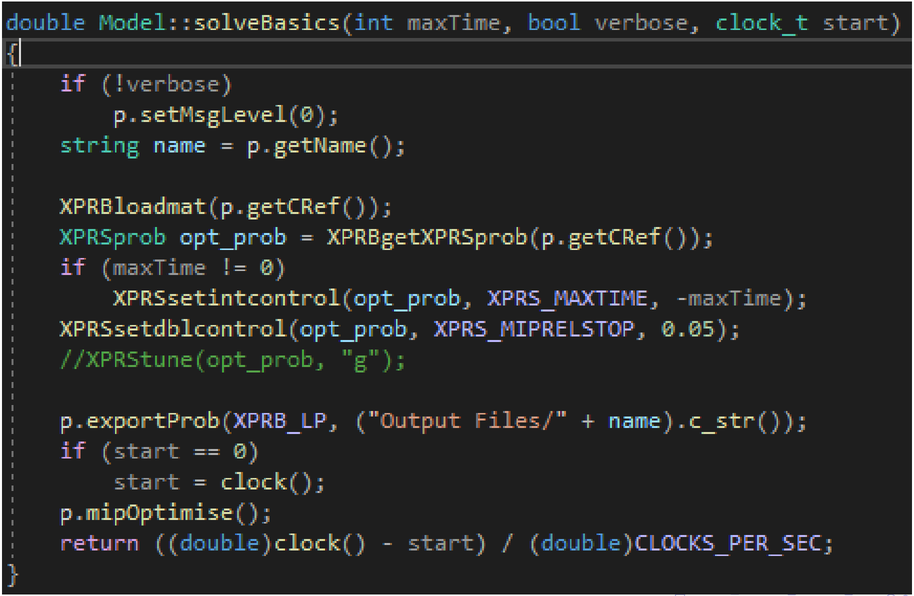
General structure

- Select mode
- Read data from file
- while not everything feasible
    - Generate year model(year data, feedback)
    - Solve year model
    - Generate month data(year solution)
    - for each month
        - Generate month model(month data)
        - Solve month model
        - If feasible -> next month
        - Generate feedback model(month data)
        - Solve feedback model
    - double maxTime

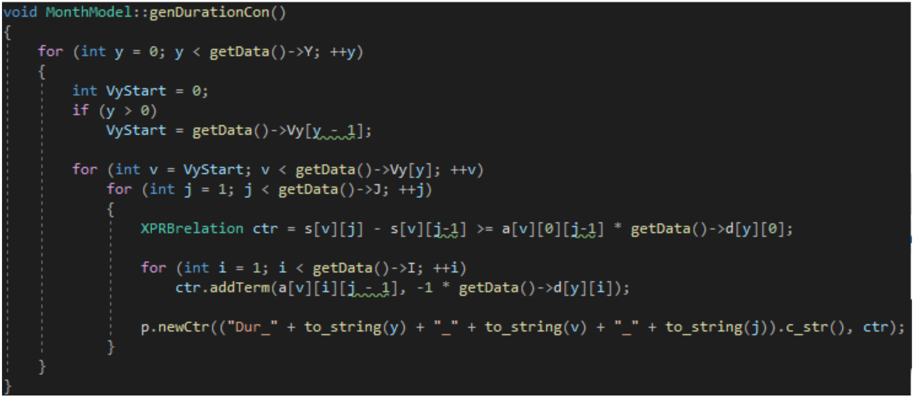
Solving function



```
double Model::solveBasics(int maxTime, bool verbose, clock_t start)
{
    if (!verbose)
        p.setMsgLevel(0);
    string name = p.getName();

    XPRBloadmat(p.getCRef());
    XPRSprob opt_prob = XPRBgetXPRSprob(p.getCRef());
    if (maxTime != 0)
        XPRSsetintcontrol(opt_prob, XPRS_MAXTIME, -maxTime);
    XPRSsetdblcontrol(opt_prob, XPRS_MIPRELSTOP, 0.05);
    //XPRStune(opt_prob, "g");

    p.exportProb(XPRB_LP, ("Output Files/" + name).c_str());
    if (start == 0)
        start = clock();
    p.mipOptimise();
    return ((double)clock() - start) / (double)CLOCKS_PER_SEC;
}
```

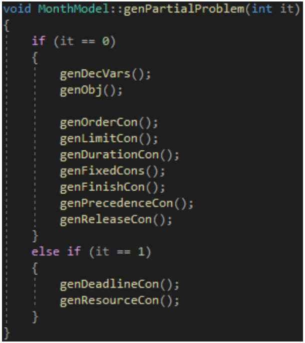
Constraint generation

$$s_{v,j} - s_{v,(j-1)} \geq \sum_{i \in I}(a_{v,i,(j-1)} \cdot d_{y,i}) \qquad \forall y \in Y, \forall v \in V_y, \forall j \in J - \{0\}$$

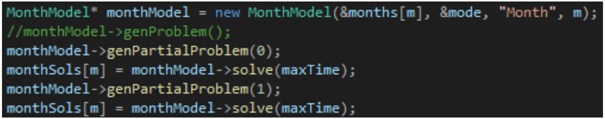


```
void MonthModel::genDurationCon()
{
    for (int y = 0; y < getData()->Y; ++y)
    {
        int VyStart = 0;
        if (y > 0)
            VyStart = getData()->Vy[y - 1];

        for (int v = VyStart; v < getData()->Vy[y]; ++v)
            for (int j = 1; j < getData()->J; ++j)
            {
                XPRBrelation ctr = s[v][j] - s[v][j-1] >= a[v][0][j-1] * getData()->d[y][0];

                for (int i = 1; i < getData()->I; ++i)
                    ctr.addTerm(a[v][i][j - 1], -1 * getData()->d[y][i]);

                p.newCtr(("Dur_" + to_string(y) + "_" + to_string(v) + "_" + to_string(j)).c_str(), ctr);
            }
    }
}
```

Splitting the problem up

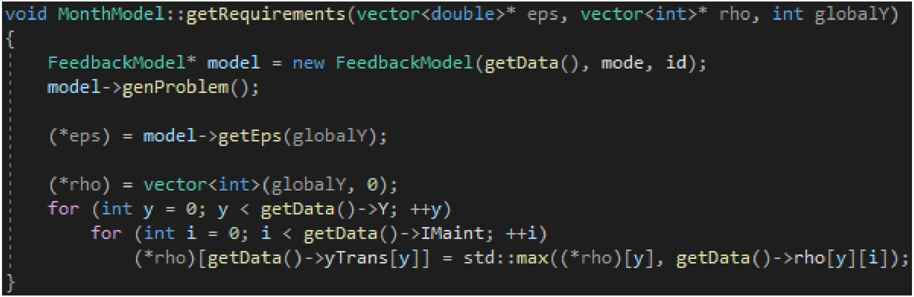


```
void MonthModel::genPartialProblem(int it)
{
    if (it == 0)
    {
        genDecVars();
        genObj();

        genOrderCon();
        genLimitCon();
        genDurationCon();
        genFixedCons();
        genFinishCon();
        genPrecedenceCon();
        genReleaseCon();
    }
    else if (it == 1)
    {
        genDeadlineCon();
        genResourceCon();
    }
}
```

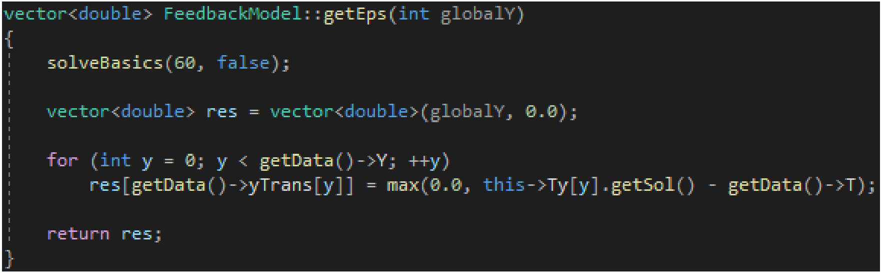


```
MonthModel* monthModel = new MonthModel(&months[m], &mode, "Month", m);
//monthModel->genProblem();
monthModel->genPartialProblem(0);
monthSols[m] = monthModel->solve(maxTime);
monthModel->genPartialProblem(1);
monthSols[m] = monthModel->solve(maxTime);
```

Feedback generation



```
void MonthModel::getRequirements(vector<double>* eps, vector<int>* rho, int globalY)
{
    FeedbackModel* model = new FeedbackModel(getData(), mode, id);
    model->genProblem();

    (*eps) = model->getEps(globalY);

    (*rho) = vector<int>(globalY, 0);
    for (int y = 0; y < getData()->Y; ++y)
        for (int i = 0; i < getData()->IMaint; ++i)
            (*rho)[getData()->yTrans[y]] = std::max((*rho)[y], getData()->rho[y][i]);
}
```



```
vector<double> FeedbackModel::getEps(int globalY)
{
    solveBasics(60, false);

    vector<double> res = vector<double>(globalY, 0.0);

    for (int y = 0; y < getData()->Y; ++y)
        res[getData()->yTrans[y]] = max(0.0, this->Ty[y].getSol() - getData()->T);

    return res;
}
```

Ideas to try

- Run program remotely on university PC (potentially faster, and can run in background)
- Experiment more with splitting constraints
- Improve feedback logic