Overview

This C++ program implements a set of interactive tasks that users can choose to perform. Each task is encapsulated within its own class, which inherits from a base abstract class called `LabTask`. The available tasks include managing a people structure, displaying a bidirectional list, constructing a family tree, and maintaining a sorted list of persons. Users can select a task from a menu, and the corresponding functionality is executed.

Key Classes and Components

LabTask

`LabTask` is an abstract base class that defines a pure virtual function `execute()`. All derived classes must implement this function to provide specific functionality for each task.

PeopleStructure

This class manages a list of people, allowing users to input and display their details.

- **`inputData()`**: Prompts users to enter the name, age, and city for three people.
- **`displayData()`**: Outputs the details of each person stored in the vector.
- **`execute()`**: Calls `inputData()` and `displayData()`, and provides a note explaining why a union cannot be used for this task.

BidirectionalList

This class implements a bidirectional linked list to store integer values.

- **Node Structure**: Represents each element in the list, holding an integer value and pointers to the previous and next nodes.
- **Constructor**: Initializes the list with values from 0 to 10.
- **`displayReverse()`**: Outputs the values of the list from end to start.
- **`execute()`**: Calls `displayReverse()` to show the values in reverse order.
- **Destructor**: Cleans up allocated memory for the list nodes.

FamilyTree

This class constructs a simple family tree structure.

- **FamilyMember Structure**: Represents each family member with pointers to their father and mother.
- **Constructor**: Initializes a child with references to its father and mother.
- **`displayFamily()`**: Outputs the names of the child, father, and mother.
- **`execute()`**: Calls `displayFamily()` to show family relationships.
- **Destructor**: Cleans up allocated memory for family members.

SortedPersonList

This class maintains a sorted linked list of persons based on their age.

- **Person Structure**: Represents each person with their name, age, and pointer to the next person.

- **addPerson()**: Inserts a new person into the list while maintaining sorted order based on age.
- **displayList()**: Outputs the names and ages of all persons in the list.
- **execute()**: Adds several persons to the list and displays them in sorted order.
- **Destructor**: Cleans up allocated memory for persons in the list.

Execution Flow

Main Menu:

The `main()` function presents a menu to the user, allowing them to select one of four tasks:

1. People Structure
2. Bidirectional List
3. Family Tree
4. Sorted Person List

Task Execution:

Based on user input, the corresponding task's `execute()` method is invoked. Each task handles its own input/output, allowing for independent execution of functionalities.

Program Termination:

After executing a selected task, the program terminates gracefully without prompting for further actions, as there is no loop implemented for repeated task selection.

Example User Interaction

People Structure

```
Select a task:
1. People Structure
2. Bidirectional List
3. Family Tree
4. Sorted Person List
>> 1
Enter name for person 1: Alice
Enter age for Alice: 30
Enter city for Alice: New York
Enter name for person 2: Bob
Enter age for Bob: 25
Enter city for Bob: Los Angeles
Enter name for person 3: Charlie
Enter age for Charlie: 35
Enter city for Charlie: Chicago
Name: Alice, Age: 30, City: New York
Name: Bob, Age: 25, City: Los Angeles
Name: Charlie, Age: 35, City: Chicago
Note: We cannot use a union for this task because a union can only hold one of its members at a time,
while we need to store multiple fields (name, age, city) simultaneously for each person.
```

Bidirectional List

```
Select a task:
1. People Structure
2. Bidirectional List
3. Family Tree
4. Sorted Person List
>> 2
Bidirectional list values from end to start: 10 9 8 7 6 5 4 3 2 1 0
```

Family Tree

```
Select a task:
1. People Structure
2. Bidirectional List
3. Family Tree
4. Sorted Person List
>> 3
Child: Child
Father: Father
Mother: Mother
```

Sorted Person List

```
Select a task:
1. People Structure
2. Bidirectional List
3. Family Tree
4. Sorted Person List
>> 4
Sorted list of persons:
Name: Alice, Age: 18
Name: Charlie, Age: 21
Name: Eve, Age: 27
Name: Bob, Age: 35
Name: David, Age: 44
```

Enhancements & Improvements

Error Handling:

- Implement error handling for invalid inputs (e.g., non-integer values when entering ages).

Dynamic Input:

- Allow users to dynamically specify how many people they want to input in `PeopleStructure`.

Additional Features:

- Extend functionality in `SortedPersonList` class to allow users to remove persons from the list.

- Add sorting capabilities in `BidirectionalList` based on node values if required.

Conclusion

This program effectively demonstrates object-oriented programming principles such as inheritance and polymorphism in C++. Each task is encapsulated within its respective class, making it modular and easy to extend or modify in future iterations. With enhancements, this program can become even more user-friendly and robust in handling various data structures and operations.