```cpp
/**
 * @file inheritance4.cpp
 * @author Humayun Kabir, Instructor, CSCI 161, VIU
 * @version 1.0.0
 * @date March 10, 2022
 *
 * Example code to demonstrate abstract class with two pure virtual function.
 *
 *
 *                 Employee (abstract)
 *                    ^
 *                    |
 *      ------------------------------------------------------
 *      |                         |                          |
 *  RegularEmployee      HourlyContractEmployee       OnetimeContractEmployee
 *
 *
 * Function payCheque() has been declared pure virtual function in base class Employee.
 * That forces every derived class to override this function, i.e., RegularEmployee,
 * HourlyContractEmployee, and OnetimeContractEmployee classes need to implement this
 * function, When this function is being called through RegularEmployee or HourlyContractEmployee
 * OnetimeContractEmployee objects or references, it executes respective overriden version of
 * the function. That also happens when this function is being called on either RegularEmployee or
 * HourlyContractEmployee or OntimeContractEmployee objects through Employee class references.
 * This is happening due to the presence of runtime polymorphism in the inheritance hierarchy.
 * Runtime polymorphism is ensured by declaring payCheque() function virtual in base class Employee.
 *
 * Function toString() has been declared pure virtual in base class Employee. All the derived
 * classes (RegularEmployee, HourlyContractEmployee, and OnetimeContractEmployee) have their
 * own version of toString() function to return a customized string based on their context.
 *
 * Class Employee becomes an abstract class now due the presence of payCheque() pure virtual function. It is
 * not allowed to instantiate an object from an abstract class.
 *
 * Function 'operator << ()' is a friend function of Employee class, which will not be
 * inherited by the derived classes, However, it will not generate a compiler error message
 * if an object of a derived class is passed through the base class reference in this function.
 * This type of call will run the function implemented by the base class, even though the derived
 * classes do not have this function implemented for themselves or inherited. This friend function
 * of Employee base class calls a virtual toString() function that retruns a customized string based
 * on the actual object that is passed by Employee class reference. For this reason, no drived class
 * now need its own friend function to overload operator << () function. Base class version of
 * 'operator << ()' function is good enough for all derived classes.
 *
 */



#include <iostream>
#include <string>

using namespace std;



/*
 * Abstract base class
 */
class Employee {
    protected:
        string employeeNumber;
        string name;
        string address;
        string email;
        string phone;
        double salary;
    public:
        Employee();
        Employee(string, string, string, string, string, double);
        Employee(const Employee&);
        Employee(Employee&&);
        Employee& operator = (const Employee&);
        Employee& operator = (Employee&&);
        virtual ~Employee();
        string getEmployeeNumber() const;
        string getName() const;
        string getAddress() const;
        string getEmail() const;
        string getPhone() const;
        double getSalary() const;
        virtual double payCheque() const = 0;       //Pure virtual functions
        virtual string toString() const = 0;        //Pure virtual functions
        friend ostream& operator << (ostream&, const Employee&);

};



class RegularEmployee: public Employee {
    public:
        RegularEmployee();
        RegularEmployee(string, string, string, string, string, double);
        RegularEmployee(const RegularEmployee&);
        RegularEmployee(RegularEmployee&&);
        RegularEmployee& operator = (const RegularEmployee&);
        RegularEmployee& operator = (RegularEmployee&&);
        ~RegularEmployee();
        double payCheque() const override;
        string toString() const override;
        //friend ostream& operator << (ostream&, const RegularEmployee&);

};
```

```cpp


class HourlyContractEmployee: public Employee {
    private:
        double hoursWorked;
    public:
        HourlyContractEmployee();
        HourlyContractEmployee(string, string, string, string, string, double, double);
        HourlyContractEmployee(const HourlyContractEmployee&);
        HourlyContractEmployee(HourlyContractEmployee&&);
        HourlyContractEmployee& operator = (const HourlyContractEmployee&);
        HourlyContractEmployee& operator = (HourlyContractEmployee&&);
        ~HourlyContractEmployee();
        double payCheque() const override;
        string toString() const override;
        void setHoursWorked(double hoursWorked);
        double getHoursWorked() const;
        //friend ostream& operator << (ostream&, const HourlyContractEmployee&);


};



class OneTimeContractEmployee: public Employee {
    public:
        OneTimeContractEmployee();
        OneTimeContractEmployee(string, string, string, string, string, double);
        OneTimeContractEmployee(const OneTimeContractEmployee&);
        OneTimeContractEmployee(OneTimeContractEmployee&&);
        OneTimeContractEmployee& operator = (const OneTimeContractEmployee&);
        OneTimeContractEmployee& operator = (OneTimeContractEmployee&&);
        ~OneTimeContractEmployee();
        double payCheque() const override;
        string toString() const override;
        //friend ostream& operator << (ostream&, const OneTimeContractEmployee&);

};



/*
 * Employee function implementations
 */
Employee::Employee():
    employeeNumber("Unkown"),
    name("Unknown"),
    address("Unknown"),
    email("Uknown"),
    phone("Unknown"),
    salary(0.0)
    {}


Employee::Employee(string employeeNumber, string name, string address,
        string email, string phone, double salary):
    employeeNumber(employeeNumber),
    name(name), address(address),
    email(email),
    phone(phone),
    salary(salary)
    {}


Employee::Employee(const Employee& other):
    employeeNumber(other.employeeNumber),
    name(other.name),
    address(other.address),
    email(other.email),
    phone(other.phone),
    salary(other.salary)
    {}


Employee::Employee(Employee&& temp):
    employeeNumber(temp.employeeNumber),
    name(temp.name),
    address(temp.address),
    email(temp.email),
    phone(temp.phone),
    salary(temp.salary)
    {}


Employee& Employee::operator = (const Employee& other) {
    if( this == &other) {
        return *this;
    }
    employeeNumber = other.employeeNumber;
    name = other.name;
    address = other.address;
    email = other. email;
    phone = other.phone;
    salary = other.salary;
    return *this;
}


```

```
207  Employee& Employee::operator = (Employee&& temp) {
208      if( this == &temp) {
209          return *this;
210      }
211      employeeNumber = temp.employeeNumber;
212      name = temp.name;
213      address = temp.address;
214      email = temp. email;
215      phone = temp.phone;
216      salary = temp.salary;
217      return *this;
218  }
219
220  Employee::~Employee() {}
221
222
223  string Employee::getEmployeeNumber() const {
224      return employeeNumber;
225  }
226
227  string Employee::getName() const {
228      return name;
229  }
230
231  string Employee::getAddress() const {
232      return address;
233  }
234
235  string Employee::getEmail() const {
236      return email;
237  }
238
239  string Employee::getPhone() const {
240      return phone;
241  }
242
243  double Employee::getSalary() const {
244      return salary;
245  }
246
247  ostream& operator << (ostream& out, const Employee& employee) {
248      out<<employee.toString();
249      return out;
250  }
251
252
253
254
255
256  /*
257   * RegularEmployee function implementations
258   */
259  RegularEmployee::RegularEmployee(): Employee() {}
260
261  RegularEmployee::RegularEmployee(string employeeNumber, string name,
262          string address, string email, string phone, double salary):
263      Employee(employeeNumber, name, address, email, phone, salary)
264      {}
265
266
267  RegularEmployee::RegularEmployee(const RegularEmployee& other):
268      Employee(other)
269      {}
270
271
272  RegularEmployee::RegularEmployee(RegularEmployee&& temp):
273      Employee(std::move(temp))
274      {}
275
276
277  RegularEmployee& RegularEmployee::operator = (const RegularEmployee& other) {
278      Employee::operator = (other);
279      return *this;
280  }
281
282
283  RegularEmployee& RegularEmployee::operator = (RegularEmployee&& temp) {
284      Employee::operator = (std::move(temp));
285      return *this;
286  }
287
288  RegularEmployee::~RegularEmployee() {}
289
290
291  double RegularEmployee::payCheque() const {
292      return salary/26.0;
293  }
294
295
296  string RegularEmployee::toString() const {
297      string empStr =
298          "Regular Employee Number: " + employeeNumber +
299          ", Name: " + name +
300          ", Adress: " + address +
301          ", Email: " + email +
302          ", Phone: " + phone +
303          ", Annual Salary: " + to_string(salary) +
304          ", Pay Cheque: " + to_string(payCheque());
305
306         return empStr;
307
308  }
309
```

```
310
311
312
313 /*
314  * HourlyContractEmployee function implementations
315  */
316
317 HourlyContractEmployee::HourlyContractEmployee():
318     Employee(),
319     hoursWorked(0.0)
320     {}
321
322
323 HourlyContractEmployee::HourlyContractEmployee(string employeeNumber, string name,
324         string address, string email, string phone, double salary, double hoursWorked):
325     Employee(employeeNumber, name, address, email, phone, salary),
326     hoursWorked(hoursWorked)
327     {}
328
329
330 HourlyContractEmployee::HourlyContractEmployee(const HourlyContractEmployee& other):
331     Employee(other),
332     hoursWorked(other.hoursWorked)
333     {}
334
335
336 HourlyContractEmployee::HourlyContractEmployee(HourlyContractEmployee&& temp):
337     Employee(std::move(temp)),
338     hoursWorked(temp.hoursWorked)
339     {}
340
341
342 HourlyContractEmployee& HourlyContractEmployee::operator = (const HourlyContractEmployee& other) {
343     Employee::operator = (other);
344     hoursWorked = other.hoursWorked;
345     return *this;
346 }
347
348 HourlyContractEmployee& HourlyContractEmployee::operator = (HourlyContractEmployee&& temp) {
349     Employee::operator = (std::move(temp));
350     hoursWorked = temp.hoursWorked;
351     return *this;
352 }
353
354 HourlyContractEmployee::~HourlyContractEmployee() {}
355
356
357 double HourlyContractEmployee::payCheque() const {
358     return salary*hoursWorked;
359 }
360
361 void HourlyContractEmployee::setHoursWorked(double hoursWorked) {
362     this->hoursWorked = hoursWorked;
363 }
364
365 double HourlyContractEmployee::getHoursWorked() const {
366     return hoursWorked;
367 }
368
369 string HourlyContractEmployee::toString() const {
370     string empStr =
371         "Hourly Contract Employee Number: " + employeeNumber +
372         ", Name: " + name +
373         ", Adress: " +address +
374         ", Email: " + email +
375         ", Phone: " + phone +
376         ", Hourly Salary: " + to_string(salary) +
377         ", Hours Worked: "+ to_string(hoursWorked) +
378         ", Pay Cheque: " + to_string(payCheque());
379        return empStr;
380
381 }
382
383
384
385
386
387 /*
388  * OneTimeContractEmployee function implementations
389  */
390
391 OneTimeContractEmployee::OneTimeContractEmployee(): Employee() {}
392
393 OneTimeContractEmployee::OneTimeContractEmployee(string employeeNumber, string name,
394         string address, string email, string phone, double salary):
395     Employee(employeeNumber, name, address, email, phone, salary)
396     {}
397
398
399 OneTimeContractEmployee::OneTimeContractEmployee(const OneTimeContractEmployee& other):
400     Employee(other)
401     {}
402
403
404 OneTimeContractEmployee::OneTimeContractEmployee(OneTimeContractEmployee&& temp):
405     Employee(std::move(temp))
406     {}
407
408
409 OneTimeContractEmployee& OneTimeContractEmployee::operator = (const OneTimeContractEmployee& other) {
410     Employee::operator = (other);
411     return *this;
412 }
```

```
413
414
415 OneTimeContractEmployee& OneTimeContractEmployee::operator = (OneTimeContractEmployee&& temp) {
416     Employee::operator = (std::move(temp));
417     return *this;
418 }
419
420
421 OneTimeContractEmployee::~OneTimeContractEmployee() {}
422
423
424 double OneTimeContractEmployee::payCheque() const {
425     return salary;
426 }
427
428
429 string OneTimeContractEmployee::toString() const {
430     string empStr =
431     "Ontime Contract Employee Number: " + employeeNumber +
432     ", Name: " + name +
433     ", Adress: " + address +
434     ", Email: " + email +
435     ", Phone: " + phone +
436     ", Onetime Salary: " + to_string(salary) +
437     ", Pay Cheque: " + to_string(payCheque());
438
439     return empStr;
440
441 }
442
443
444
445
446
447 /*
448  * Function to return Employee rvalue
449  */
450 //Employee createEmployee(string employeeNumber, string name, string address,
451 //                  string email, string phone, double salary) {
452 //  return Employee(employeeNumber, name, address, email, phone, salary);
453 //}
454
455
456 RegularEmployee createRegularEmployee(string employeeNumber, string name, string address,
457         string email, string phone, double salary) {
458     return RegularEmployee(employeeNumber, name, address, email, phone, salary);
459 }
460
461
462 HourlyContractEmployee createHourlyContractEmployee(string employeeNumber, string name,
463         string address, string email, string phone, double salary, double hoursWorked) {
464     return HourlyContractEmployee(employeeNumber, name, address, email, phone, salary, hoursWorked);
465 }
466
467
468 OneTimeContractEmployee createOneTimeContractEmployee( string employeeNumber, string name,
469         string address, string email, string phone, double salary){
470     return OneTimeContractEmployee(employeeNumber, name, address, email, phone, salary);
471 }
472
473
474
475
476
477 int main() {
478
479     /*
480      * Since Employee is now an abstract class, you are not allowed to create objects from Employee class.
481      */
482     //Employee employee0;
483     //cout<<employee0<<endl;
484     //Employee employee1("EMP001", "Humayun", "Victoria", "humayun@viu.ca", "250-727-0980", 200000.0);
485     //cout<<employee1<<endl;
486     //Employee employee2(employee1);
487     //cout<<employee2<<endl;
488     //Employee employee3 = createEmployee("EMP001", "Humayun", "Victoria", "humayun@viu.ca", "250-727-0980", 200000.0);
489     //cout<<employee3<<endl;
490     //Employee employee4;
491     //employee4 = createEmployee("EMP001", "Humayun", "Victoria", "humayun@viu.ca", "250-727-0980", 200000.0);
492     //cout<<employee4<<endl;
493
494
495     /*
496      * You can create objects from non-abstract classes
497      */
498     cout<<"Regular Objects from non-abstract subclasses....................................."<<endl;
499     RegularEmployee regEmp0;
500     // Calls operator<<() friend function of Employeee class.
501     // Passes RegularEmployee object reference through Employee reference
502     // in operator<<() function.
503     // Runtime polymorphism takes care of using toString() function from
504     // RegularEmployee class in operator<<() function.
505     cout<<regEmp0<<endl;
506     RegularEmployee regEmp1("REMP001", "Humayun", "Victoria", "humayun@viu.ca", "250-727-0980", 200000.0);
507     cout<<regEmp1<<endl;
508     RegularEmployee regEmp2(regEmp1);
509     cout<<regEmp2<<endl;
510     RegularEmployee regEmp3 = createRegularEmployee("REMP001", "Humayun", "Victoria", "humayun@viu.ca", "250-727-0980",
    200000.0);
511     cout<<regEmp3<<endl;
512     RegularEmployee regEmp4;
513     regEmp4 = createRegularEmployee("REMP001", "Humayun", "Victoria", "humayun@viu.ca", "250-727-0980", 200000.0);
514     cout<<regEmp4<<endl;
```

```
515
516
517         HourlyContractEmployee hourEmp0;
518         // Calls operator<<() friend function of Employeee class.
519         // Passes HourlyContractEmployee object reference through Employee reference
520         // in operator<<() function.
521         // Runtime polymorphism takes care of using toString() function from
522         // HourlyContractEmployee class in operator<<() function.
523         cout<<hourEmp0<<endl;
524         HourlyContractEmployee hourEmp1("HEMP001", "Humayun ", "Victoria ", "humayun@viu.ca ", "250-727-0980 ", 200.0, 40.0);
525         cout<<hourEmp1<<endl;
526         HourlyContractEmployee hourEmp2(hourEmp1);
527         cout<<hourEmp2<<endl;
528         HourlyContractEmployee hourEmp3 = createHourlyContractEmployee("HEMP001", "Humayun", "Victoria", "humayun@viu.ca",
    "250-727-0980", 200.0, 40.0);
529         cout<<hourEmp3<<endl;
530         HourlyContractEmployee hourEmp4;
531         hourEmp4 = createHourlyContractEmployee("HEMP001", "Humayun", "Victoria", "humayun@viu.ca", "250-727-0980", 200.0,
    40.0);
532         cout<<hourEmp4<<endl;
533
534
535         OneTimeContractEmployee onetimeEmp0;
536         // Calls operator<<() friend function of Employeee class.
537         // Passes OneTimeContractEmployee object reference through Employee reference
538         // in operator<<() function.
539         // Runtime polymorphism takes care of using toString() function from
540         // OneTimeContractEmployee class in operator<<() function.
541         cout<<onetimeEmp0<<endl;
542         OneTimeContractEmployee onetimeEmp1("TEMP001", "Humayun ", "Victoria ", "humayun@viu.ca ", "250-727-0980 ", 30000.0);
543         cout<<onetimeEmp1<<endl;
544         OneTimeContractEmployee onetimeEmp2(onetimeEmp1);
545         cout<<onetimeEmp2<<endl;
546         OneTimeContractEmployee onetimeEmp3 = createOneTimeContractEmployee("TEMP001", "Humayun", "Victoria", "humayun@viu.ca",
    "250-727-0980", 30000.0);
547         cout<<onetimeEmp3<<endl;
548         OneTimeContractEmployee onetimeEmp4;
549         onetimeEmp4 = createOneTimeContractEmployee("TEMP001", "Humayun", "Victoria", "humayun@viu.ca", "250-727-0980",
    30000.0);
550         cout<<onetimeEmp4<<endl;
551
552         cout<<endl;
553         cout<<"Objects from non-abstract subclasses with abstract base class references (pointers) to achieve runtime
    polymorphism"<<endl;
554
    cout<<"*****************************************************************************************************************"<<endl;
555         Employee* employees[] = {
556         new RegularEmployee ("REMP001", "Humayun Regular 1", "Victoria", "humayun@viu.ca", "250-727-0980", 200000.0),
557         new RegularEmployee ("REMP002", "Humayun Regular 2", "Victoria", "humayun@viu.ca", "250-727-0980", 400000.0),
558         new RegularEmployee ("REMP003", "Humayun Regular 3", "Victoria", "humayun@viu.ca", "250-727-0980", 300000.0),
559         new HourlyContractEmployee ("HEMP001", "Humayun Hourly 1", "Victoria", "humayun@viu.ca", "250-727-0980", 200.0, 40.0),
560         new HourlyContractEmployee ("HEMP002", "Humayun Hourly 2", "Victoria", "humayun@viu.ca", "250-727-0980", 300.0, 20.0),
561         new RegularEmployee ("REMP004", "Humayun Regular 4", "Victoria", "humayun@viu.ca", "250-727-0980", 100000.0),
562         new RegularEmployee ("REMP005", "Humayun Regular 5", "Victoria", "humayun@viu.ca", "250-727-0980", 500000.0),
563         new OneTimeContractEmployee ("TEMP001", "Humayun Onetime 1", "Victoria", "humayun@viu.ca", "250-727-0980", 20000.0),
564         new OneTimeContractEmployee ("REMP001", "Humayun Onetime 2", "Victoria", "humayun@viu.ca", "250-727-0980", 80000.0)
565         };
566
567
568         for(int i=0; i<9; i++) {
569             cout<<*employees[i]<<endl;
570             delete employees[i];
571         }
572
573         return 0;
574     }
```