```
Script started on 2021-11-04 13:10:59-05:00 [TERM="xterm" TTY="/dev/pts/1" COLUMNS=
k_gandhi6@ares:~$ cat Money.info
Name: Kush Gandhi

Class: CSC122-001

Project: Balance in all things...

levels: 7.5



4.5 = base project

2 = overloaded operators Check Class

1 = overloaded operators Money Class



Description: This program add checks like deposists

and get new balance and record checks.

k_gandhi6@ares:~$ cat Money.h
#ifndef MONEY_H

#define MONEY_H



#include <iostream>



class Money

{

    long all_cents;//monetary value stored as pennies



public:



    //initializes the object to dollars *100 cents

    Money(long dollars, int cents = 0) :

        all_cents(dollars * 100 + cents) {}
```

```
    //Initializes the object to $0.00

    Money(void) : all_cents(0) {}



    //overloaded operators

    Money operator+(const Money& amount) const

    {

        Money temp(*this);

        temp.all_cents += amount.all_cents;

        return temp;

    }



    Money operator-(const Money& amount) const

    {

        Money temp(*this);

        temp.all_cents -= temp.all_cents;

        return temp;

    }



    Money operator-(void) const

    {

        Money temp(*this);

        temp.all_cents = -temp.all_cents;

        return temp;

    }



    Money operator-=(const Money& amount)

    {
```

```
    this->all_cents -= amount.all_cents;
    return *this;
}

Money& operator+=(const Money& amount)
{
    this->all_cents += amount.all_cents;
    return *this;
}

//bool operators
bool operator==(const Money& amount) const
{
    return all_cents == amount.all_cents;
}
bool operator>=(const Money& amount) const
{
    return all_cents >= amount.all_cents;
}
bool operator>(const Money& amount) const
{
    return all_cents > amount.all_cents;
}
bool operator<(const Money& amount) const
{
    return all_cents < amount.all_cents;
}
bool operator<=(const Money& amount) const
{
    return all_cents <= amount.all_cents;
}
bool operator==(const Money& amount)
{
    return all_cents == amount.all_cents;
}
bool operator!=(const Money& amount)
{
    return all_cents != amount.all_cents;
}

//input and output function
void input(std::istream& ins = std::cin);
void output(std::ostream& outs = std::cout) const;

friend std::istream& operator >>(std::istream& ins, Money& amount)
{
    amount.input(ins);
    return ins;
}
friend std::ostream& operator<<(std::ostream& outs, const Money& amount)
{
    amount.output(outs);
    return outs;
}
```

```
    //getter

    double get_value(void) const

    {

        return all_cents/100.00;

    }


};


#endif

k_gandhi6@ares:~$ cat Money.cpp
#include <iostream>

#include <fstream>

#include <cctype>

#include <string>

#include "Money.h"

using namespace std;


//input and output func. for money class

void Money::input(std::istream& ins)

{

    long dollars, cents;

    char temp;

    ins >> temp >> dollars >> temp >> cents;

    all_cents = dollars * 100 + cents;

    return;

}
```

```
void Money::output(std::ostream& outs) const

{

    outs << '$' << all_cents / 100 << '.'

         << (all_cents % 100 < 10 ? '0' : '\0') << all_cents % 100;

    return;

}


//Now the check class

class Check

{

    long num;

    Money balance;

    bool cashed;

public:

    Check(void) : num(0), balance(0), cashed(false) {}

    Check(long n, Money b, bool c) : num(n), balance(b), cashed(c) {}


    Check(const Check & other) : num(other.num), balance(other.balance),

                                 cashed(other.cashed) {}

    //getters

    long get_num(void) const

    {

        return num;

    }

    Money get_balance(void) const

    {
        return balance;
    }
    bool get_cashed(void) const
    {
        return cashed;
    }

    //setters
    bool set_num(const long& n);//n = num
    bool set_balance(const Money& b);//b = balance
    bool set_cashed(const bool& c);//c = cashed

    bool operator>(const Check& c) const;

    void input(std::istream& is = std::cin);
    void output(std::ostream& os = std::cout) const;

    friend std::ostream& operator<<(std::ostream& out, const Check& c)
    {
        c.output(out);
        return out;
    }
    friend std::istream& operator>>(std::istream& in, Check& c)
    {
        c.input(in);
        return in;
    }
    friend Money operator+(const Money& amount, const Check& c)
    {
        return amount + c.get_balance();
    }
    friend Money operator-(const Money& amount, const Check& c)
    {
        return amount - c.get_balance();
    }

};

//basic swap function with 3 variables
void sort(Check*& p, size_t size);
inline void swap(Check& a, Check& b)
{
    Check t(a);
    a = b;
    b = t;
    return;
}

inline Money obtainBalance(Money original, Money check_total, Money deposit_total)
{//returns the balance
    return original + deposit_total - check_total;
}

bool Check::operator>(const Check& c) const
```

```
{
    Check temp(*this);
    //if block for comparison of non/cahed checks
    if (temp.cashed != temp.cashed)
    {
        return !temp.cashed;
    }
    return temp.num > c.num;
}

bool Check::set_num(const long& n)
{
    if (n >= 0)
    {
        num = n;
        return true;
    }
    return false;
}

bool Check::set_balance(const Money& b)
{
    balance = b;
    return true;
}

bool Check::set_cashed(const bool& c)
```

```
{
    cashed = c;
    return true;
}

void Check::output(ostream& os) const
{
    Check c(*this);
    os << c.num << '\t' << c.balance;
    return;
}

void Check::input(istream& is)
{//check for y or Y input from user
    char chr;
    is >> num >> balance >> chr;
    cashed = (toupper(chr) == 'Y') ? true : false;
}

//the holy sort function
void sort(Check*& p, size_t size)
{
    bool check = false;//bool flag for sorting
    int i = 0;//counter
    while (i < size && !check)//loop
    {
        check = true;
        for (size_t j = 0; j + i + 1 < size; ++j)
        {
            if (p[j] > p[j + 1])
            {
                swap(p[j], p[j + 1]);
                check = false;
            }
        }
        ++i;
    }
    return;
}

int main(void)
{
    //last balance and new balance variables
    Money lastBalance, newBalance;

    //variables for the check class
    Check* chekBook;
    size_t bookSize;
    Money calcTotal;
    size_t book = 0;

    //vatiables for Money class
    Money* chekDeposist;
    size_t deposistSize;
```

```
    Money deoposistTot;
    size_t deposist = 0;

    cout << "\t\tWelcome to the Check Balance Program!!" << endl;
    cout << "\nINSTRUCTIONS: ENTER FORMAT ($00.00)" << endl;
    cout << "\n\nEnter the amount of checks to process: ";
    cin >> bookSize;

    chekBook = new Check[bookSize];
    if (chekBook == NULL)
    {//if no space to allocate...
        cerr << "\nNo space left to allocate " << bookSize <<
                " Please free up space!!" << endl;
    }

    for (size_t i = book; i < bookSize; i++)
    {
        cin >> *(chekBook + i);
    }

    cout << "=============================================="
         << "\n\nEnter the balance on the account: ";
    cin >> lastBalance;
    cout << "\nHow many deposists you need: ";
    cin >> deposistSize;

    chekDeposist = new Money[deposistSize];
```

```
if (chekDeposist == NULL)
{//if no space to allocate...
    cerr << "No space left to allocate " << bookSize <<
            "Please free up space!!" << endl;
}

for (size_t i = 0; i < deposistSize; i++)
{
    cout << "\nEnter the amount of deposit " << i + 1 << ":  ";
    cin >> *(chekDeposist + i);
}

//call the sort function
sort(chekBook, bookSize);

//loops for calcuation operations
for (size_t i = deposist = 0; i < deposistSize; ++i)
{
    deoposistTot += *(chekDeposist + i);
}

book = 0;
while (book < bookSize && chekBook[book].get_cashed())
{
    calcTotal += chekBook[book].get_balance();
    ++book;
}
```

```

//call to balance function to get total
newBalance = obtainBalance(lastBalance, calcTotal, deoposistTot);

while (book < bookSize)
{
    calcTotal += chekBook[book].get_balance();
    ++book;
}
book = 0;

//prints the calculation of the checks and deposists and new balance
cout << "\n=============================================";
cout << "\nThe total of all checks are: " << calcTotal
     << "\nThe total of the deposits are: " << deoposistTot
     << "\nYour new balance is: " << newBalance << endl;

//next thing to do is look for cashed checks
cout << "\n\nCashed Checks" << endl;

while (book < bookSize && chekBook[book].get_cashed())
{
    cout << "\n" << *(chekBook + book);
    ++book;
}

//now for the uncashed checks!!
```

```
    cout << "\n\nUncashed Checks" << endl;


    while (book < bookSize)

    {

        cout << "\n" << *(chekBook + book);

        ++book;

    }

    cout << endl;

    //release the memmory

    delete[] chekBook;

    delete[] chekDeposist;

    chekBook = NULL;

    chekDeposist = NULL;


    return 0;

}

k_gandhi6@ares:~$ CPP Money
Money.cpp***
In file included from Money.cpp:5:
Money.h: In member function ‘Money
Money::operator-(const Money&) const’:
Money.h:27:34: warning: unused parameter
‘amount’ [-Wunused-parameter]
   27 |     Money operator-(const Money& amount) const
      |                     ~~~~~~~~~~~~~^~~~~~
Money.h: In member function ‘double
Money::get_value() const’:
Money.h:101:16: warning: conversion
from ‘long int’ to ‘double’ may
change value [-Wconversion]
  101 |         return all_cents/100.00;
      |                ^~~~~~~~~
Money.cpp: In function ‘void swap(Check&,
Check&)’:
Money.cpp:87:9: warning:
implicitly-declared ‘constexpr Check&
Check::operator=(const Check&)’ is deprecated
[-Wdeprecated-copy]
   87 |     a = b;
      |         ^
Money.cpp:35:5: note:
because ‘Check’ has user-provided
‘Check::Check(const Check&)’
   35 |     Check(const Check & other) :
   num(other.num), balance(other.balance),
      |     ^~~~~
Money.cpp:88:9: warning:
implicitly-declared ‘constexpr Check&
Check::operator=(const Check&)’ is deprecated
[-Wdeprecated-copy]
   88 |     b = t;
      |         ^
Money.cpp:35:5: note:
because ‘Check’ has user-provided
‘Check::Check(const Check&)’
   35 |     Check(const Check & other) :
   num(other.num), balance(other.balance),
      |     ^~~~~
Money.cpp: In member function ‘bool
Check::operator>(const Check&) const’:
Money.cpp:101:21: warning:
self-comparison always evaluates to false
[-Wtautological-compare]
  101 |     if (temp.cashed !=
  temp.cashed)
      |         ~~~~~~~~~~~ ^~
      ~~~~~~~~~~~
Money.cpp: In function ‘void sort(Check*&,
size_t)’:
Money.cpp:149:14: warning: comparison of
integer expressions of different signedness: ‘int’
and ‘size_t’ {aka ‘long unsigned
int’} [-Wsign-compare]
  149 |     while (i < size && !check)//loop
      |            ~~^~~~~~


k_gandhi6@ares:~$ ./Money.out
                Welcome to the Check Balance Program!!

INSTRUCTIONS: ENTER FORMAT ($00.00)


Enter the amount of checks to process: 4
1001 $23.43 y
1002 $45.32 y
1003 $12.53 y
1004 $57.90 n
==============================================

Enter the balance on the account: $99.99
```

```
How many deposists you need: 3

Enter the amount of deposit 1:  $23.49

Enter the amount of deposit 2:  $12.44

Enter the amount of deposit 3:  $75.99

===============================================
The total of all checks are: $139.18
The total of the deposits are: $111.92
Your new balance is: $0.00


Cashed Checks

1001    $23.43
1002    $45.32
1003    $12.53

Uncashed Checks

1004    $57.90
k_gandhi6@ares:~$ exit
exit

Script done on 2021-11-04 13:13:34-05:00 [COMMAND_EXIT_CODE="0"]
```