Compiler Design

Today's Class Topics

-Basic Block
-Control flow graph(CFG)
-Code optimization
-Machine Independent optimization
-Machine dependent optimization
-Liveness analysis
-Run time environment

Compiler Design

Basic Block: Basic block is a set of statements that always executes in a sequence one after the other.

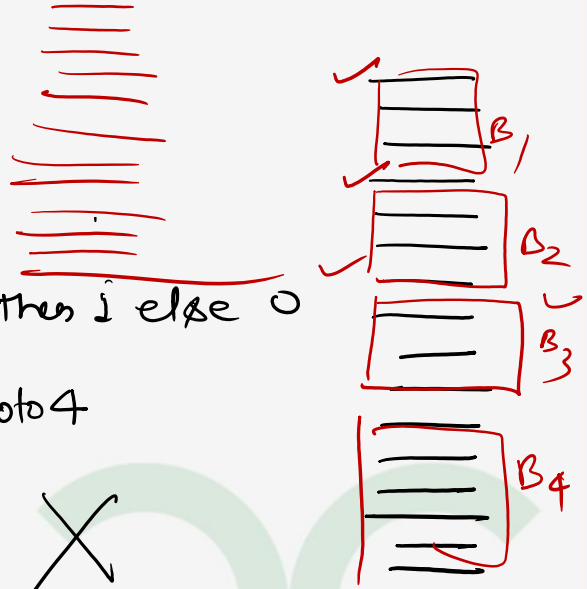
The characteristics of basic blocks are-

-They do not contain any kind of jump statements in them.
-There is no possibility of branching or getting halt in the middle.
-All the statements execute in the same order they appear.
-They do not lose the flow control of the program.

Ex:- $a = \overbrace{b + \overbrace{c * d}^{t_1}}^{t_2}$

$t_1 = c * d$
$t_2 = b + t_1$
$a = t_2$

Basic Block.



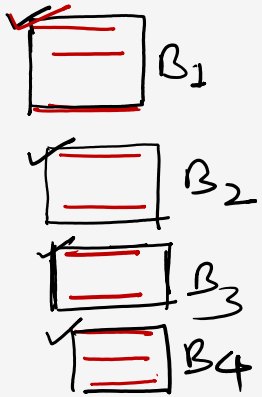
Ex: if A < B then 1 else 0

1) if A < B goto 4
2) $t_1 = 0$
3) goto 5
4) $t_1 = 1$
5) ——

Compiler Design

How to find the Basic Blocks

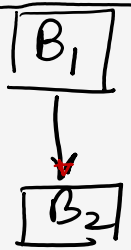
1. First, find the set of leaders from intermediate code, the first statements of basic blocks. The following are the steps for finding leaders:
   1. The first instruction of the three-address code is a leader.
   2. Instructions that are the target of conditional/unconditional goto are leaders.
   3. Instructions that immediately follow any conditional/unconditional goto/jump statements are leaders.
2. For each leader found, its basic block contains itself and all instructions up to the next leader.



Compiler Design

Control Flow Graph(CFG): used in code optimization & code generation

- A CFG gives a flow information between the basic blocks
  Each Basic block is a node in CFG
- A CFG is a directed graph between the basic blocks
  There will be a directed edge from basic block B_1 to the Basic Block B_2 if B_2 comes immediately after B_1

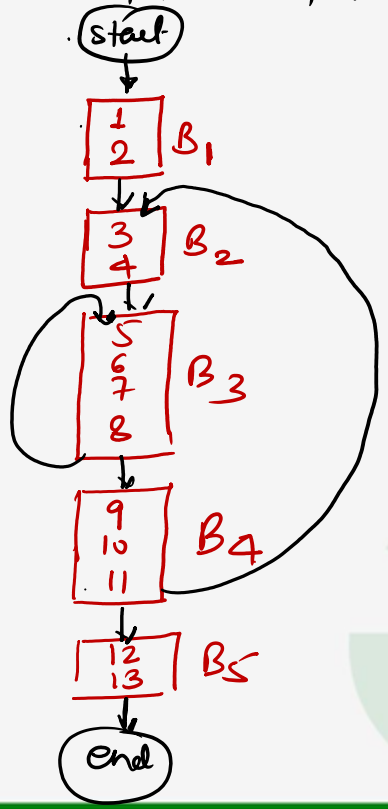


Compiler Design

Question: Find the No. of nodes & edges in the CFG for the following code segment

```
1  a = 1          B1
2  i = 0
3  t1 = a+5       B2
4  t2 = x*a
5  t3 = 4*b       B3
6  a = a+t3
7  i = i+1
8  if i<t3 goto5
9  J = J+1        B4
10 K = K+1
11 if J<K goto3
12 i = J+K        B5
13 K = 0
```

CFG:



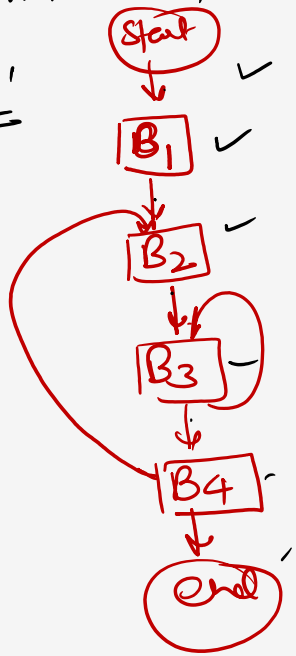
7 nodes
8 Edges
Any

Compiler Design

Question:- Consider the following intermediate code given below. The no. of nodes & edges in the Control flow graph constructed for the code respectively are ?

a) 5 and 7
b) 6 and 7
c) 5 and 5
d) 7 and 8

CFG:

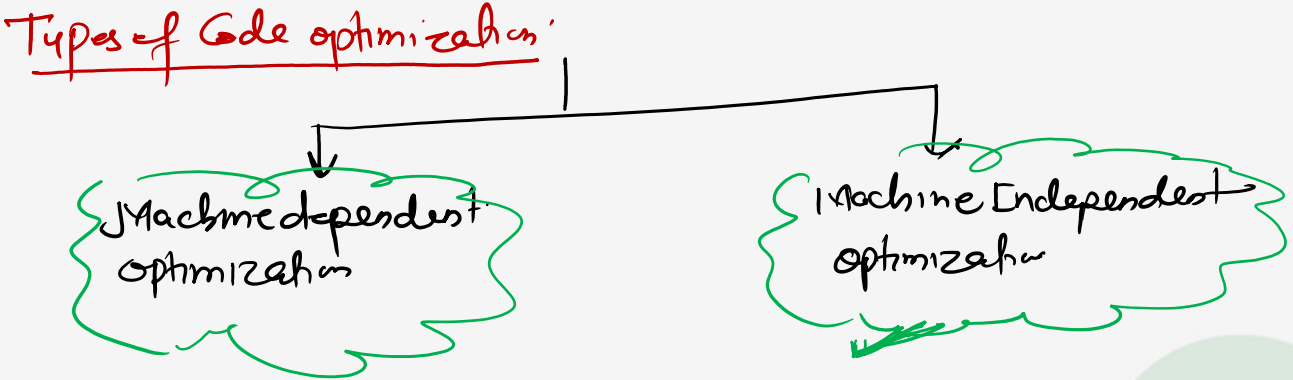



```
1)  i = 1                    B1
2)  J = 1                    B2
3)  t1 = 5 * i               B3
4)  t2 = t1 + J
5)  t3 = 4 * t2
6)  t4 = t3
7)  a[t4] = -1
8)  J = J + 1
9)  if (J <= 5) goto 3
10) i = i + 1                B4
11) if (i < 5) goto 2
```

Compiler Design

Code optimization: Code optimization is a process of reducing the number of instructions without affecting the outcome of the source program.

Code optimization means improvement of the program, it increase the speed and performance of the programs it reduce the space consumed and reduces the time required to execute the program



Compiler Design

Compiler Design

Machine independent optimization Technique:-

① Common Subexpression elimination:-

$a = b + c$
$t_1 = a - b$
$t_2 = b + c = a$
$b = b - a$
$t_3 = a - b$
$c = a + b$

$\Rightarrow$

$a = b + c$
$t_1 = a - b$
$t_2 = a$
$b = b - a$
$t_3 = a - b$
$c = a + b$

✓

Compiler Design

② Constant folding :-

$a = 3 * 5 \Rightarrow a = 15$

$b = 2 + 10 \Rightarrow b = 12$

③ Copy Propogation :-

$a = b$
$c = a$
$d = c$
$e = d + e$

$\Rightarrow$

$a = b$
$c = b$
$d = b$
$e = b + e$

④ Constant Propogation :-

$a = 5$
$b = a + 3$
$c = b + 6$

$\Rightarrow$

$a = 5$
$b = 5 + 3$
$c = b + 6$

$\Rightarrow$

$a = 5$
$b = 15$
$c = 21$

Compiler Design

⑤ Deadcode elimination:-

```
i=0
if(i>10)
{
  ---
  ---
}
x=x+y
```

⇒

```
i=0
x=x+y
```

✓

⑥ Loop invariant computation:-

Ex: `i=0;` `n=100;` `a=1, b=2, c=3` `x=10, y=20`

```
for(i=0; i<=n; i++)
{ x=x+y;
  a=b+c;
}
```

⇒

```
for(i=0; i<=100; i++)
{ x=x+y;
}
a=b+c;
```

✓

Compiler Design

(7) Loop fusion / Loop jamming :-

```
for(i=1; i<=100; i++)
{ a=a+b; }
for(i=1; i<=100; i++)
{ x=y*i; }
```

⇒

```
for(i=1; i<=100; i++)
{ a=a+b;
  x=y*i; }
```

(8) Loop Unrolling :-

```
i=1
while(i<=100)
{ x=x+5;
  i=i+1; }
```

⇒

```
i=1
while(i<=100)        (100 → 50)
{ x=x+5;
  i=i+1;
  x=x+5;
  i=i+1; }
```

Compiler Design

⑨ Induction variable elimination :-

```
i=0; J=0;
while(i≤100)
{ x=y+J;
  J=J+3;
  i=i+1;
}
```

⇒

```
i=0;
J=0;
while(J≤300)
{
  x=y+J;
  J=J+3;
}
```

⑩ Reduction in Strength :-

$2*x = x+x$

$x^2 = x*x$

Multiplication :- Left Right operator

Right shift $a>>b = \frac{a}{2^b}$

$a>>4 \Rightarrow \frac{a}{2^4} = \frac{a}{16}$

$a<<b = a*2^b$

$a<<3 \Rightarrow a*2^3 = a*8$

Ex:

```
Count=0
for(i=1; i≤100; i++)
{Count=i*7;
}
```

Count = 7, 14, 21, 28 ...

⇒

```
Count=0
for(i=1; i≤100; i++)
{
  Count=Count+7;
}
```

Count = 7, 14, 21, 28 ...

Compiler Design

Machine dependent optimization :-

⇒ Peephole optimization Technique :-

(1) Redundant load & store optimization :-

load R_1, A	load R_1, X	store X, R_1
store A, R_1	load R_1, X	load R_1, X

(2) Constant folding : ✓

(3) Strength reduction : ✓

(4) Elimination of Null sequence :-

$a + 0 \Rightarrow a$
$a * 1 \Rightarrow a$
$a / 1 \Rightarrow a$
$a - 0 \Rightarrow a$

Compiler Design

(5) Dead code elimination:

(6) use of machine idioms:

```
MOV R1, X
MOV R2, 1
Add R1, R2
MOV X, R1
```

$R_1 = X$
$R_2 = 1$
$R_1 = R_1 + R_2 = X + 1$
$X = X + 1$

Incr(X)

(7) flow control optimization:

```
if (a>b) goto L1
L1: goto L2
L2: goto L3
L3 goto L4
L4: a=a+10
```

=>

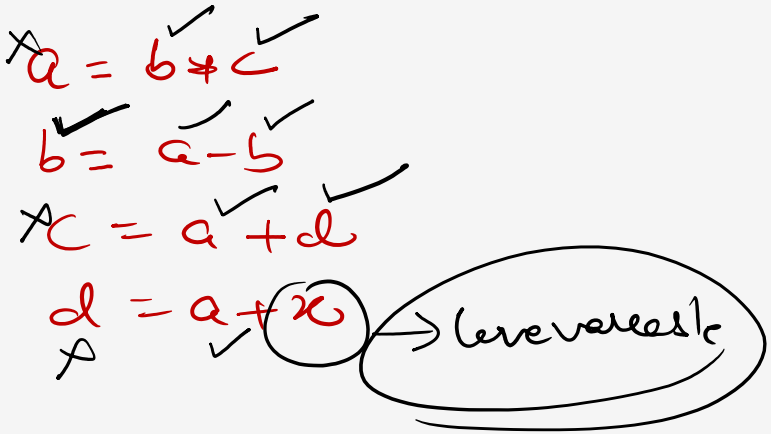
```
if (a>b) goto L4
L4: a=a+10
```

Compiler Design

Compiler Design

Liveness analysis: In a computer the number of registers are limited and a program may contain Hues number of variable whenever we execute the program we allocate the registers to the variables those are in live
Those variables are dead they will be move back from register to the main memory, Moving a variables from the registers to the main memory is called as memory **spilling.**

A variable x is said to be live at any statement if it is used in the statement or any where in the subsequent program before it is defined



Compiler Design

Compiler Design

Application of Liveness analysis:

- Register allocation
- Dead code elimination

Compiler Design

1) $a = b - c$
2) $c = a + b$
3) $b = d - a$
4) $a = c + b$
5) $c = d - b$
6) $a = a + c$
7) $d = a + 10$

	a	b	c	d
1	X	✓	✓	✓
2	✓	✓	X	✓
3	✓	X	✓	✓
4	X	✓	✓	✓
5	✓	✓	X	✓
6	✓	X	✓	X
7	✓	X	X	X

The No. of statements where all variable live = 0

⇒ The No of statements where atleast 3 variables live = 5 Ans

Compiler Design

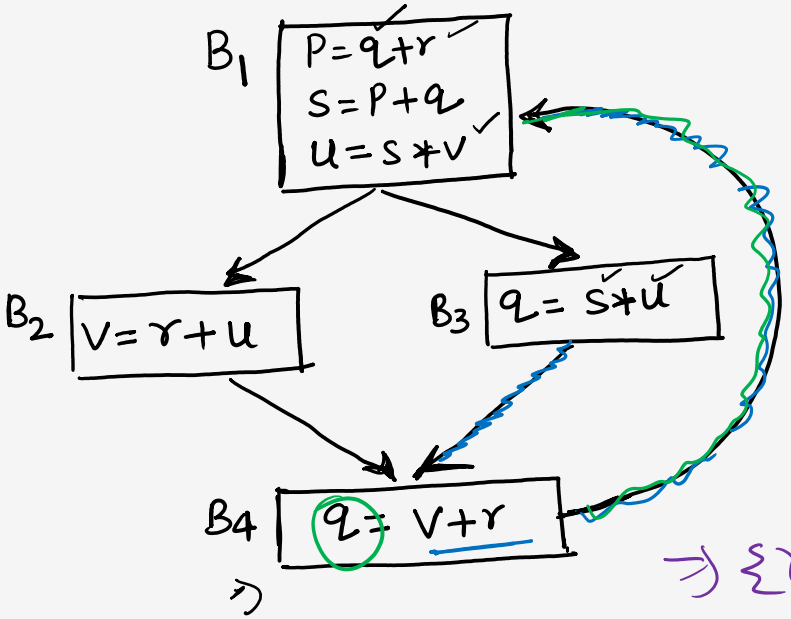
Question:-

1) a = 10
2) b = a * 5
3) c = a + b
4) d = c % 5
5) e = a + b - c
6) a = b + c + d
7) if (a > b) goto 5
8) return (e * a)

	a	b	c	d	e
1	X	X	X	X	X
2	✓	X	X	X	X
3	✓	✓	X	X	X
4	✓	✓	✓	X	X
5	✓	✓	✓	✓	X
6	X	✓	✓	✓	✓
⇒ 7	✓	✓	✓	✓	✓
8	✓	X	X	X	✓

Compiler Design

Question:- Find the variables that are live in both blocks B_2 & B_3 Consider the following Control flow Graph



	p	q	r	s	u	v
1	X	✓	✓	X	X	✓
2	X	X	✓	X	✓	X
3	X	X	✓	✓	✓	✓
4	X	X	✓	X	X	✓

⇒ {r, u} Ans

Compiler Design

Break 15 mins | Class: 2:20 pm

Question:- Assume that all operations take their operands from registers the mini. no. of registers required to execute the following code without memory spilling

1) a = 1
2) b = 2
3) c = 3
4) d = a + b
5) e = c + d
6) f = c + e
7) b = c + e
8) e = b + f
9) d = 5 + e
10) return (d + b)

min No of Registers = 3 Ans

	a	b	c	d	e	f
1	X	X	X	X	X	X
2	✓	X	X	X	X	X
3	✓	✓	X	X	X	X
4	✓	✓	✓	X	X	X
5	X	X	✓	✓	X	X
6	X	X	✓	X	✓	X
7	X	X	✓	X	✓	✓
8	X	✓	X	X	X	✓
9	X	✓	X	X	✓	X
10	X	✓	X	✓	X	X

minimum Registers required = max Variable live at a time = 3

Compiler Design

Run time Environment :- SL → IR → m/c code → Run time Environment

Whatever a necessary support required by O.S. to run a program is runtime environment, Runtime environment deal with the issues

1) Linkage among Procedure or function
2) Parameter Passing
3) Interface to OS.
4) Mechanism of accessing Variables
5) Allocation of Storage etc.

Compiler Design

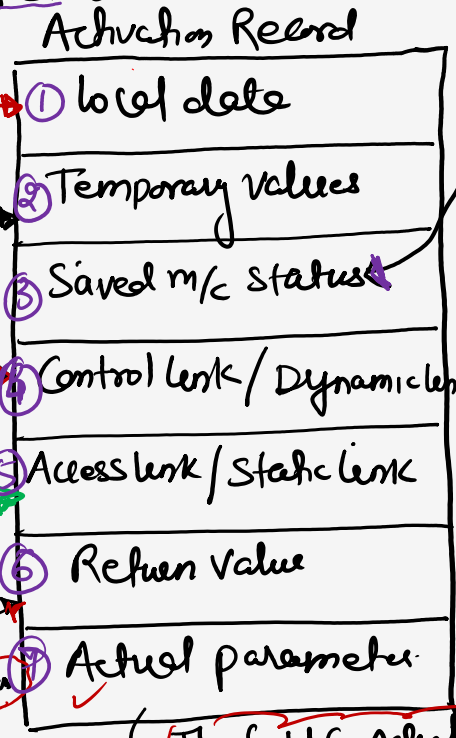
Program is a set of Procedure, whenever a Procedure is in execution its activation record will be created in STACK called Control STACK

Ex:-

```
fact(int n)
{
if(n<=1)
   return 1;
else
 return n*fact(n-1);
}
```

Compiler Design

Activation Records:- has 7-fields



Holds the data that is local to the execution of the function

Stores the values that arises in the execution of an expression

Stores the Address of Activation record of the caller function

Store the information of data which is outside the local scope (i.e. data of calling function)

Used by the called procedure to return a value to calling function

Stores m/c status such as register, program Counter etc. before the procedure is called Holds the information about status of machine just before the function call

Ex:

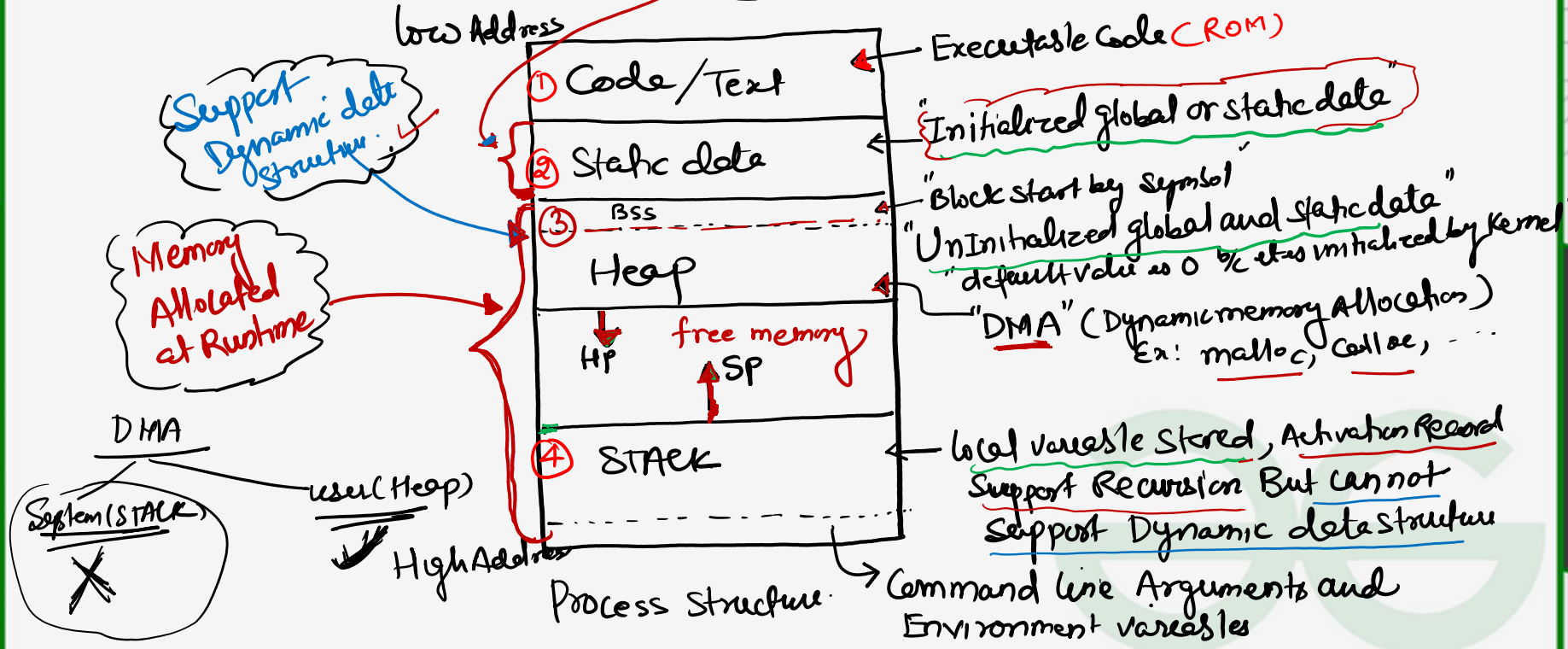
```
A()  /* Caller or Calling function */
{
  int a, b;
  B(); /* Called function */
}
B()
{
  int c;
  c = a + b
}
```

The field for Actual parameters is used by the Calling Procedure to supply parameters to the Called procedure

Compiler Design

Runtime Environment of C-program :

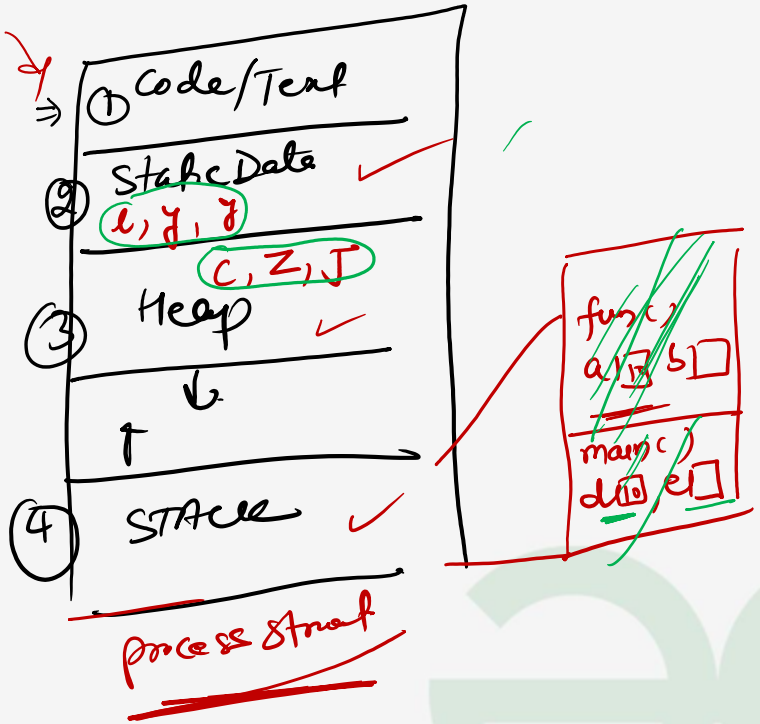
memory allocation at Compile time it is fixed, it can not support Dynamic data structure



Compiler Design

Ex-

```
int i = 10, j;
fun (int x)
{
    static int y = 10, z;
    int a = 10, b;
}
main()
{
    static int f = 20, c;
    int d = 10, e;
    fun()
}
```



Compiler Design

⇒ Static Allocation:

1) Memory Allocation is done at Compile time
2) Binding do not change at runtime
3) Recursion is not Supported.
4) Size of data objects must be Known at Compile time
5) Dynamic data structure not Supported

⇒ Heap Allocation:

1) Allocation and deallocation will be done at any time based on user requirement
2) Recursion Supported
3) Dynamic data structure Supported

⇒ Stack Allocation:

1) Recursion Supported
2) local variable belongs to new Activation Record.
3) Dynamic data structure not supported
4) local variables can not be retained once Activation end.

Compiler Design

Q-45: Consider the following intermediate program in three address code

$p = a - b$

$q = p * c$

$p = u * v$

$q = p + q$

Which one of the following corresponds to a static single assignment from the above code

A)	B)	C)	D)
$p_1 = a - b$	$p_3 = a - b$	$p_1 = a - b$	$p_1 = a - b$
$q_1 = p_1 * c$	$q_4 = p_3 * c$	$q_1 = p_2 * c$	$q_1 = p * c$
$p_1 = u * v$	$p_4 = u * v$	$p_3 = u * v$	$p_2 = u * v$
$q_1 = p_1 + q_1$	$q_5 = p_4 + q_4$	$q_2 = p_4 + q_3$	$q_2 = p + q$

Thank You !