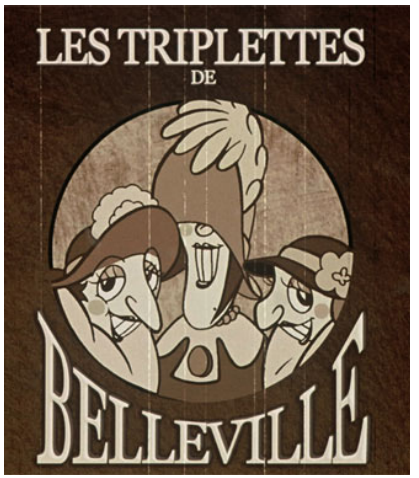
Annotated slides

CS319: Scientific Computing (with C++)

Niall Madden (Niall.Madden@NUIGalway.ie)

Week 10: Sparse Matrices, and The STL

9am, 20 April, and 4pm, 21 April, 2021



CS319 – Week 10: Sparse Matrices, and The STL

Start of ...

PART 0: Assessment for CS319

Part 0: Assessment for CS319

I have tweaked the original assessment plan for CS319:

1. 50% based on lab assignments:
   - 10% for each of Lab 2 and Lab 3;
   - 15% for each of Lab 4+5 and Lab 7.
2. 50% based on your project work.

So, no in-class test

IS THAT OK?

Projects: Your project will consist of

1. An idea you select/propose, in (1:1) discussion with me;
2. A demonstration of various programming techniques in C++, including, minimally, coding your own class(es);
3. You will submit an initial, 250 word project proposal by Friday. (Word count is indicative: I won't check).
4. You will submit a 3 page project report along with your code. (DEADLINE???)
5. The report should be organised as follows.
   - 5.1 A summary of what you have done. This should be 300-500 words, and written in a non-technical style: anyone should be able to understand it. The emphasis should be on the problem solved, and why it is interesting, and not on the code.
   - 5.2 A technical discussion of the code.
   - 5.3 A note on what you have learned/discovered.
   - 5.4 An example of typical input and output.
   - 5.5 Details any limitations of your code that you have noted, and a comparison with your original project proposal.
   - 5.6 The highlights of your code: What took the most effort? What are you most proud of?

   The report must be in PDF and submitted here (via TurnItIn)

in lab or by email.

and working with files

i.e., half page.

roughly.

might include a class definitions for illustration.

not group projects.

The code for your project is a chance for your to show your skills in C++ programming, and in scientific computing.
Each project will be independently negotiated with Niall.
Each will involve, at the very least, all of the following

1. An external data source, so that you can show your expertise in read from and/or writing to files.
2. A `class` (or set of classes) that you design yourself
3. An algorithm that preforms some type of useful calculation

The projects will be graded, and will contribute 50% to the over-all grade for CS319. The break-down of marks is as follows.

(a) Negotiating the project topic with Niall completed by 17:00, Friday 23 April [**5 Marks**]
(b) The project proposal completed by 17:00, Monday 26 April [**5 Marks**]
(c) C++ code [**25 Marks**]
(d) The Project report [**15 Marks**]

Deadline

Some ideas for projects

Here is a random selection of topics. Their purpose is to stimulate discussion.

1. A class for storing *symmetric* matrices, and an implementation of the *Conjugate Gradients* algorithm for solving linear systems.
2. *LU*-decomposition for linear systems (see MA385) or Cholesky factorization
3. Problems in Cryptography I: Shift encryption, and decyphering it with a frequency analysis.
4. Other cryptography methods?
5. Algorithms on Graphs: minimum spanning trees, computing the chromatic number, searching (depth-first V breath-first), shortest path on weighted graphs, ... (see MA284),
6. Prime factorization with arbitrary precision integers.
7. Image enhancement: edge detection, softening/blurring.
8. Data analysis - clustering methods
9. Triangulation of points in $\mathbb{R}^2$ (Delaunay).
10. ??? — statistical ~ fitting,

Anything in Numerical Analysis.. ~ Quadrature.

CS319 – Week 10: Sparse Matrices, and The STL

END OF PART 0

CS319 – Week 10: Sparse Matrices, and The STL

Start of ...

PART 1: Sparse Matrices

Part 1: Sparse Matrices

In Week 9 we looked at a social network analysis algorithm, PageRank. For the problem to be interesting, our network should have thousands, perhaps millions, of nodes/vertices.

Compared to the over-all number of entries in the matrix, the **number of non-zeros** (**NNZs**) is relatively small. So it does not make sense to store them all. Instead, one uses one of the following formats:

- **Triplet** (which we'll look at presently),
- **Compressed Row Storage** (CRS) (after triplet)
- **Compressed Column Storage** (CCS) ← after triplet

And the following formats for very specialised matrices, which we won't study in CS319:

- **Block Compressed Row/Column Storage**
- **Compressed Diagonal Storage**
- **Skyline**

Although the representation and manipulation of sparse matrices is an major topic in Scientific Computing, there isn't a universally agreed definition of an (abstract) *sparse matrix*.

This is because, when coding, we should ask the question: **"When is it worth the effort to store a matrix in a sparse format, rather than in standard (dense) format?"**

The answer is often context-dependent. But roughly, use a sparse format when

- The memory required by the sparse format is less then the "dense" (or "full") one;
- The expense of updating the sparse format is not excessive;
- Computing a `MatVec` is faster for a sparse matrix.

"not sparse" = "dense" = "full".

Triplet

The basic idea for triplet form is: to store a **sparse** matrix with `NNZ` non-zeros we ...

- define integer arrays `I[NNZ]` and `J[NNZ]`,
- a double array `X[NNZ]`.
- Then entry a_{ij} is stored as `I[k]=i`, `J[k]=j`, `X[k]=`a_{ij}, for some k.

Example: write down the triplet form of the following matrix:

$$\begin{pmatrix} 1 & 0 & 11 & 0 \\ 1 & 0 & 2 & 0 \\ 9 & 19 & 0 & 29 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

i: 0, 1, 2, 3 — j: 0, 1, 2, 3

NNZ = 8

I = [0, 0, 1, 1, 2, 2, 2, 3]

J = [0, 2, 0, 2, 0, 1, 3, 3]

X = [1, 11, 1, 2, 9, 19, 29, 5]

Note: Matrix had 16 entries
Triplet Version: 24 entries

} so not a good example of why we store in triplet.

Our next goal is implement a triplet matrix as a `class`. The main tasks are:

- Decide what private data elements are needed.
- Decide what public methods are needed.
- Implement a matrix-vector multiplication algorithm.

} Code in next section

Discussion...

Private	Public
Int arrays I & J, Double array X, int N — number of rows & cols in matrix int NNZ - number of non-zeros ... ?	Constructors + Destructors All methods for Matrix: getij, setij, etc Overloaded: =, * where: locate an entry. ie, find K s.t. I[K] = i, J[K] = j.

CS319 – Week 10: Sparse Matrices, and The STL

END OF PART 1

CS319 – Week 10: Sparse Matrices, and The STL

Start of ...

PART 2: Coding `triplet`

Part 2: Coding triplet

Triplet.h

See git repo on bitbucket.

```
1  // Triplet.h:  For 2021-CS319 Week 10
   // Author: Niall Madden
3
   #ifndef _TRIPLET_H_INCLUDED
5  #define _TRIPLET_H_INCLUDED

7  #include "Vector09.h"
   #include "Matrix10.h"
```

like Matrix09.h but with zero function.

Part 2: Coding triplet

used to convert a Matrix class to Triplet.

Triplet.h

```
10  class Triplet {
      friend Triplet full2Triplet(Matrix &F, unsigned int NNZ_MAX);
12  private:
      unsigned int *I, *J;
14    double *X;
      unsigned int N;
16    unsigned int NNZ;
      unsigned int NNZ_MAX;
18
    public:
20    Triplet (unsigned int N, unsigned int nnz_max); // Constructor
      Triplet (const Triplet &t); // Copy constructor
22    ~Triplet(void);

24    Triplet &operator=(const Triplet &B); // overload assignment operator
```

memory allocated in constructor.

size of matrix.

number of (current) non-zeros

used in OMA.

// destructor.

Part 2: Coding triplet

Triplet.h

```
26    unsigned int size(void) {return (N);};
      int where(unsigned int i, unsigned int j); // negative return on error
28    unsigned int nnz(void) {return (NNZ);};
      unsigned int nnz_max(void) {return (NNZ_MAX);};
30
      double getij (unsigned int i, unsigned int j);
32    void setij (unsigned int i, unsigned int j, double x);

34    unsigned int getI (unsigned int k) { return I[k];};
      unsigned int getJ (unsigned int k) { return J[k];};
36    double getX (unsigned int k) { return X[k];};

38    Vector operator*(Vector u);
      void print(void);
40  };
    #endif
```

Part 2: Coding triplet

Triplet.cpp

```
1  // Triplet.cpp  for 2021-CS319 Week 10
   //   What: Methods for the Triplet class
3  // Author: Niall Madden
   #include <iostream>
5  #include <iomanip>
   #include "Vector09.h"
7  #include "Matrix10.h"
   #include "Triplet.h"
```

Triplet.cpp (Constructor)

```
10  // Standard  constructor.
    Triplet::Triplet (unsigned int N, unsigned int nnz_max) {
12    this->N = N;
      this->NNZ_MAX = nnz_max;
14    this->NNZ = 0;

16    X = new double [nnz_max];
      I = new unsigned int [nnz_max];
18    J = new unsigned int [nnz_max];
      for (unsigned int k=0; k<nnz_max; k++)  {
20      I[k]=-1;
        J[k]=-1;
22      X[k]=(double)NULL;
      }
24  }
```

Part 2: Coding triplet

When using a Triplet object to represent a matrix, T, we often need to find where in the array X, the value of $t_{i,j}$ is stored. That is done by the following function.

Triplet.cpp (where)

```
56 int Triplet::where(unsigned int i, unsigned int j)
   {
     unsigned int k=0;
58   do {
       if ( (I[k]==i) && (J[k]==j) )
60       return(k);
       k++;
62   } while (k<NNZ);
     return(-1);
64 }
```

if not found.

Triplet.cpp (setij)

```
void Triplet::setij (unsigned int i, unsigned int j, double x)
{
  if (i>N-1)
    std::cerr << "Triplet::setij(): i Index out of bounds." << std::endl;
  else if (j>N-1)
    std::cerr << "Triplet::setij(): j Index out of bounds." << std::endl;
  else if (NNZ > NNZ_MAX-1)
    std::cerr << "Triplet::setij(): Matrix full." << std::endl;
  else
  {
    int k=where(i,j);
    if (k == -1)
    {
      I[NNZ]=i;
      J[NNZ]=j;
      X[NNZ]=x;
      NNZ++;
    }
    else
      X[k]=x;
  }
}
```

← check if that entry is set

↖ if not, set it

← overwrite old value.

Triplet.cpp (operator *)

```
Vector Triplet::operator*(Vector u)
{
  Vector v(N); // v = A*u, where A is the implicitly passed Triplet
  v.zero();
  if (N != u.size())
    std::cerr << "Error: Triplet::operator* - dimension mismatch"
              << std::endl;
  else
  {
    for (unsigned int k=0; k<NNZ; k++)
      v.seti(I[k], v.geti(I[k]) + X[k]*u.geti(J[k]));
  }
  return(v);
}
```

Read yourself

To demonstrate the use of the Triplet class, I've included a program called 01triplet_example which shows how to use the Jacobi method to solve a linear system where the matrix is stored in triplet format.

CS319 – Week 10: Sparse Matrices, and The STL

END OF PART 2

CS319 – Week 10: Sparse Matrices, and The STL

Start of ...

PART 3: ~~CSR~~ CCS

The Compressed Column Sparse Format

This is one of the many alternatives to triplet.

Part 3: CCS

If we know that the entries in our matrix are stored in order, then it is possible to store the matrix more efficiently that in Triplet format. One way of doing this is to use **CCS**: **Compressed Column Storage**, also known as *Harwell-Boeing*

The matrix is stored in 3 vectors:

- a `double` array, x of length `nnz` ("number of nonzero entries") storing the non-zero entries matrix, in column-wise order.
- an `int` array, r of length `nnz` storing **row** index of the entries. That is, $x[k]$ would be found in row $r[k]$ of the full matrix.
- an `int` array, c of length $N + 1$, where $c[k]$ stores that starting point of column k as it appears in the arrays x and r, and $c[N] = nnz$.

Example

Show how the matrix below would be stored in CCS

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 2 & -1 & 0 & -2 \\ -3 & 5 & -1 & 0 \\ 0 & -2 & 4 & 0 \\ -3 & 0 & 0 & 4 \end{pmatrix} \end{array}$$

$C = [0, 3, 6, 8, 10]$

$R = [0, 1, 3, 0, 1, 2, 1, 2, 0, 3]$

$X = [2, -3, 3, -1, 5, -2, -1, 4, -2, 4]$

The process of multiplying a matrix (in CCS) by a vector is rather simple:

```
int index=0;
for (int col=0; col<N; col++)
   for (j=c[col]; j<c[col+1]; j++)
   {
     i=r[index];
     v[i] += x[index]*b[j];
     index++;
   }
```

I don't provide code for implementing a CCS class here: that is an exercise.

CS319 – Week 10: Sparse Matrices, and The STL

END OF PART 3

CS319 – Week 10: Sparse Matrices, and The STL

Start of ...

PART 4: The Standard Template Library (STL)

Or "How not to reinventing the wheel

focus on sets, multisets & vectors.

During the semester, we've focused on designing classes that can be used to solve problems. These included classes: `Stack`, `Vector` and `Matrix`.

However, most of you worked out that, to some extent, these are already supported in C++. The motivations for reinventing them included

- our implementation is simple to use;
- we learned important aspects of C++/OOP;
- we needed to achieve specific tasks efficiently: this is particularly true of our design of sparse matrix classes.

Now we'll look at how to use the built-in implementation that comes with the C++ **Standard Template Library (STL)**.

The **STL** provides

(1) **Containers:** ways of collecting/storing items of some type (template....)

(2) **Iterators:** for accessing items in the containers (like an index)

(3) **Algorithms:** for operating on the contents of containers, such as finding a particular item, or sorting (a subset) of them.

(4) **functors:** essentially, a class which defines the operator(). We won't say more than this right now. (or, indeed, at all).

We'll now look at examples of (1)–(3), and then consider an application to our Password Frequency Problem from a few weeks ago.

It has to be noted, though: the STL is not that easy to use. In particular the error messages generated are rather verbose and unhelpful.

Eg: stack, set, vector are all containers.

Eg: i is an iterator for vector, can write v[i].

A **container** stores objects/elements. These elements can have basic data-type (e.g., `char`, `int`, `double`, ...) or can be objects (e.g., `string`, or user-defined objects).

The most important types of containers are:

`vector`: an indexed sequence (often called "*random access*", though this would be better called "*arbitrary access*". All the items are of the same type. It can be resized, and have new items added to the end. One can also add items to positions not the end, but this is slow.

`set`: a collection of unique items (of the same type), stored in order. When defined relative to a user-defined class, an overloaded `operator<` (less than) must be provided for correct operation.

`multiset`: an ordered collection, like a set, but can have repeated values.

`list`: a doubly linked list.

`stack`: a stack.

... etc...

We'll focus on `set`s, `multiset`s and `vector`s.

An `iterator` is an object used to select (or move between) elements in a container.

We can think of them as pointers, that allow us to reference particular elements.

They come in particular flavours:

- forward, reverse, and bidirectional iterators;
- random-access/indexed-access iterators;
- input and output iterators;

CS319 – Week 10: Sparse Matrices, and The STL

END OF PART 4

CS319 – Week 10: Sparse Matrices, and The STL

Start of ...

PART 5: sets and multisets in STL

Part 5: sets and multisets

To use a `set` or `multiset`, we must

```
#include <set>
```

Suppose we want to create a `multiset` to store `string`s (which just happen to be passwords...), and an iterator for it, we could define

```
1 std::multiset <std::string> multi_pwd;
  std::multiset <std::string>::iterator multi_pwd_i;
```

To add an item to the (multi)set, we could used

```
multi_pwd.insert(MyString);
```

This will add the new string to the `multiset`, automatically choosing its position so that it remains ordered. (If we use a set, it gets inserted into the correct position, providing this does not result in duplication).

Container type : multiset
Container contains : string
Container name : multi_pwd

Other important methods include

- `begin()` (returns an iterator that points to the first element)
- `end()` (returns an iterator that points to *one past* the end of the set).
- `clear()` (remove contents)
- `count()` (count number of occuences)
- `empty()` (is the set empty?)
- `erase()` (remove an element, or range of elements)
- `find()` (locate an element; return an iterator)
- `size()` (number of elements)
- `swap()` (swap contents of two sets of same type)
- `for_each()` (apply a particular function to each item in a container)

see later.

An example of using `begin` and `end` with a set and `multiset`:

01set_and_multiset.cpp

```
10  int main(void )
    {
12    std::set <int> set_int;
      std::set <int>::iterator set_int_i;
14    std::multiset <int> multi_int;
      std::multiset <int>::iterator multi_int_i;
16
      for (int i=0; i<=20; i+=3) // (0,3,6,9,12,15,18)
18    {
        set_int.insert(i);
20      multi_int.insert(i);
      }
22    for (int i=20; i>0; i-=2) // (20,18,16,...,4,2)
      {
24      set_int.insert(i);
        multi_int.insert(i);
26    }
```

Notice
① the elements are not inserted in order
② Some entries, e.g., 18, are inserted twice.

Part 5: sets and multisets

First, we will see how to iterate over the `multiset`:

01set_and_multiset.cpp

```
28  std::cout << "The multiset has " << multi_int.size() <<
      " items." << std::endl;
30  std::cout << "\t They are: ";
    for (multi_int_i = multi_int.begin();
32       multi_int_i != multi_int.end();
         multi_int_i++)
34    std::cout << std::setw(3) << *multi_int_i;
    std::cout << std::endl;
36  std::cout << "\t 6 occurs " << multi_int.count(6) <<
      " time(s)." << std::endl;
```

using iterator in for loop.

The output is

```
1  The multiset has 17 items.
     They are:   0   2   3   4   6   6   8   9  10  12  12  14  15  16  18  18  20
3    6 occurs 2 times.
```

Next we will iterate over the `set`:

02set_and_multiset.cpp

```
  std::cout << "The set has " << set_int.size() <<
40    " items." << std::endl;
  std::cout << "\t They are: ";
42  for (set_int_i = set_int.begin();
       set_int_i != set_int.end();
44     set_int_i++)
    std::cout << std::setw(3) << *set_int_i;
46  std::cout << std::endl << "\t 6 occurs " << set_int.count(6)
          << " time(s)." << std::endl;
```

The output is

```
1 The set has 14 items.
         They are:   0  2  3  4  6  8  9 10 12 14 15 16 18 20
3        6 occurs 1 time.
```

← no repeated entries

CS319 – Week 10: Sparse Matrices, and The STL

END OF PART 5

CS319 – Week 10: Sparse Matrices, and The STL

Start of ...

PART 6: vector

ie the vector container from the STL.

Part 6: vector

To use vector, we must

```
#include <vector>
```

Unlike a set, we can access a vector by index. Moreover, by default it is not sorted, though there are algorithms to sort its contents.

Since it is unordered, a new item usually gets added to the end, using push_back

This can be removed, using pop_back

Other important methods include

- at
- operator[]
- back (not the same as end)

Part 6: vector

03STL_vector.cpp

```
10 #include <vector>        // vector
   #include <algorithm>     // sort
12 void print_int (int i) { std::cout << std::setw(3) << i; }
   int main(void )
14 {
     std::vector <int> vec_int;
16   std::vector <int>::iterator vec_int_i;
     std::cout << "Vector has " << vec_int.size() <<
18     " elements." << std::endl ;

20   for (int i=3; i>=0; i--)
       vec_int.push_back(i*3); // (9,6,3,0)

     std::cout << "Vector has " << vec_int.size() << " elements: ";
24   for (unsigned int i=0; i<vec_int.size(); i++)
       std::cout << std::setw(3) << vec_int[i];
```

Output (so far):

```
1 Vector has 0 elements.
  Vector has 4 elements:   9   6   3   0
```

This snippet demonstrates the use of

- the find and insert methods;
- the for_each iterate through an entire container.

find where 3 occurs in vector: returns a iterator for that location

03STL_vector.cpp

```
28  vec_int_i = find (vec_int.begin(),vec_int.end(),3);
    vec_int.insert(vec_int_i,10);

30  std::cout << std::endl;
    std::cout << "Vector has " << vec_int.size() << " elements: ";
32  for_each (vec_int.begin(), vec_int.end(), print_int);
```

insert 10 just before 3.

name of function called at each iteration

Output (continued):

```
1   Vector has 0 elements.
    Vector has 4 elements:  9  6  3  0
3   Vector has 5 elements:  9  6 10  3  0
```

Finally, we show how to sort the items in the list:

03STL_vector.h

```
    std::cout << "Sorting the vector..." << std::endl;
36  sort(vec_int.begin(), vec_int.end());
    std::cout << "Now vector is: ";
38  for_each (vec_int.begin(), vec_int.end(), print_int);
```

Output (all):

```
1 Vector has 0 elements.
  Vector has 4 elements:   9  6  3  0
3 Vector has 5 elements:   9  6 10  3  0
  Sorting the vector...
5 Now vector is:  0  3  6  9 10
```

Other important methods include

- `begin()` (returns an iterator that points to the first element)
- `end()` (returns an iterator that points to *one past* the end of the set).
- `clear()` (remove contents)
- `count()` (count number of occuences)
- `empty()` (is the set empty?)
- `erase()` (remove an element, or range of elements)
- `find()` (locate an element; return an iterator)
- `size()` (number of elements)
- `swap()` (swap contents of two sets of same type)
- `for_each()` (apply a particular function to each item in a container)

The **ranged-based** *for* loop is a recent addition to C++, so it might not work with old compilers. With g++, you may need to enable the `c++11` option.

-std = c++11

In the code above, the line

```
1   for_each(vec_int.begin(), vec_int.end(), print_int);
```

could be replaced with

```
1   for (int i : vec_int)
        print_int(i);
```

take i to be each entry of vec_int in turn.

CS319 – Week 10: Sparse Matrices, and The STL

END OF PART 6

Finish here

CS319 – Week 10: Sparse Matrices, and The STL

Start of ...

PART 7: Algorithm

Part 7: Algorithm

To use `algorithm`, we must

```
#include <algorithm>
```

Useful functions that this provides include

- `for_each`
- `sort` and `partial_sort`
- `search`
- `copy` and `fill`
- `merge`
- `set_union`, `set_difference`
- etc.

(See lecture notes for details)

```
#include <set>       // multiset
#include <vector>    // vector
#include <algorithm> // sort
```

```
1  class pwd {
   private:
3    std::string word;
     int freq;
5  public:
     pwd(std::string s, int f) {word=s; freq=f;};
7    std::string getword(void) const {return(word);};
     int getfreq(void) const {return(freq);};
9  };

11 bool compare (pwd p, pwd q)
   {
13   return (p.getfreq() > q.getfreq());
   }
15
   int FileLength(std::ifstream &InFile, int &LongestWord);
```

```
int main(void )
{
  std::ifstream InFile;
  std::string InFileName="UserAccount-1e5.txt";
  std::multiset <std::string> multi_pwd;
  std::multiset <std::string>::iterator multi_pwd_i;
  std::vector <pwd> vector_pwd;

  InFile.open(InFileName.c_str());
  if (InFile.fail() )
  {
    std::cerr << "Error: Cannot open " << InFileName <<
      " for reading." << std::endl;
    exit(1);
  }

  // Need to know the number of lines, and the length of the longest one
  int LineCount=0, LongestLine;
  LineCount =  FileLength(InFile, LongestLine);
  std::cout  << InFileName << " has " << LineCount << " lines.\n";
  std::cout  << "\tThe longest has " << LongestLine << " characters.\n";
```

```
// Read the lines
char *c_string_word;
c_string_word = new char [LongestLine+1];
for (int i=0; i<LineCount; i++)
{
  InFile.getline(c_string_word, LongestLine+1);
  multi_pwd.insert(c_string_word);
}
```

```
// Copy the passwords to the pwd vector
multi_pwd_i = multi_pwd.begin();
vector_pwd.push_back(pwd(*multi_pwd_i,
      multi_pwd.count(*multi_pwd_i)));
multi_pwd_i++;
while (multi_pwd_i != multi_pwd.end())
{
  if ((vector_pwd.back()).getword() != *multi_pwd_i)
    vector_pwd.push_back(pwd(*multi_pwd_i,
       multi_pwd.count(*multi_pwd_i)));
  multi_pwd_i++;
}
```

```
  std::sort (vector_pwd.begin(), vector_pwd.end(), compare);

  std::cout << "Top 10 passwords are: " << std::endl;
  for (unsigned int i=0; i<10; i++)
    std::cout <<  std::setw(12) << (vector_pwd[i]).getword() <<
      std::setw(6) << (vector_pwd[i]).getfreq() << std::endl;

  InFile.close();

  return(0);
}
```