HW7, Exception

HW7

```
List<string> grocery;
   grocery.add("milk"); grocery.add("eggs"); grocery.add("bread");
   check("add()/size()", grocery.size(), 3 );
   grocery.display(); // 각 줄에 milk, eggs, bread가 출력됨 (세줄)
   grocery.clear();
   check("clear()", grocery.size(), 0 );

   List<Student> students;
   Student s1("Shin", 88); Student s2("Kim", 72); Student s3("Cho", 91);
   students.add( s1 ); students.add( s2 ); students.add( s3 );
   Student s4("Shin", 90);

   check("exist()", students.exist( s4 ), false );
   check("exist()", students.exist( s3 ), true );
```

HW7Sol

```
template <typename T>
class List{
   vector<T> data;
public:
   List() { }
   void add(T item) { data.push_back(item); }
   int size() { return data.size(); }
   void clear() { data.clear(); }
   void display() {
      for(int i = 0; i < data.size(); ++i) {
         cout << data[i] << endl;
   }}
   bool exist(T item) {
      return (find( data.begin(), data.end(), item ) != data.end());
   }
};
```

Exception Handling

- Handle errors (exception) to avoid crash
- Error occurs
  - throw an exception
  - Any object can be thrown
  - Better use std::exception or its subclasses
- Handle error
  - catch the exception
  - Catches based on the type of thrown object

Constant throw-catch

- No error handling

```
int a=10, b=0;
if( !b ) cout << “ERROR” << endl;
else cout << a/b;
```

- Throw-catch handling

```
try {
    if( b == 0 ) throw 1;
    cout << a / b;
} catch (int e) {
    cout<< “EXCEPTION: “ << e << endl;
}
```

- Throw “const int DeivByZero=1” instead of 1

Constant throw-catch: Limitation

```
try {
    if( b == 0 ) throw 1;
    if( a == 0 ) throw 2;
    cout << a / b;
} catch (int e) {
    if( e == 1 ) cout<< “ERR: DivideByZero“ << endl;
    else if( e == 2 ) cout << “ERR: DivideZero” <<
endl;
}
```

- Hard to distinguish between errors

Better way: Custom Exception Type

```
try {
        if( b == 0 ) throw DivideByZero();
        if( a == 0 ) throw DivideZero();
        cout << a / b;
} catch (DivideByZero &e) {
        cout<< “ERR: DivideByZero“ << endl;
} catch (DivideZero &e) {
        cout << “ERR: DivideZero” << endl;
} catch (…) {
        cout << “ERR: Unknown” << endl;
}
```

Exception Class

```
class DivideByZero : public runtime_error
{
public:
   DivideByZero() :
   runtime_error("Divide by zero exception") {}
};
```

Exception Class

```
double number1, number2, ratio;
  cout << "Enter a numerator: ";
  cin >> number1;
  cout << "Enter a denominator: ";
  cin >> number2;
  try {
     ratio = quotient(number1, number2);
     cout << "Result is: " << ratio << endl;
  }
  catch (DivideByZero &except) {
     cout << except.what() << endl;
  }
```