Code optimization

The basic objective of code optimization is improving the quality of intermediate code such that it occupies less space in the memory and minimize the time taken to execute the program. There are 2 types of code optimization methods -:

1. machine dependent optimization
2. machine independent optimization.

Machine dependent optimization Techniques -:

1. Register Allocation
2. Interleaving of data and instructions
3. Use of special features of the machine. ie machine dependent optimization is performed using machine features.

Machine Independent optimization Techniques

1. Here compiler uses mathematical properties to improve the quality of the intermediate code.
   eg. Use of simple mathematical properties such as identity and null operations to remove useless code.
2. Identification or removal of the common subexpression.
3. Strength reduction.
4. Code motion.
5. Constant folding
6. Deadcode elimination.

eg 1. Identify the null

Before	After
✗ A = A*1	✗
✗ B = B+0	✗
✓ B = B+2	B = B+2

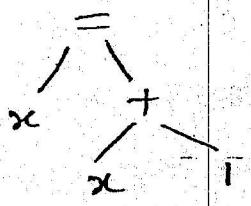
2nd problem

Common Subexpressions - 0.

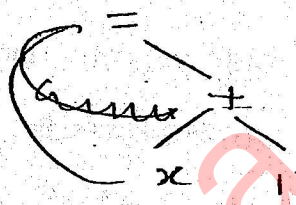
The common subexpression in a given expression identified using directed acyclic graphs. (DAG)

eg. $x = x + 1$

↳ syntax tree

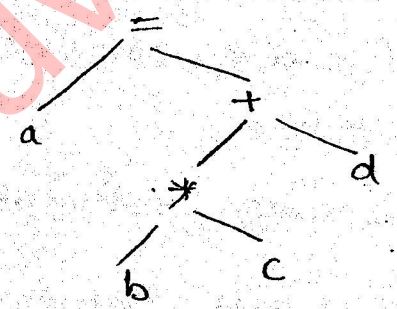


DAG tree



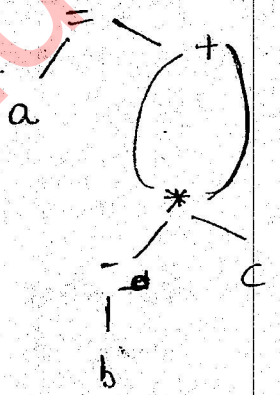
eg. $a = b * c + d$

↳ No common, so DAG (same as syntax directed)

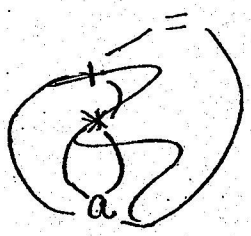


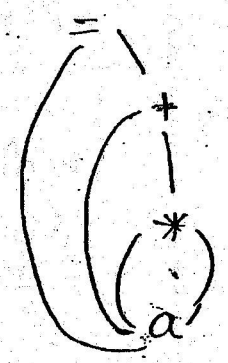
eg. $a = -b * c + -b * c$

↳ DAG



eg. $a = a * a + a$



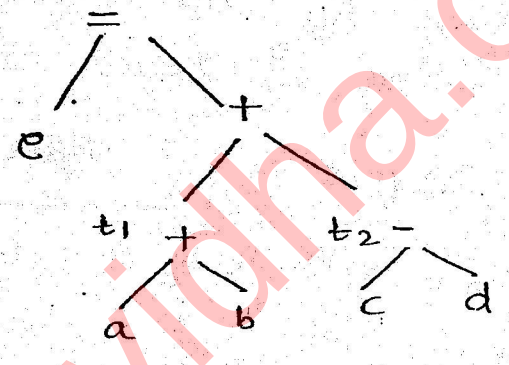


eg. a $t_1 = a + b$

$t_2 = c - d$

$e = t_1 + t_2$

⇒



* Strength reduction

Replacing expensive operator by equivalent and cheapes operator i.e called strength reduction.

	Before	After	Replace
①	$B = C + 2 * x$	$B = C + x + x$	* by +
②	$B = C + x \wedge 2$	$B = C + x * x$	$\wedge$ by *

* Code motion (frequency reduction)

The most targeted statement in the program is loops. because the so called (90 - 10) states that 90% of execution time will be spend on 10% of loop, remaining 90% code consumes 10% time.

eg.

Before	After

while (A = B*C+D)
{
Prinf (Hi);
}
	E = B*C+D
	while (A = E)
	{
	printf ();
	}

* constant folding :- Replacing the arithmetic numerical expression by its value during compilation time itself is called constant folding.

eg.

Before	After
A = B+2.11*3.11	A = B + 6.56

fold the constant

* Deadcode elimination { ~~unreachable~~ Unreachable codes } elimination.

eg.
```
A = 10
if (A < 10)
    printf (Hy);
else
    printf (Bye);
```
} unreachable code or dead code.

* Inside a Loop, following optimization techniques can be used :-
  - code ~~unreachable~~ motion
  - Strength Reduction
  - Constant folding.

* Code Generation :-

The most difficult phase among all phases of compiler is code generation both practically and theoretically.

↳ code generator can produce target code or object code in various forms.

It may be :-

① Absolute machine code → .exe file

② Reallocatable Machine code → .obj file majority of compiler gives this.

③ ~~to~~ Assembly language code.

④ ~~Another~~ High level language code.

If in this, we go from Top to Bottom order, they are very easy to implement, but if we go Bottom to up, it ~~becomes complex~~, execution time can be minimized.

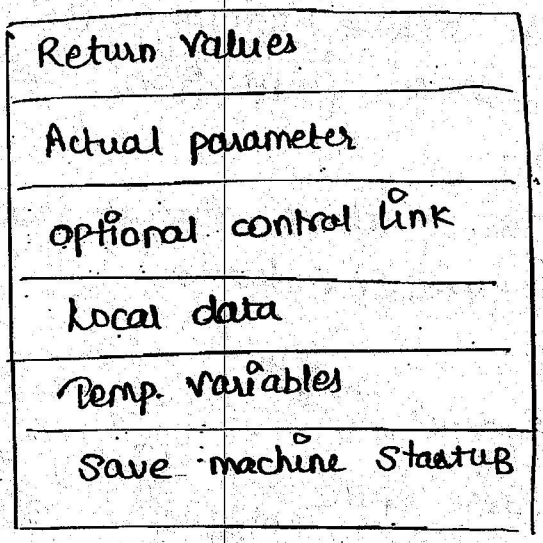
* Run Time Environment :-

The snapshot of the program during execution is called Run Time Environment.

The information needed by single executed procedure is manage using a record structure & framework is called activation record.

It is customary to push the activation record of a procedure on run time stack when procedure is called and to pop the activation record of the stack when control returns to the caller. The general fields in activation record

As shown in the figure -:



* Storage allocation strategy -:
A different storage allocation strategy is used in each of the three data areas in the organisation of typical sub division of run time memory into code and data area.
There are 2 types of allocation strategy of for activation record -:
1. static allocation
2. dynamic allocation

* Static allocation
In static allocation size of data should be known at compile time therefor it is not suitable for recursive procedure based languages.
Static allocation generally uses in scientific programming languages such as FORTRAN, SNOWBALL

* Dynamic allocation.

In dynamic allocation, storage would be provided for activation record during run time. There are 2 types of run time storage allocations :-

(1). Stack allocation.
(2). Heap allocation.

Stack allocation strategy is more suitable for recursive based procedure languages.

whereas heap allocation strategy is suitable for programming languages where dynamic data structures are created.

Q. Consider following lists. Match the list 1 with list 2.

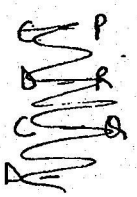
List 1

A. Lexical analysis
B. Code optimization
C. Parsing
D. Intermediate generated code

List 2

P. PDA
Q. DAGs
R. Syntax Trees
S. Regular expressions.

A – S
B – Q
C – P
D – R.



Q. Match the following :- (GATE 2016 Set 2)

A	Lexical analysis	1.	LMD
B	Top down parsing	2.	Typechecking
C	Semantic analysis	3.	FA
D	Run time Environment	4.	Activation Record

Sol.

A - 3
B - 1
C - 2
D - 4

Q. Heap allocation is required for a language (GATE 2010)

(a). that support recursion
(b). that use dynamic scope rules
(c). that support dynamic data structure
(d). None of the above.