**Person.h:**  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Person Class Declaration

\* Written by Laith Assaf

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <string>

using namespace std;

class people;

class person

{

private:

string lastName; // Person's last name

string firstName; // Person's first name

int age; // Person's age

int id; // Person's unique identifier

double gpa; // Person's grade point average

public:

// Constructors

person(); // Null constructor

person(string last, string first, int age, int id, double gpa); // Parameterized constructor

// Input/Output methods

bool get(istream& in); // Input person data

void put(ostream& out) const; // Output person data

// Stream operators

friend ostream& operator<<(ostream& out, const person& p);

friend istream& operator>>(istream& in, person& p);

// Comparison operators

bool operator>(const person& other) const; // Greater than

bool operator<(const person& other) const; // Less than

bool operator==(const person& other) const; // Equal to

bool operator==(int searchId) const; // Compare with ID

// Accessor methods

double getGPA() const; // Returns person's GPA

int getAge() const; // Returns person's age

// Friend class declaration

friend class people; // Give people class access to private members

};

**Person.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Person Class Implementation

\* Written by Laith Assaf

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "Person.h"

#include <iomanip>

#include <iostream>

#include <string>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor: person

\* Parameters: none

\* Return: none

\* Initializes empty person object

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

person::person()

{

firstName = "";

lastName = "";

age = 0;

id = 0;

gpa = 0.0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor: person

\* Parameters: string last - last name

\* string first - first name

\* int a - age

\* int i - id

\* double g - gpa

\* Return: none

\* Creates person with given values

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

person::person(string last, string first, int a, int i, double g)

{

lastName = last;

firstName = first;

age = a;

id = i;

gpa = g;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: get

\* Parameters: istream& in - input stream

\* Return: bool - true if successful

\* Reads person data from input stream

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool person::get(istream& in)

{

bool success; // Return value

// Read ID first

in >> id;

if (in.fail())

{

success = false;

}

else

{

// Read first and last name

in >> firstName >> lastName;

if (in.fail())

{

success = false;

}

else

{

// Read age and GPA

in >> age >> gpa;

success = !in.fail();

}

}

return success;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: put

\* Parameters: ostream& out - output stream

\* Return: none

\* Outputs formatted person data

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void person::put(ostream& out) const

{

out << setw(4) << id << " "

<< setw(14) << lastName << " "

<< setw(8) << firstName << " "

<< setw(3) << age << " "

<< fixed << setprecision(2) << setw(4) << gpa;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: operator<<

\* Parameters: ostream& out - output stream

\* const person& p - person to output

\* Return: ostream& - output stream

\* Stream insertion operator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ostream& operator<<(ostream& out, const person& p)

{

p.put(out);

return out;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: operator>>

\* Parameters: istream& in - input stream

\* person& p - person to input

\* Return: istream& - input stream

\* Stream extraction operator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

istream& operator>>(istream& in, person& p)

{

p.get(in);

return in;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: operator>

\* Parameters: const person& other - person to compare

\* Return: bool - true if lastName greater

\* Greater than comparison operator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool person::operator>(const person& other) const

{

return lastName > other.lastName;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: operator<

\* Parameters: const person& other - person to compare

\* Return: bool - true if this lastName less

\* Less than comparison operator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool person::operator<(const person& other) const

{

return lastName < other.lastName;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: operator==

\* Parameters: const person& other - person to compare

\* Return: bool - true if last names equal

\* Equality comparison operator for two persons

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool person::operator==(const person& other) const

{

return lastName == other.lastName;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: operator==

\* Parameters: int searchId - ID to compare

\* Return: bool - true if IDs equal

\* ID equality comparison operator (keep this one as is)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool person::operator==(int searchId) const

{

return id == searchId;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: getGPA

\* Parameters: none

\* Return: double - person's GPA

\* Returns the person's GPA

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double person::getGPA() const

{

return gpa;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: getAge

\* Parameters: none

\* Return: int - person's age

\* Returns the person's age

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int person::getAge() const

{

return age;

}

**People.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* People Class Declaration

\* Written by Laith Assaf

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "Person.h"

class people

{

private:

person\* map; // Pointer to dynamic array of person objects

int len; // Number of persons currently in the array

static const int MAX\_SIZE = 20; // Maximum size of the array

public:

// Constructor and Destructor

people(); // Null constructor - creates storage for array

~people(); // Destructor - deletes array storage

// Array operations

bool insert(const person& p); // Add person to array in ascending order

bool remove(int searchId); // Remove person with given ID

bool find(int searchId) const; // Find person with given ID

// Display operations

void display(ostream& out) const; // Display array with statistics

void dean() const; // Display persons with GPA >= 3.75

};

**People.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* People Class Implementation

\* Written by Laith Assaf

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "People.h"

#include <iomanip>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor: people

\* Parameters: none

\* Return: none

\* Creates dynamic array storage

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

people::people()

{

map = new person[MAX\_SIZE];

len = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Destructor: ~people

\* Parameters: none

\* Return: none

\* Deallocates dynamic array storage

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

people::~people()

{

delete[] map;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: insert

\* Parameters: const person& p - person to insert

\* Return: bool - true if successful

\* Inserts person in ascending ID order

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool people::insert(const person& p)

{

int i; // Loop counter

bool success; // Return value

// Check if array is full

if (len >= MAX\_SIZE)

{

success = false;

}

else

{

// Find insertion point while shifting larger elements

for (i = len - 1; i >= 0 && map[i] > p; i--)

{

map[i + 1] = map[i];

}

// Insert at correct position

map[i + 1] = p;

len++;

success = true;

}

return success;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: remove

\* Parameters: int searchId - ID to remove

\* Return: bool - true if found and removed

\* Removes person with matching ID

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool people::remove(int searchId)

{

int i; // Loop counter

bool found; // Return value

found = false;

// Find person with matching ID

for (i = 0; i < len && !found; i++)

{

if (map[i] == searchId)

{

// Shift remaining elements left

for (int j = i; j < len - 1; j++)

{

map[j] = map[j + 1];

}

len--;

found = true;

}

}

return found;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: find

\* Parameters: int searchId - ID to find

\* Return: bool - true if found

\* Searches for person with matching ID

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool people::find(int searchId) const

{

int i; // Loop counter

bool found; // Return value

found = false;

// Search array for matching ID

for (i = 0; i < len && !found; i++)

{

if (map[i] == searchId)

{

found = true;

}

}

return found;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: display

\* Parameters: ostream& out - output stream

\* Return: none

\* Displays list with statistics

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void people::display(ostream& out) const

{

int i; // Loop counter

double totalAge; // Sum of ages

double totalGPA; // Sum of GPAs

double avgAge; // Average age

double avgGPA; // Average GPA

// Initialize accumulator variables

totalAge = 0;

totalGPA = 0;

// Display header

out << "\nID Last First Age GPA\n"

<< "---- -------------- -------- --- ----\n";

// Display each person

for (i = 0; i < len; i++)

{

map[i].put(out);

out << endl;

// Accumulate totals

totalAge += map[i].getAge();

totalGPA += map[i].getGPA();

}

// Display footer line

out << "---- -------------- -------- --- ----\n";

// Calculate and display averages

if (len > 0)

{

avgAge = totalAge / len;

avgGPA = totalGPA / len;

}

else

{

avgAge = 0;

avgGPA = 0;

}

out << setw(12) << len << " Persons "

<< "Average " << setw(2) << static\_cast<int>(avgAge) << " "

<< fixed << setprecision(2) << avgGPA << endl << endl;

}  
  
  
  
  
**Output:**

**Family.txt:**

Enter data file name: Family.txt

Initial List

ID Last First Age GPA

---- -------------- -------- --- ----

4732 Bowman David 45 3.03

9440 Bowman Frank 37 4.00

2635 Bowman John 30 2.63

1132 Bowman Mark 42 2.50

7166 Bowman Mark 13 4.00

3333 Bowman Richard 47 3.96

2487 Christensen Ann 70 3.71

9636 Cox Susan 36 3.89

7390 Gueller Kathleen 34 1.86

6706 Morales Carlos 68 3.50

---- -------------- -------- --- ----

10 Persons Average 42 3.31

Find 4732 Found

Find 1132 Found

Find 7437 Not Found

Find 6706 Found

Updated List

ID Last First Age GPA

---- -------------- -------- --- ----

9440 Bowman Frank 37 4.00

2635 Bowman John 30 2.63

7166 Bowman Mark 13 4.00

3333 Bowman Richard 47 3.96

2487 Christensen Ann 70 3.71

9636 Cox Susan 36 3.89

7390 Gueller Kathleen 34 1.86

---- -------------- -------- --- ----

7 Persons Average 38 3.44

laithassaf@Laiths-MacBook-Air-2 Program02 %  
  
**Friends.txt**

Enter data file name: Friends.txt

Initial List

ID Last First Age GPA

---- -------------- -------- --- ----

1132 Bowman Mark 56 3.80

6798 Gleason Kent 62 2.70

1163 Hall Josh 32 3.85

8546 Haney Robert 33 3.14

8610 Smith Gary 28 4.00

1147 Taylor Brian 36 2.52

---- -------------- -------- --- ----

6 Persons Average 41 3.34

Find 4732 Not Found

Find 1132 Found

Find 7437 Not Found

Find 6706 Not Found

Updated List

ID Last First Age GPA

---- -------------- -------- --- ----

6798 Gleason Kent 62 2.70

1163 Hall Josh 32 3.85

8546 Haney Robert 33 3.14

8610 Smith Gary 28 4.00

1147 Taylor Brian 36 2.52

---- -------------- -------- --- ----

5 Persons Average 38 3.24

laithassaf@Laiths-MacBook-Air-2 Program02 %