Finding Subgroups of a Factor Group

Data Structures

group element (i.e. point group operation): Eigen::Matrix3d

group: std::vector<Eigen::Matrix3d>

list_of_groups: std::vector<std::vector<Eigen::Matrix3d>>

Methods: Functions

```
bool check_for_closure(std::vector<Eigen::Matrix3d> group)
{
        returns true if group is closed;
        else returns false;
}

bool check_for_identity(std::vector<Eigen::Matrix3d> group)
{
        returns true if Identity is an element of the group;
        else returns false;
}
```

Methods: Functions

```
Eigen::Matrix3d multiply_group_elements(Eigen::Matrix3d element1, Eigen::Matrix3d element2)
{
        returns product of two group elements;
}

bool is_element_in_group(Eigen::Matrix3d element, std::vector<Eigen::Matrix3d> group)
{
        returns true if given element is a member of the given group;
        else returns false;
}
```

Methods: Functions

bool compare_groups(std::vector<Eigen::Matrix3d> group1, std::vector<Eigen::Matrix3d> group2)

{

returns true if two groups are equal size and contain same elements;

else returns false;

}

bool is_group_in_list_of_subgroups(std::vector<std::vector<Eigen::Matrix3d>> list_of_subgroups, std::vector<Eigen::Matrix3d> group)

{

returns true if given group is a member of the given list;

else returns false;

}

Methods: Functions

std::vector<Eigen::Matrix3d> group combine_groups(std::vector<Eigen::Matrix3d> group1, std::vector<Eigen::Matrix3d> group2)

{

combines two groups into one group;

while combining the groups, removes duplicate elements;

returns combined group with no duplicates;

}

Methods: Functions

std::vector<std::vector<Eigen::Matrix3d>> list_of_groups generate_subgroups(std::vector<Eigen::Matrix3d> total_group)

{

takes an element of the total group, initializes a subgroup with it, and begins multiplying element by itself;

if the product is not present in subgroup, product is added as element of the subgroup and will again by multiplied by initial element;

if the product is present in the subgroup, subgroup is completed;

if the size of the subgroup is equal to the size of the total group, the total group is already smallest possible subgroup and loop is exited;

combines smaller subgroups into larger subgroups (using previously outlined function) and if subgroup is closed, it is added to list of subgroups;

returns a list of subgroups;

}

Main Function Outline

```
int main(total_group)
{
        checks total_group for closure and identity, returns error if false;
        generate subgroups using outlined functions;
        print multiplication table for group??
}
```