Recursion Drill

③ K th Finding

Q: Find K th move recursively (For Tower of Hanoi)

No. of moves needed to move n disks

$F(x) = F(x-1) + 1 + F(x-1)$

$F(x) = 2F(x-1) + 1$

$F(1) = 1$ move , $F(2) = 3$ moves

$F(0) = 0$ move , $F(3) = 7$ moves

$$\boxed{F(x) = 2^{x} - 1}$$

Code :-

```
void movedisk (int disks, int from, int to, int extra, int k) {
    if (disks ==0) return;
    if (k<=(1<< disks-1)) -1) {   //left side
        movedisk (disks -1 , from, extra, to, k);

    }
    else if (k==(1<<(disks -1))) {   //Middle
        cout << "From << from << "To " << to << endl;

    }
    else {
        movedisk (disk -1, extra, to, from, k- (1<<disks -1)));

    }
```

Q: K-th symbol in Grammar

$$\begin{tikzcd}
 & 0 \arrow[dl] \arrow[dr] & \\
0 & & 1
\end{tikzcd}
\qquad
\begin{tikzcd}
 & 1 \arrow[dl] \arrow[dr] & \\
1 & & 0
\end{tikzcd}$$

$$\begin{tikzcd}
\text{① row} & & & 0 \arrow[dll] \arrow[drr] & & & \\
\text{② row} & 0 \arrow[d] \arrow[dr] & & & & 1 \arrow[dl] \arrow[d] & \\
\text{③} & 0 & 1 & & 1 & 0 &
\end{tikzcd}$$

Generate from. 0 to level 3

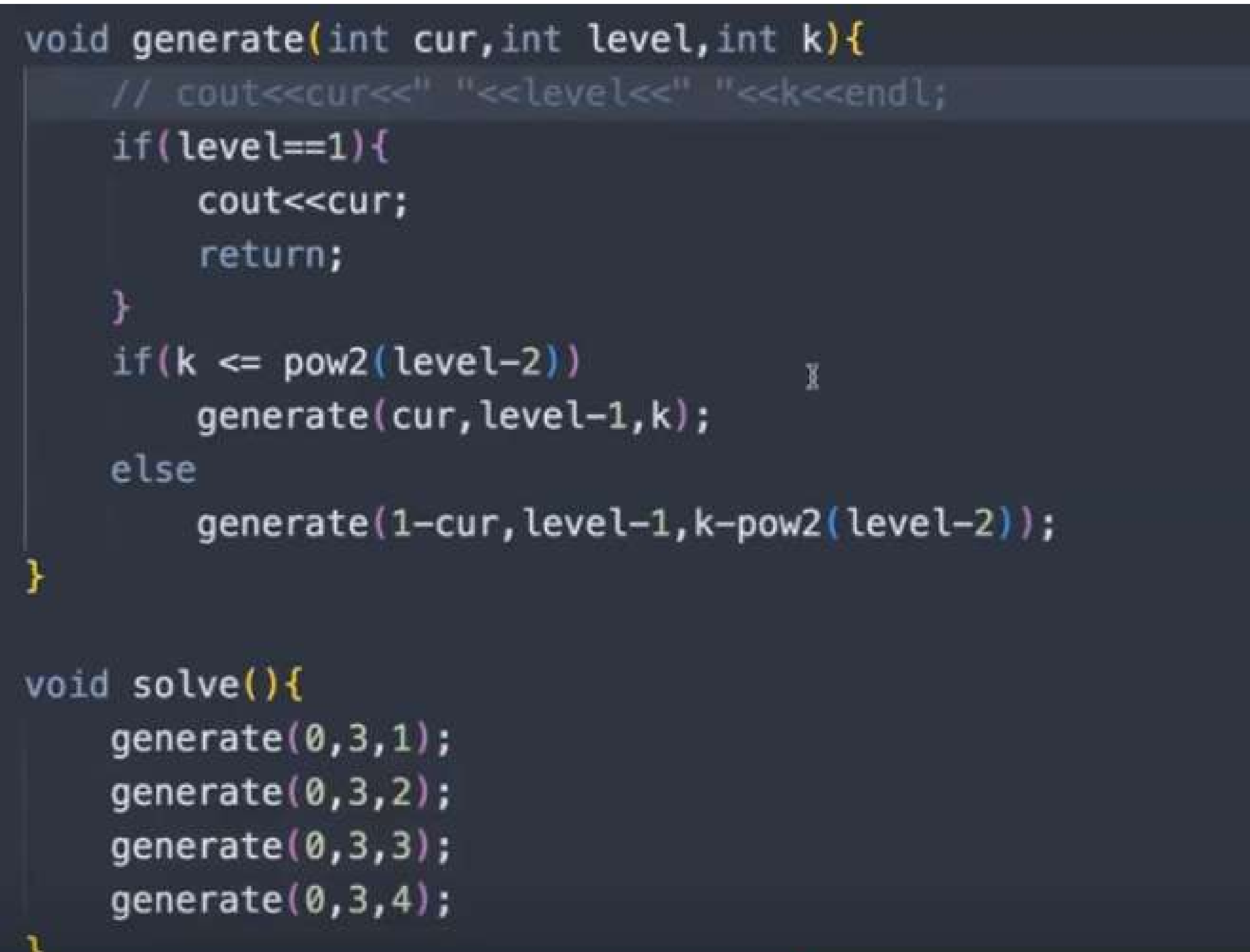
$$\begin{tikzcd}
 & & & \text{generate }(0,3) \arrow[dll] \arrow[drr] & & & \\
 & \text{gen }(0,2) \arrow[dl] \arrow[dr] & & & & \text{gen }(1,2) \arrow[dl] \arrow[dr] & \\
\text{gen}(0,1) \arrow[d] & & \text{gen }(1,1) \arrow[d] & & \text{gen}(1,1) \arrow[d] & & \text{gen}(0,1) \arrow[d] \\
\boxed{0} & & \boxed{1} & & \boxed{1} & & \boxed{0}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{generate }(0 \arrow[r, bend left] \arrow[dr, bend right] \arrow[ddr, bend right] & \text{1st level} \\
& \text{gene }(0, \text{level}-1) \\
& \text{gen }(1, \text{level}-1)
\end{tikzcd}$$

generate $(\overset{\text{1st level}}{0}, \text{level})$

↳ gene $(0, \text{level}-1)$

↳ gen $(1, \text{level}-1)$



```
void generate(int cur,int level,int k){
    // cout<<cur<<" "<<level<<" "<<k<<endl;
    if(level==1){
        cout<<cur;
        return;
    }
    if(k <= pow2(level-2))
        generate(cur,level-1,k);
    else
        generate(1-cur,level-1,k-pow2(level-2));
}

void solve(){
    generate(0,3,1);
    generate(0,3,2);
    generate(0,3,3);
    generate(0,3,4);
}
```

3. Christmas

Level - 0 burger $\rightarrow$ Patty (P)

~~Level-L burger (L>=1) is B, level-(L-1) burger, P.~~

Level-L-burger = "B" + level-(n-1) + "P" + level-(n-1) + "B"

		0 level	1 level	2 level
B	–	0	2	4
P	–	1	3	7

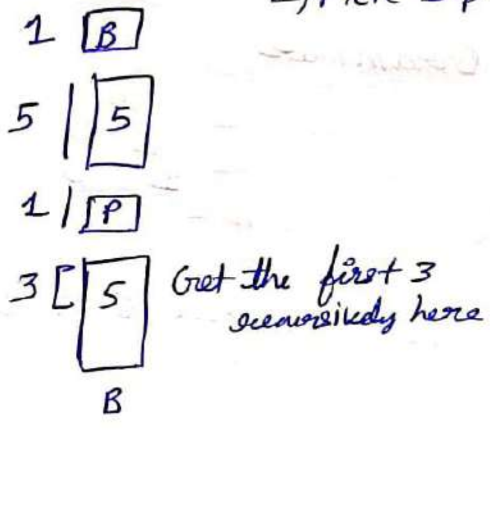
$\downarrow$ Previous value $\times 2 + B + P + B = 3$

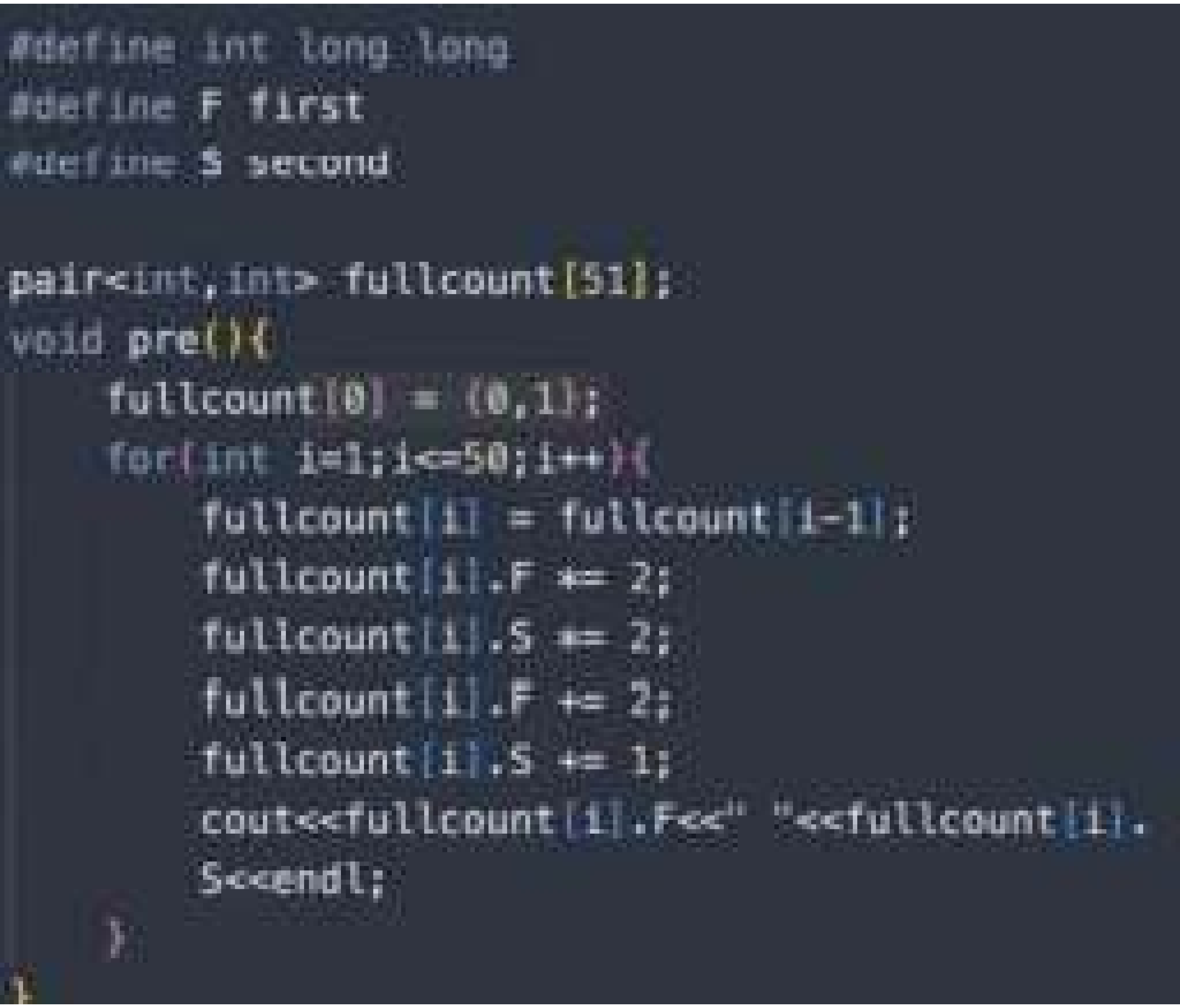
We can pre compute the above values.

If we want the top x at level, x = 10 (say)

↳ Pick top 10 items

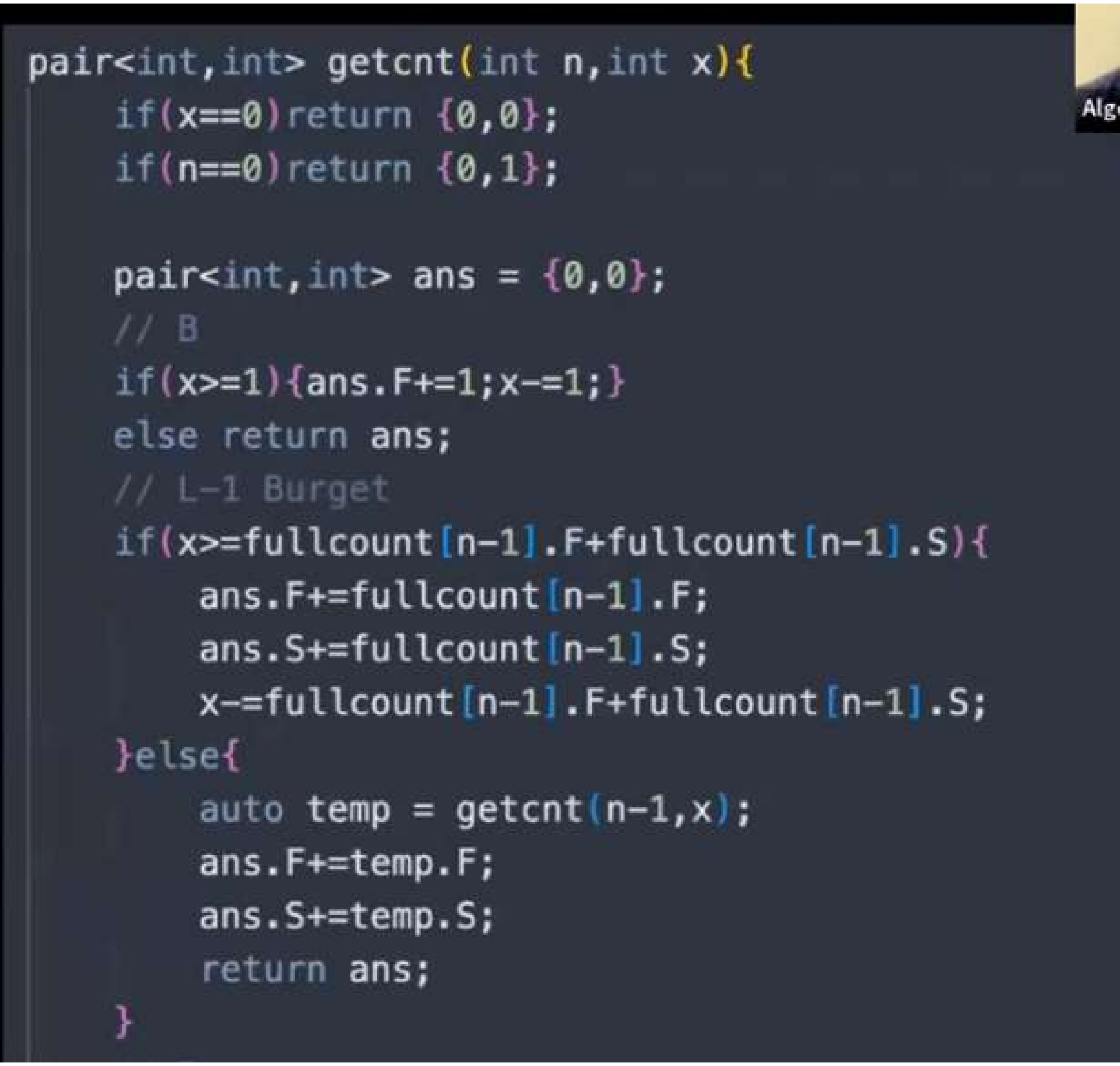
$$\boxed{1+5+1+3 = 10}$$





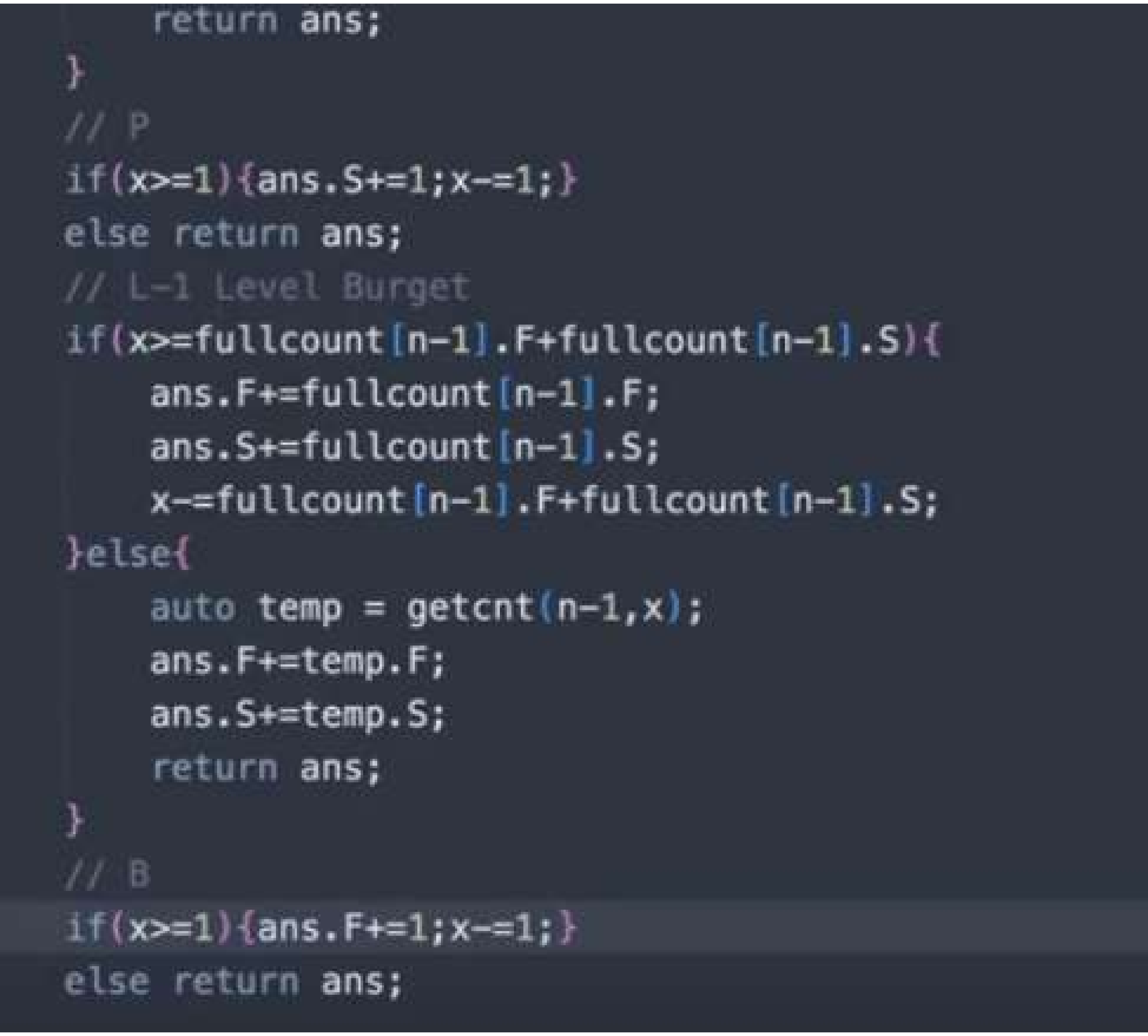
```cpp
#define int long long
#define F first
#define S second

pair<int,int> fullcount[51];
void pre(){
    fullcount[0] = {0,1};
    for(int i=1;i<=50;i++){
        fullcount[i] = fullcount[i-1];
        fullcount[i].F += 2;
        fullcount[i].S += 2;
        fullcount[i].F += 2;
        fullcount[i].S += 1;
        cout<<fullcount[i].F<<" "<<fullcount[i].
        S<<endl;
    }
}
```



```
pair<int,int> getcnt(int n,int x){
    if(x==0)return {0,0};
    if(n==0)return {0,1};

    pair<int,int> ans = {0,0};
    // B
    if(x>=1){ans.F+=1;x-=1;}
    else return ans;
    // L-1 Burget
    if(x>=fullcount[n-1].F+fullcount[n-1].S){
        ans.F+=fullcount[n-1].F;
        ans.S+=fullcount[n-1].S;
        x-=fullcount[n-1].F+fullcount[n-1].S;
    }else{
        auto temp = getcnt(n-1,x);
        ans.F+=temp.F;
        ans.S+=temp.S;
        return ans;
    }
```



```
    return ans;
}
// P
if(x>=1){ans.S+=1;x-=1;}
else return ans;
// L-1 Level Burget
if(x>=fullcount[n-1].F+fullcount[n-1].S){
    ans.F+=fullcount[n-1].F;
    ans.S+=fullcount[n-1].S;
    x-=fullcount[n-1].F+fullcount[n-1].S;
}else{
    auto temp = getcnt(n-1,x);
    ans.F+=temp.F;
    ans.S+=temp.S;
    return ans;
}
// B
if(x>=1){ans.F+=1;x-=1;}
else return ans;
```