Course C^{++}, Exercise Number 10

Date: 23.05.2013

This exercise is about templates. The task is to implement the class below. It cannot be called `union`, because `union` is a reserved word in C^{++}.

```
template< typename A, typename B >
class unionof
{
   A* a;
   B* b;
      // Invariant: Exactly one of them is non-zero.

public:
   unionof( const A& a );
   unionof( const B& b );
   unionof( const unionof& u );

   void operator = ( const A& a );
   void operator = ( const B& b );
   void operator = ( const unionof& u );

   const A& first( ) const;
   A& first( );

   const B& second( ) const;
   B& second( );

   bool hasfirst( ) const;
   bool hassecond( ) const;

   ~unionof( );
};

template< typename A, typename B >
std::ostream& operator << ( std::ostream& stream,
                            const unionof< A, B > & u );
```

1. Implement the constructors of `unionof`.
2. Implement the assignment operators of `unionof`.
3. Implement the desctructor of `unionof`.
4. Implement the `first( )` methods and the the `second( )` methods.
5. Implement `hasfirst( ) const` and `hassecond( ) const`.
6. Implement `operator << ( std::ostream& , const unionof< > & )`. You will need to make it friend of `class unionof`, which is not as easy at it seems. Write, before the definition of class unionof:

   ```
   template< typename A, typename B > class unionof;

   template< typename A, typename B >
   std::ostream& operator << ( std::ostream& stream,
                               const unionof< A, B > & );
   ```

   The friend decleration has the following form:

   ```
   friend std::ostream& operator << < > ( std::ostream& stream,
                                          const unionof< A, B > & );
   ```

   Now it should work. If you declare something friend, that is not a template, then two things happen: (1) it is declared, (2) it is made a friend. With templates, step (1) is not done automatically, so you have to declare the object (in this case `operator <<`) in advance. Because the declaration of `operator <<` requires `unionof`, it has to be declared incompletely before `operator <<`.
7. Make sure that `unionof< >` has no memory leaks. Test it on a few different classes, e.g. `double, int, std::string`, etc.