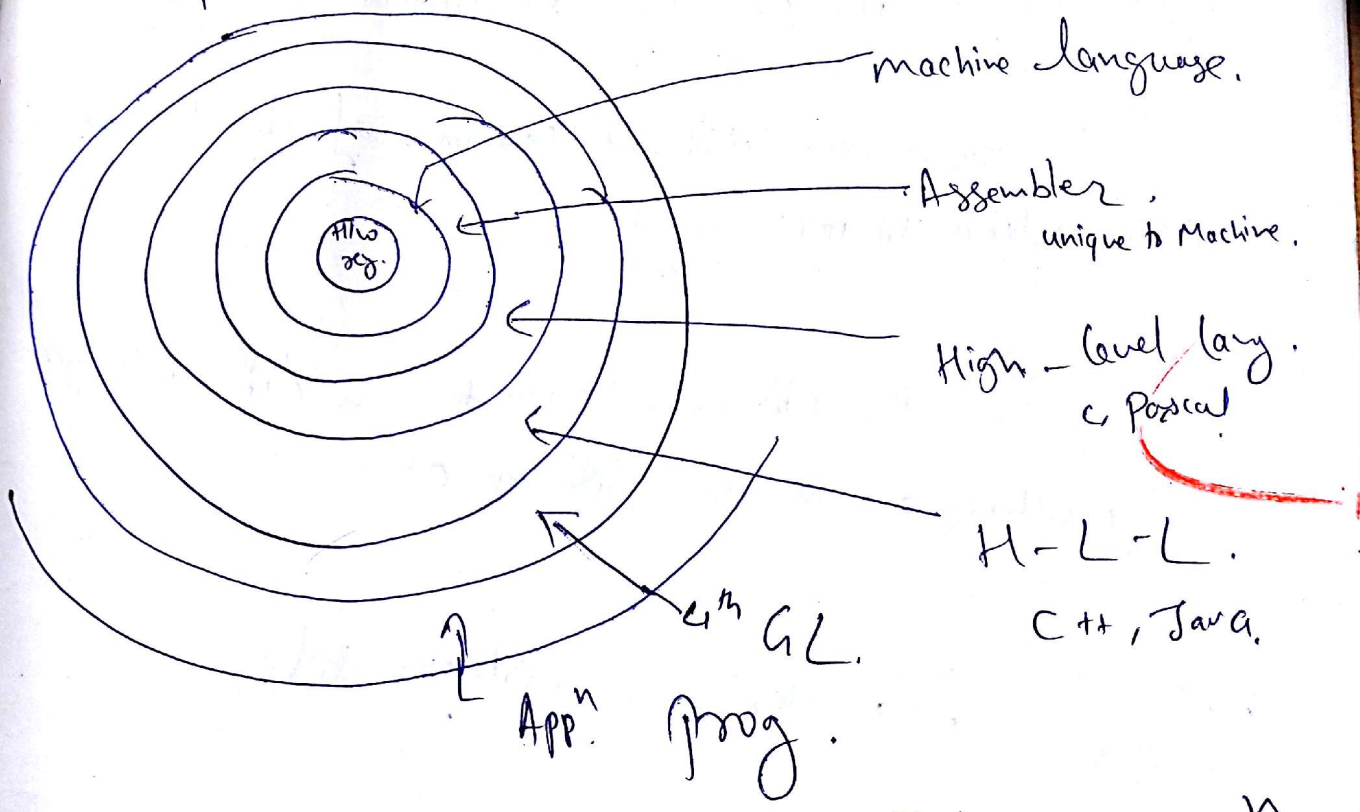
Software.

" H/w is merely a vechicle for allowing the S/w to express itself."



From H/w to App^n.

Combining H/w & S/w

⇒ Natural language discription.

⇒ Translation of the problem Statement into a more Computer-Compatible form.

↳ use high-L-L (e.g C)

depends on Problem Constraints & performing the translation.

Additional levels of Translations are Necessary too.

→ HLL must be translated to a form which can understand → i.e. 0's & 1's. i.e. object code.

→ how are different versions of target language accomodated if necessary?

→ Can the obj code execute on different target machines or run on different OS?

Source code. → [Compiler] → object code

Translating from Source Code to obj. Code.

⇒ Preprocessor : tool to help in translation process Performing.

↳ Several operations to prepare Source file for Compiler.
- ↳ inclusion of header file's
- ↳ line control
- ↳ Conditional Compilation.

↳ ~~inst~~ Macro Processing

↳ Selecting Source text for Compilation

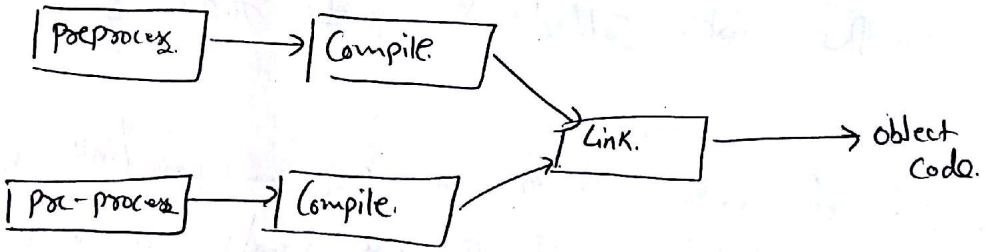
↳ incorporating different Shared files into the file that will ultimately be compiled.

Source code → [preprocess → Compile] → object code.

Cross Compiler tool for translating programs into variety of forms.

⇒ a Compiler that runs on one machine typically the development platform, and generates code for a different machine, typically the target machine.

Assembler ⇒ Converts the collection of Assembly language steps into machine language (i.e. 0 1's) object codes



Linker and loader ⇒ Although the prog. is now in machine language, it is not ready to be executed.

⇒ Prob: All variables and data structures used in the prog. must reside in Comp. memory and each need's addr in memory.

Q. which address should be used?

A ⇒ Assembler generate relocatable code.

Q. Is it benifical to use existing code.

A. yes, reduce development time & cost.

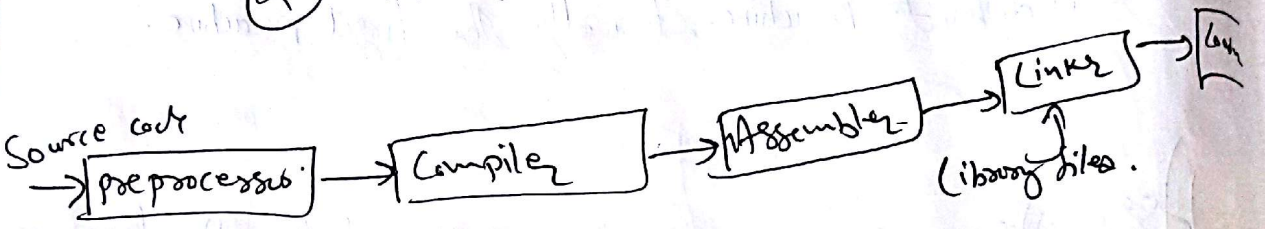
Q How to incorporate such code into the prog. without typing

A ⇒ with linker - loader

Linker loader does.

(1.) links a collection of prog. modules together, and it resolves addr prob.

(2.) it resolves add problems.



⇒ During the Process of translation, the names of all variables are identified and an entry of each is made in the table called "Symbol table".

⇒ if when linker arrives and the linker can't find that definition, the process ends with a link error.

Linking ⇒ involves combining all the machine code into a single file and re-referencing all addresses to the start (addr 0.)

⇒ Storing app^n goes to flash-type ROM.

An Embedded C Prog ⇒

A. Prog: any set of instructions.

⇒ Performance (Certain Characteristics & attributes)
e.g. size, Speed, utility. etc.

⇒ Robust ⇒ tolerant of failure conditions, misuse unexpected i/p's, side effects,

⇒ Ease of Change:- modularity., Supports modification reuse, addition. of new features.

⇒ Good design & Coding style.
i.e.
clarity in Algo and control flow through Code, modularity., readibility, proper use of indentation.
indentation.
Style of coding.

Developing Embedded Sw ⇒

↳ Specification. & design. (70-80%)
↳ Abstraction ⇒ ability to minimize or eliminate details that are unimportant
↳ focus on essential elements of problem

Embedded S/w

↳ watches, VCR, Cellular phones.
↳ guided missiles, Control Satellites.
↳ Medical instrumtz.

→ does not terminate.
→ Consumes power.
→ takes time.

Requirement of Em. S/w

↳ Reliability
↳ Cost effective
↳ low power consumption
↳ efficient to memory.
↳ timeliness
  ↳ satisfy timing constraints
    ↳ cache schemes
    ↳ speculative Inst. executing
    ↳ Branching.

→ Concurrency :- n/w of variety of sensors an retain control over actuator.

→ livelyness :- WDT

→ Interfaces :- Static Behaviour

→ Heterogeneity :- → Different Computational Styles & implementation technologies.

→ interact with event occuring.
  ↳ irregularly in time.
  ↳ Regular in time.

→ Reactivity → React cont to env. at speed of enviornment.
Safety Critical
Can't fail arbitarily.

Running a Prog.

↳ Run on host Syst.
  ↳ Inst level Sim. — Fast-Slow
  Cycle-accurate Sim — Slow
↳ Run on target Syst.

Model of Programs

↳ Why not use the source code directly?
  ↳ source code → Assembly language, C-code etc.

↳ fundamental model for program is:
  Control/data flow graph (CDFG)
    ↳ both data operation & control operations.

Data flow Graph

↳ model of program with no conditionals.
  (one entry & exit point) ⇒ basic block.

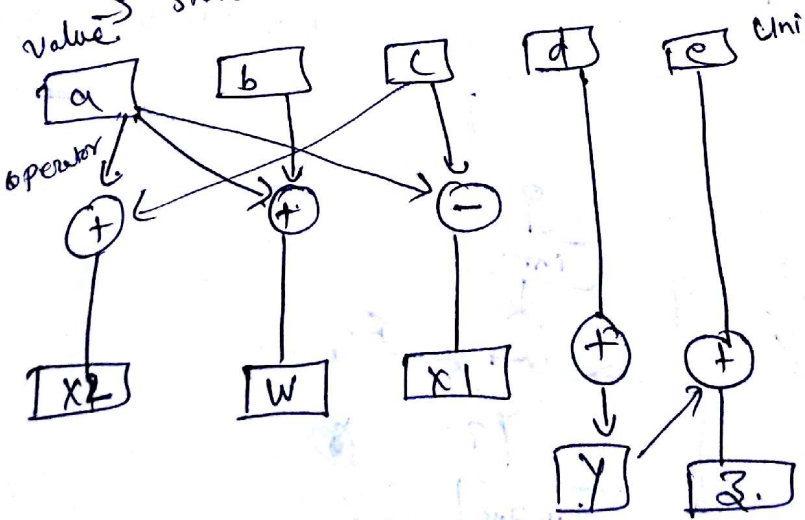
```
w = a+b          |          w = a+b
x = a-c          |          x1 = a-c
y = x+d          |          y = x1+d
x = a+c          |          x2 = a+c
z = y+e          |          z = y+e
```

A basic block in C.

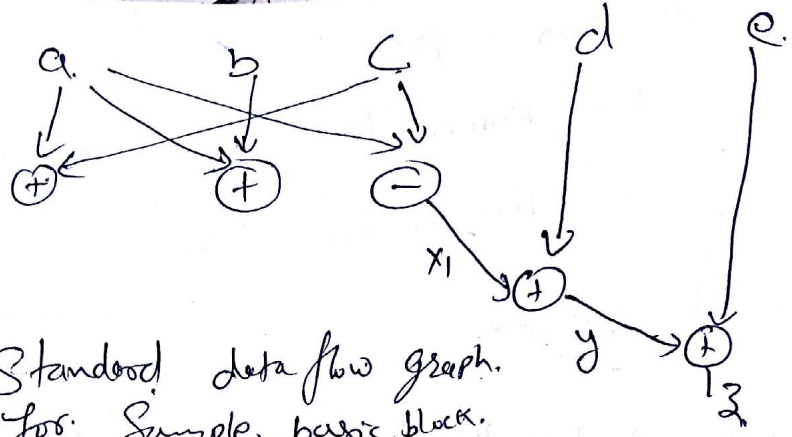
rewrite

Sign-assignment form.
(variable appears once on LHS.
allows to identify unique location in code.

↳ Statement executed sequentially.



An extended data flow graph for our Sample basic block



Standard data flow graph. for Sample. basic block.

CDFG ⇒ uses data flow graph as an element.

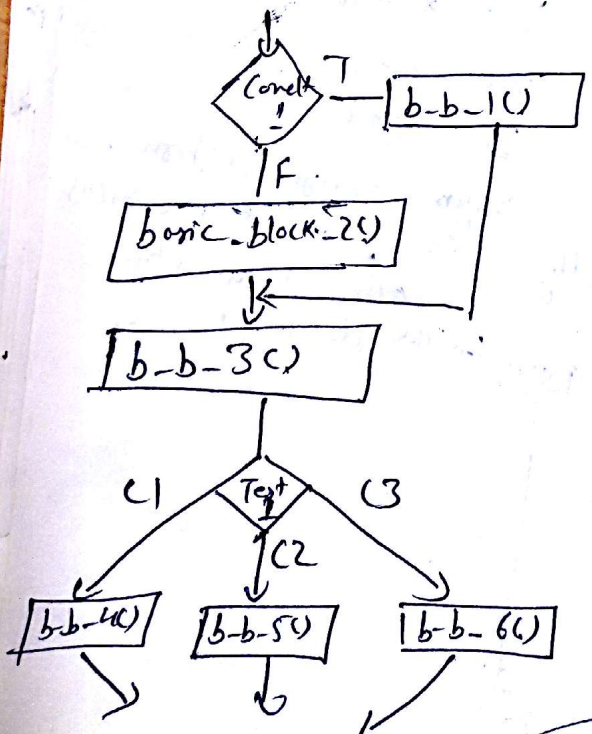
decision nodes : describe all types of Control in prog.

data flow nodes :- encapsulates Complete DFG

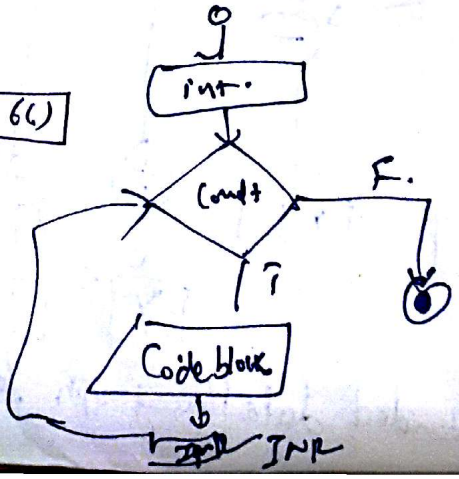
```
for (i=0; i=N; i++) {
    loop-body ();
}
```

⇒

```
if (cond1)
    basic block_1();
else
    basic_block_2();
basic_block-3();
Switch (test1) {
    case c1 : b_b_4(); break;
    case c2 : b-b-5(); break;
    case c3 : b-b_6(); break;
}
```

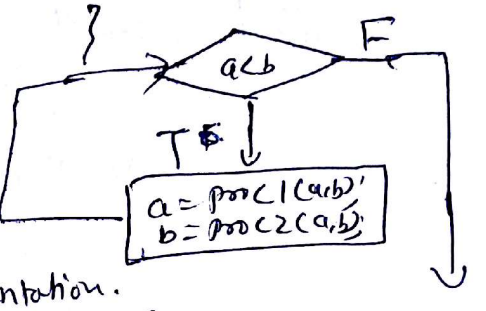


CDFG



```
i = 0
while (i<N) {
  loop body();
  i++;
}
```

```
while (a<b) {
  a = proc1 (a,b);
  b = proc2 (a,b);
}
```



CDFG is hierarchical representation.
↳ a.

CDFG

Prog. lang.

→ Procedural: C.

→ OOP's
↳ reusability.

→ Java: Code mobility. (code accessing multi specification implementation instance.
↳ if JVM supported by OS
↳ n/w application
(no use 3rd party source's.)

Use of C.

↳ Processor dependent Object's like Characters, bytes, bits and addresses can be handled directly.

↳ To be manipulated to control interrupts Controllers, CPU registers etc →

↳ C provides Special Variable types like registers, Volatile, Static & Constant.

Variable Types in C.

Register → Var to be very often used, Compiler places in a register.

Static → Private Comm" in module
↳ stored Globally.

Volatile → useful for handling memory mapped I (don't map them to cache)

Compiler use Volatile KW to avoid putting I Content in Cache. C by generating code in such a

Memory:— when memory is allocated in a large no. of contagious blocks or Chunks, leaving a good % of memory unallocated and unusable: results out of memory allocation errors!

→ weakness
→ Internal fragm

Dynamic
int *a = v
↳ ma
Automat
in
Static
ex
↳
glob
↳
Heap
(mal
Allo
for
array
elements.
call
for

1) Dynamic memory Alloc^n.

int *a = malloc. (Size of (int));

↳ malloc ↳ Calloc.

~~Automatic~~ memory Alloc^n

↳ Stack memory.

int a = 43

Static M.A ⇒ at Compile time.

extern ⇒ so T.U can have access to G.V of 2nd T.U

↳ Variable is defined elsewhere.

Static qualifier

e of link time

Size

most commonly assigned variable's,

ack for variables

Support for allo^n &

access These Variable.

↳ Dynamic. memory Allocation

↳ Allo^n at run time on demand

↳ Allo^n & deallocation under prog- Control.

Heap

(malloc -calloc free)

Allo^n for array elements.

↳ Accommodated in Heap.

T* P = (T*) malloc (Sizeof (T))

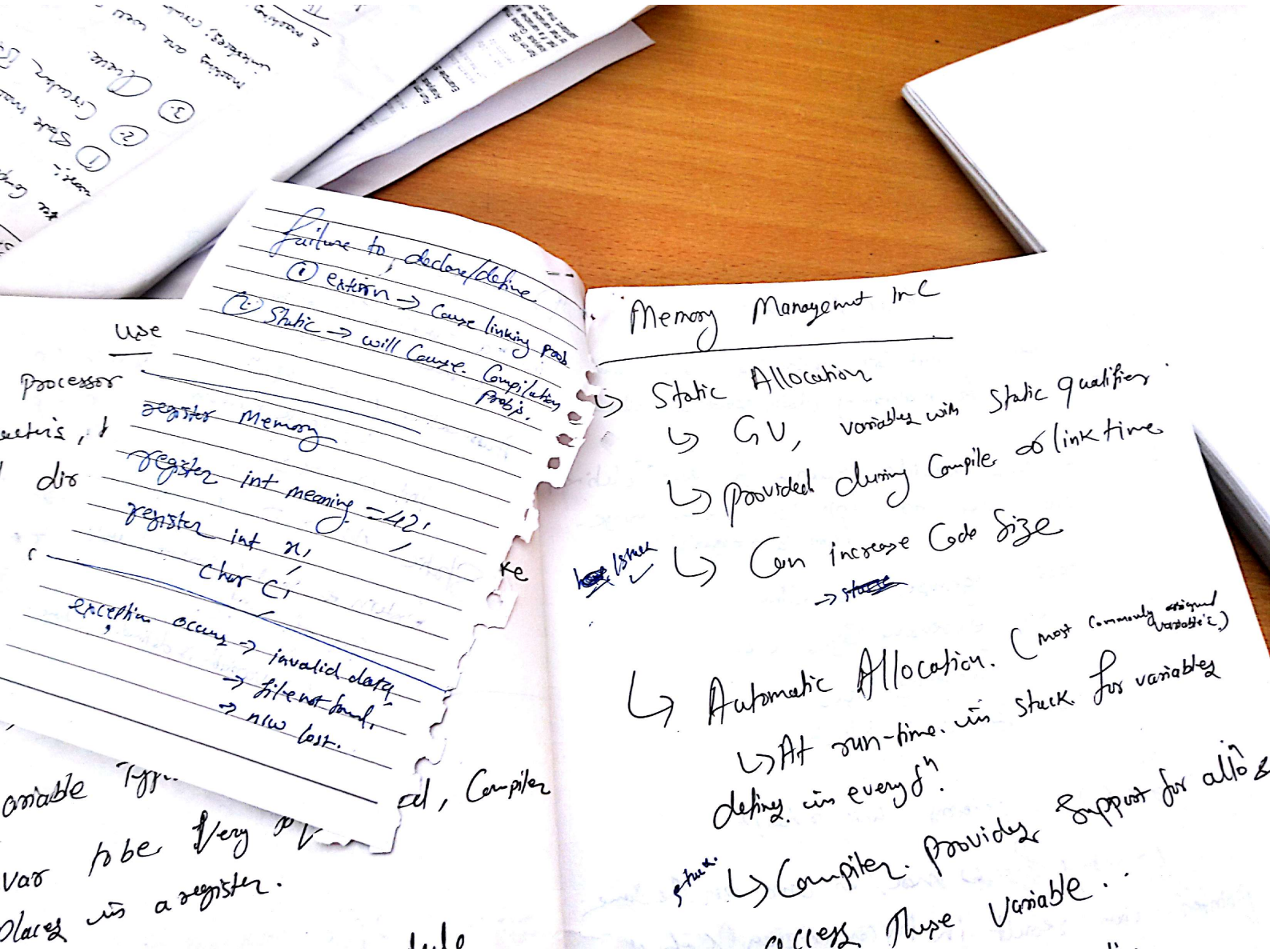
free(P)

→ worst case

→ Internal fragmentation → Allo^n in Heap area.

↳ Chunk of size 16, 32 etc.

↳ actually acquired or released in units in which memory is divided



Stack

→ Computer's memory that stores temp. variables created by f".

→ managed & optimized by CPU

→ every time when f" declares a new variables, it is pushed on to the Stack. when f" exits all variables are popped off.

→ Once freed, region of stack is available for other stack variables.

Very fast

don't have to explicitly de-allocate variables

CPU manage stack memory not become fragmented.

- local variables only

limit on stack size (os-dependent)

Variable can't be resized

Heap

→ Comp memory that is not managed automatically managed for you. (manual) not tightly managed by CPU.

→ free floating region (and is larger)

→ To allocate memory on heap: malloc() or calloc() f"s. use free() to deallocate that memory once you don't need it. else there will be "memory leak"

memory leak ⇒ memo on heap will still be set aside. (& won't be available to other processes).

→ Global variables

→ no size restrictions

→ slower

→ variable can be by any f" any where in prog.

→ Variable's can access globally

→ no limit on memory size

→ slower access (relatively)

→ no guaranteed eff use of space, fragmentation possible

→ manual memory management

→ variables can be resized using realloc().