```cpp
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string>
#include <vector>
using std::cout;
using std::endl;
using std::string;


class Person
{
public:
    Person(const string& fname, const string& lname, const string& dateBirth) : firstName(fname), lastName(lname), date_of_birth(dateBirth) {};
    virtual ~Person() {}
    string getFirstName() {return firstName;};
    string getLastName() { return lastName; };  // accesses implicit and returns private member
    string getBirthDate() { return date_of_birth; };
    bool operator < (const Person& person) const  //compares implicit person object to parameter person object
    {
        return (this->firstName.compare(person.firstName) == -1); // uses compare function for strings to
    }                                                                   //determine if firstName of the implicit person
                                                                        //is less than the parameter person's firstName
                                                                        //firstName used as a key
                                                                        //returns bool to ComparePointers() function
private:
    string firstName;
    string lastName;
    string date_of_birth;

protected:
    struct ComparePointers    //comparator used to sort children vector of Person*
    {
        bool operator () (const Person* person1, const Person* person2)
        {
            return (*person1 < *person2);  // dereferences pointers person1 and person2 to
                                           //person objects and uses operator < function to compare
        }                                  //bool returned to STL sort function
        found in <algorithm> header
    };
};

class Mother : public Person
{
public:
    Mother(const string& fname, const string& lname, const string& dateBirth) : Person(fname, lname, dateBirth) {};  //constructor assigns fname and lname to parent Person
                                                                                                                     // member variables firstName and lastName
                                                                                                                     //for newly created Mother object. Can access via
                                                                                                                     //inherited getlastName() and getfirstName() functions
    virtual ~Mother() {}  // destructor
    Person* hasBaby(const string& f_name, const string& birth_date);
    void print_children();
    void removeChild(Person*);
private:
    std::vector<Person*> children; //stores children of mother (deleted at time program ends)


};

//adds a child to mother object via member children object (vector of Person*)
Person* Mother::hasBaby(const string& f_name, const string& birth_date)
{
    Person* newBaby = new Person(f_name, getLastName(), birth_date); // dynamically allocates memory to store person object that newbaby points to
                                                                       //getLastName() returns mother's last name
    children.push_back(newBaby);  // adds to vector
    return newBaby;    // returns pointer newBaby of type Person*
}

void Mother::removeChild(Person* person)  //removes child from mother object via member child object (vector of Person*)
{
    int count = 0; // used to detail the position in the vector to erase the child
    std::vector<Person*> ::iterator it; // iterator used to transit vector from one Person* to the next
    for (it = children.begin(); it != children.end(); ++it)
    {
        if ((*it) == person) //compares address of dereferenced iterator to
```

```cpp
            that of the person object parameter
        {
            children.erase(children.begin() + count); // uses vector STL erase function to remove child from vector if found
            delete person;  // deletes data pointed to by person
            person = NULL;  // sets person to point to nothing
            break;  //exits for loop
        }
        count++;
        if (it == (children.end() - 1)) //lets user know the child searched for of the implicit mother object doesn't exist
        {
            cout << "A child by the name of " << person->getFirstName() << " " << person->getLastName();
            cout << " was not found and erased for " << this->getFirstName() << " " << this->getLastName() << "  ";
        }
    }
}

//sorts list of mother's children and displays mother and children
void Mother::print_children()
{
    cout << std::left;
    cout << std::setw(4) << "" << this->getFirstName() << " " << this->getLastName() << "'s children are: " << endl;
    cout << std::setw(4) << "" << "-------------------------" << endl;
    sort(children.begin(), children.end(), ComparePointers()); //function uses a form of a selection sort O(n^2) to arrange children
                                                                        //using comparator ComparePointers()
    std::vector<Person*> ::iterator it; // iterator used to transit vector from one Person* to the next
    for (it = children.begin(); it != children.end(); ++it)
    {
        cout << std::setw(10) << "" << (*it)->getFirstName() << " " << (*it)->getLastName();      //dereferenced iterator it is address of Person*
        cout << " DOB: " << (*it)->getBirthDate() << endl;                                  //with -> pointing to object member functions
    }
        cout << endl;


    cout << endl;
}

int main(void)
```

```
{
    Mother sue("Sue", "Smith","5/7/60");
    Mother irene("Irene", "DeWalt", "6/4/55");  // creates mother object
    Person* joe = sue.hasBaby("Joe", "8/12/80");
    Person* kay = sue.hasBaby("Kay","3/2/85"); // creates pointer kay of type
      Person*
    Person* mike = sue.hasBaby("Mike", "4/2/93");
    Person* jeremy = irene.hasBaby("Jeremy","5/3/73");
    cout << "Baby Joe's last name is: " << joe->getLastName() << endl; //
      children have mother's last name
    cout << "Baby Kay's last name is: " << kay->getLastName() << endl;
    cout << endl;
    sue.print_children();  //mother function print_children called for sue
      object
    irene.print_children();
    sue.removeChild(kay);
    sue.print_children();
    irene.removeChild(joe);

    system("pause > nul");
    return 0;
}
```