1 Reading

Read and complete all in-text exercises for the following chapters:

- Ch 1 & 2

2 Goals

- Practice using inheritance in `C++`;
- Ensure your own `C++` editing and compilation environment is set up; and
- Ensure you can submit your assignment for grading.

Note you are free to use whatever editor and environment you like for this class, but your programs must be able to compile and run on `ada` (which is running Ubuntu) using `g++`.

3 Instructions

1. Implement the `Security`, `Stock`, and `StockOption` classes defined in the listings below. Each class implementation must be in a separate file: the implementation for the `Security` class definition in `security.h` should go in the file `security.cpp`), the implementation for `Stock` should go in the file `stock.cpp`, and the implementation for `StockOption` should go in the file `stock_option.cpp`. Be sure you follow both the course style guide and the guidelines on the cover sheet in your implementation.
2. Ensure your program passes the tests (given as `assert` statements) in `hw1.cpp`. Note that there is a small "bug" in the class header files given below that you will need to find and fix for all the tests to pass. Fixing the bug must not involve making any changes to `hw1.cpp`.
3. Hand in a hard-copy printout of your source code, with a cover sheet. Be sure to *carefully* read over and follow all guidelines outlined in the cover sheet. Your hard-copy should be stapled and turned in during class on the due date.
4. Submit your source code using the `dropoff` command on `ada`. Your source code must be submitted by class on the due date.

4 Addtional Details

Security, Stock, and Option Details. Each `Security` class object represents the ownership of a financial asset consisting of a symbol identifying a company, the security's current share value

(i.e., the current value of one share in the security), and the number of holdings of the security. The `Stock` class is a subclass of `Security` (a type of security) that denotes ownership in a stock, which adds the purchase price of each individual share of the stock (where the holdings is the number of shares owned). The `StockOption` class is a subclass of `Stock` denoting a "sell" stock option. `StockOption` adds a strike price to the notion of a stock, which is the value that each share can be sold at during the lifetime of the option contract. Note that for a `StockOption`, the purchase price is the amount the option costs per share and not the actual stock price. For instance, a given option for a stock may cost $2 per share, even though the price to purchase an actual share is $42. If the strike price were $48 per share with a current share price of $42, the option would be worth $4 per share, which is the difference between the strike price (what you are allowed to sell the stock for) and the current share price (what you can buy the stock for) less the original option's price.

Compiling and Running. To compile `hw1.cpp` on `ada`, the Lubuntu Virtual Machine, or another system running Unix/Linux, from a terminal type:

```
$ g++ -g -Wall security.cpp stock.cpp stock_option.cpp hw1.cpp -o hw1
```

where `$` denotes the command prompt (i.e., you don't type `$` as part of the command). Note that for the above command to work, you must run it from within the directory where your files are stored. Also note that you can rerun the command by using the up-arrow key to cycle through previous commands (this saves you from having to type the command over and over). Finally, the `-g` and `-Wall` are compiler "flags": the `-g` flag tells the compiler to generate debug symbols in the executable file and the `-Wall` flag tells the compiler to provide a number of "standard" warnings during the compilation process.

5 Code Listings

Listing 1: `security.h`

```
#ifndef SECURITY_H
#define SECURITY_H

#include <string>

class Security
{
public:
  // create a security with the given symbol
  Security(std::string the_symbol);

  // company symbol
  std::string get_symbol();

  // update the current share value
  void set_share_value(double current_share_value);

  // current share value
  double get_share_value() const;

  // update the number of holdings
  void set_holdings(int number_of_holdings);

  // current number of holdings
  int get_holdings() const;

  // total current value of all holdings
  double market_worth() const;

private:
  std::string symbol;           // security identifier
  double share_value = 0;       // each share's current market value
  int holdings = 0;             // number of total shares of security held
};

#endif
```

Listing 2: `stock.h`

```
#ifndef STOCK_SHARE_H
#define STOCK_SHARE_H

#include <string>
#include "security.h"

class Stock : public Security
{
public:
  // create a stock with the given company symbol
  Stock(std::string the_symbol);

  // set the purchase price of the holdings
  void set_purchase_price(double the_purchase_price);

  // purchase price
  double get_purchase_price() const;

  // compute the net worth of the stock holdings
  double sell_value() const;

 private:
  double purchase_price = 0;    // price per holding
};

#endif
```

Listing 3: stock_option.h

```
#ifndef STOCK_OPTION_H
#define STOCK_OPTION_H

#include <string>
#include "stock.h"

class StockOption : public Stock
{
public:
  // crate a stock option with the given symbol
  StockOption(std::string the_symbol);

  // set the strike price per share
  void set_strike_price(double the_strike_price);

  // strike price per share
  double get_strike_price() const;

  // the net worth of the option
  double sell_value() const;

private:
  double strike_price = 0;      // price to sell per holding
};
```

Listing 4: `hw1.cpp`

```cpp
#include <iostream>
#include <assert.h>
#include "security.h"
#include "stock.h"
#include "stock_option.h"

using namespace std;

// returns true if stock value is positive
bool should_sell(Stock& the_stock);


int main(int argc, char** argv)
{
  Security s1("GOOG");
  s1.set_share_value(1245);
  s1.set_holdings(120);
  assert(s1.get_share_value() == 1245);
  assert(s1.get_holdings() == 120);
  assert(s1.market_worth() == 1245*120);

  Stock s2("APPL");
  s2.set_share_value(204);
  s2.set_holdings(76);
  s2.set_purchase_price(175);
  assert(s2.get_share_value() == 204);
  assert(s2.get_holdings() == 76);
  assert(s2.get_purchase_price() == 175);
  assert(s2.market_worth() == 204*76);
  assert(s2.sell_value() == (204-175)*76);

  StockOption s3("AMZN");
  s3.set_share_value(1823);
  s3.set_holdings(500);
  s3.set_purchase_price(5.25);
  s3.set_strike_price(1828);
  assert(s3.get_share_value() == 1823);
  assert(s3.get_holdings() == 500);
  assert(s3.get_purchase_price() == 5.25);
  assert(s3.get_strike_price() == 1828);
  assert(s3.market_worth() == 1823*500);
  assert(s3.sell_value() == (1828-5.25)*500 - 1823*500);

  assert(should_sell(s2) == true);
  assert(should_sell(s3) == false);
}

bool should_sell(Stock& the_stock)
{
  return the_stock.sell_value() > 0;
}
```