Search Elements

This lesson will cover the different ways of searching for values using 'find'.

We can search for elements in three different ways:

Returns an iterator to the first element that matches the search criteria.

```
InpIt find(InpIt first, InpI last, const T& val)
InpIt find(ExePol pol, FwdIt first, FwdIt last, const T& val)

InpIt find_if(InpIt first, InpIt last, UnPred pred)
InpIt find_if(ExePol pol, FwdIt first, FwdIt last, UnPred pred)

InpIt find_if_not(InpIt first, InpIt last, UnPred pre)
InpIt find_if_not(ExePol pol, FwdIt first, FwdIt last, UnPred pre)
```

Returns an iterator to the first element in the range:

```
FwdIt1 find_first_of(InpIt1 first1, InpIt1 last1, FwdIt2 first2, FwdIt2 last2)
FwdIt1 find_first_of(ExePol pol, FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2)

FwdIt1 find_first_of(InpIt1 first1, InpIt1 last1, FwdIt2 first2, FwdIt2 last2, BiPre pre)
FwdIt1 find_first_of(ExePol pol, FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2, Bi
```

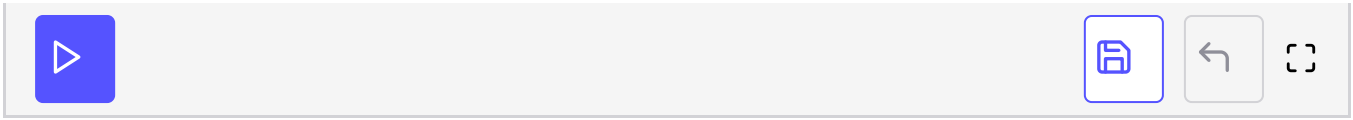
Returns identical, adjacent elements in a range:

```
FwdIt adjacent_find(FwdIt first, FwdIt last)
FwdIt adjacent_find(ExePol pol, FwdIt first, FwdIt last)

FwdIt adjacent_find(FwdIt first, FwdI last, BiPre pre)
FwdIt adjacent_find(ExePol pol, FwdIt first, FwdI last, BiPre pre)
```

The algorithms require input or forward iterators as arguments and return an iterator on the element when successfully found. If the search is not successful, they return an end iterator.

```
#include <iostream>
#include <algorithm>
```

```
#include <set>
#include <list>
using namespace std;


bool isVowel(char c){
  string myVowels{"aeiouäöü"};
  set<char> vowels(myVowels.begin(), myVowels.end());
  return (vowels.find(c) != vowels.end());
}

int main(){
  list<char> myCha{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
  int cha[]= {'A', 'B', 'C'};

  cout << *find(myCha.begin(), myCha.end(), 'g') << endl;             // g
  cout << *find_if(myCha.begin(), myCha.end(), isVowel) << endl;      // a
  cout << *find_if_not(myCha.begin(), myCha.end(), isVowel) << endl;  // b

  auto iter= find_first_of(myCha.begin(), myCha.end(), cha, cha + 3);
  if (iter == myCha.end()) cout << "None of A, B or C." << endl;      // None of A, B or C.

  auto iter2= find_first_of(myCha.begin(), myCha.end(), cha, cha+3,
                            [](char a, char b){ return toupper(a) == toupper(b); });
  if (iter2 != myCha.end()) cout << *iter2 << endl;;                   // a
  auto iter3= adjacent_find(myCha.begin(), myCha.end());
  if (iter3 == myCha.end()) cout << "No same adjacent chars." << endl;
  // No same adjacent chars.

  auto iter4= adjacent_find(myCha.begin(), myCha.end(),
                            [](char a, char b){ return isVowel(a) == isVowel(b); });
  if (iter4 != myCha.end()) cout << *iter4;                    // b
}
```



std::find, std::find_if, std::find_if_not, std::find_of, and std::adjacent_fint

---

In the next lesson, we'll discuss how to count the number of elements in a given range.