Example Project B (C++)

This project will demonstrate:
1. Virtual functions.
2. Inline member functions.
3. Function templates.
4. Class templates.
5. Multiple inheritance.

Activities
1. Create an `Assignment` object called `homework` in the stack. Call the display grade function.

   SOLUTION
   ```
   The activity's numeric score is 80.0
   The activity's letter grade is B
   ```

2. Create a `FinalExam` object called `exam` in the stack. Call the display grade function.

   SOLUTION
   ```
   The activity's numeric score is 97.0
   The activity's letter grade is A
   ```

3. Create a `PassFailActivity` object called `classwork` in the stack. Call the display grade function.

   SOLUTION
   ```
   The activity's numeric score is 100.0
   The activity's letter grade is P
   ```
   CAVEAT The output was this:
   ```
   The activity's letter grade is A
   ```
   To fix this we must tell the compiler to dynamically link the `getLetterGrade()` method. *(See step 4.)*

4. Add the `virtual` keyword to the `getLetterGrade()` function both in the Assignment and PassFailActivity classes. As C++ classes can alway be derived (no Java equivalent to a `final` class) it is best practice to declare any method that could potentially be redefined in derived classes as `virtual`. Run activity 3 again.

   SOLUTION
   ```
   The activity's numeric score is 100.0
   The activity's letter grade is P
   ```

5. Create an `AverageFunctions.h` file and write a template function called `totalAssignmentAverage()`. The function should take in an array of pointers to the `homework`, `exam` and `classwork` and return the average score of all these assignments.

Test this template function by producing an average as a `double` and `integer` precision value. Don't forget to delete the memory allocated for the array of assignments after the test.

SOLUTION

AverageFunctions.h

```
template <class T>
T totalAssignmentAverage(Assignment ** assignments, int
count)
{

    T sum;

    sum = static_cast<T>(0);

    for( int i = 0; i < count; i++ ) {
        sum += assignments[i]->getScore();
    }

    return sum / static_cast<T>(count);
}
```

main.cpp

```
Assignment ** assignments = new Assignment*[3];
assignments[0] = &homework;
assignments[1] = &exam;
assignments[2] = &classwork;

{
    double assignmentAverage;
    assignmentAverage =
totalAssignmentAverage<double>(assignments, 3);
    cout << "Average with floating point precision: " <<
assignmentAverage << endl;
}

{
    int assignmentAverage;
    assignmentAverage =
totalAssignmentAverage<int>(assignments, 3);
    cout << "Average with integer precision: " <<
assignmentAverage << endl;
}

delete [] assignments;
```

6. Rewrite and test `totalAssignmentAverage()` such that `<int>` and `<double>` do not need to be declared when calling `totalAssignmentAverage()` and we can assign the sum to a variable simply by passing it as a parameter. Call the rewritten version `totalAssignmentAverageRewrite()`.

SOLUTION

AverageFunctions.h

```
template <class T>
void totalAssignmentAverage(T & val, Assignment **
assignments, int count)
{

    T sum = totalAssignmentAverage<T>(assignments, count);

    val = sum;

}
```

main.cpp

```
Assignment ** assignments = new Assignment*[3];
assignments[0] = &homework;
assignments[1] = &exam;
assignments[2] = &classwork;

{
    double assignmentAverage;
    totalAssignmentAverageRewrite(assignmentAverage,
assignments, 3);
    cout << "Average with floating point precision: " <<
assignmentAverage << endl;
}

{
    int assignmentAverage;
    totalAssignmentAverageRewrite(assignmentAverage,
assignments, 3);
    cout << "Average with integer precision: " <<
assignmentAverage << endl;
}

delete [] assignments;
```

7. Use the `SimpleVector` class as the array to store the assignments for the `totalAssignmentAverage()` function. Rewrite and test the `totalAssignmentAverage()` function to work with the `SimpleVector` class and call it `totalAssignmentAverageWithVector()`.

SOLUTION

AverageFunctions.h

```
template <class T, class V>
void totalAssignmentAverageWithVector(T & val, V vector)
{

    Assignment ** assignments = new
Assignment*[vector.size()];

    for( int i = 0; i < vector.size(); i++ ) {
        assignments[i] = vector[i];
    }

    totalAssignmentAverageRewrite(val, assignments,
vector.size());

}
```

main.cpp

```
SimpleVector<Assignment *> assignmentVector(3);
assignmentVector[0] = &homework;
assignmentVector[1] = &exam;
assignmentVector[2] = &classwork;

{
    double assignmentAverage;
    totalAssignmentAverageWithVector(assignmentAverage,
assignmentVector);
    cout << "Average with floating point precision: " <<
assignmentAverage << endl;
}

{
    int assignmentAverage;
    totalAssignmentAverageWithVector(assignmentAverage,
assignmentVector);
    cout << "Average with integer precision: " <<
assignmentAverage << endl;
}
```

8. Create a class called `DateTime` that is derived from the `Date` and `Time` class. Use the `setDateTime()` and `getDateTime()` member functions in `Assignment` to output the date for each assignment.

`- char dateTimeString[20]`
`+ DateTime()`
`+ DateTime(int, int, int, int, int, int)`
`+ const char * getDateTime() const`

Note: `strcpy` and `sprintf` requires a string as `const char *`.

SOLUTION

main.cpp

```
homework.setDateTime(14, 9, 2013, 11, 45, 12);
cout << "Homework DateTime: " << homework.getDateTime() <<
endl;

exam.setDateTime(10, 9, 2013, 9, 25, 33);
cout << "Exam DateTime: " << exam.getDateTime() << endl;

classwork.setDateTime(26, 8, 2013, 4, 51, 00);
cout << "Classwork DateTime: " << classwork.getDateTime()
<< endl;
```

DateTime.h

```
class DateTime : public Date, public Time
{

protected:
    char dateTimeString[20];

public:
    // Default constructor
    DateTime();

    // Constructor
    DateTime(int, int, int, int, int, int);

    // Accessor function
    const char * getDateTime() const
    {
        return dateTimeString;
    }

};
```

DateTime.cpp

```
DateTime::DateTime() : Date(), Time()
{
    strcpy(dateTimeString, "1/1/1900 00:00:00");
}

DateTime::DateTime(int dy, int mon, int yr, int hr, int mt,
int sc) : Date(dy, mon, yr), Time(hr, mt, sc)
{

    sprintf(dateTimeString, "%d/%d/%d %d:%d:%d",
getMonth(), getDay(), getYear(), getHour(), getMin(),
getSec());

}
```