CSC215

Math and Computer Science

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

A Legacy of Excellence

STL Permutations

- Include the library
  - #include <algorithm>
- Must have a pointers or bidirectional iterators
- You insert your items into a container class
  - That is the first permutation
- Gives you the next lexicographically greater permutation if it exists
- Gives you the next lexicographically lesser permutation if it exists

SOUTH DAKOTA
SCHOOL OF MINES & TECHNOLOGY

Next Permutation

- Requires 2 parameters
  - Starting iterator / pointer
  - Ending iterator / pointer
  - Swap must work with your data
- Returns a boolean
  - true for it found another permutation
  - False it did not find another permutation
- Does not handle duplicates or items out of lexicographical order
  - Returns false when if it can not generate another permutation

Integer Arrays

```
Note order of items, low to high
int array[10] = {0,1,2,3,4};

do
{
    for( i=0; i<5; i++)
        cout << array[i] << " ";
    cout << endl;
}while( next_permutation(array, array+5) );
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Integer Vector

```
vector<int> = {0,1,2,3,4};

do
{
    for( i=0; i<5; i++)
        cout << v1[i] << " ";
    cout << endl;
}while( next_permutation( v1.begin(), v1.end() ) );
```

String

```
string s1 = "abcde";

do
{
    cout << s1 << endl;
}while( next_permutation(s1.begin(), s1.end()) );
```

SOUTH DAKOTA
M
SCHOOL OF MINES
& TECHNOLOGY

Integer List

```
list<int> l1 = {0,1,2,3,4};
list<int>::iterator it;

do
{
    for( it=l1.begin(); it != l1.end(); it++)
        cout << *it << " ";
    cout << endl;
}while( next_permutation( l1.begin(), l1.end() ) );
```

Array of Strings

```
// Note order of items, low to high
string vals[10] = {"black", "blue", "green",
                   "orange", "red"};
do
{
    for( i=0; i<5; i++)
        cout << vals[i] << " ";
    cout << endl;
}while( next_permutation( vals, vals+5 ) );
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Problems (must be next greater)

```
string vals[10] = {"red", "blue", "black",
                    "orange", "green"};
do
{
    for( i=0; i<5; i++)
        cout << vals[i] << " ";
    cout << endl;
}while( next_permutation( vals, vals+5 ) );
```

Produces

red blue black orange green
red blue green black orange
red blue green orange black
red blue orange black green
red blue orange green black
red green black blue orange
red green black orange blue
red green blue black orange
red green blue orange black
red green orange black blue
red green orange blue black
red orange black blue green
red orange black green blue
red orange blue black green
red orange blue green black
red orange green black blue
red orange green blue black

Previous Permutation

- Requires 2 parameters
  - Starting iterator / pointer
  - Ending iterator / pointer
  - Swap must work with your data
- Returns a Boolean result
  - True for it found another permutation
  - False it did not find another permutation
- Does not handle duplicates or items out of lexicographical order
  - Returns false when if it can not generate another permutation

Array of integers

```
Note order high to low
int array[10] = {4,3,2,1,0};

do
{
    for( i=0; i<5; i++)
        cout << array[i] << " ";
    cout << endl;
}while( prev_permutation(array, array+5) );
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Integer Vectors

```
vector<int> v1 = {4,3,2,1,0};

do
{
    for( i=0; i<5; i++)
        cout << v1[i] << " ";
    cout << endl;
}while( prev_permutation(v1.begin(), v1.end()) );
```

SOUTH DAKOTA
SCHOOL OF MINES & TECHNOLOGY

Strings

```
string s1 = "edcba";
// strings
do
{
    cout << s1 << endl;
}while( prev_permutation( s1.begin(),s1.end()) );
```

Integer List

```
list<int> l1= {4,3,2,1,0};
list<int>::iterator it;

do
{
    for( it=l1.begin(); it != l1.end(); it++)
        cout << *it << " ";
    cout << endl;
}while( prev_permutation( l1.begin(), l1.end() ) );
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Array of Strings

```
string vals[10] = {"red", "orange", "green",
                   "blue", "black" };
do
{
    for( i=0; i<5; i++)
        cout << vals[i] << " ";
    cout << endl;
}while( prev_permutation( vals, vals+5 ) );
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Lexicographical Problem

```
string vals[10] = {"black", "blue", "orange",
"red", "green"};
do
{
    for( i=0; i<5; i++)
        cout << vals[i] << " ";
    cout << endl;
}while( prev_permutation( vals, vals+5 ) );
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Produces

black blue orange red green

black blue orange green red

black blue green red orange

black blue green orange red

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Conditional – Red in 2nd, green in 3rd

```
vector<string>data={"black","blue","green","red","white"};
do
{
    if( data[1] == "red" && data[2] == "green")
    {
        for( auto x: data)
            cout << x << " ";
        cout << endl;
    }
}while( next_permutation( data.begin(), data.end()));
```

SOUTH DAKOTA M SCHOOL OF MINES & TECHNOLOGY

Produces

```
black red green blue white
black red green white blue
blue red green black white
blue red green white black
white red green black blue
white red green blue black
```

Descramble a word – 1st and last are in position

```
string data = "Vctoer";
string::iterator first, last;

first = data.begin(); first++;
last = data.end(); last--;
sort(first, last);
do
{
    cout << data << endl;
}while( next_permutation( first, last) );
```

Produces

Vceotr	Veoctr	Votcer
Vcetor	Veotcr	Votecr
Vcoetr	Vetcor	Vtceor
Vcoter	Vetocr	Vtcoer
Vcteor	Vocetr	Vtecor
Vctoer	Vocter	Vteocr
Vecotr	Voectr	Vtocer
Vector	Voetcr	Vtoecr