Main Function

```
int main()
{
    while (true)
    {
       cout << "\nOptions:\n";
       cout << "1. Add\n";
       cout << "2. Delete\n";
       cout << "3. Search\n";
       cout << "4. Display\n";
       cout << "5. Exit\n";

       int c;
       char name_c[256];
       char number_c[256];
       cin >> c;

       switch (c)
       {
       case 1:
       {
             cout << "\nType person data:\n";

             cin.ignore(1, '\n');

             cout << "\nName:";
             cin.getline(name_c, sizeof(name_c));

             cout << "\nNumber:";
             cin.getline(number_c, sizeof(number_c));

             string name = name_c;
             string number = number_c;

             p person;
             person.name = name;
             person.phone = number;

             add_dup(name, person);
             add_dup(number, person);
```

```
            break;
        }
        case 2:
        {
            cout << "\nType value:\n";
        }
        case 3:
        {
            string blank = "";
            blank = inputName(blank);

            break;
        }
        case 4:
        {
            cout << "\nLinked list:\n";
            for (auto& item : map)
            {
                item.second.print_numbered = true;
                item.second.print_prefix = "\t";
                cout << "Map key: " << item.first << "\tMap entry: \n"
<< item.second;
            }
            break;
        }
        case 5:
        {
            return 0;
            break;
        }
        }
    }
}
```

In the main function, input arrays for the name and number of the entries are declared. The user is then prompted to input the desired operation from the available options. After the operation is performed, the function loops to receive user input again.

Add Data Functionality

```
	case 1:
	{
		cout << "\nType person data:\n";

		cin.ignore(1, '\n');

		cout << "\nName:";
		cin.getline(name_c, sizeof(name_c));

		cout << "\nNumber:";
		cin.getline(number_c, sizeof(number_c));

		string name = name_c;
		string number = number_c;

		p person;
		person.name = name;
		person.phone = number;

		add_dup(name, person);
		add_dup(number, person);

		break;
	}
```

Add Data Functionality

When the user types 1, the Add functionality is carried out. First, the user is prompted to type in a name and number for the new contact. The use of getline ensures that the entire line the user enters is used. Then, a person node is created, and the variables name and phone number are assigned. Finally, the person node is added to an unordered_map using the function add_dup. Both the name and number are used as keys so they can both be searched for later.

```
void add_dup(string name, p& person)
{
    int i = 0;
```

```
    string tmp = "";
    while (name[i] != '\0')
    {
       tmp += name[i];
       map[tmp].add_back(person);
       i++;
    }
}
```

When the add_dup function is called, an iterative loop is started. For each pass of the loop the next character of the name is added to a temporary string. This string is then used to add another entry in the unordered_map for that person. This results in there being an entry in the unordered_map for each additional letter of the name e.g. one for D, Da and Dan. The linked list from the map then has a new node attached to the back of it.

This is quite inefficient, e.g. given contact name: daVID, phone number: 1000, a total of 5 + 4 linked lists are modified/made:

1. map[“d”]
2. map[“da”]
3. map[“dav”]
4. map[“davi”]
5. map[“david”]

And for the numbers:

6. map[“1”]
7. map[“10”]
8. map[“100”]
9. map[“1000”]

And to each of these lists, we add a node containing the data contact name: daVID, phone number: 1000.

This however does make the later functionalities much easier to implement.

Delete Data Functionality

```
       case 2:
       {
             cout << "\nType value:\n";
       }
       case 3:
       {
             string blank = "";
```

```
            blank = inputName(blank);

            break;
      }
      case 4:
      {
            cout << "\nLinked list:\n";
            for (auto& item : map)
            {
                  item.second.print_numbered = true;
                  item.second.print_prefix = "\t";
                  cout << "Map key: " << item.first << "\tMap entry: \n"
<< item.second;
            }
            break;
      }
      case 5:
      {
            return 0;
            break;
      }
      }
   }
}
```

The delete functionality calls inputName to allow the user to input the name character by character. Then the user can select which index of the displayed contacts to delete.

Search Functionality

```
       case 3:
       {
              string blank = "";
              blank = inputName(blank);

              break;
       }


Search functionality
```

When the user enters 3, the search functionality is carried out. The string blank is created and initialised, then passed to the function inputName which handles the search.

```
void addLetter(string& prevWord, const char& newLetter) {
    if (newLetter == 8) prevWord.pop_back(); //backspace
    else prevWord.push_back(newLetter);
    return;
}

void printNames(const string& name) {
    cout << "\nContacts: ";
    if (name == "") return;
    else
    {
       map[name].print_numbered = true;
       map[name].print_prefix = "\t";
       cout << "\n" << map[name];
    }
}

string inputName(string& name) {
    cout << "\nInput name: ";
    cout << name;
    char letter = tolower(_getch());
    if (letter == '\r') return name; //enter
    else addLetter(name, letter);
    printNames(name);
    return inputName(name);
}
```

```
Functions involved in search
```

inputName simulates a real time search by printing out all the possible names and numbers that could be accessed from the current user input: this can be either the contact name or phone number.

The user input is: a character, which is added to the previous input; a backspace, which removes the last input; or an enter which finalises the input.

To do this, it prints out the user input and then calls the function printNames and addLetter. This loops recursively until an enter is detected, which is the break statement. printNames simply accesses the relevant doubly linked list and prints it, which is behaviour defined in the class template.

Print Functionality

```
	case 4:
	{
		cout << "\nLinked list:\n";
		for (auto& item : map)
		{
			item.second.print_numbered = true;
			item.second.print_prefix = "\t";
			cout << "Map key: " << item.first << "\tMap entry: \n"
<<	 item.second;
		}
		break;
	}

Print functionality
```

When the user enters 4, the entire unordered_map is printed to the console. This includes nodes which contain partial names and numbers, which are used to implement the preview search.