POSD2019f Midterm Test 2

Readme first

- Exam time: 1:30 pm to 5:30 pm, 2019/11/30
- Commit and push often.
- Push disabled at 5:30 pm sharp.
- Code:40% TA-Unit-Test: 40% Your Unit-Test:20%
- Gitlab url : https://ssl-gitlab.csie.ntut.edu.tw/users/sign_in .
- Jenkins url : https://ssl-jenkins.csie.ntut.edu.tw/login?from=/ .
- Go to https://ssl-gitlab.csie.ntut.edu.tw/course/posd2019f_midterm2, and get the file and code that TA gives to you, copy these code and files to your homework repository. Do not push any commit to posd2019f_midterm2 repository.

Resources allowed to use

- Design Patterns (GoF) textbook
- projects in class (https://ssl-gitlab.csie.ntut.edu.tw/yccheng/posd2019f)
- Your own homework repository on Gitlab
- cplusplus.com (http://www.cplusplus.com)
- Prescribed Dictionary(https://dictionary.cambridge.org/zht)

Attention!!

You cannot visit any other website and you must turn off your mobile phone during the midterm exam, or you will be considered as cheating.

Violation of the rules:

First time : Deduct 50 points

Second time: Calculated by zero

Midterm Project Structure

- bin
  - ut_all
- obj
  - evaluate_visitor.o
  - flattened_set_visitor.o
- src
  - element.h
  - evaluate_visitor.cpp
  - evaluate_visitor.h
  - flattened_set_visitor.cpp
  - flattened_set_visitor.h
  - integer.h
  - iterator.h
  - null_iterator.h
  - set.h
  - visitor.h
- test
  - ut_midterm2.cpp
- makefile

Write the corresponding makefile to generate executable file which named ut_all in bin folder!

Problems

You will find that "Element" class is given to you already.

1. (Code:5% TA-Unit-Test:6% Your Unit-Test: 4%)

   We want an "Integer" object which is a kind of "Element".

   The "Integer" object has the following member functions:

   You should inherit "Element" class!

Integer(int i)	// (1)
void add(Element * element)	// (2)
int size()	// (3)
std::string toString()	// (4)
Iterator * createIterator()	// (5)
void setOperator(char c)	// (6)
void accept(Visitor * visitor)	// (7)

(1) Create an "Integer" object with int.
For example:
Integer integer(1);
is to create an "Integer" object whose value is 1.

(2) This is for uniformity, so throw a string of "It's an integer!"

(3) Always return 1 in this function.

(4) Return the string of the value of "Integer" object.
For example:
ASSERT_EQ("1", integer. toString());

(5) You should return NullIterator. (I will give you skeleton in problem3)

(6) This is for uniformity, so throw a string of "Cannot set operator on it!"

(7) This is reserved for Visitor pattern. We will explain the visitor you need to implement in problem 4 and 5.

2. (Code:5% TA-Unit-Test:6% Your Unit-Test: 4%)

We want a Set object whose child elements can be integers or sets of integers of arbitrary level of nesting. For example,

```
{1, {2}, {3, 4, {5, { }}}}
```

is a Set object.

Note that { } is the empty set.

The Set object has the following member functions.

You should inherit "**Element**" class!

Set ()	// (1)
void add(Element * element)	// (2)
int size()	// (3)
std::string toString()	// (4)
Iterator * createIterator()	// (5)
void setOperator(char ope)	// (6)
void accept(Visitor * visitor)	// (7)

(1) Create an "Set" object (**Default operator is +**)

(2) Add child element to set

(3) return the size of set

(4) Return the string of the value of "Set" object.

For example:

```
Integer integer(1);
Set set();
Set.add(integer);
ASSERT_EQ("{1}", Set. toString());
```

(5) Create Iterator for accessing the child element in Set

'SetIterator' should be inner class In 'Set'class

(I will give you skeleton in problem3)

(6) Set operator for "set" object

You need to support + - * /

If operator **not** + - * /, you should throw a string of "Invalid operator!"

(7) This is reserved for Visitor pattern. We will explain the visitor you need to implement in problem 4 and 5

To construct the set in the example: {1, {2}, {3, 4, {5, { }}}}, first create an empty set, add Integer object 1, then add Set object {2}, and finally add Set object {3, 4, {5, { }}}.

Then the result of toString() will be "{1, {2}, {3, 4, {5, { }}}}"

3. (Code:10% TA-Unit-Test:9% Your Unit-Test: 4%)

Itrerator pattern for accessing the child element

Iterator skeleton

```
class Iterator {
public:
 virtual void first() = 0;
 virtual Element* currentItem() = 0;
 virtual void next() = 0;
 virtual bool isDone() = 0;
};
```

SetIterator skeleton

```
class SetIterator:XXX {  // You should inherit Iterator
  public:
    SetIterator(Set * s){}
    void first(){}
    Element* currentItem(){}
      // If iterator is done, you should throw string "No current item!"
    void next(){}
      /* If iterator does not exist  next item,
         you should throw string "Moving past the end!" */
    bool isDone(){}
  };
```

NullIterator skeleton

```
class NullIterator:XXX{  // You should inherit Iterator
public:
   NullIterator(){}
      // You should throw sting of  "No child member!"
   void first(){}
      // You should throw sting of  "No child member!"
   Element* currentItem(){}
      // You should throw sting of  "No child member!"
   void next(){}
      // You should throw sting of  "No child member!"
   bool isDone(){}
      // You should throw sting of  "No child member!"
};
```

In problem4 and 5, you will have to implement two visitors.

The visitor will have the following skeleton:

Visitor skeleton:

```
class Visitor{
public:
    virtual void visitInteger(Integer* i) = 0;
    virtual void visitSet(Set* s) = 0;
};
```

"FlattenedSetVisitor" and "EvaluateVisitor" should inherit "Visitor" class.

You should use Iterator in problem 3 to access child element

4. (Code:10% TA-Unit-Test:10% Your Unit-Test: 4%)

   Implement a feature to let a set generate a *flattened* set. For example,

   the flattened set of

   {1, {2}, {3, 4, {5, { }}}}

   Should be

   {1,2,3,4,5}.

   the flattened set of an integer is also a integer.

   1's flattened set should be 1.

   Implement a "**FlattenedSetVisitor**" class for this feature.

   Add a function "***getResult()***" to "**FlattenedSetVisitor**" class

   Element * getResult(); // return the flattened element

You can use the same 'FlattenedSetVisitor' more than one time, so you should initialize the status each time.

5. (Code:10% TA-Unit-Test:9% Your Unit-Test: 4%)

Implement a "**EvaluateVisitor**" class for this feature.

Add a function "***getResult()***" to "**EvaluateVisitor**" class

double getResult(); // return the result

In problem 3, we can set an operator on a set.

We can compute the set by the operator.

For example,

1. {1, 2, 3} and its operator is +, so the result of evaluation is

   $1 + 2 + 3 = 6$.

2. {1, 2, {2,1}} and its operator is +

   Subset {2,1} and its operator is -

   , so the result of evaluation is $1 + 2 + (2 - 1) = 4$

You should support + - * /

{} return 0

If operator is / and divisor is zero, you should throw string of "Divide by zero!"

You can use the same 'EvaluateVisitor' more than one time, so you should initialize the status each time.

9