

# Lab 1

---

## Lab 1

This lab will focus on using polymorphism while interacting with a contact list. There will be a main class `Contacts` that controls the “view” the user sees and responds to user input. The contact information (personal and work) will be an instance of the `Information` class.

## The Contacts Class

This lab is built around the `Contacts` class, which has four attributes:

- \* `view` - This string attribute controls what the user sees. When the value for `view` changes, the information changes. There are four different views.
  - \* `List view` - Shows the list of all of the contacts.
  - \* `Information view` - Shows the work and personal information for a particular contact.
  - \* `Add view` - Add information for a new contact.
  - \* `Quit view` - Leave a message for the user and then end the program.
  - \* `names` - Vector of strings that stores the names of each person in the contact list.
  - \* `titles` - Vector of strings that stores the titles for each person in the contact list.
  - \* `workPhoneNumbers` - Vector of strings that stores the work phone numbers for each person in the contact list.
  - \* `workEmails` - Vector of strings that stores the work email addresses for each person in the contact list.
  - \* `personalPhoneNumbers` - Vector of strings that stores the personal phone numbers for each person in the contact list.
  - \* `personalEmails` - Vector of strings that stores the personal email addresses for each person in the contact list.
- 
- `choice` - This string attribute represents input from the user and is used to change `view`.
  - `index` - This integer attribute keeps track of the particular contact whose information is to be displayed.
  - `length` - This integer attribute keeps track of the length of the above vectors.

To put polymorphism into practice, we are going to create the abstract class `Information`. This will have the pure abstract functions `DisplayInfo` and `AddInfo`. The `Contacts` class inherits from `Information`, and therefore it

must override the DisplayInfo and AddInfo functions.

```
//add class definitions below this line

class Information {
public:
    virtual void DisplayInfo() = 0;
    virtual void AddInfo() = 0;
};

class Contacts : public Information {
public:
    void DisplayInfo() {

    }

    void AddInfo() {

    }
};

//add class definitions above this line
```

Add the attributes and constructor for the Contacts class. For testing purposes, set view to "quit". We will change this to a more appropriate value later on. The other attributes do not need a value when instantiating the object. Instantiate each vector attribute, set choice to an empty string, and set index and length to 0. Additionally, add the Display function, which we will go in more detail later on.

```
//add class definitions below this line

class Information {
public:
    virtual void DisplayInfo() = 0;
    virtual void AddInfo() = 0;
};

class Contacts : public Information {
public:
    Contacts() {
        view = "quit";
        names = {};
        titles = {};
        workPhoneNumbers = {};
        workEmails = {};
        personalPhoneNumbers = {};
    }
};
```

```

        personalEmails = {};
        choice = "";
        index = 0;
        length = 0;
    }

    void DisplayInfo() {

    }

    void AddInfo() {

    }

    void Display() {

    }

private:
    string view;
    vector<string> names;
    vector<string> titles;
    vector<string> workPhoneNumbers;
    vector<string> workEmails;
    vector<string> personalPhoneNumbers;
    vector<string> personalEmails;
    string choice;
    int index;
    int length;
};

//add class definitions above this line

```

## The Display Function

The Display function is designed to be a loop that runs until the user tells the program to end. The function checks the value of the `view` attribute and calls the appropriate function that displays the information for each view. Since the loop is `while (true)`, be sure to include a `break` statement otherwise the loop would never stop (C++ would eventually stop the program with an error message). Fill out the Display function like below.

```

void Display() {
    while (true) {
        if (view=="list") {
        }
        else if (view=="info") {
            DisplayInfo();
        }
        else if (view=="add") {
            cout << " " << endl;
            AddInfo();
        }
        else if (view=="quit") {
            cout << ("\nClosing the contact list...\n");
            break;
        }
    }
}

```

## Starting the Other Functions

The Display function calls three other functions; DisplayInfo and AddInfo have already been declared. Trying to test the code would cause your program to crash as the ShowList function has not yet been defined. Create another empty function ShowList just as was done with DisplayInfo and AddInfo. We will come back later and add working code to each function. However, to get your code to run we must provide a definition for virtual functions in your base class.

```

void DisplayInfo() {

}

void AddInfo() {

}

void ShowList() {

}

```

## Testing Your Code

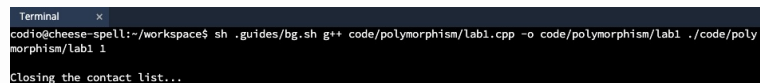
Before moving on to the next part of the script, we want to check that our code is working. To do that, instantiate a Contacts object and call the Display function.

```
//add code below this line

Contacts contacts;
contacts.Display();

//add code above this line
```

Run your program. Because the view attribute is "quit", the script should immediately display a message and then stop. Your output should look something like this.



```
Terminal
codio@cheese-spell:~/workspace$ sh .guides/bg.sh g++ code/polymorphism/lab1.cpp -o code/polymorphism/lab1 ./code/polymorphism/lab1 1
Closing the contact list...
```

[.guides/img/Polymorphism/output\\_lab1\\_updated](#)

#### ▼ Code

Your code should look like this:

```
#include <iostream>
#include <vector>
using namespace std;

class Information {
public:
    virtual void DisplayInfo() = 0;
    virtual void AddInfo() = 0;
};

class Contacts : public Information {
public:
    Contacts() {
        view = "quit";
        names = {};
        titles = {};
        workPhoneNumbers = {};
        workEmails = {};
        personalPhoneNumbers = {};
        personalEmails = {};
        choice = "";
        index = 0;
        length = 0;
    }
}
```

```

void DisplayInfo() {

}

void AddInfo() {

}

void ShowList() {

}

void Display() {
    while (true) {
        if (view=="list") {
            ShowList();
        }
        else if (view=="info") {
            DisplayInfo();
        }
        else if (view=="add") {
            cout<< " "<<endl;
            AddInfo();
        }
        else if (view=="quit") {
            cout<< ("\nClosing the contact list...\n");
            break;
        }
    }
}

private:
    string view;
    vector<string> names;
    vector<string> titles;
    vector<string> workPhoneNumbers;
    vector<string> workEmails;
    vector<string> personalPhoneNumbers;
    vector<string> personalEmails;
    string choice;
    int index;
    int length;
};

int main(){

```

*//add code below this line*

```
Contacts contacts;  
contacts.Display();
```

*//add code above this line*

```
return 0;  
}
```

# Lab 2

---

## Adding a Contact

The first thing we need to do is to change the default view when a Contacts object is instantiated. The list view should be the first view shown to the user. Change the value for view from "quit" to "list" in the constructor. The rest of the constructor should not be changed.

### ▼ Code from Lab 1

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Information {
public:
    virtual void DisplayInfo() = 0;
    virtual void AddInfo() = 0;
};

class Contacts : public Information {
public:
    Contacts() {
        view = "quit";
        names = {};
        titles = {};
        workPhoneNumbers = {};
        workEmails = {};
        personalPhoneNumbers = {};
        personalEmails = {};
        choice = "";
        index = 0;
        length = 0;
    }

    void DisplayInfo() {

    }
```



```

void AddInfo() {

}

void ShowList() {

}

void Display() {
    while (true) {
        if (view=="list") {
            ShowList();
        }
        else if (view=="info") {
            DisplayInfo();
        }
        else if (view=="add") {
            cout<< " "<<endl;
            AddInfo();
        }
        else if (view=="quit") {
            cout << ("\nClosing the contact list...\n");
            break;
        }
    }
}

private:
    string view;
    vector<string> names;
    vector<string> titles;
    vector<string> workPhoneNumbers;
    vector<string> workEmails;
    vector<string> personalPhoneNumbers;
    vector<string> personalEmails;
    string choice;
    int index;
    int length;
};

//add class definitions above this line

int main(){

    //add code below this line

    Contacts contacts;
    contacts.Display();

```

```

        //add code above this line

    return 0;

}

```

```

public:
    Contacts() {
        view = "list";
        // rest of the constructor remains unchanged
    }

```

Next we want to modify the ShowList function to show the list of people in the contact list. There are two possible states for the list view: the list is empty or there are contacts in the list. When the list is empty, the user will be provided with the choice to add a contact or quit the program. Use a conditional to represent these two states. For now, set view to "quit" in the else branch.

The print statement is to add a blank line for legibility. We also need a cin object to collect input from the user. If length is 0, then present the user with a choice. Store their input in choice. The .tolower() function will convert the user choice to a lowercase letter. This will make comparisons easier. Remember, C++ is case sensitive; q and Q are not the same. By forcing all input to lowercase, we only need to test for the lowercase letter. The ShowList function ends by calling another function to handle the user's choice.

```

void ShowList() {
    cout << endl;
    char sc;
    if (length == 0) {
        cout << "(A)dd a new contact \n(Q)uit \n> ";
        cin >> sc;
        choice = putchar(tolower(sc));
    }
    else {
        view = "quit";
    }
    HandleChoice();
}

```

## Handling User Choices

Every time the user makes a choice, we want to evaluate that choice and perform the appropriate action. In this case, the user can choose between adding a contact or quitting the program. Notice that view only changes to "add" if "a" is entered **and** we are in list view. We only want to add new contacts from the list view.

```
void HandleChoice() {  
    if (choice=="q") {  
        view = "quit";  
    }  
    else if (choice=="a" && view=="list") {  
        view = "add";  
    }  
}
```

## Adding a Contact

When the user enters "a", we need to create a new contact. To do this, we are going to modify the AddInfo function. Use cout and the getline function to ask the user to enter the name, personal phone number, personal email, work title, work phone number, and work email, and then temporarily store that information. Each piece of information should go into the corresponding vector attribute after. Notice that we include cin >> ws within getline because by doing so, we are able to consume the whitespace ws. Once the information has been added, increase the length attribute and revert back to the list view.

```

void AddInfo() {
    cout << endl;
    string sc2;
    cout << ("Enter their name: ");
    getline(cin >> ws, sc2);
    string name = sc2;
    names.push_back(name);

    cout << ("Enter their personal phone number: ");
    getline(cin >> ws, sc2);
    string personalPhone = sc2;
    personalPhoneNumbers.push_back(personalPhone);

    cout << ("Enter their personal email: ");
    getline(cin >> ws, sc2);
    string personalEmail = sc2;
    personalEmails.push_back(personalEmail);

    cout << ("Enter their work title: ");
    getline(cin >> ws, sc2);
    string title = sc2;
    titles.push_back(title);

    cout << ("Enter their work phone number: ");
    getline(cin >> ws, sc2);
    string workPhone = sc2;
    workPhoneNumbers.push_back(workPhone);

    cout << ("Enter their work email: ");
    getline(cin >> ws, sc2);
    string workEmail = sc2;
    workEmails.push_back(workEmail);
    length++;
    view = "list";
}

```

## Testing Your Code

Before moving on to the next part of the program, we want to check that our code is adding a contact to the list. To do that, we need to create a getter for the length attribute.

```
int GetLength() {  
    return length;  
}
```

Now call print the result from the GetLength function.

```
//add code below this line  
  
Contacts contacts;  
contacts.Display();  
cout<< contacts.GetLength()<< endl;  
  
//add code above this line
```

Run the program and enter a when prompted, then add the following contact:

```
Rachel Kim  
555 123-4567  
rachel_k@personalMail.com  
Senior Software Engineer  
555 890-1234  
rkim@workMail.com
```

If everything worked properly, your program should print 1 in the terminal as there is one person in our contact list.

#### ▼ Code

Your code should look like this:

```
#include <iostream>  
#include <vector>  
using namespace std;  
  
//add class definitions below this line  
  
class Information {  
    public:  
        virtual void DisplayInfo() = 0;  
        virtual void AddInfo() = 0;  
};  
  
class Contacts : public Information {  
    public:
```

```

Contacts() {
    view = "list";
    names = {};
    titles = {};
    workPhoneNumbers = {};
    workEmails = {};
    personalPhoneNumbers = {};
    personalEmails = {};
    choice = "";
    index = 0;
    length = 0;
}

void DisplayInfo() {

}

void AddInfo() {
    cout << endl;
    string sc2;
    cout << ("Enter their name: ");
    getline(cin >> ws, sc2);
    string name = sc2;
    names.push_back(name);

    cout << ("Enter their personal phone number: ");
    getline(cin >> ws, sc2);
    string personalPhone = sc2;
    personalPhoneNumbers.push_back(personalPhone);

    cout << ("Enter their personal email: ");
    getline(cin >> ws, sc2);
    string personalEmail = sc2;
    personalEmails.push_back(personalEmail);

    cout << ("Enter their work title: ");
    getline(cin >> ws, sc2);
    string title = sc2;
    titles.push_back(title);

    cout << ("Enter their work phone number: ");
    getline(cin >> ws, sc2);
    string workPhone = sc2;
    workPhoneNumbers.push_back(workPhone);

    cout << ("Enter their work email: ");
    getline(cin >> ws, sc2);
    string workEmail = sc2;

```

```

        workEmails.push_back(workEmail);
        length++;
        view = "list";
    }

    int GetLength() {
        return length;
    }

    void HandleChoice() {
        if (choice=="q") {
            view = "quit";
        }
        else if (choice=="a" && view=="list") {
            view = "add";
        }
    }

    void ShowList() {
        cout << endl;
        char sc;
        if (length == 0) {
            cout << "(A)dd a new contact \n(Q)uit \n> ";
            cin >> sc;
            choice = putchar(tolower(sc));
        }
        else {
            view = "quit";
        }
        HandleChoice();
    }

    void Display() {
        while (true) {
            if (view=="list") {
                ShowList();
            }
            else if (view=="info") {
                DisplayInfo();
            }
            else if (view=="add") {
                cout << endl;
                AddInfo();
            }
            else if (view=="quit") {
                cout << "\nClosing the contact list...\n";
                break;
            }
        }
    }

```

```

    }
    }
}

private:
    string view;
    vector<string> names;
    vector<string> titles;
    vector<string> workPhoneNumbers;
    vector<string> workEmails;
    vector<string> personalPhoneNumbers;
    vector<string> personalEmails;
    string choice;
    int index;
    int length;
};

//add class definitions above this line

int main(){

    //add code below this line

    Contacts contacts;
    contacts.Display();
    cout << contacts.GetLength() << endl;

    //add code above this line

    return 0;

}

```



# Lab 3

---

## Displaying the List View

Now that we can add a contact to the list, we will want to show all of the contacts in the list.

### ▼ Code from Lab 2

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Information {
public:
    virtual void DisplayInfo() = 0;
    virtual void AddInfo() = 0;
};

class Contacts : public Information {
public:
    Contacts() {
        view = "list";
        names = {};
        titles = {};
        workPhoneNumbers = {};
        workEmails = {};
        personalPhoneNumbers = {};
        personalEmails = {};
        choice = "";
        index = 0;
        length = 0;
    }

    void DisplayInfo() {

    }

    void AddInfo() {
        cout << endl;
    }
}
```

```

string sc2;
cout << ("Enter their name: ");
getline(cin >> ws, sc2);
string name = sc2;
names.push_back(name);

cout << ("Enter their personal phone number: ");
getline(cin >> ws, sc2);
string personalPhone = sc2;
personalPhoneNumbers.push_back(personalPhone);

cout << ("Enter their personal email: ");
getline(cin >> ws, sc2);
string personalEmail = sc2;
personalEmails.push_back(personalEmail);

cout << ("Enter their work title: ");
getline(cin >> ws, sc2);
string title = sc2;
titles.push_back(title);

cout << ("Enter their work phone number: ");
getline(cin >> ws, sc2);
string workPhone = sc2;
workPhoneNumbers.push_back(workPhone);

cout << ("Enter their work email: ");
getline(cin >> ws, sc2);
string workEmail = sc2;
workEmails.push_back(workEmail);
length++;
view = "list";
}

int GetLength() {
    return length;
}

void HandleChoice() {
    if (choice=="q") {
        view = "quit";
    }
    else if (choice=="a" && view=="list") {
        view = "add";
    }
}

```

```

void ShowList() {
    cout << endl;
    char sc;
    if (length == 0) {
        cout << "(A)dd a new contact \n(Q)uit \n> ";
        cin >> sc;
        choice = putchar(tolower(sc));
    }
    else {
        view = "quit";
    }
    HandleChoice();
}

void Display() {
    while (true) {
        if (view=="list") {
            ShowList();
        }
        else if (view=="info") {
            DisplayInfo();
        }
        else if (view=="add") {
            cout << endl;
            AddInfo();
        }
        else if (view=="quit") {
            cout << ("\nClosing the contact list...\n");
            break;
        }
    }
}

private:
    string view;
    vector<string> names;
    vector<string> titles;
    vector<string> workPhoneNumbers;
    vector<string> workEmails;
    vector<string> personalPhoneNumbers;
    vector<string> personalEmails;
    string choice;
    int index;
    int length;
};

```

*//add class definitions above this line*

```

int main(){

    //add code below this line

    Contacts contacts;
    contacts.Display();
    cout << contacts.GetLength() << endl;

    //add code above this line

    return 0;

}

```

But before doing that, we need to remove the print statement at the end of the script. The final two lines of code should look like this:

```

//add code below this line

Contacts contacts;
contacts.Display();

//add code above this line

```

We are going to iterate through the list of contacts and print the name. To help users select a contact, a number will appear before each name. This way, the user types the number, and the contact's information appears. Previously, the `cin` object stored the user input as a `char`, but since we want the user to be able to input numbers now, we have to change the input `sc` to a string instead. Then we create a loop in which `sc` gets iterated and each character will be converted into lowercase and added to the string `choice`. In the `else` branch function, create a `for` loop to go from `0` to `length`. We want a number followed by the name. The numbers should start at `1`, so print the loop index plus `1` followed by the element from the `names` vector. After displaying the list of names, ask the user to make a selection. Entering a number will show all of the information about a contact. After `HandleChoice` is called, be sure to set `choice` back to an empty string; otherwise, the user's input will continue to be stored in `choice` and negatively affect the acceptable input requirement.

```

void ShowList() {
    cout << endl;
    string sc;
    if (length == 0) {
        cout << "(A)dd a new contact \n(Q)uit \n> ";
        cin >> sc;
        for (char c : sc) {
            choice += (tolower(c));
        }
    }
    else {
        for (int i = 0; i < length; i++) {
            cout << (i + 1) << " " << names.at(i) << endl;
        }
        cout << ("\n(#) Select a name \n(A)dd a new\ncontact\n(Q)uit \n> ");
        cin >> sc;
        for (char c : sc) {
            choice += (tolower(c));
        }
    }
    HandleChoice();
    choice = "";
}

```

## Handling Numeric Input

Add an `else if` branch to the `HandleChoice` function that asks if the user input is numeric (remember, the `cin` object now stores user input as a string) and if the user is in the list view. If yes, then convert the user input to an integer, subtract 1, and store it in the variable `num`. Remember, we added one to the loop index in the `ShowList` function. If the user made a mistake in entering the number, the script will crash if you try to access an index that is outside of the vector. So we need to verify that the number is between 0 and `length`. Finally, set `index` to `num` and set `view` to "info".

```

void HandleChoice() {
    if (choice=="q") {
        view = "quit";
    }
    else if (choice=="a" && view=="list") {
        view = "add";
    }
    else if (IsNumeric(choice) && view=="list") {
        int num = (stoi(choice) - 1);
        if (num >= 0 && num < length) {
            index = num;
            view = "info";
        }
    }
}

```

We need to create the boolean helper function `IsNumeric` which takes the user input and determines if they entered a number. First see if the user input (a string) is empty. Return `false` if either condition is true. Then try to convert the string to an integer. If this works, return `true`. If C++ throws an exception because the string cannot be converted to an integer, then return `false`. Since `IsNumeric` is a helper function, you should encapsulate it as `private`.

```

bool IsNumeric(string s) {
    int value;

    if (s == "") {
        return false;
    }

    try {
        value = stoi(s);
        return true;
    }
    catch (runtime_error& e) {
        return false;
    }
}

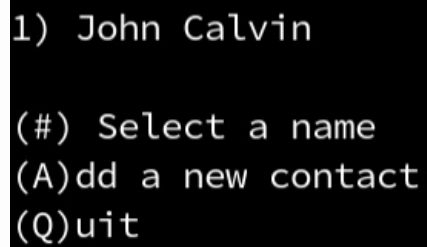
```

## Testing Your Code

Before moving on to the next part of the program, we want to check that our code is displaying all of the contacts in the list. To do that, enter two different contacts. The first one is:

```
John Calvin  
555 111-2222  
john.calvin@email.net  
Philosopher  
555 333-4444  
jcalvin@work.org
```

You should see a list that looks like this:



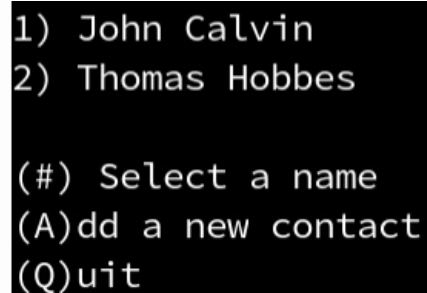
```
1) John Calvin  
  
(#) Select a name  
(A)dd a new contact  
(Q)uit
```

[.guides/img/Polymorphism/lab3\\_pic1](#)

Now add a second contact to the list:

```
Thomas Hobbes  
555 666-7777  
t_hobbes@email.net  
Philosopher  
555 888-9999  
tom_hobbes@work.org
```

Your program should now show the following output:



```
1) John Calvin  
2) Thomas Hobbes  
  
(#) Select a name  
(A)dd a new contact  
(Q)uit
```

[.guides/img/Polymorphism/lab3\\_pic2](#)

#### ▼ Code

Your code should look like this:

```
#include <iostream>  
#include <vector>  
using namespace std;
```

*//add class definitions below this line*

```
class Information {
public:
    virtual void DisplayInfo() = 0;
    virtual void AddInfo() = 0;
};

class Contacts : public Information {
public:
    Contacts() {
        view = "list";
        names = {};
        titles = {};
        workPhoneNumbers = {};
        workEmails = {};
        personalPhoneNumbers = {};
        personalEmails = {};
        choice = "";
        index = 0;
        length = 0;
    }

    void DisplayInfo() {

    }

    void AddInfo() {
        cout << endl;
        string sc2;
        cout << ("Enter their name: ");
        getline(cin >> ws, sc2);
        string name = sc2;
        names.push_back(name);

        cout << ("Enter their personal phone number: ");
        getline(cin >> ws, sc2);
        string personalPhone = sc2;
        personalPhoneNumbers.push_back(personalPhone);

        cout << ("Enter their personal email: ");
        getline(cin >> ws, sc2);
        string personalEmail = sc2;
        personalEmails.push_back(personalEmail);

        cout << ("Enter their work title: ");
        getline(cin >> ws, sc2);
        string title = sc2;
```



```

titles.push_back(title);

cout << ("Enter their work phone number: ");
getline(cin >> ws, sc2);
string workPhone = sc2;
workPhoneNumbers.push_back(workPhone);

cout << ("Enter their work email: ");
getline(cin >> ws, sc2);
string workEmail = sc2;
workEmails.push_back(workEmail);
length++;
view = "list";
}

int GetLength() {
    return length;
}

void HandleChoice() {
    if (choice=="q") {
        view = "quit";
    }
    else if (choice=="a" && view=="list") {
        view = "add";
    }
    else if (IsNumeric(choice) && view=="list") {
        int num = (stoi(choice) - 1);
        if (num >= 0 && num < length) {
            index = num;
            view = "info";
        }
    }
}

void ShowList() {
    cout << endl;
    string sc;
    if (length == 0) {
        cout << ("Add a new contact \n(Q)uit \n> ");
        cin >> sc;
        for (char c : sc) {
            choice += (tolower(c));
        }
    }
    else {
        for (int i = 0; i < length; i++) {
            cout << (i + 1) << " " << names.at(i) << endl;

```

```

    }
    cout << ("\n(#) Select a name \n(A)dd a new
    contact\n(Q)uit \n> ");
    cin >> sc;
    for (char c : sc) {
        choice += (tolower(c));
    }
}
HandleChoice();
choice = "";
}

void Display() {
    while (true) {
        if (view=="list") {
            ShowList();
        }
        else if (view=="info") {
            DisplayInfo();
        }
        else if (view=="add") {
            cout << endl;
            AddInfo();
        }
        else if (view=="quit") {
            cout << ("\nClosing the contact list...\n");
            break;
        }
    }
}
}

```

```

private:
    string view;
    vector<string> names;
    vector<string> titles;
    vector<string> workPhoneNumbers;
    vector<string> workEmails;
    vector<string> personalPhoneNumbers;
    vector<string> personalEmails;
    string choice;
    int index;
    int length;

    bool IsNumeric(string s) {
        int value;

        if (s == "") {
            return false;
        }
    }
}

```

```
    }

    try {
        value = stoi(s);
        return true;
    }
    catch (runtime_error& e) {
        return false;
    }
}

};

//add class definitions above this line

int main(){

    //add code below this line

    Contacts contacts;
    contacts.Display();

    //add code above this line

    return 0;

}
```

# Lab 4

## Displaying Contact Info

The next step is to display the contact information for a selected contact.

### ▼ Code from Lab 3

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Information {
public:
    virtual void DisplayInfo() = 0;
    virtual void AddInfo() = 0;
};

class Contacts : public Information {
public:
    Contacts() {
        view = "list";
        names = {};
        titles = {};
        workPhoneNumbers = {};
        workEmails = {};
        personalPhoneNumbers = {};
        personalEmails = {};
        choice = "";
        index = 0;
        length = 0;
    }

    void DisplayInfo() {

    }

    void AddInfo() {
        cout<< endl;
        string sc2;
        cout<<("Enter their name: ");
```

```

getline(cin >> ws, sc2);
string name = sc2;
names.push_back(name);

cout<<("Enter their personal phone number: ");
getline(cin >> ws, sc2);
string personalPhone = sc2;
personalPhoneNumbers.push_back(personalPhone);

cout<<("Enter their personal email: ");
getline(cin >> ws, sc2);
string personalEmail = sc2;
personalEmails.push_back(personalEmail);

cout<<("Enter their work title: ");
getline(cin >> ws, sc2);
string title = sc2;
titles.push_back(title);

cout<<("Enter their work phone number: ");
getline(cin >> ws, sc2);
string workPhone = sc2;
workPhoneNumbers.push_back(workPhone);

cout<<("Enter their work email: ");
getline(cin >> ws, sc2);
string workEmail = sc2;
workEmails.push_back(workEmail);
length++;
view = "list";
}

int GetLength() {
    return length;
}

void HandleChoice() {
    if (choice=="q") {
        view = "quit";
    }
    else if (choice=="a" && view=="list") {
        view = "add";
    }
    else if (IsNumeric(choice) && view=="list") {
        int num = (stoi(choice) - 1);
        if (num >= 0 && num < length) {
            index = num;
            view = "info";
        }
    }
}

```

```

    }
}

void ShowList() {
    cout<< endl;
    string sc;
    if (length == 0) {
        cout<< "(A)dd a new contact \n(Q)uit \n> ";
        cin >> sc;
        for (char c : sc) {
            choice += (tolower(c));
        }
    }
    else {
        for (int i = 0; i < length; i++) {
            cout<< (i + 1) << " " << names.at(i)<<endl;
        }
        cout<< ("\n(#) Select a name \n(A)dd a new contact\n(Q)uit \n> ");
        cin >> sc;
        for (char c : sc) {
            choice += (tolower(c));
        }
    }
    HandleChoice();
    choice = "";
}

void Display() {
    while (true) {
        if (view=="list") {
            ShowList();
        }
        else if (view=="info") {
            DisplayInfo();
        }
        else if (view=="add") {
            cout<< endl;
            AddInfo();
        }
        else if (view=="quit") {
            cout<< ("\nClosing the contact list...\n");
            break;
        }
    }
}

```

```

private:
    string view;
    vector<string> names;
    vector<string> titles;
    vector<string> workPhoneNumbers;
    vector<string> workEmails;
    vector<string> personalPhoneNumbers;
    vector<string> personalEmails;
    string choice;
    int index;
    int length;

    bool IsNumeric(string s) {
        int value;

        if (s == "") {
            return false;
        }

        try {
            value = stoi(s);
            return true;
        }
        catch (runtime_error& e) {
            return false;
        }
    }
};

//add class definitions above this line

int main(){

    //add code below this line

    Contacts contacts;
    contacts.Display();

    //add code above this line

    return 0;

}

```

On the previous page, we already modified `HandleChoice` to deal with numeric input. So let's update the `DisplayInfo` function to display the information. Print the elements from each vector using the attribute `index`. After the information is displayed, change view back to "list" and then call `ShowList`.

```
void DisplayInfo() {
    cout<< endl;
    cout<<names.at(index)<<endl;
    cout<<"Personal email address: " << personalEmails.at(index)
        <<endl;
    cout<<"Personal phone number: " <<
        personalPhoneNumbers.at(index)<<endl;
    cout<<"Work title: " << titles.at(index)<<endl;
    cout<<"Work email address: " << workEmails.at(index)<<endl;
    cout<<"Work phone number: " << workPhoneNumbers.at(index)
        <<endl;

    view = "list";
    ShowList();
}
```

## Testing Your Code

This program should be complete now. To test it, add at least two contacts. Select one of the contacts by entering the correlated number. Keep entering a number to display the relevant contact information, a to add additional contacts, or q to quit the program.

### ▼ Code

Your code should look like this:

```
#include <iostream>
#include <vector>
using namespace std;

//add class definitions below this line

class Information {
public:
    virtual void DisplayInfo() = 0;
    virtual void AddInfo() = 0;
};

class Contacts : public Information {
public:
    Contacts() {
        view = "list";
```



```

names = {};
titles = {};
workPhoneNumbers = {};
workEmails = {};
personalPhoneNumbers = {};
personalEmails = {};
choice = "";
index = 0;
length = 0;
}

void DisplayInfo() {
    cout<< endl;
    cout<<names.at(index)<<endl;
    cout<<"Personal email address: " <<
        personalEmails.at(index)<<endl;
    cout<<"Personal phone number: " <<
        personalPhoneNumbers.at(index)<<endl;
    cout<<"Work title: " << titles.at(index)<<endl;
    cout<<"Work email address: " << workEmails.at(index)
        <<endl;
    cout<<"Work phone number: " << workPhoneNumbers.at(index)
        <<endl;
    view = "list";
    ShowList();
}

void AddInfo() {
    cout<< endl;
    string sc2;
    cout<<("Enter their name: ");
    getline(cin >> ws, sc2);
    string name = sc2;
    names.push_back(name);

    cout<<("Enter their personal phone number: ");
    getline(cin >> ws, sc2);
    string personalPhone = sc2;
    personalPhoneNumbers.push_back(personalPhone);

    cout<<("Enter their personal email: ");
    getline(cin >> ws, sc2);
    string personalEmail = sc2;
    personalEmails.push_back(personalEmail);

    cout<<("Enter their work title: ");
    getline(cin >> ws, sc2);
    string title = sc2;
    titles.push_back(title);
}

```

```

        cout<<("Enter their work phone number: ");
        getline(cin >> ws, sc2);
        string workPhone = sc2;
        workPhoneNumbers.push_back(workPhone);

        cout<<("Enter their work email: ");
        getline(cin >> ws, sc2);
        string workEmail = sc2;
        workEmails.push_back(workEmail);
        length++;
        view = "list";
    }

    int GetLength() {
        return length;
    }

    void HandleChoice() {
        if (choice=="q") {
            view = "quit";
        }
        else if (choice=="a" && view=="list") {
            view = "add";
        }
        else if (IsNumeric(choice) && view=="list") {
            int num = (stoi(choice) - 1);
            if (num >= 0 && num < length) {
                index = num;
                view = "info";
            }
        }
    }

    void ShowList() {
        cout<< endl;
        string sc;
        if (length == 0) {
            cout<< ("(A)dd a new contact \n(Q)uit \n> ");
            cin >> sc;
            for (char c : sc) {
                choice += (tolower(c));
            }
        }
        else {
            for (int i = 0; i < length; i++) {
                cout<< (i + 1) << " " << names.at(i)<<endl;
            }
        }
    }

```

```

        cout<< ("\n(#) Select a name \n(A)dd a new
        contact\n(Q)uit \n> ");
        cin >> sc;
        for (char c : sc) {
            choice += (tolower(c));
        }
    }
    HandleChoice();
    choice = "";
}

void Display() {
    while (true) {
        if (view=="list") {
            ShowList();
        }
        else if (view=="info") {
            DisplayInfo();
        }
        else if (view=="add") {
            cout<< endl;
            AddInfo();
        }
        else if (view=="quit") {
            cout<< ("\nClosing the contact list...\n");
            break;
        }
    }
}
}

```

```

private:
    string view;
    vector<string> names;
    vector<string> titles;
    vector<string> workPhoneNumbers;
    vector<string> workEmails;
    vector<string> personalPhoneNumbers;
    vector<string> personalEmails;
    string choice;
    int index;
    int length;

    bool IsNumeric(string s) {
        int value;

        if (s == "") {
            return false;
        }
    }
}

```

```
    }

    try {
        value = stoi(s);
        return true;
    }
    catch (runtime_error& e) {
        return false;
    }
}

};

//add class definitions above this line

int main(){

    //add code below this line

    Contacts contacts;
    contacts.Display();

    //add code above this line

    return 0;

}
```

# Lab Challenge

---

## Problem

In the IDE to the left, the class Chef is already defined, as is the Display function. However, it does not have a constructor. Create three constructors that take one, two, and three parameters respectively. **Note** that the attributes name and cuisine are set to "null" while michelinStars is set to 0 by default.

### ▼ Given Code

```
#include <iostream>
using namespace std;

//add class definitions below this line

class Chef {
public:

    //add constructors below this line

    //add constructors above this line

    string GetName() {
        return name;
    }

    string GetCuisine() {
        return cuisine;
    }

    int GetStars() {
        return michelinStars;
    }

    void Display() {
        cout << GetName() << " is known for " << GetCuisine() << "
            cuisine and has " << GetStars() << " Michelin stars." <<
            endl;
    }
}
```

```

private:
    string name = "null";
    string cuisine = "null";
    int michelinStars = 0;
};

//add class definitions above this line

int main() {

    //DO NOT EDIT code below this line

    Chef c1("Marco Pierre White");
    Chef c2("Rene Redzepi", "Nordic");
    Chef c3("Thomas Keller", "French", 3);

    c1.Display();
    c2.Display();
    c3.Display();

    //DO NOT EDIT code above this line

    return 0;

}

```

## Expected Output

Three Chef objects are instantiated, each one using a different constructor. Calling the Display function for each object should return the following text:

| Function Call | Return Value   |
|---------------|--|
| c1.Display()  | Marco Pierre White is known for null cuisine and has 0 Michelin stars. |
| c2.Display()  | Rene Redzepi is known for Nordic cuisine and has 0 Michelin stars.     |
| c3.Display()  | Thomas Keller is known for French cuisine and has 3 Michelin stars.    |