```
// Example features of modern C++
//

#include <iostream>
#include <vector>
#include <list>

#define endl "\n"

using namespace std;

// typedef allows you to define your own type.
// What other ways allow us to define a data type??
typedef vector<int> vecInt;
typedef vector<vecInt> twoDvecInt;


/**
 * @brief builds a randomly populated 2D vectpr
 *        of integers.
 *
 * @param int rows
 * @param int cols
 * @return twoDvecInt
 */
twoDvecInt buildArray(int rows, int cols)
{
    twoDvecInt A;

    A.resize(rows);

    for (int i = 0; i < A.size(); i++)
    {
        A[i].resize(cols);
        for (int j = 0; j < A[i].size(); j++)
            A[i][j] = rand() % 100;
    }

    return A;
}


twoDvecInt buildStaggered(){
  twoDvecInt A;

  A.resize(rand()%10);

  for (int i = 0; i < A.size(); i++)
  {
      A[i].resize(rand()%10);
      for (int j = 0; j < A[i].size(); j++)
          A[i][j] = 0;
```

```
    }

    return A;

}



void dumpVector(twoDvecInt A)
{
    for (int i = 0; i < A.size(); i++)
    {
        for (int j = 0; j < A[i].size(); j++)
        {
            if (A[i][j] < 10)
            {
                cout << "0";
            }
            cout << A[i][j] << " ";
        }
        cout << endl;
    }
}

void dumpVector2(twoDvecInt A){
    for (auto row : A)
    {
      for (auto col : row)
      {
          if (col < 10)
          {
              cout << "0";
          }
          cout << col << " ";
      }
      cout<<endl;
    }

}

void dumpVector3(twoDvecInt A){
    for (auto i = A.rbegin(); i != A.rend(); i++)
    {
      for (auto j = i->rbegin(); j != i->rend(); j++)
      {
          if (*j < 10)
          {
              cout << "0";
          }
          cout << *j << " ";
      }
      cout<<endl;
    }
```

```
}

int main(int argc, char **argv)
{
    srand(345);
    // Initializer_list<> type. Defined in STL; Make a habit of using
    // {} to init a vector or other containers.

    initializer_list<int> ilist = {1, 2, 3, 4, 5};

    // What is generic programming?
    vector<int> B = {2, 4, 6, 8}; // one example of init
    vector<int> A{ilist};         // initialized using
initializer_list<int> constructor
    vector<int> C = ilist;        // works also by calling the same
constructor
    vector<int> D{B};             // Note that I am not using () in the
copy constructor

    list<int> Clst = {5, 7, 9, 10}; // also initializes a stl linked list
as well.

    // this is a doubly linked list.  There is also a forward linked list
in STL

    // Iterators.  STL has a type called iterator that is used to traverse
the elements
    // of a stl container.
    list<int>::iterator itor; // declares an iterator called itor that
points to an element
                              // in list<int>.
    // Iterators can be thought of as a class that encapsulates pointers
to the item.
    // STL types have two methods called begin() and end() among others.
    //      type.begin(): returns the itor that points to the first item
in the container
    //      type.end(): points to one after the last item in the
container.
    //      itor++: overloaded operator that increments the itor to point
to the next item.

    // Here is how you use an iterator to traverse a list container
    for (list<int>::iterator itor = Clst.begin(); itor != Clst.end();
itor++)
    {
        cout << *itor << " "; // Note they also have dereferencing via *
    }
    cout << endl;

    for (auto itor = Clst.begin(); itor != Clst.end(); itor++)
    {
        cout << *itor << " "; // Note they also have dereferencing via *
    }
    cout << endl;
```

```
    // doing the same on a vector
    for (vector<int>::iterator itor = A.begin(); itor != A.end(); itor++)
    {
        cout << *itor << " "; // Note they also have dereferencing via *
    }
    cout << endl;

    // Note that they both have the SAME interface even though one is a
list and the other
    // is a vector.  Works the way on most of the stl containers.

    // C++11 has a new mechanism called auto.  This is very nice.  It will
automatically
    // determine (ie deduce) the type that is needed via context

    //For example:

    auto x = 5.0;               // this determines the type from the 5
which is integer.
    auto rx = &x;               // rx is a reference to x
    cout<<*rx<<endl;
    auto d = {1, 2};            // d is an initializer_list<int>
    auto e = {1.1, 2.2, 4.65};  // e is an initializer_list<double>
    auto zp = make_pair(2.3, 5); // zp is a pair<double,int>

    // Or how about this. `itor` becomes whatever type is in `A`
    // note the !=. We stop looping when we go off the end of the
container
    for (auto itor = A.begin(); itor != A.end(); itor++)
    {
        cout << *itor << " "; // Note they also have dereferencing via *
    }
    cout << endl;
    // Note that the above only works for stl containers.  If you want it
to work
    // for say your own class you would need to add iterators and begin()
and end()
    // to your class.  We might attemp this late in the semester.

    // Range loops . Very Pythonish
    for (auto x : A)
    { // for every x in A. the type of x is the type of whats in A
        cout << x << " ";
    }
    cout << endl;
    // and for a linked list
    for (auto x : Clst)
    { // for every x in A. the type of x is the type of whats in A
        cout << x << " ";
    }
    cout << endl;

    // Suppose that you want to modify the values of a vector , say A
```

```
    // Here we need x to be a reference to the item in A.
    for (auto &x : A)
    { // for every x in A, double it
        x *= 2;
    }
    for (auto x : A)
    { // Now print A
        cout << x << " ";
    }
    cout << endl;

    twoDvecInt vv = buildArray(rand()%20,rand()%20);

    dumpVector(vv);
    cout<<endl;
    dumpVector2(vv);
    cout<<endl;
    dumpVector3(vv);

    twoDvecInt v2 = buildStaggered();
    cout<<endl;
    dumpVector(v2);

    return 0;
}
```