Function Output Iterator

Author: David Abrahams, Jeremy Siek, Thomas Witt
Contact: dave@boost-consulting.com, jsiek@osl.iu.edu, witt@ive.uni-hannover.de
Organization: Boost Consulting, Indiana University Open Systems Lab, University of Hanover Institute for Transport Railway Operation and Construction
Date: 2004-01-13
Copyright: Copyright David Abrahams, Jeremy Siek, and Thomas Witt 2003. All rights reserved

abstract: The function output iterator adaptor makes it easier to create custom output iterators. The adaptor takes a unary function and creates a model of Output Iterator. Each item assigned to the output iterator is passed as an argument to the unary function. The motivation for this iterator is that creating a conforming output iterator is non-trivial, particularly because the proper implementation usually requires a proxy object.

Table of Contents

```
template <class UnaryFunction>
class function_output_iterator {
public:
  typedef std::output_iterator_tag iterator_category;
  typedef void                     value_type;
  typedef void                     difference_type;
  typedef void                     pointer;
  typedef void                     reference;

  explicit function_output_iterator();

  explicit function_output_iterator(const UnaryFunction& f);

  /* see below */ operator*();
  function_output_iterator& operator++();
  function_output_iterator& operator++(int);
private:
  UnaryFunction m_f;     // exposition only
};
```

`function_output_iterator` requirements

`UnaryFunction` must be Assignable and Copy Constructible.

`function_output_iterator` models

`function_output_iterator` is a model of the Writable and Incrementable Iterator concepts.

`function_output_iterator` operations

`explicit function_output_iterator(const UnaryFunction& f = UnaryFunction());`

Effects: Constructs an instance of `function_output_iterator` with `m_f` constructed from `f`.

`operator*();`

Returns: An object `r` of unspecified type such that `r = t` is equivalent to `m_f(t)` for all `t`.

`function_output_iterator& operator++();`

Returns: `*this`

`function_output_iterator& operator++(int);`

Returns: `*this`

Example

```
struct string_appender
{
    string_appender(std::string& s)
        : m_str(&s)
    {}

    void operator()(const std::string& x) const
    {
        *m_str += x;
    }

    std::string* m_str;
};

int main(int, char*[])
{
  std::vector<std::string> x;
  x.push_back("hello");
  x.push_back(" ");
  x.push_back("world");
  x.push_back("!");

  std::string s = "";
  std::copy(x.begin(), x.end(),
            boost::make_function_output_iterator(string_appender(s)));
  std::cout << s << std::endl;

  return 0;
}
```