Q1:

Map:

#include<iostream>

#include<string>

#include<map>

using namespace std;

int main() {

map<string, int> grades;

grades.insert(pair<string, int>("Felicity", 100));

grades.insert(pair<string, int>("Lily", 98));

grades.insert(pair<string, int>("Amy", 95));

grades.insert(pair<string, int>("Cindy", 93));

grades.insert(pair<string, int>("Lucy", 96));

for (map<string, int>::iterator itr=grades.begin(); itr!=grades.end(); itr++) {

cout << itr->first << " " << itr->second << endl;

}

return 0;

}

---

Amy 95

Cindy 93

Felicity 100

Lily 98

Lucy 96

---

Multimap:

#include <iostream>  
#include <string>  
#include <map>  
#include <list>  
  
using namespace std;  
  
int main()  
{  
 multimap<string,string> students;  
 students.insert(make\_pair("Felicity","100"));  
 students.insert(make\_pair("Lily","98"));  
  
 for (auto itr = students.begin(); itr != students.end(); ++itr)  
 {  
 cout << itr->first << '\t' << itr->second << '\n';  
 }  
  
}

---

Felicity 100

Lily 98

---

The map and multimap containers are specialized containers that should only be used when you are working with associative data. The map class is used when the keys in your application need to be unique and the multimap class is used when the keys in the application can be duplicates.

---

Q2：

1):

#include<iostream>

using namespace std;

int factorial(int n);

int main() {

int n;

cout << "Enter a positive integer: ";

cin >> n;

cout << "Factorial of " << n << " = " << factorial(n);

return 0;

}

int factorial(int n) {

if(n > 1)

return n \* factorial(n - 1);

else

return 1;

}

--

Enter a positive integer: 3

Factorial of 3 = 6%

---

2):

#include<iostream>

using namespace std;

int fib(int x) {

if ((x==1) || (x==0)) {

return(x);

} else {

return(fib(x-1) + fib(x-2));

}

}

int main() {

int x, i = 0;

cout << "Enter the number of terms of series: ";

cin >> x;

cout << "\nFibonnaci Sequence: ";

while(i < x) {

cout << " " << fib(i);

i++;

}

return 0;

}

---

Enter the number of terms of series: 5

Fibonnaci Sequence: 0 1 1 2 3%

---

Q3:

#include <iostream>

#include <set>

using namespace std;

struct classcomp {

bool operator() (const int& lhs, const int& rhs) const

{return lhs>rhs;}

};

int main() {

// Setup a standard set

set<int> a;

a.insert(5);

a.insert(1);

a.insert(6);

a.insert(3);

a.insert(7);

a.insert(2);

set<int>::iterator it;

for ( it=a.begin(); it != a.end(); it++ )

cout << " " << \*it;

cout << endl << endl;

return 0;

}

----

1 2 3 5 6 7

----

Q4:

Birthday.h

#ifndef BIRTHDAY\_BIRTHDAY\_H  
#define BIRTHDAY\_BIRTHDAY\_H  
  
#include <iostream>  
  
using namespace std;  
  
class birthday  
{  
 friend ostream& operator << (ostream&, const birthday&); // method overloading  
 friend istream& operator >> (istream&, birthday&);  
  
public:  
  
 // default constructor  
 birthday();  
  
 // specific constructor  
 birthday(int d, int m, int y);  
  
 // setters  
 void setDay(int d);  
 void setMonth(int m);  
 void setYear(int y);  
 void setDmy(int d, int m, int y);  
  
 // getters  
 int getDay() const;  
 int getMonth() const;  
 int getYear() const;  
  
 // printf  
 void print() const;  
  
 // Overloading operators  
 bool operator<(const birthday& otherDate) const;  
 bool operator>(const birthday& otherDate) const;  
 bool operator==(const birthday& otherDate) const;  
 bool operator<=(const birthday& otherDate) const;  
 bool operator>=(const birthday& otherDate) const;  
 bool operator!=(const birthday& otherDate) const;  
  
 // Overloading math operations  
 birthday operator++();  
 birthday operator++(int);  
 birthday operator--();  
 birthday operator--(int);  
  
private:  
  
 // data members  
 int day;  
 int month;  
 int year;  
};  
  
#endif //BIRTHDAY\_BIRTHDAY\_H

Birthday.cpp

#include <iostream>  
#include "birthday.h"  
  
using namespace std;  
  
// default constructor  
birthday::birthday()  
{  
 day = 1;  
 month = 1;  
 year = 2000;  
}  
  
// specific constructor  
birthday::birthday(int d, int m, int y)  
{  
 day = d;  
 month = m;  
 year = y;  
}  
  
// setters  
void birthday::setDay(int d)  
{  
 day = d;  
}  
  
void birthday::setMonth(int m)  
{  
 month = m;  
}  
  
void birthday::setYear(int y)  
{  
 year = y;  
}  
  
void birthday::setDmy(int d, int m, int y)  
{  
 day = d;  
 month = m;  
 year = y;  
}  
  
// getters  
int birthday::getDay() const  
{  
 return day;  
}  
  
int birthday::getMonth() const  
{  
 return month;  
}  
  
int birthday::getYear() const  
{  
 return year;  
}  
  
// printf  
void birthday::print() const  
{  
 cout << day << "/"  
 << month << "/"  
 << year;  
}  
  
// ostream operator  
ostream& operator << (ostream& osObject, const birthday& date1) //creating an instance of a date class, i.e. date1  
{  
 osObject << date1.day  
 << "/" << date1.month  
 << "/" << date1.year;  
 return osObject;  
}  
  
// istream operator  
istream& operator >> (istream& isObject, birthday& date1)  
{  
 isObject >> date1.day >> date1.month >> date1.year;  
 return isObject;  
}  
  
// Overloading "equal to" operator  
bool birthday::operator==(const birthday& otherDate) const  
{  
 if (day == otherDate.day && month == otherDate.month  
 && year == otherDate.year)  
 return true;  
 else  
 return false;  
}  
  
// Overloading "not equal to" operator  
bool birthday::operator!=(const birthday& otherDate) const  
{  
 return !(\*this == otherDate);  
}

Person.h

#ifndef BIRTHDAY\_PERSON\_H  
#define BIRTHDAY\_PERSON\_H  
  
#include <iostream>  
#include <string>  
  
using namespace std;  
  
class person  
{  
 friend ostream& operator << (ostream&, const person&);  
 friend istream& operator >> (istream&, person&);  
  
public:  
 // default constructor: must have the same name as the file  
 person();  
  
 // specific constructor  
 person(string fn, string mn, string ln);  
  
 // setters  
 void setFirstName(string fn);  
 void setMiddleName(string mn);  
 void setLastName(string ln);  
 void setName(string fn, string mn, string ln);  
  
 //getters  
 string getFirstName() const;  
 string getMiddleName() const;  
 string getLastName() const;  
  
 // printf  
 void print() const;  
  
 // Overloading operators  
 bool operator<(const person& otherPerson) const;  
 bool operator>(const person& otherPerson) const;  
 bool operator==(const person& otherPerson) const;  
 bool operator<=(const person& otherPerson) const;  
 bool operator>=(const person& otherPerson) const;  
 bool operator!=(const person& otherPerson) const;  
  
protected: // change from private to protected so that studentType can access these  
  
 string firstName; // data members  
 string middleName;  
 string lastName;  
};  
  
#endif //BIRTHDAY\_PERSON\_H

Person.cpp

#include <iostream>  
#include <string>  
#include "person.h"  
  
  
using namespace std;  
  
// default constructor  
person::person() // header file name :: header file function  
{  
 firstName = "Me";  
 middleName = "My";  
 lastName = "Mine";  
}  
  
// specific constructor  
person::person(string fn, string mn, string ln)  
{  
 firstName = fn;  
 middleName = mn;  
 lastName = ln;  
}  
  
// setters  
void person::setFirstName(string fn)  
{  
 firstName = fn;  
}  
  
void person::setMiddleName(string mn)  
{  
 middleName = mn;  
}  
  
void person::setLastName(string ln)  
{  
 lastName = ln;  
}  
  
void person::setName(string fn, string mn, string ln)  
{  
 firstName = fn;  
 middleName = mn;  
 lastName = ln;  
}  
  
// getters  
string person::getFirstName() const  
{  
 return firstName;  
}  
  
string person::getMiddleName() const  
{  
 return middleName;  
}  
  
string person::getLastName() const  
{  
 return lastName;  
}  
  
//printf  
void person::print() const  
{  
 cout << firstName << " "  
 << middleName << " "  
 << lastName << " ";  
}  
  
// ostream operator  
ostream& operator << (ostream& osObject, const person& person1) //creating an instance of a date class, i.e. date1  
{  
 osObject << person1.firstName  
 << " " << person1.middleName  
 << " " << person1.lastName;  
  
 return osObject;  
}  
  
istream& operator >> (istream& isObject, person& person1)  
{  
 isObject >> person1.firstName >> person1.middleName >> person1.lastName;  
  
 return isObject;  
}  
  
// Overloading "equal to" operator  
bool person::operator==(const person& otherPerson) const  
{  
 if (firstName == otherPerson.firstName && middleName == otherPerson.middleName  
 && lastName == otherPerson.lastName)  
 return true;  
 else  
 return false;  
}  
  
// Overloading "not equal to" operator  
bool person::operator!=(const person& otherPerson) const  
{  
 return !(\*this == otherPerson);  
}  
  
// Overloading "less than or equal to" operator  
bool person::operator<=(const person& otherPerson) const  
{  
 return (\*this < otherPerson || \*this == otherPerson);  
}  
  
// Overloading "less than" operator  
bool person::operator<(const person& otherPerson) const  
{  
 if ((lastName < otherPerson.lastName) ||  
 (lastName == otherPerson.lastName && middleName < otherPerson.middleName) ||  
 (lastName == otherPerson.lastName && middleName == otherPerson.middleName && firstName < otherPerson.firstName))  
 return true;  
 else  
 return false;  
}  
  
// Overloading "greater than or equal to" operator  
bool person::operator>=(const person& otherPerson) const  
{  
 return !(\*this < otherPerson);  
}  
  
// Overloading "greater than" operator  
bool person::operator>(const person& otherPerson) const  
{  
 return !(\*this <= otherPerson);  
}

Main.cpp

#include <iostream>  
#include <string>  
#include <map>  
  
#include "person.h"  
#include "birthday.h"  
  
using namespace std;  
  
int main()  
{  
 // Declaring a map object  
 map<person, birthday> myMap;  
  
 // Creating instances of personType  
 person p1("Yan", "Ted", "Willy");  
 person p2("Ang", "Wan", "Xin");  
 person p3("Sally", "Billy", "Chad");  
 person p4("Donna", "Emily", "Francis");  
  
 // Creating instances of date  
 birthday d1(3, 8, 2014);  
 birthday d2(12, 12, 2018);  
 birthday d3(5, 19, 2012);  
 birthday d4(8, 6, 2019);  
  
 // Inserting values into map called myMap  
 myMap.insert(pair<person, birthday>(p1, d1));  
 myMap.insert(pair<person, birthday>(p2, d2));  
 myMap.insert(pair<person, birthday>(p3, d3));  
 myMap.insert(pair<person, birthday>(p4, d4));  
  
 // Defining an iterator called "it"  
 map<person, birthday>::iterator it;  
  
 // Displaying the map  
 for (it = myMap.begin(); it != myMap.end(); ++it)  
 {  
 cout << it->first<< ": "  
 << it->second << '\n';  
 }  
  
  
 return 0;  
}

Sally Billy Chad: 5/19/2012

Donna Emily Francis: 8/6/2019

Yan Ted Willy: 3/8/2014

Ang Wan Xin: 12/12/2018