Course C^{++}

Exercise List 5

Date: 21.03.2013*+ 1 week

This week we complete the differentation program. Despite its length, I think that this exercise is easy.

1. In case you didn't do it yet in the previous task, add a method `nrsubtrees( ) const` to class `tree`.

   In order to be able to write a reasonable differentation program, we need to be able to represent numbers. If one would do this in serious code, one would have to define different types of `trnodes`, and use inheritance to deal with different types of trees. Since we didn't cover inheritance yet, we have to find another solution in this exercise. We will simply represent `unsigned int` by its string representation. This would be not acceptable in real life code, but it is fine for this exercises.

   In order to deal with the unsigneds, add the following code to `tree.cpp`:

```
#include <sstream>

bool tree::isunsigned( ) const
{
   // Numbers don't have subtrees:

   if( pntr -> subtrees. size( ) != 0 )
      return false;

   const std::string& s = pntr -> f;
   for( auto p = s. begin( ); p != s. end( ); ++ p )
   {
      if( *p < '0' || *p > '9' ) return false;
   }
   return true;
}

unsigned int tree::getunsigned( ) const
{
```

*start of astronomical spring

```
      // We make a stringstream from the string, and
      // read the unsigned from it.

      std::istringstream s( pntr -> f );
      unsigned int res = 0;
      s >> res;
      return res;
   }

   tree::tree( unsigned int i )
      : pntr(0)
   {
      // This is a rare case where immediate initialization is not possible.

      std::ostringstream s;
      s << i;

      pntr = new trnode( s. str( ), { }, 1 );
   }
```

The methods must also be declared in `tree.h` of course.

2. Write a function `tree diff( const tree& t, const std::string& var )` (I don't think it should be a member of `tree`) that can do the differentation. Add some usual rules for $+, -, /, \star$ and some more primitive functions, like $\sin, \cos, e^x$, etc.

3. We still have to take a look at R-value references. `std::vector` is implemented pretty much in the same way as the stack in Exercise 3. This means that `class std::vector` looks like this

```
template< typename X > class vector<X>
{
   X* ref;
   unsigned int current_capacity;
   unsigned int current_size;
   ...
};
```

The content of the vector is stored on the heap.

R-value references were introduced to solve the following problem:

```
std::vector< int > somevector( )
{
   Construct some big std::vector v1;
```

```
    Construct some big std::vector v2;

    if( something ) return v1; else return v2;
}

main(... ) { std::vector< int > res = somevector( ); ... }
```

When the vector is returned, it has to be copied into the variable `res` of `main`. For this, the copy constructor will be used. After copying, the original vector will be destroyed. This implies that all data in the selected vector will be copied, and after that destroyed.

Since `std::vector` contains only a pointer to the heap, it would be much nicer to copy only the `ref` of the returned vector into `res.ref`. After that, the original `ref` can be replaced by a pointer to a zero-length segment, which will then be destroyed. From the outside, nobody would notice the difference.

In order to allow such pointer transfers (with transfer of ownership), the R-value references were introduced in C^{++}-11. An R-value reference is a reference to an object that is about to be destroyed. Put differently, the compiler constructs an R-value reference when it is certain that the object is being used for the last time. If it doesn't find a method that fits the R-value reference, it will try a usual reference, etc. The copy constructor `std::vector<X> ( std::vector<X> && v )` could be implemented as

```
    : ref( v. ref )
{
    v. ref = new X[0];
        // One could also change the class invariant,
        // and to allow ref = 0, when the vector is empty.
}
```

Write print statements in the two constructors of `tree` (with usual reference, and with R-value reference), and find for each of the constructors a situation where it is used.

N.B. I originally wrote the R-value reference as `const`, but that makes no sense. Remove `const` in the R-value reference constructor.