Basic Optimization Techniques

Technical P.I.

1. Strings are stored :-
   1. Stack Area
   2. Heap Area
   3. Data Area

   } Depend upon declaration.

Heap $\boxed{\text{char *str}}$ = (char*)malloc(100);

~~1828027~~

char str[ ] = "KIIT";
$\boxed{\text{stack}}$

char *str = "KIIT"
Data/shared area.

2. Structure padding :-

```
struct st
{
    int a;
    float b;
    char c;
    char d;
};
```

12 bytes

a	b	c

a	b	c	d

Undergo till 4 bytes are covered.

~~After 4 bytes completed~~ one more 4 byte space is created.

According

```
struct st
{
    int x;
    double y;
    float z;
```

4
8
4

12

12	4

1828027

3. Bit Field :-

Memory saving member of structure.

4+4+4+4 = 16

int roll_no:8 → int :8 Anonymous → Bit field without any name.

```
struct student
{
    int roll_no:8;   4
    int age:8;       4
    char name[20];   28
};
```

Limitations :-

* Can't use scanf to intialize.
* Can't have pointer to bit field.

4. Diff b/w array/structure

5. " " st/union.

Basic Optimization Techniques

1. Avoid calculation inside loop.
   - Inner loops have min. possible calculations
   - Calculations dependent outside loop.
2. Avoid pointer de-reference in loop :-
   - Creates lots of trouble in memory.

```
int temp = *p
for(i)
{
    temp+=i;
}
*p = temp;
```

1828027.

Satpathy

3. Unroll small loops :—

fact[i] = fact[i-1] * i;

4. Use register variable as counters :—

register int x;
register int y → CPU register
Little bit fast.

5. Operators using :—

— Minimize division

$\frac{a}{b/c} = \frac{a*c}{b}$

✗ int d = a/b/c;
✓ int d = a/b*c;

— ~~Multiplication~~ Bitwise shift.

x = x/2 , x = x >> 1;
x = x*2 , x = x << 1; — work on binary digit

— Pre-increment more than post increment
— In post increment lots of things happen

→ short circuit AND.

(max < b) && (max = b)

If left false, right won't be evaluated.

— Prefer bitwise more than arithmetic :—

a ^ = b ^ = a ^ = b ;

1828027