```
#include <iostream>
#include "HourlyEmployee.h"
#include "SalariedEmployee.h"
using std::cout;
using std::endl;
using SavitchEmployees::HourlyEmployee;
using SavitchEmployees::SalariedEmployee;

int main( )
{
    HourlyEmployee joe;
    joe.setName("Mighty Joe");
    joe.setSsn("123-45-6789");
    joe.setRate(20.50);
    joe.setHours(40);
    cout << "Check for " << joe.getName( )
         << " for " << joe.getHours( ) << " hours.\n";
    joe.printCheck( );
    cout << endl;

    SalariedEmployee boss("Mr. Big Shot", "987-65-4321", 10500.50);
    cout << "Check for " << boss.getName( ) << endl;
    boss.printCheck( );

    return 0;
}
```

```cpp
//
//  Employee.hpp
//  CppPlayground
//
//  Created by Liwei on 2020/3/20.
//  Copyright © 2020 Liwei. All rights reserved.
//

#ifndef Employee_h
#define Employee_h

#include <string>
using std::string;

namespace SavitchEmployees
    {
    class Employee
    {
    public:
        Employee( );
        Employee(const string& theName, const string& theSsn);
        string getName( ) const;
        string getSsn( ) const;
        double getNetPay( ) const;
        void setName(const string& newName);
        void setSsn(const string& newSsn);
        void setNetPay(double newNetPay);
        void printCheck( ) const;
    private:
        string name;
        string ssn;
        double netPay;
    };
    }//SavitchEmployees

#endif /* Employee_hpp */
```

```cpp
//
//  Employee.cpp
//  CppPlayground
//
//  Created by Liwei on 2020/3/20.
//  Copyright © 2020 Liwei. All rights reserved.
//

//This is the file employee.cpp.
//This is the implementation for the class Employee.
//The interface for the class Employee is in the header file employee.h.
#include <string>
#include <cstdlib>
#include <iostream>
#include "Employee.h"

using std::string;
using std::cout;
namespace SavitchEmployees
    {
    Employee::Employee( ) : name("No name yet"),
    ssn("No number yet"), netPay(0)
    {
        //deliberately empty

    }
    Employee::Employee(const string& theName, const string& theNumber)  :
     name(theName), ssn(theNumber), netPay(0)
    {
        //deliberately empty
    }
    string Employee::getName( ) const
    {
        return name;
    }
    string Employee::getSsn( ) const
    {
        return ssn;
    }

    double Employee::getNetPay( ) const
    {
        return netPay;
    }
    void Employee::setName(const string& newName)
    {
        name = newName;
    }
    void Employee::setSsn(const string& newSsn)
    {
        ssn = newSsn;
    }
    void Employee::setNetPay (double newNetPay)
    {
        netPay = newNetPay;
    }
    void Employee::printCheck( ) const
    {
        cout << "\nERROR: printCheck FUNCTION CALLED FOR AN \n"
        << "UNDIFFERENTIATED EMPLOYEE. Aborting the program.\n" << "Check
         with the author of the program about this bug.\n";
        exit(1); }
    }//SavitchEmployees
```

```cpp
//
//  HourlyEmployee.hpp
//  CppPlayground
//
//  Created by Liwei on 2020/3/20.
//  Copyright © 2020 Liwei. All rights reserved.
//

#ifndef HourlyEmployee_h
#define HourlyEmployee_h

#include <stdio.h>
#include "Employee.h"

using std::string;
namespace SavitchEmployees{
class HourlyEmployee : public Employee
{

public:
    HourlyEmployee( );
    HourlyEmployee(const string& theName, const string& theSsn,
                   double theWageRate, double theHours);
    void setRate(double newWageRate);
    double getRate( ) const;
    void setHours(double hoursWorked);
    double getHours( ) const;
    void printCheck( );

private:
    double wageRate;
    double hours;
};
}//SavitchEmployees


#endif /* HourlyEmployee_h */
```

```cpp
//
//  HourlyEmployee.cpp
//  CppPlayground
//
//  Created by Liwei on 2020/3/20.
//  Copyright © 2020 Liwei. All rights reserved.
//

#include <string>
#include <iostream>
#include "HourlyEmployee.h"
using std::string;
using std::cout;
using std::endl;

namespace SavitchEmployees{
HourlyEmployee::HourlyEmployee( ):Employee(),wageRate(0), hours(0)
{
    //deliberately empty
}

HourlyEmployee::HourlyEmployee(const string& theName, const string&
 theNumber, double theWageRate,
                               double theHours)
: Employee(theName, theNumber), wageRate(theWageRate),
hours(theHours){
    //deliberately empty
}

void HourlyEmployee::setRate(double newWageRate)
{
    wageRate = newWageRate;
}
double HourlyEmployee::getRate( ) const
{
    return wageRate;
}

void HourlyEmployee::setHours(double hoursWorked)
{
    hours = hoursWorked;
}
double HourlyEmployee::getHours( ) const
{
    return hours;
}

void HourlyEmployee::printCheck( )
{
    setNetPay(hours * wageRate);
    cout << "\n__________________________________________\n";
    cout << "Pay to the order of " <<getName()<< endl;
    cout << "The sum of " <<getNetPay()<< " Dollars\n";
    cout << "__________________________________________\n";
    cout << "Check Stub: NOT NEGOTIABLE\n";
    cout << "Employee Number: " << getSsn( ) << endl;
    cout << "Hourly Employee. \nHours worked: " << hours << " Rate: " <<
     wageRate << " Pay: " << getNetPay( ) << endl;

    cout << "__________________________________________\n";
}

}//SavitchEmployees
```

```cpp
//
//  SalariedEmployee.hpp
//  CppPlayground
//
//  Created by Liwei on 2020/3/20.
//  Copyright © 2020 Liwei. All rights reserved.
//

#ifndef SalariedEmployee_h
#define SalariedEmployee_h

#include <stdio.h>
#include <string>
#include "Employee.h"

using std::string;
namespace SavitchEmployees{
class SalariedEmployee : public Employee
{

public:
    SalariedEmployee( );
    SalariedEmployee (const string& theName, const string& theSsn,
                      double theWeeklySalary);
    double getSalary( ) const;
    void setSalary(double newSalary);
    void printCheck( );

private:
    double salary;//weekly
};

}//SavitchEmployees
#endif /* SalariedEmployee_h */
```

```
//
//  SalariedEmployee.cpp
//  CppPlayground
//
//  Created by Liwei on 2020/3/20.
//  Copyright © 2020 Liwei. All rights reserved.
//


#include <iostream>
#include <string>
#include "SalariedEmployee.h"
using std::string;
using std::cout;
using std::endl;

namespace SavitchEmployees
    {
    SalariedEmployee::SalariedEmployee( ) : Employee( ), salary(0)  {
        //deliberately empty
    }

    SalariedEmployee::SalariedEmployee(const string& theName,
                                       const string& theNumber,
                                       double theWeeklyPay)
    : Employee(theName, theNumber), salary(theWeeklyPay){
        //deliberately empty
    }
    double SalariedEmployee::getSalary( ) const
    {
        return salary;
    }

    void SalariedEmployee::setSalary(double newSalary)
    {
        salary = newSalary;
    }

    void SalariedEmployee::printCheck( )
    {
        setNetPay(salary);

        cout << "\n__________________________________________\n";
        cout << "Pay to the order of " <<getName()<< endl;
        cout << "The sum of " <<getNetPay()<< " Dollars\n";
        cout << "__________________________________________\n";
        cout << "Check Stub NOT NEGOTIABLE \n";
        cout << "Employee Number: " << getSsn( ) << endl;
        cout << "Salaried Employee. Regular Pay: " << salary << endl;
        cout << "__________________________________________\n";
    }
    }//SavitchEmployees
```

```cpp
//Program to demonstrate the class PFArrayDBak.
#include <iostream>
#include "pfarraydbak.h"
using std::cin;
using std::cout;
using std::endl;

void testPFArrayDBak( );
//Conducts one test of the class PFArrayDBak.

int main( )
{
    cout << "This program tests the class PFArrayDBak.\n";

    char ans;
    do
    {
        testPFArrayDBak( );
        cout << "Test again? (y/n) ";
        cin >> ans;
    }while ((ans == 'y') || (ans == 'Y'));

    return 0;
}

void testPFArrayDBak( )
{
    int cap;
    cout << "Enter capacity of this super array: ";
    cin >> cap;
    PFArrayDBak a(cap);

    cout << "Enter up to " << cap << " nonnegative numbers.\n";
    cout << "Place a negative number at the end.\n";

    double next;

    cin >> next;
    while ((next >= 0) && (!a.full( )))
    {
        a.addElement(next);
        cin >> next;
    }

    if (next >= 0)
    {
        cout << "Could not read all numbers.\n";
        //Clear the unread input:
        while (next >= 0)
            cin >> next;
    }

    int count = a.getNumberUsed( );
    cout << "The following " << count
         << " numbers read and stored:\n";
    int index;
    for (index = 0; index < count; index++)
        cout << a[index] << " ";
    cout << endl;

    cout << "Backing up array.\n";
    a.backup( );

    cout << "emptying array.\n";
    a.emptyArray( );
    cout << a.getNumberUsed( )
         << " numbers are now stored in the array.\n";

    cout << "Restoring array.\n";
    a.restore( );
    count = a.getNumberUsed( );
    cout << "The following " << count
         << " numbers are now stored:\n";
    for (index = 0; index < count; index++)
        cout << a[index] << " ";
    cout << endl;
}
```

```cpp
//This is the header file pfarrayd.h. This is the interface for the class
//PFArrayD. Objects of this type are partially filled arrays of doubles.
#ifndef PFARRAYD_H
#define PFARRAYD_H

class PFArrayD
{
public:
    PFArrayD( );
    //Initializes with a capacity of 50.

    PFArrayD(int capacityValue);

    PFArrayD(const PFArrayD& pfaObject);

    void addElement(double element);
    //Precondition: The array is not full.
    //Postcondition: The element has been added.

    bool full( ) const;
    //Returns true if the array is full, false otherwise.

    int getCapacity( ) const;

    int getNumberUsed( ) const;

    void emptyArray( );
    //Resets the number used to zero, effectively emptying the array.

    double& operator[](int index);
    //Read and change access to elements 0 through numberUsed - 1.

    PFArrayD& operator =(const PFArrayD& rightSide);

    ~PFArrayD( );
protected:
    double *a; //for an array of doubles.
    int capacity; //for the size of the array.
    int used; //for the number of array positions currently in use.
};

#endif //PFARRAYD_H
```

```cpp
//This is the implementation file pfarrayd.cpp.
#include <iostream>
using std::cout;
#include "pfarrayd.h"

PFArrayD::PFArrayD( ) : capacity(50), used(0)
{
    a = new double[capacity];
}

PFArrayD::PFArrayD(int size) : capacity(size), used(0)
{
    a = new double[capacity];
}

PFArrayD::PFArrayD(const PFArrayD& pfaObject)
  :capacity(pfaObject.getCapacity( )), used(pfaObject.getNumberUsed( ))
{
    a = new double[capacity];
    for (int i =0; i < used; i++)
        a[i] = pfaObject.a[i];
}

double& PFArrayD::operator[](int index)
{
    if (index >= used)
    {
        cout << "Illegal index in PFArrayD.\n";
        exit(0);
    }

    return a[index];
}

PFArrayD& PFArrayD::operator =(const PFArrayD& rightSide)
{
    if (capacity != rightSide.capacity)
    {
        delete [] a;
        a = new double[rightSide.capacity];
    }

    capacity = rightSide.capacity;
    used = rightSide.used;
    for (int i = 0; i < used; i++)
        a[i] = rightSide.a[i];
    return *this;
}

PFArrayD::~PFArrayD( )
{
    delete [] a;
}

void PFArrayD::addElement(double element)
```

```
{
    if (used >= capacity)
    {
        cout << "Attempt to exceed capacity in PFArrayD.\n";
        exit(0);
    }
    a[used] = element;
    used++;
}

bool PFArrayD::full( ) const
{
    return (capacity == used);
}

int PFArrayD::getCapacity( ) const
{
    return capacity;
}

int PFArrayD::getNumberUsed( ) const
{
    return used;
}

void PFArrayD::emptyArray( )
{
    used = 0;
}
```

```
//This is the header file pfarraydbak.h. This is the interface for the
 class
//PFArrayDBak. Objects of this type are partially filled arrays of doubles.
//This version allows the programmer to make a backup copy and restore
//to the last saved copy of the partially filled array.
#ifndef PFARRAYDBAK_H
#define PFARRAYDBAK_H
#include "pfarrayd.h"

class PFArrayDBak : public PFArrayD
{
public:
    PFArrayDBak( );
    //Initializes with a capacity of 50.

    PFArrayDBak(int capacityValue);

    PFArrayDBak(const PFArrayDBak& Object);

    void backup( );
    //Makes a backup copy of the partially filled array.

    void restore( );
    //Restores the partially filled array to the last saved version.
    //If backup has never been invoked, this empties the partially filled
     array.

    PFArrayDBak& operator =(const PFArrayDBak& rightSide);

    ~PFArrayDBak( );
private:
    double *b; //for a backup of main array.
    int usedB; //backup for inherited member variable used.
};

#endif //PFARRAYD_H
```

```cpp
//This is the file: pfarraydbak.cpp.
//This is the implementation of the class PFArrayDBak.
//The interface for the class PFArrayDBak is in the file pfarraydbak.h.
#include "pfarraydbak.h"
#include <iostream>
using std::cout;

PFArrayDBak::PFArrayDBak( ) : PFArrayD( ), usedB(0)
{
    b = new double[capacity];
}

PFArrayDBak::PFArrayDBak(int capacityValue) : PFArrayD(capacityValue),
 usedB(0)
{
    b = new double[capacity];
}

PFArrayDBak::PFArrayDBak(const PFArrayDBak& oldObject)
              : PFArrayD(oldObject), usedB(0)
{
    b = new double[capacity];
    usedB = oldObject.usedB;
    for (int i = 0; i < usedB; i++)
        b[i] = oldObject.b[i];
}

void PFArrayDBak::backup( )
{
    usedB = used;
    for (int i = 0; i < usedB; i++)
        b[i] = a[i];
}

void PFArrayDBak::restore( )
{
    used = usedB;
    for (int i = 0; i < used; i++)
        a[i] = b[i];
}

PFArrayDBak& PFArrayDBak::operator =(const PFArrayDBak& rightSide)
{
    PFArrayD::operator =(rightSide);
    if (capacity != rightSide.capacity)
    {
        delete [] b;
        b = new double[rightSide.capacity];
    }

    usedB = rightSide.usedB;
    for (int i = 0; i < usedB; i++)
        b[i] = rightSide.b[i];

    return *this;
}

PFArrayDBak::~PFArrayDBak( )
{
    delete [] b;
}
```