一、简答

使用的全局变量：

```
pair<int, int> cursor, spot;  // 光标位置
int sz[N];  // 维护每行字符串长度
int row;    // 当前有效行数
// 记事本
string notepad;
// 剪贴板
string clipboard;
// 粘滞功能
bool v;
// 选中状态
bool selected;
```

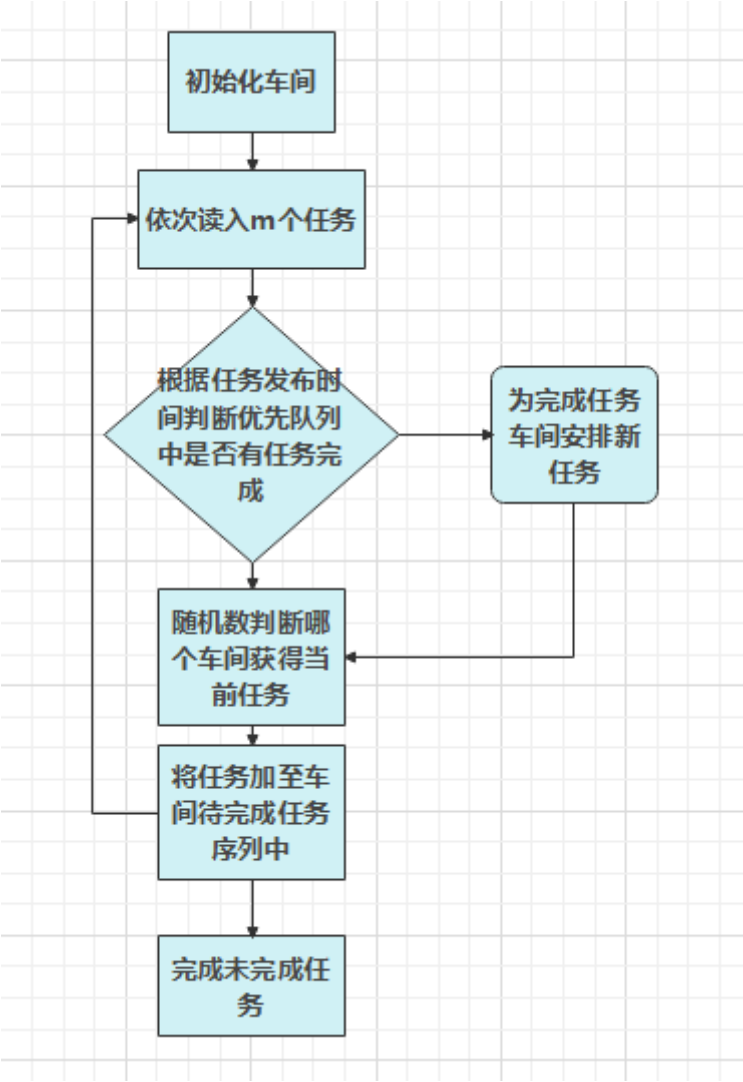
· 使用string存储文本内容，空格以及'\n'也作为内容存储。
· 使用spot记录开启粘滞功能时光标的位置(记录点)，和当前光标位置cursor共同确定选中内容。
· copy功能的实现：

```
void copy() {
    if (selected) {
        int a = getPos(spot);
        int b = getPos();
        if (b < a) swap(a, b);
        clipboard = notepad.substr(a, b - a);
    }
    else if (sz[cursor.first]) {    // 不为空
        // 复制当前行的内容
        int pos = getPos() - cursor.second;
        clipboard = notepad.substr(pos, sz[cursor.first]);
    }
}
```

首先判断是否选中状态，若是则将选中内容复制到剪贴板clipboard；否则复制当前行，由于文本内容存于单个string，通过数组sz[]记录每行长度，以此确定当前行在string中的起止。

二、代码及注释

流程图：


初始化车间
依次读入m个任务
根据任务发布时间判断优先队列中是否有任务完成
为完成任务车间安排新任务
随机数判断哪个车间获得当前任务
将任务加至车间待完成任务序列中
完成未完成任务

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

static unsigned int p = 1;

int myrand(void) {
    p = p * 1103515245 + 12345;
    return((unsigned int)(p/65536) % 100);
}

struct task
{
    string name;
    int t;
};

struct room
{
    string name;
    list<task> job;     // 待执行任务
    task going;         // 当前执行任务
    int attr;
} R[1005];

map<string, int> mp;
map<int, int> mp2;       // 任务添加时间 <-> 车间号
map<string, int> mp3;    // 任务名 <-> 添加时间
ll clk[1005];           // 当前任务完成时间
priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<pair<ll, int>>> pq;     // 当前工作

void Push(int i, int t, task tk) {
    // 当前处于空闲状态,则直接执行
    if (clk[i] <= t) {
        clk[i] = t + tk.t;
        R[i].going = tk;
        pq.push({clk[i], t});
        return;
    }

    if (!R[i].attr)
        R[i].job.push_back(tk);

    // 有序插入
    if (R[i].attr == 1) {
        auto bn = R[i].job.begin();
        auto end = R[i].job.end();
        for (auto it = bn; it != end; it++) {
            if (it->t > tk.t || (it->t == tk.t && it->name < tk.name)) {
                R[i].job.insert(it, tk);
```

```
                return;
            }
        }
        R[i].job.push_back(tk);
    }
    if (R[i].attr == 2) {
        auto bn = R[i].job.begin();
        auto end = R[i].job.end();
        for (auto it = bn; it != end; it++) {
            if (it->t < tk.t || (it->t == tk.t && it->name < tk.name)) {
                R[i].job.insert(it, tk);
                return;
            }
        }
        R[i].job.push_back(tk);
    }
}

void solve() {
    int n, m;
    cin >> n >> m;

    string name, attr;
    // 初始化车间
    for (int i = 1; i <= n; i++) {
        cin >> name >> attr;
        R[i].name = name;
        mp[name] = i;
        if (attr == "TF") {
            R[i].attr = 0;
        }
        else if (attr == "SF") {
            R[i].attr = 1;
        }
        else {
            R[i].attr = 2;
        }
    }

    int t, d, k;
    string s, c;
    for (int i = 1; i <= m; i++) {
        cin >> t >> s >> d >> k;
        // 初始化新任务
        mp3[s] = t;
        task tk;
        tk.name = s;
        tk.t = d;
        // t为当前时间
        // 检查是否有车间完成任务
        while (!pq.empty()) {
```

```
            auto top = pq.top();
            if (top.first <= t) {
                // 有大量同时完成的任务
                // 该任务已完成,属于u车间
                pq.pop();
                int u = mp2[top.second];
                cout << clk[u] << ' ' << R[u].going.name << '\n';
                if (!R[u].job.empty()) {
                    auto bn = R[u].job.begin();
                    R[u].going = *bn;     // 为该车间设置新任务
                    R[u].job.pop_front();
                    clk[u] = clk[u] + R[u].going.t;
                    pq.push({clk[u], mp3[R[u].going.name]});
                }
            }
            else break;
        }
        bool flag = 0;
        for (int j = 1; j <= k; j++) {
            cin >> c;
            if (flag) continue;
            if (j < k && (100 / k) > myrand()) {
                mp2[t] = mp[c];
                Push(mp[c], t, tk);
                flag = 1;
            }

            if (j >= k && (!flag)) {
                mp2[t] = mp[c];
                Push(mp[c], t, tk);
            }
        }
    }

    // 最后任务
    while (!pq.empty()) {
        auto top = pq.top();
        pq.pop();
        int u = mp2[top.second];
        cout << clk[u] << ' ' << R[u].going.name << '\n';
        if (!R[u].job.empty()) {
            R[u].going = *R[u].job.begin();     // 为该车间设置新任务
            R[u].job.pop_front();
            clk[u] = clk[u] + R[u].going.t;
            pq.push({clk[u], mp3[R[u].going.name]});
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
```

```
    cin.tie(nullptr);
    solve();
    return 0;
}
```