CD U-5

① Basic Blocks

↳ it is the set of statements which executes one after another.

Rules to find leader:

① first three address instr. of code is leader
② wherever the condition statmt is there, it will be leader
③ statmt following the condition statmt will be also leader

functions preserving transformation:

(1) Compile time evaluation
  (i) constant folding
  (ii) constant propagation
(2) Common sub expression elimination
(3) Copy propagation
(4) Code movement
(5) Strength Reduction
(6) Dead code elimination
(7) Loop optimization

(1) Compile Time evaluation

↳ shifting of computation from runtime to compile time.

(i) Constant folding: computation of constnt is done at compile time instead of execution time.

example: length = $\left(\frac{22}{7}\right) * d$

(ii) constant propagation: computation of expression is done at compile time instead of execution time

example: pi = 3.14; r = 5;
Area = pi * r * r

(2) Common subexpression elimination

↳ It is an expression which is appearing repeatedly in program.

↳ if the operands of this sub expression do not get changed then result of the expression is used instead of recomputing it each time.

example:

$t_1 = 4 * i$
$t_2 = a[t_1]$
$t_3 = 4 * j$
$t_4 = 4 * i$
$t_5 = n$
$t_6 = b[t_4] + t_5$

$\Rightarrow$

$t_1 = 4 * i$
$t_2 = a[t_1]$
$t_3 = 4 * j$
$t_5 = n$
$t_6 = b[t_4] + t_5$

Here, t_4 is repeated, so it is eliminated and result of the t_1 is taken

(3) Copy Propagation

↳ It is the type of propagation which uses one variable instead of another

example: $x = pi$

$area = x * r * r$

$\Rightarrow$ $area = pi * r * r$

Here, x is elimited and only one variable is used

(4) Code Movement

↳ The two goals of code movement is:

(i) To obtain space complexity

(ii) To obtain time complexity

example:

```
for(i=0; i<=10; i++)
{ x=y*5;
  ----
  k=(y*5)+50; }
```

$\Rightarrow$

```
z=y*5
for(i=0; i<=10; i++){
x=z; ---- ; k=z+50; }
```

Here, (y*5) is generated only once.

(5) Strength Reduction:

→ in strength reduction technique, the higher strength operators can be replaced by lower strength operator

* is higher than +

example:

```
for(i=1; i<=50; i++)
{ ----
  cont = i * 7;
  ---
}
```

⇒

```
temp = 7;
for(i=1; i<=50; i++){
----
cont = temp;
temp = temp + 7;
}
```

(6) Read code elimination:

↳ A variable is said to be live in a program if the value is used frequently.

↳ A variable is said to be dead in a program if the value is never used.

example:

```
i = j;
----
x = i + 10;
---
```

i=j can be eliminted, cuz it is never used.

(7) loop optimization:

↳ It is a technique in which the optimization is performed on inner loop

example

```
while(i<=max-1)
{ sum = sum + a[i]
}
```

⇒

```
n = max-1
while(i<=n)
{ sum = sum + a[i]
}
```

optimize the code:

② i=1, s=0;
for(i=1; i=3; i++)
for(j=1; j<=3; j++)
c[i][j] = c[i][j] + a[i][j] + b[i][j]

Three address code:

i = 1
s = 0
s = 1
L2: j = 1
L4: t1 = 4*i
t1 = t1 + j
t2 = t1 * 4
t3 = a[t2]
t4 = i * 20
t4 = t4 + j
t5 = t4 * 4
t6 = b[t5]
t7 = t3 + t6
t8 = i * 20
t9 = t8 + j
t10 = t8 * 4
t11 = c[t9]
t12 = t7 + t11
c[i][j] = t12
j = j + 1
if j <= 3 goto L1
i = i + 1
if i <= 3 goto L2

Using common sub expression elimination

s = 0
i = 1
L2: j = 1
L1: t1 = 4 * i
t1 = t1 + j
t2 = t1 * 4
t3 = a(a2)
t4 = b[t2]
t5 = t3 + t4
t6 = c[t2]
t7 = t5 + t6
c[i][j] = t7
j = j + 1
if j <= 3 goto L1
i = i + 1
if j <= 3 goto L2

(3) Global Data Flow Analysis

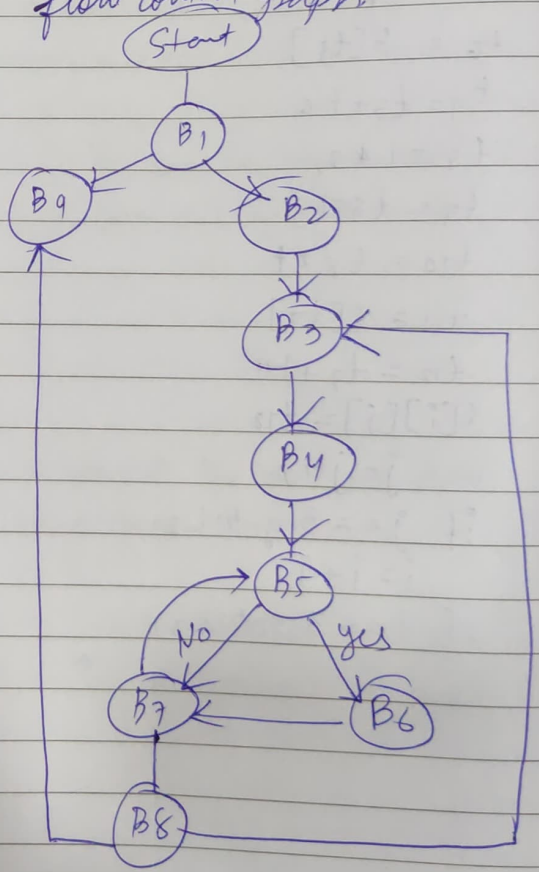
- It is used for wide application
- Here a program is represented in the form of program flow graph
- program flow graph is the graphical representation in which node is basic block and edge is the flow control
- Two types of flow analysis:

(1) control flow Analysis (2) Data flow Analysis

(1) Control flow Analysis: It is used to determine the information regarding arrangent, graph nodes, loops, nested loops of a specific code

(2) Data flow Analysis: It is based on the data flow. It determines the information regarding the definition and use of data in program flow control graph

```
                                    ] B1
if (i>0)
{sum = B[0];                        ] B2
  i = 0;
L1: if(A[i] < B[i])                 ] B3
    { j = 1;                        ] B4
L2: if( B[i] >= 0)                  ] B5
    { sum = sum + B[j]; }           ] B6
    j = j+1;                        ] B7
    if(j < N) goto L2;
  }
  i = i+1;                          ] B8
}
  i = i+1;
  if (i < N) goto L1;
}
printf("sum = %d\n", sum)           ] B9
```



④ Storage Allocation Strategies

There are 3 types of storage Allocation strategies

(i) Static Allocation : It is for all the data at compile time

(ii) Stack Allocation : It is used to manage the runtime storage.

(iii) Heap Allocation : It is used to manage the dynamic memory allocation

(i) Static Allocation: Data structure cannot be created dynamically

→ It is for all the data at compile time

→ Size of every data item is determined by compiler

→ here the compiler can decide the amount of storage need for the data.

→ it is easy to identify the address of data.

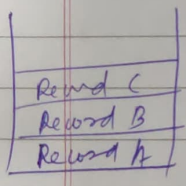
Adv: easy to implmt
: does not run out of memy
: allows type checking

Disadv: cannt use variables
: not compatible with subprograms

(ii) Stack Allocation : Data struct: can be created dynically

→ It is used to manage the runtime storage

→ It is based on stack

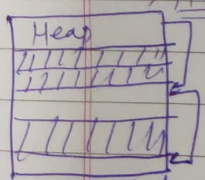


→ push() is used to start
→ pop() is used to end
→ uses LIFO

Adv: supports recursion
: dynamic in nature

Disadv: it is slow.

(iii) Heap allocation: Data struct can be created dynically

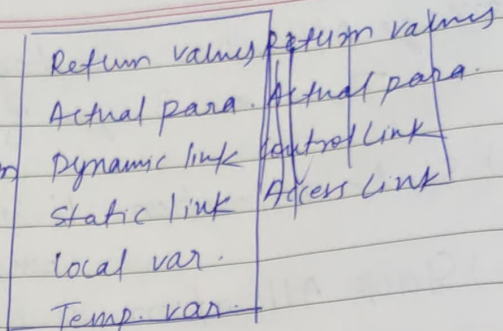


→ used to manage the dynamic memy allocation
→ helpful for executing data
→ heap is allocated for activation record.

Adv: large blocks can be break into small blocks
: efficient

Disadv: problem of fragmentation

Activation Records



Temp var: used for the evaluation of expression

Local var: local data is stored here used for execution

Static link: This is optional field non local data in Activation Record

Dynamic link: This is also optional field Points to the AR

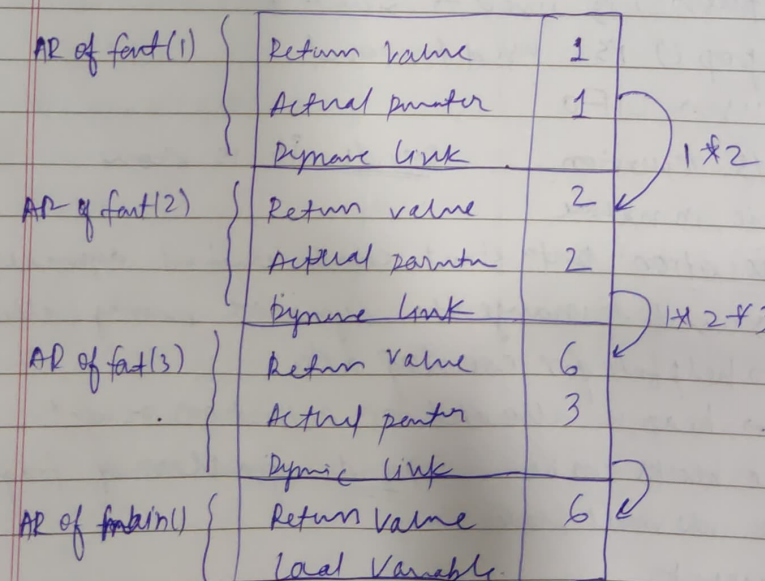
Actual para: has the infos of actual param they are passed to next field called procedure

Return value: Stores the result

example:

```
main (){
   int f
   f = fact (3);
}
int fact( int n)
 if( n == 1)
    return 1;
else
   return n * fact(n-1);
```

main()
fact(3)
fact(2)
fact(1)



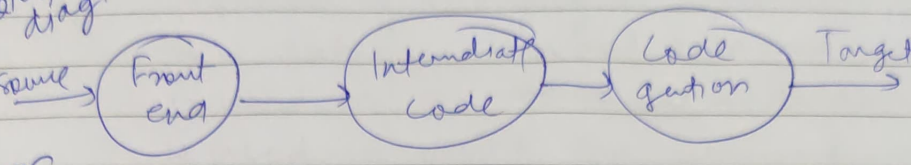
Qn

① code optimization

used to produce the efficient code

pattern for code optimization

block diag.



Machine depnt: based on the char. of the target

Machine indepnt: based on prog. langage such as algorithm etc.

② Peephole optimization

- used to optimize the small part of code.
- it is performed on very small set of instruction
- this small set of instruction is called peephole

Objectives: To improve performe
To reduce code size
To optimize code

ex

```
y = x + 5                y = x + 5
i = y ;          →       w = i * 3
z = i ;
w = z * 3 ;
```