Chapter:- 6 Compilers & Interpreter

2011]

* What is an overall procedure for compilation of expression? Briefly explain the use of operand Descriptors & Register Descriptors in expression compilation.

Ans:-

⇒ A toy code generator for Expressions & Intermediate codes for Expressions are an overall procedure for compilation of expression.

* Operand Descriptors:-

⇒ An operand descriptors has the following fields:

1. Attributes: Contains the subfields Type, length and miscellaneous information.

2. Addressability:- Specifies where the operand is located, & how it can be accessed. It has two subfields.

   a. Addressability Code:
      M = operand in Memory.
      R = operand in Register.
      ⇒ addressability Codes.
      e.g.
      AR = Address in Register.
      AM = Address in memory.

(b) Address:
   Address of a CPU register or memory word.

Madhvi Berg.

⇒ An operand descriptor is built for every operand participating in an expression, i.e. for id's constants & partial results.

⇒ A descriptor is built for an id when the id is reduced during parsing.

⇒ A partial result p_i is the result of evaluating some operator op_i. A descriptor is built for p_i immediately after code is generated for operator op_i.

⇒ Exc: The code generated for expression a*b is as follows:

```
MOVER    AREG, A
MULT     AREG, B
```

⇒ Three operand descriptors are used during code generation. Assuming a, b to be integers occupying 1 memory word, these are:

1	(int, 1)	M, addr(a)	Descriptor for a
2	(int, 1)	M, addr(b)	" " b
3	(int, 1)	R, addr(AREG)	" " a*b

Madhuri Bera

* Register Descriptors:

A register descriptor has two fields.

1. status: Contains the code free or occupied to indicate register status.
2. Operand Descriptor #: If status = occupied, this field contains the descriptor # for the operand contained in the register.

⇒ Register descriptors are stored in an array called Register_descriptor. one register descriptor exists for each CPU register.

Ex: The register descriptor for AREG after generating code for a * b as in above Ex. would be.

occupied	#3

⇒ This indicates that register AREG contains the operand described by descriptor #3.

Madhui Bera.

[Jan-2011]

* Explain with example - The role of static & dynamic pointer for accessing local & nonlocal variable in block structured language. [07]

Ans:

* Static pointer:

→ Access to nonlocal variables is implemented using the second reserved pointer in AR. Which has the address 1(ARB), is called the static pointer.

→ When an AR is created for a block b, its static pointer is set to point to the AR of the static ancestor of b.

→ The code to access a nonlocal variable nl-var declared in a level m ancestor of b-use, $m \geq 1$ is as follows:

1. r1 = ARB; where r is some register.
2. Repeat step-3 m times.
3. r1 = 1(r1); i.e. load the static pointer into register r.
4. Access nl-var using the address <r1> + dnl_var.

* Dynamic pointer:

→ The first reserved pointer in a block's AR points to the activation record of its dynamic parent. This is called the dynamic pointer & has the address 0(ARB).

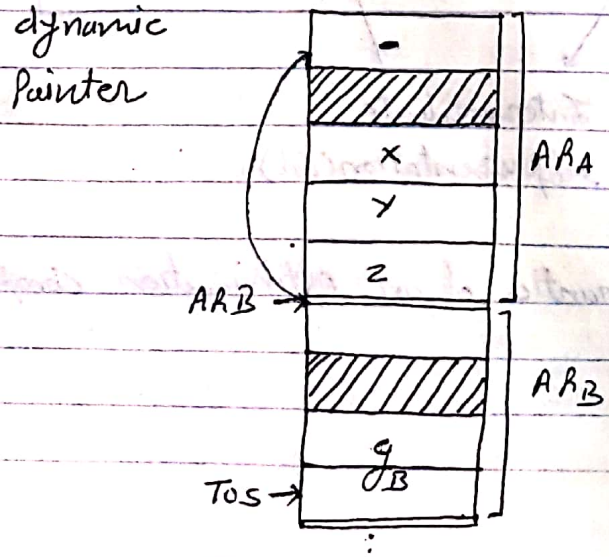
Madhu Bera

2) The AR dynamic pointer is used for deallocating an AR.

* Example:- Consider the block-structured program. The variables accessible within the various block as follows:

```
A [ x, y, z : integer;
    B [ g : real;
        C [ h, z : real; ]
      j, I : integer; ]
```

Block	Accessible Variables	
	local	nonlocal
A	x_A, y_A, z_A	–
B	g_B	x_A, y_A, z_A
C	h_C, z_C	x_A, y_A, g_B
D	j_D, I_D	x_A, y_A, z_A



Madhvi Beri

[Jan-2011] * Write a shorte note on "Code optimization" [07]

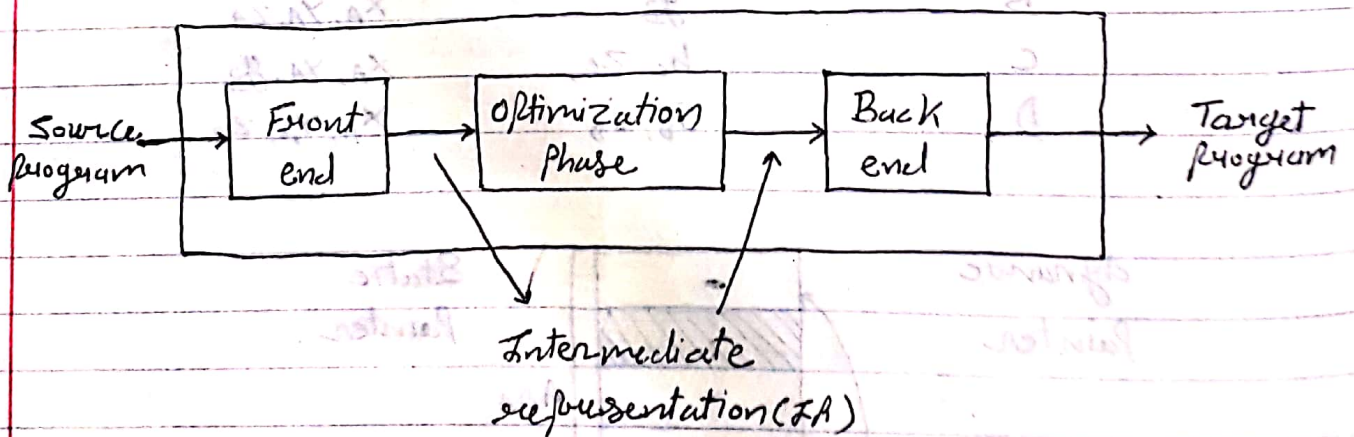
Ans:-

⇒ Code optimization aims at improving the execution efficiency of a program. This is achieved in two ways:

1. Redundancies in a program are eliminated.
2. Computations in a program are rearranged or rewritten to make it execute efficiently.

⇒ The optimization techniques are thus independent of both the PL & the target machine.

⇒ Following fig. contains a schemantic of an optimizing compiler.



Schemantic of an optimization compiler.

→ The compiler was found to consume 40 percent extra compilation time due to optimization. The optimized program occupied 25 percent less storage and executed three times as fast as the un-optimized program.

Optimizing Transformations:-

→ An optimizing transformation is a rule for re-writing a segment of a program to improve its

→ A few optimizing transformations commonly used in compilers are discussed below.

Compile time evaluation:-

→ Execution efficiency can be improved by performing certain actions specified in a program during compilation itself.

→ When all operands in a operation are constants, the operation can be performed at compilation time.

→ The result of the operation, also a constant, can replace the original evaluation in the program.

→ Thus, an assignment a := 3.14157/2 can be replaced by a := 1.570785, thereby eliminating a division operation.

Madhvi Beri

★ Elimination of Common subexpression :-

1) Common sub expressions are occurrences of expressions yielding the same value.

2) Exc:

```
a := b*c                t := b*c;
---                 ⇒   a := t;
x := b*c+5.2;           ...
                        x := t+5.2;
```

3) In above contains the two occurrences of b*c. The second occurrence of b*c can be eliminated bec'z the first occurrence of b*c is always evaluated before the second occurrence is reached during execution of the program. The value computed at the first occurrence is saved int in t. This value is used in the assignment to x.

★ Dead Code elimination:

1) Code which can be omitted from a program without affecting its results is called dead code.

2) Dead code is dedecated by checking whether the value assigned in an assignment statement is used anywhere in the program.

Madhu Babu.

★ Frequency Reduction:

⇒ Execution time of a program can be reduced by moving code from a part of a program which is executed very frequently to another part of the program which is executed fewer times.

⇒ For Example; the transformation of loop optimization move loop invariant code out of a loop & place it prior to loop entry.

⇒ Ex:-

```
for i:=1 to 100 do
begin
    z:=j;
    x:=25*a;
    y:=x+z;
end;
```

⇒

```
x=25*a;
for i:=1 to 100 do
begin
    z:=j;
    y:=x+z;
end;
```

★ Strength Reduction:

⇒ The strength reduction optimization replaces the occurence of a time consuming operation by an occurance of a transfer operation e.g. replacement of a multiplication by an addition.

Madhui Bera.

* Local & Global optimization :-

2) Local optimization :- The optimizing transformations are applied over small segments of a program consisting of a few statements.

2) Global optimization :- The optimizing transformations are applied over a program unit i.e. over a function or a procedure.

[June-2011]

* Define optimizing transformations. Explain with suitable example. [07]

Ans :-

Note :- Read answer from previous question.

[June-2011]

* Explain allocation & access of local and non-local variable for a block structure lang. with below given code.

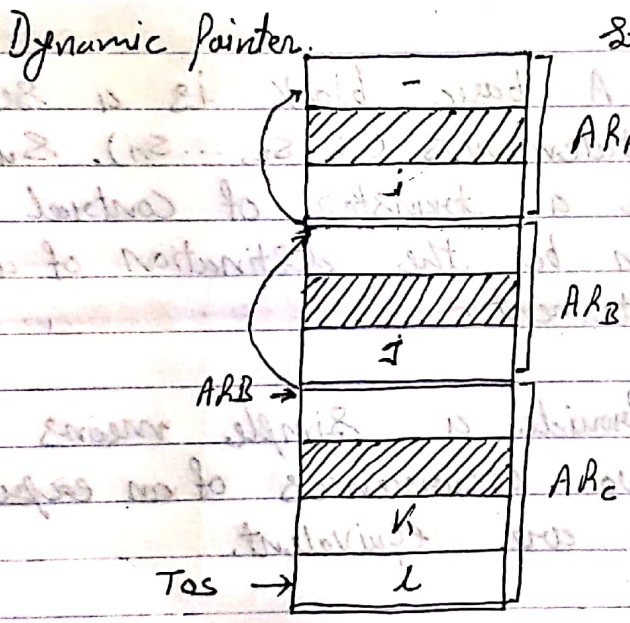
```
Void main()
{
    int i = 5;
    {
        int i = 7;
A       {
B           c{ int K = 10, L = 15;
               K = i;
            }
        }
    }
}
```

Madhuri Bera.

Ans:-

Block	Accessible Variables	
	local	Non-local
A	i_A	–
B	i_B	i_A
C	k_C, l_C	i_A, i_B

block Structure program.



↳ The 1st reserved pointer in block's AR points to the activation record of its dynamic parent. This is called dynamic pointer.

↳ Access to nonlocal variables is implemented using the second reserved pointer in AR. This is called the static pointer. which has the add. I(ARB).

Madhui Berg

[Dec. 2011]

⇒ Define local optimization & Basic block. Explain with suitable example. [07]

Ans:

2) Local optimization:

The optimizing transformations are applied over small segments of program consisting of a few statements.

2) Basic Block:

A basic block is a sequence of a program statements ($S_1, S_2, \ldots S_n$). Such that only S_n can be a transfer of control statement and only S_1 can be the destination of a transfer of control statement.

2) Value number provide a simple means to determine if two occurrences of an expression in a basic block are equivalent.

2) The value numbering technique is applied on the fly while identifying basic blocks in a source program.

2) Example: The Symbol table & the quadruples table showing local optimization.

Madhu Bera.

- Symbol Table.

Symbol	...	Value number.
y		0
x		15
g		14
z		0
d		5
w		0

- Quadruples Table.

	operator	operand-1		operand-2		Result	use
		operand	Value no.	operand	Value no.	name	flag
20	:=	g	–	25.2	–	t_{20}	f
21	+	z	0	2	–	t_{21}	f
22	:=	x	0	t_{21}	–	t_{22}	f
23	*	x	15	y	0	t_{23}	f
24	+	t_{23}	–	d	5	t_{24}	f
⋮							
57	:=	w	0	t_{23}	–	t_{57}	f

Stmt no.	Statement.
14	g := 25.2;
15	x := z+2;
16	h := x*y+d;
...	...
34	w := x*y;

Madhvi Bera.

2) Local optimization proceeds as follows:

- All variables are assumed to have the value numbers '0' to start with.
- Processing of statements 14 & 15 leads to generation of quadruples numbered 20-22 shown in the table.
- Value no. of y, or & y at this stage are 14, 15 & 0. respectively.
- Quadruple for x*y is generated next, & the value no: of x & y are copied from the symbol table.
- This quadruple is assigned the result name t_{23}, & its save flag is set to false.
- This quadruple is entered in entry no. 23 of the quadruple table. This process is continous up to end of the program.

Madhu Bera

[Dec-2011]

* Write short note on Parameter passing mechanism. [07M]

Ans:
- Language rules for parameter passing define the semantics of parameter usage inside a function.

- Call by Value.

- Values of actual parameters are passed to the called functions. These values are assigned to the corresponding formal parameters.

- Call by value is commonly used for built-in functions of the language.

- A called function may allocate memory to a formal parameter & copy the value of the actual parameter into this location at every call.

- Call by Value - result:-

- This mechanism extends the capabilities of the call by value mechanism by coping the values of formal parameters back into corresponding actual parameters at return.

- Side effects are realized at return. It is inherits the simplicity of the call by value mechanism but ~~into~~ incurs higher overheads.

Madhu Beig.

- Call by references:

↳ The address of an actual parameter is passed to the called functions.

↳ If the parameter is an expression, its value is computed and stored in a temporary location & the address of the temporary location is passed to the called function.

↳ It is very popular because it has 'cleaner' semantics than call by value-result.

- Call by name:

↳ This parameter transmission mechanism has the same effect as if every occurence of a formal parameter in the body of the called function is replaced by the name of the corresponding actual parameter.

Dec-2011]

* Write short note on - Control flow analysis [03]

Ans: ↳ Control flow analysis analyses a program to collect information concerning its structure e.g. presence & nesting of loops in the program.

↳ Information concerning program structure is used to answer specific questions of interest

↳ The control flow concepts of interest are:

1. Predecessors & successors:
If $(b_i, b_j) \in E$, b_i is a predecessors of b_j & b_j is a successors of b_i.

2. Paths:
A path is a sequence of edges such that the destination node of one edge is the source node of the following edge.

3. Ancestors & descendants:
If a path exists from b_i to b_j, b_i is an ancestor of b_j & b_j is a descendant of b_i.

4. Dominators & post-dominators:
Block b_i is a dominator of block b_j if every path from n_0

to b_j passes through b_i, b_i is a post-dominator of b_j if every path from b_j to an exit node passes through b_i.

↳ Control flow concepts can be used to answer certain questions in a straight forward manner.

* Write short note on Triples & Quadruples with suitable example. [07]

Ans:-

* Triples:

↳ A triples is a representation of an elementary operation in the form of a ~~fsed~~ pseudo machine instruction.

	operator	operand 1	operand 2

↳ Triples are numbered in some convenient manner. Each operand of a triple is either a variable / constant or the result of some evaluation represented by another triple.

↳ A program representation called indirect triples is useful in optimizing compilers.

↳ In this representation, a table is built to contain all distinct triples in the program.

Madhvi Berg-

2) A program statement is represented as a list of triple numbers. This arrangement is useful to detect the occurrences of identical expressions in a program.

2) The indirect triples representation provides memory economy.

2) Ex:

$$Z := a^{2} + b^{1} * c^{5} + d^{4} * e \uparrow^{3} f;$$
$$Y := x^{6} + b^{1} * c;$$

	operator	operand 1	operand 2
1	*	b	c
2	+	(1)	a
3	↑	e	f
4	*	d	(3)
5	+	(2)	(4)
6	+	x	(1)

triple's table.

	stmt no.	triples no.
1	1	1, 2, 3, 4, 5
2	2	1, 6

statement table.

Madhu Bera.

* Quadruple :

2) A quadruple represents an elementary evaluation in the following format :

operator	operand 1	operand 2	result name.

2) Here, result name designates the result of the evaluation. It can be used as the operand of another quadruple.

Exc :-

ex $z := a^{2} + b^{1} * c^{5} + d^{4} * e^{3} \uparrow f;$

operator	operand 1	operand 2	result name.
*	b	c	t1
+	t1	a	t2
↑	e	f	t3
*	d	t3	t4
+	t2	t4	t5

Quadruple's table.

Madhuri Bera.

* Explain Data flow Concept - Available Expressions with suitable example. [07]

Ans:-

→ Data flow analysis techniques analyze the use of data in a program to collect information for the purpose of optimization. This information, called data flow information, is computed at the entry & exit of each basic block in a program.

→ It is used to decide whether an optimizing transformation can be applied to a segment of code in the program.

→ Data flow Concepts :-

Data flow Concept	optimization in which used.
Available expression	- Common subexpression elimination.
Live Variable	- Dead code elimination.
Reaching definition	- Constant & Variable propagation.

1. Available Expressions :-

→ The transformation of global common sub expression elimination can be defined as follows :

Madhuri Bera.

2) Consider subexpression x*y occuring at program point P_i in basic block b_i. This occurrence can be eliminated if

i) Conditions of global optimization are satisfied at entry to b_i.

ii) No assignments to x or y precede the occurrence of x * y in b_i.

2) The data flow concept of available expressions is used to implement common sub expression elimination.

2) The availability of an expression at entry or exit of basic block b_i is computed using the following rules:

i) Expression e is available at the exit of b_i if

   i) b_i contains an evaluation of e which is not followed by assignments to any operands of e, or

   ii) Expression e is available at entry to b_i if it is available at the exit of each predecessor of b_i in Grp.

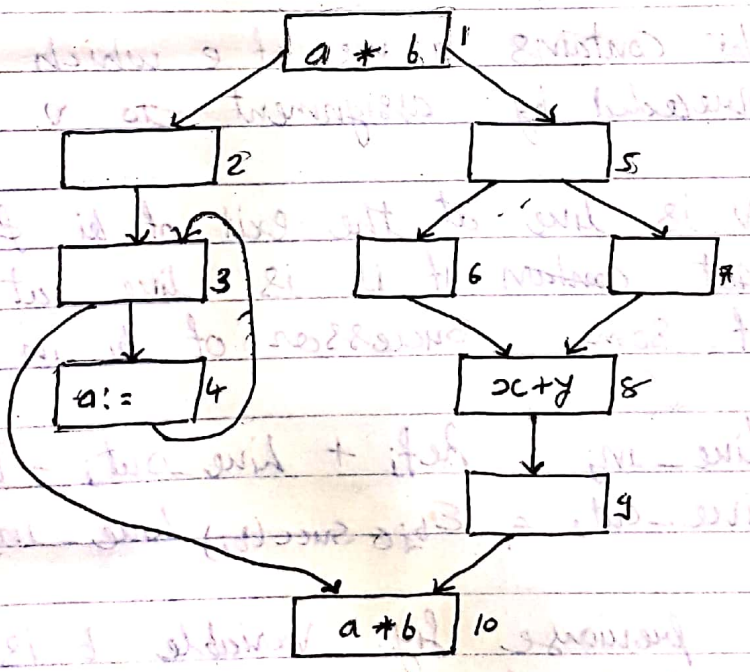
Madhu Bera.

⇒ Available expressions is termed a forward data flow concept because availability at the exit of a node determines availability at the entry of its successors.

⇒ Example: Available expression analysis for the PFG,

a*b : Avail_in = true. for blocks 2, 5, 6, 7, 8, 9
Avail_out = true. for blocks 1, 2, 5, 6, 7, 8, 9, 10

x+y : Avail_in = true. for blocks 6, 7, 8, 9
Avail_out = true for blocks 5, 6, 7, 8, 9.



II Live Variables:

2> A Variable Var is said to be live at a program point p_j in basic block b_i if the value contained in it at p_j is likely to be used during subsequent execution of the program.

2> The liveness property of a variable can be determined as follows:

1. Variable v is live at the entry of b_i if

   i. b_i contains a use of e which is not preceded by assignment to v. or

   ii. v is live at the exit of b_i & b_i does not contain if it is live at the entry of some successor of b_i in cp.

$$Live\_in_j = Ref_j + Live\_out_j - Def_j$$
$$Live\_out_j = \varepsilon_{b_i \in Succ(b_i)} Live\_in_j$$

2> Ex: In previous fig. Variable b is live at the entry of all blocks, a is live at the entry of all bo blocks excepting block 4 & variables x, y are live at the entry of blocks 1, 5, 6, 7 & 8.

Madhu Bera.

[Dec. 2011]

* Explain static and dynamic memory allocation in block structured programming language. [03]

Ans:

Note: Read the answer from page No :- 4. (Jan - 2011 Que. Paper)

[May-2012]

* Define the control structure.

Ans:

The control structure of a language is the collection of language features for altering the flow of control during the execution of a program.

[May-2012]

* What is the format of Triples?

Ans:

operator	operand 1	operand 2

[May-2012]

* Explain the design and operation of an interpreter. [07]

Ans:

2) The interpreter is itself a Pascal program, which is compiled by a Pascal programx compiler. The data store of the interpreter consists of two large arrays named rvar & ivar which are used to store real & integer values respectively.

2) Last few locations in the rvar & ivar arrays are used as stacks for expression evaluation with the help of the pointers r_tos & i_tos respectively.

Madhu Bala

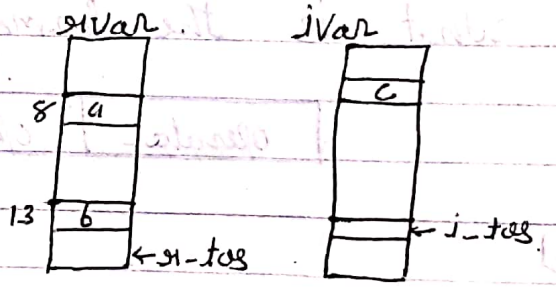
⇒ The interpreter program contains a set of data manipulation routines for use in expression evaluation.

⇒ Example: Consider the basic program.

```
real a, b
integer c
let c = 7.
let b = 1.2
a = b + c.
```

⇒ Soln:

Symbol	type	add.
a	real	8
b	real	13
c	int	5



⇒ Interpretation of a = b+c proceeds as follows: The interpreter procedure add is called with the symbols table entries of b & c. add analyses the type of b & c and decides that procedure addrealint will have to be called to realize the addition.

⇒ addrealint now executes a single Pascal stmt which is equivalent to

```
rVar[r-tos] := rVar[13] + iVar[5];
```

Madhuri Beru