Adaptors

Iterators can do more than just search through data, they can now insert values!

WE'LL COVER THE FOLLOWING

- Insert iterators
- Stream Iterators

Iterator adaptors enable the use of iterators in insert mode or with streams. They need the header `<iterator>`.

Insert iterators

With the three insert iterators `std::front_inserter, std::back_inserter` and `std::inserter` you can insert an element into a container at the beginning, at the end or an arbitrary position respectively. The memory for the elements will automatically be provided. The three map their functionality on the underlying methods of the container `cont`.

The table below gives you two pieces of information: Which methods of the containers are internally used and which iterators can be used depends on the container's type.

Name	Internally-used Method	Container
`std::front_inserter(val)`	`cont.push_front(val)`	`std::deque` `std::list`
`std::back_inserter(v`		

al)	cont.push_back(val)	std::vector std::deque std::list std::string
std::inserter(val, pos)	cont.insert(pos, val)	std::vector std::deque std::list std::string std::map std::set

The three insert iterators

You can combine the algorithms in the STL with the three insert iterators.

```
#include <iostream>
#include <iterator>
#include <queue>
#include <vector>
#include <unordered_map>
#include <algorithm>

int main(){

  std::deque<int> deq{5, 6, 7, 10, 11, 12};
  std::vector<int> vec{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

  std::copy(std::find(vec.begin(), vec.end(), 13), vec.end(), std::back_inserter(deq));

  for (auto d: deq) std::cout << d << " "; // 5 6 7 10 11 12 13 14 15
  std::cout<<std::endl;

  std::copy(std::find(vec.begin(), vec.end(), 8),
  std::find(vec.begin(), vec.end(), 10),
  std::inserter(deq, std::find(deq.begin(), deq.end(), 10)));
  std::inserter(deq, std::find(deq.begin(), deq.end(), 10)));
  for (auto d: deq) std::cout << d << " "; // 5 6 7 8 9 10 11 12 13 14 15
  std::cout<<std::endl;


  std::copy(vec.rbegin()+11, vec.rend(),
  std::front_inserter(deq));
  for (auto d: deq) std::cout << d << " "; // 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
  std::cout<<std::endl;

  return 0;
}
```

Stream Iterators

Stream iterator adaptors can use streams as a data source or data sink. C++ offers two functions to create istream iterators and two to create ostream iterators. The created istream iterators behave like input iterators, the ostream iterators like insert iterators.

Function	Description
`std::istream_iterator<T>`	Creates an end-of-stream iterator.
`std::istream_iterator<T>(istream)`	Creates an istream iterator for `istream`.
`std::ostream_iterator<T>(ostream)`	Creates an ostream iterator for `ostream`
`std::ostream_iterator<T>(ostream, delim)`	Creates an ostream iterator for `ostream` with the delimiter `delim`.

The four stream iterators

Thanks to the stream iterator adapter you can directly read from or write to a stream.

The following interactive program fragment reads in an endless loop natural numbers from `std::cin` and pushes them onto the vector `myIntVec`. If the input is not a natural number, an error in the input stream will occur. All

numbers in `myIntVec` are copied to `std::cout`, separated by `:`. Now you can see the result on the console.

```cpp
#include <iostream>
#include <iterator>
#include <queue>
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <iterator>

int main(){
  std::vector<int> myIntVec;
  std::istream_iterator<int> myIntStreamReader(std::cin);
  std::istream_iterator<int> myEndIterator;

  // Possible input
  // 1
  // 2
  // 3
  // 4
  // z
  while(myIntStreamReader != myEndIterator){
    myIntVec.push_back(*myIntStreamReader);
    ++myIntStreamReader;
  }

  std::copy(myIntVec.begin(), myIntVec.end(), std::ostream_iterator<int>(std::cout, ":"));
          // 1:2:3:4:

  return 0;
}
```