CSC215

Math and Computer Science

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

List

- Include the List library

```
#include <list>
```

- Gives us a bi directional list (doubly linked list)
- Has forward and backwards iterators
- Does not provide random access ([], .at(), +, -)
- Can hold any object.

Creating a list

```
list<int> intList;       // list of integers
list<float> floatList;   // list of floats
list<string> strList;    // list of strings
list<rec> recList;       // list of rec structures
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Initializing at Instantiation

```
// 11 ways to declare and initialize a list
int array[5] = {1,2,3,4,5};

// list with 1,2,3,4,5
list<int> l1(array, array+5);
// list with 1,2,3,4,5
list<int> l2(l1.begin(), l1.end() );
// 100 integers with value 0
list<int> l3( 100, 0);
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

=, assign() Member functions

- Both will resize the data set if needed. May use them to assign other vectors that hold the same type, or the same type of arrays.
- Assign function has 2 different ways of usage

```
list<int> list1 = {1,2,3,4,5};
list<int> list2, list3;

list2 = list1; // {1,2,3,4,5}
// {1,2,3,4,5}
list2.assign(list1.begin(), list1.end());
list3.assign(5,100); // {100,100,100,100,100}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Swap Member Function

- Will exchange the values within two lists

```
list<int> list1 = {1,2,3,4,5 };
list<int> list2 = {6,7,8,9,10};

list1.swap(list2);
// list1 is now 6,7,8,9,10
// list2 is now 1,2,3,4,5
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Push Back Member Function

- Add an element to the end of the list

```
list<int> list1= {1,2,3,4,5};     // 1,2,3,4,5

list1.push_back(100);             // 1,2,3,4,5,100
```

Pop Back Member Function

- Removes an element at the end of the list

```
list<int> list1 = {1,2,3,4,5};     // 1,2,3,4,5

list1.pop_back();          // 1,2,3,4
```

Note: *WILL CRASH YOUR PROGRAM IF LIST IS EMPTY

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Push Front Member Function

- Add an element to the front of the list

```
list<int> list1 = {1,2,3,4,5};

list1.push_front(-1);              // -1,1,2,3,4,5
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Pop Front Member Function

- Removes an element at the front of the list

```
list<int> list1 = {1,2,3,4,5};

list1.pop_front( );          // 2,3,4,5
```

Note: *WILL CRASH YOUR PROGRAM IF LIST IS EMPTY

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Insert Member Function

- Insert will position one or more elements into an existing list.
  - 4 ways to call the insert function

```
list<int> list1 = {1,2,3,4,5};
list<int> list2 = {6,7,8,9,10};
list<int>::iterator it;
it = list1.begin();
it++;
list1.insert( it, list2.begin(), list2.end() );
// 1,6,7,8,9,10,2,3,4,5
list1.insert( list1.begin(),-10); // -10, 1,6,7,8,9,10,2,3,4,5
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Erase Member Function

- Erase, removes Elements from a list
  - 2 ways to call the erase function

```
list<int> list1 = {1,2,3,4,5};
list<int> list2 = {6,7,8,9,10};

list<int>::iterator it;
it = list1.begin();
it++;
list1.erase(it, list1.end() );       // 1
list2.erase( list2.begin() );     // 7,8,9,10
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Clear Member Functions

- Clear, Removes all elements from the list object.

```
list<int> list1 = {1,2,3,4,5};
list<int> list2 = {6,7,8,9,10};

list1.clear();     //
list2.clear();     //
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Resize Member Function

- Resize, changes the number of elements in the list.
  - Shrinks or increases the capacity of the list.
  - 2 ways to call it.

```
list<int> list1 = {1,2,3,4,5};
list<int> list2 = {6,7,8,9,10};

list1.resize(8,0);      // 1,2,3,4,5,0,0,0
list2.resize( 3 );      // 6,7,8
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Size Member Function

- Size returns the number of elements in the list

```
list<int> list1 = {1,2,3,4};
list<int> list2 = {5,6,7,8,9,10};

cout << list1.size( );   // outputs 4
cout << list2.size( );   // outputs 6
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Empty Member Function

- Empty, returns a true / false value based on if the list contains any data.

```
list<int> list1 = {1,2,3,4,5};
list<int> list2;
if( list1.empty() ) // no output
    cout << "list1 is empty" << endl;
if( list2.empty() ) // output
    cout << "list2 is empty" << endl;
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Max Size Member Functions

- max_size, returns an integer representing how large the list can become.

```
list<int> list1 = {1,2,3,4,5};
cout <<"Max Size = " << list1.max_size() << endl;
```

Max Size = 357913941

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Front and Back Member functions

- Front allows you to access the element that comes first.
- Back allows you to access the element that is last.

```
list<int> list1 = {1,2,3,4,5};

// 1
cout << list1.front();
// 5
cout << list1.back();
```

```
// list1 = 20,2,3,4,5
list1.front() = 20;
// list1 = 20,2,3,4,100
list1.back() = 100;
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Merge Member Function

- Combine sorted lists into first list
  - L1.merge(L2);
  - L2 becomes empty
  - L1 contains the list L2

```
list<int> list1 = { 1,2,3,4,8,10};
list<int> list2 = { 0,3,5,6,9};
list1.merge(list2);     // 0,1,2,3,3,4,5,6,8,9,10
                        // list2 is empty
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Splice Member Function

- Transfers elements from a second list and insert them into the first list
  - List 2 becomes empty
  - 6 ways to call the splice

```
list<int> list1 = {1,2,4,8,10};
list<int> list2 = {0,3,5,6,9};
list<int>::iterator it;
it = list1.begin();
it++; it++;              // move iterator down 2 spots

list1.splice( it, list2 ); // 1,2,0,3,5,6,9,4,8,10
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Remove Member Function

- All elements in the list that match the supplied value are deleted from the list.

```
list<int> list1 = {1,2,3,1,10,11,53,1,6,9};

list1.remove(1);   // 2 3 10 11 53 6 9
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Remove if Member Function

- All elements in the list that match the supplied function are deleted from the list.

```
list<int> list1 = {1,2,4,6,10,11,53,1,6,9};

list1.remove_if( isEven ); // 1,11,53,1,9
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Remove If Function

- Must return true to remove it, false to keep it.
- Expects a const data type to be passed by reference into the function.

```
// list<int> must match const int &
bool isEven ( const int &item )
{
    return ( (item & 1) == 1 ? false : true);
}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Unique Member Function

- Removes adjacent elements that are duplicates from the list

```
list<int> list1 = {1,2,2,5,5,5,1,1,9,9};

list1.unique(); // 1,2,5,1,9
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Sort Member Function

- Puts the list into ascending order or order based on a function.
- Operator < must work on the data type
- 2 ways to call the function.

```
list<int> list1 = {1,2,4,6,10,11,53,1,6,9};

list1.sort(); // 1,1,2,4,6,6,9,10,11,53
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Sort Member Function

- Puts the list into ascending order or order based on a function.
- 2 ways to call the function.

```
list<int> list1 = {1,2,4,6,10,11,53,1,6,9};

list1.sort( partition );// 1,1,9,11,53,2,4,6,6,10
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Comparison Function for Sort

- List<int> - write function for left item compared to right item
  - Return true if in order
  - Return false if out of order

```
bool partition( int leftitem,
                int rightitem)
{  bool odd1, odd2;
   odd1 = (leftitem & 1 ) == 1;
   odd2 = (rightitem & 1) == 1;
   if( odd1 && !odd2 )
        return true;
   if( odd2 && !odd1 )
        return false;
   return leftitem <
                rightitem;
}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Reverse Member Function

- Swap elements within the list so that the first item becomes the last, the second item becomes the second from the end.

```
list<int> list1 = {1,2,3,4,5};

list1.reverse(); // 5,4,3,2,1
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Iterator Functions

- begin() and end() forward iterators

```
list<int>::iterator iit;
list<double>::iterator dit;
```

- rbegin() and rend() reverse iterators

```
list<int>::reverse_iterator irit;
list<int>::reverse_iterator drit;
```

Note: iit = iit + 3; iit = iit – 5; Not available since it is not stored in consecutive memory

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Passing Lists to Functions – By Value

- Function Prototypes

```
void func1( list<int> lt );
void func2( list<double> lt );
```

- Function Calls

```
list<int> l1;
list<double> l2;
func1(l1);
func2(l2);
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Passing Lists to Functions – By Reference

- Function Prototypes

```
void func1( list<int> &lt );
void func2( list<double> &lt );
```

- Function Calls

```
list<int> l1;
list<double> l2;
func1(l1);
func2(l2);
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Traversing the List

```
list<int> l = {1,2,3,1,10,11,53,1,6,9};
list<int>::iterator it;

it = l.begin();                 // temp = headptr
while( it != l.end() )          // temp != nullptr
{
    cout << *it << " ";
    it++;                       // temp = temp->next
}
cout << endl;
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Range based for loop - value

```
list<int> lt = {1,2,3,1,10,11,53,1,6,9};
for( auto x : lt)
{   ++x;
    cout << x << " "; } //2 3 4 2 11 12 54 2 7 10

for( auto v : lt)
    cout << v << " ";  //1 2 3 1 10 11 53 1 6 9
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Range based for loop - reference

```
list<int> lt = {1,2,3,1,10,11,53,1,6,9};
for( auto &x : lt)
{   ++x;
    cout << x << " "; } //2 3 4 2 11 12 54 2 7 10

for( auto v : lt)
    cout << v << " ";  // 2 3 4 2 11 12 54 2 7 10
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Partial List

```
list<int> lt = {1,2,3,1,10,11,53,1,6,9};
list<int>::iterator it, startSpot, stopSpot;

startSpot = lt.begin();
startSpot++;                        // iterator at 2
stopSpot = lt.end();
stopSpot--; stopSpot--;             // iterator at 6

for( it = startSpot; it != stopSpot; it++)
    cout << *it << " ";             // 2 3 1 10 11 53 1
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Find Function

```
list<int>::iterator listfind( list<int> &lt, int tgt)
{   list<int>::iterator it;

    it = lt.begin();
    while( it != lt.end())
    {
        if( *it == tgt)
            return it;
        it++;
    }
    return it;              // lt.end();
}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Using the Function

```
list<int> lt = {1,2,3,1,10,11,53,1,6,9};
list<int>::iterator it;

it = listfind( lt, 10);
if( it != lt.end())
    cout << "Found the item: " << *it << endl;
else
    cout << "Didn't find the item" << endl;

it = listfind( lt, 12);
if( it != lt.end())
    cout << "Found the item: " << *it << endl;
else
    cout << "Didn't find the item 12" << endl;
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Algorithm - Transform

```
list<int> l1 = {1,2,3,1,10,11,53,1,6,9};
list<int> l2;
list<int>::iterator it;
l2.resize( l1.size(), 0);
transform(l1.begin(), l1.end(), l2.begin(), inc);

for( auto x: l2)
    cout << x << " ";     // 2 3 4 2 11 12 54 2 7 10
cout << endl;
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Inc Function

```
int inc( int v)
{
    return v+1;
}
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Algorithm – find, find_if, find_end …..

```
list<int> l1 = {1,2,3,1,10,11,53,1,6,9};
list<int>::iterator it;

it = find( l1.begin(), l1.end(), 10); // operator==
if( it != l1.end())
    *it += 10;

for( auto x: l1)// 1 2 3 1 20 11 53 1 6 9
    cout << x << " ";
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY