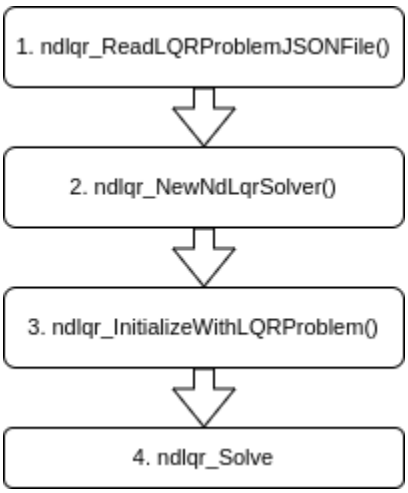
Data flow before ndlqr_Solve()

As stated in our report, the flow of function calls in the rsLQR algorithm can be summarized, on a high-level, as follows:



The main function of the rsLQR algorithm looks like this:

```c
int main() {
    printf("This is the example using an installed rsLQR package!.\n\n");
    // Read an LQRProblem from a json file
    const char* filename = LQRPROBFILE;
    LQRProblem* lqrprob = ndlqr_ReadLQRProblemJSONFile(filename);
    int nstates = lqrprob->lqrdata[0]->nstates;
    int ninputs = lqrprob->lqrdata[0]->ninputs;
    int nhorizon = lqrprob->nhorizon;

    // Initialize the solver and solve
    NdLqrSolver* solver = ndlqr_NewNdLqrSolver(nstates, ninputs, nhorizon);
    if (ndlqr_InitializeWithLQRProblem(lqrprob, solver) == 0) {
        ndlqr_Solve(solver);
    }

    // Print the solve summary
    ndlqr_PrintSolveSummary(solver);

    // Free the problem data and the solver
    ndlqr_FreeLQRProblem(lqrprob);
    ndlqr_FreeNdLqrSolver(solver);
    return 0;
}
```

In a separate file, we already discussed the data flow in 4. ndlqr_Solve(). Thus, we now discuss the data flow in the first three functions. For the sake of understanding, we will take an example case to discuss the data flow. In our example, ninputs = m = 5, nstates = n = 100, and nhorizon = N = 4.

1. **ndlqr_ReadLQRProblemJSONFile(): The known values in the equations of the LQR problem get populated into an object of the LqrProblem class.**
2. **ndlqr_NewNdLqrSolver():** Contiguous memory is allocated for the following:
    a. diagonals: An array of size 2N. diagonals[2k] and diagonals[2k+1] together make one block. Thus, diagonals has 4 blocks. The first element in the block is of size nxn = 100x100. The second element in the block is of size mxm = 5x5.
    b. data (NdData): nsegments = N - 1 = 3, depth = log2(N) = 2, factors = an array of size N * depth = 8. Each element in factors is an NdFactor, in which lambda matrix is of size 100x100, state matrix is of size 100x100, and input matrix is of size 5x100.
    c. fact (NdData): same memory allocation size as data.
    d. soln (NdData): same memory allocation size as data, except that depth = 1 and factors = an array of size N * depth = 4.
3. **ndlqr_InitializeWithLQRProblem():** This is used to initialize the factors array, of both data and soln.
    a. data (NdData): Indices 0, 1, 2, 3, 5, and 6 of factors get initialized with values from LqrProblem. Indices 4 and 7 are skipped. This is due to the structure of the LqrProblem H matrix. Further, lambda matrices are not initialized. Only state and input matrices are.
    b. soln (NdData): All indices (0, 1, 2, and 3) get initialized. And, all the lambda, state, and input matrices are initialized, except for the input matrix of index 3. This is because N = 4, meaning, **index 3 corresponds to the final time step, which does not require an input**.