Manipulating Sets

Examples

- From manipulating arrays of integers
- To manipulating CdfTrackView, in a B physics example
- Using the same manipulations
- Introducing the STL, predicates and comparisons

Reference Book:

- Generic Programming and the STL, Matthew H Austern

Code Examples:

- In “Examples” Package : Examples/TUTORIAL01

Arrays

```
#include <iostream>

void print (int *begin, int *end) {
    int *current=begin;
    while (current!=end) {std::cout << *current << std::endl;
                          current++;}
}

int main(int, char **){
        int array[100];
        for (int i=0; i<100; i++) array[i]=i;
        //Print out the 40th to 50th elements
        print(array+40,array+50);
}
```

```
40
41
42
43
44
45
46
47
48
49
```

STL and Arrays

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

void print (int *begin, int *end) {
    int *current=begin;
    while (current!=end) std::cout << *(current++) << std::endl;
}

int main(int, char **){
        int array[100];
        for (int i=0; i<100; i++) array[i]=i;
        //Print out the 42nd to 52nd elements
        print(array+42,array+52);
        //Reverse the 45th to 50th elements
        std::reverse(array+45,array+50);
        //Print out the 42nd to 52nd elements
        print(array+42,array+52);

}
```

42
43
44
45
46
47
48
49
50
51

42
43
44
49
48
47
46
45
50
51

STL and Arrays

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

int main(int, char **){
        int array[100];
        for (int i=0; i<100; i++) array[i]=i;
        //Print out the first 5 elements of array
        print(array,array+5);

        //Randomly shuffle the whole array
        std::random_shuffle(array+0,array+100);

        //Print out the first 10 from the shuffled array
        print(array,array+10);

        //Find and print the maximum element
        int *maxInteger = std::max_element(array,array+100);
        std::cout << *maxInteger << std::endl;
}
```

0
1
2
3
4

98
3
22
43
59
60
17
88
43
12

99

Predicates and Comparisons

Predicates

A predicate makes a “yes” or “no” decision on a single object

e.g.: Does this track have pt > 20 GeV : returns true or false

```
class IsHighPt {
public:
        //Truth operator
        bool operator () ( const CdfTrack & track ) const {
                if ( track.pt() > 20 ) { return true; }
                else { return false; }
        }
};

IsHighPt ptCut;    // Make an object
if (ptCut(track))   // Then, use it like a subroutine.
```

A “Predicate” is an object which has a function call operator and it therefore looks like a subroutine

Comparisons

A comparison compares 2 objects to each other, used in std::sort

e.g.: Does track a have magnitude < track b : returns true if condition is satisfied

```
class IncrMagnitude {
public:
    bool operator () (const Hep3vector &a, const Hep3Vector &b ) const {
        return a.mag()<b.mag();
    }
};
```

Finding Prime Numbers

```
#include <iostream>
#include <algorithm>
#include  <math.h>

int main(int,  char **){
        int array[100];
        for (int i=0; i<100; i++) array[i]=i;
        //Make a predicate object that returns whether number is prime
        IsPrime isPrime;
        //How many prime numbers are in this range?
        int nPrime = std::count_if(array,array+100,isPrime);
        std::cout << "There are " << nPrime << "prime numbers " << std::endl;
        //The array is unchanged, print the 4th to 9th elements
        print(array+4,array+9);
        //Make a partition
        int *endOfPrimes = std:: partition(array,array+100,isPrime);
        std::cout << "Here are the prime numbers: " << std::endl;
        print(array, endOfPrimes);
        std::cout << "Here are the nonprime numbers: " << std::endl;
        print(endOfprimes,array+100);
}
```

There are 27 prime numbers

4
5
6
7
8

Here are the prime numbers

11
3
23
83
..
1

Here are the nonprime numbers

78
77
9
27
..
69
32

Vector Containers

```
#include <iostream>
#include <algorithm>
#include <math.h>
#include <vector>
#include "CLHEP/Vector/ThreeVector.h"

void print (std::vector<Hep3Vector>::iterator begin,
                std::vector<Hep3Vector>::iterator end )   {
  std::vector<Hep3Vector>::iterator current = begin;
  while (current!=end) {std::cout << *current <<std::endl;
                          current++;}
}

int main(int, char **){
        //Make a vector container called varray
        std::vector<Hep3Vector> varray;
        //Fill it with some vectors
        varray.push_back(Hep3Vector(0,0,0));
        varray.push_back(Hep3Vector(1,0,0));
        varray.push_back(Hep3Vector(0,1,0));
        varray.push_back(Hep3Vector(1,1,1));

        std::cout << "Size of  vector is" << varray.size() << std::endl;
        std::cout << "And its elements are"<< std::endl;
        print(varray.begin(),varray.end());
}
```

Vectors are containers, as are arrays

Vectors can grow and shrink and keep track of their size

Iterators behave as pointers, i.e. ++ and *

Size of vector is
4

And its elements are

0,0,0
1,0,0
0,1,0
1,1,1

STL and Vector Containers

```
#include <iostream>
#include <algorithm>
#include <math.h>
#include <vector>
#include <iterator.h>
#include "CLHEP/Vector/ThreeVector.h"

int main(int, char **){
        //Make a vector container and fill it with some vectors
        std::vector<Hep3Vector> varray;
        varray.push_back(Hep3Vector(0,0,0));
        varray.push_back(Hep3Vector(1,0,0));
        varray.push_back(Hep3Vector(0,1,0));
        varray.push_back(Hep3Vector(1,1,1));

        std::cout << "The vector elements are"<< std::endl;
        //Make an output iterator
        std::ostream_iterator<Hep3Vector> terminal(std::cout,"\n");
        std::copy(varray.begin(),varray.end(), terminal);

        //Randomly shuffle – size of vector is unchanged
        std::random_shuffle(varray.begin(),varray.end());
        std::copy(varray.begin(),varray.end(), terminal);
}
```

The vector elements are

0,0,0
1,0,0
0,1,0
1,1,1

0,1,0
1,1,1
0,0,0
1,0,0

Sort by Magnitude

```
#include <iostream>
#include <algorithm>
#include <math.h>
#include <vector>
#include <iterator.h>
#include "CLHEP/Vector/ThreeVector.h"

//IncrMagnitude is a Comparison
class IncrMagnitude {
public:
      bool operator () (const Hep3vector &a, const Hep3Vector &b ) const {
                return a.mag()<b.mag();
      }
};
int main(int, char **){
      //Make a container called varray and fill it with vectors
      std::vector<Hep3Vector> varray;
      varray.push_back(Hep3Vector(0,0,0));
      varray.push_back(Hep3Vector(1,0,0));
      varray.push_back(Hep3Vector(0,1,0));
      varray.push_back(Hep3Vector(1,1,1));

      //Shuffle the vectors in the container and print them to the screen
      std:: random_shuffle(varray.begin(),varray.end());
      std::ostream_iterator<Hep3Vector> destination(std::cout,"\n");
      std::copy(varray.begin(),varray.end(), destination);

      //Sort the vector by magnitude and print them to the screen
      IncrMagnitude incrMagnitude;
      std::sort(varray.begin(),varray.end(), incrMagnitude);
      std::copy(varray.begin(),varray.end(), destination);

      std::cout <<  " Vector of greatest magnitude "<< std::endl;
      std::vector<Hep3Vector> ::iterator mxVector =
                std::max_element(varray.begin(),varray.end(), incrMagnitude);
      std::cout <<  *mxVector << std::endl;
}
```

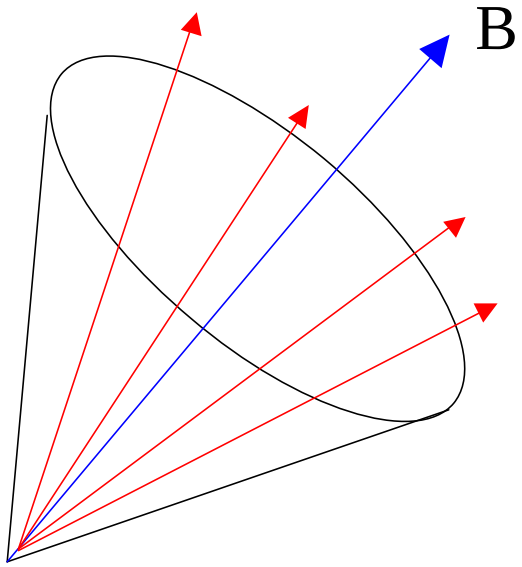
0,1,0
1,1,1
0,0,0
1,0,0

0,0,0
0,1,0
1,0,0
1,1,1

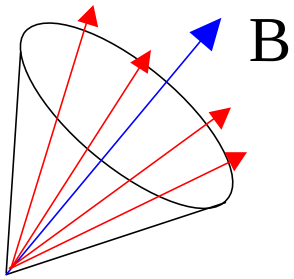
Vector of greatest Magnitude
1,1,1

B physics example

Find the track within a cone
around the B direction,
with the smallest transverse
momentum (wrt a B track
direction)
and return its charge



B physics example




```
#include “TrackingObjects/Tracks/CdfTrack.hh”
#include “TrackingObjects/Storable/CdfTrackView.hh”
#include “TrackingObjects/Tracks/track_cut.hh”
#include “TrackingObjects/Tracks/track_sort.hh”
#include <algorithm>

int singleTrackTag (CdfTrackView &tv, Hep3Vector & direction) {
        CdfTrackView::iterator begin = tv.contents().begin(),  end = tv.contents().end();
        //Select default tracks in a cone of 0.7 around the B direction
        const CdfTrackCut & cut = track_cut::SelectDefault() &&
                                  track_cut::InConeAround (direction,0.7);
        //Restrict to the subset satifying the cuts
        end = std::partition(begin,end,CdfTrackCutPredicate(&cut));
        //Make a comparison object that orders by Pt Rel wrt b direction
        track_sort::IncrPtRel incrPtRel(direction);
        //Find the minimum Pt track in this subset
        CdfTrackView::iterator mnPtTrack = std::min_element(begin,end,incrPtRel);
        //return: 0 for no track found, 1 for positive track found, -1 for negative track:
        if(mnPtTrack==end){ return 0;}
        if((**mnPtTrack).charge()> 0){ return 1;}
        else {return –1;}
}
```

Examples/Tutorial01_07

Summary

- STL library allows you to manipulate sets of objects
- And provides containers (vectors, lists, maps)
- Same routines can work on all kinds of collections (including arrays) of all types of objects
- CdfTrackSet, CdfMuonView, CdfMuonSet, CdfMuonView all have collections
  (get them with the contents() method)
- Standard useful predicates and comparisons are provided

Reference Book:

- Generic Programming and the STL, Matthew H Austern

Code Examples:

- In “Examples” Package : Examples/TUTORIAL01