CSC215

Math and Computer Science

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

A Legacy of Excellence

What do you dislike about arrays?

- Overstepping the bounds of arrays
- Constant size (dynamic helps)
- Can not pass by value
- Assignment operator missing (a = b)

Vector

- Grows and Shrinks as needed
- Has an assignment operator
- Can be passed by value
- Boundary checking with the at() function

Declaring a Vector

- Use the library

```
#include <vector>
```

- Declaring a vector
  - Can only contain 1 data type
- Examples (11 ways to initialize vectors)

```
vector<int> v1;
vector<double> v2;
vector<string> v3;
vector<studentRec> v4;
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Initializing Vectors

```
int arr[10] = {1,2,3,4,5,6,7,8,9,10};

vector<int> v1;
vector<int> v2(arr, arr+10);
vector<int> v3(v2);
Vector<int> v4 = {1,2,3,4,5,6,7,8,9,10};
```

=, 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.

Examples:

```
v1 = v2;
v1.assign(4, 10);                // v1 = 10,10,10,10
v1.assign( v2.begin(), v2.end());
                                 // 1,2,3,4,5,6,7,8,9,10
v1.assign( arr+1, arr+5);        // v1 = 2,3,4,5
```

Swap Member Function

- Will exchange the values within two vectors.

```
vector<int> v1 = {1,2,3,4};
vector<int> v2 = {5,6,7};

v1.swap(v2);
// v1 = 5,6,7    v2=1,2,3,4
```

Push Back Member Functions

- push_back, adds an element to the end of the vector.

```
vector<int> v1 = {1,2,3,4};
vector<int> v2 = {5,6,7};

v1.push_back(v2[0]);                 // 1,2,3,4,5
v2.push_back(42);                    // 5,6,7,42
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Pop Back Member Function

- pop_back, removes the last element in the vector. It does not return the value. It is a non valued function.

```
vector<int> v1 = {1,2,3,4};
vector<int> v2 = {5,6,7};

v1.pop_back();              // 1,2,3
v2.pop_back();              // 5,6
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Insert Member Function

- Insert will position one or more elements into an existing vector.
  - Works with arrays, and vectors.
  - 4 ways to call the insert function

```
vector<int> v1 = {1,2,3,4};
vector<int> v2 = {5,6,7};
v1.insert( v1.begin()+1, v2.begin(), v2.end());
                                        // 1,5,6,7,2,3,4
v2.insert( v2.begin()+1, 3, 42 );   // 5,42,42,42,6,7
v2.insert( v2.begin(), 42 );        // 42, 5,6,7
```

SOUTH DAKOTA
SCHOOL OF MINES & TECHNOLOGY

Erase Member Function

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

```
vector<int> v1 = {1,2,3,4,5,6,7,8,9,10};

v1.erase( v1.begin()+2 );    // 1,2,4,5,6,7,8,9,10
v1.erase( v1.begin()+2, v1.end()-3);
                                // 1,2,8,9,10
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Clear Member Function

- Clear, Removes all elements from the vector object.

```
vector<int> v1 = {1,2,3,4,5,6,7,8,9,10};
vector<int> v2 = {5,6,7};

v1.clear();             //
v2.clear();             //
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Resize Member Function

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

```
vector<int> v1 = {1,2,3,4,5,6,7,8,9,10};
vector<int> v2 = {5,6,7};
v1.resize(5);                     // 1,2,3,4,5
V1.resize(10);                    // 1,2,3,4,5,0,0,0,0,0
v2.resize(10,1);                  // 5,6,7,1,1,1,1,1,1,1
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Size Member Functions

- Size returns the number of elements in the vector object.

```
vector<int> v1 = {1,2,3,4,5,6,7,8,9,10};
vector<int> v2 = {5,6,7};

cout << v1.size() << endl;  // 10
cout << v2.size() << endl;  // 3
```

SOUTH DAKOTA M SCHOOL OF MINES & TECHNOLOGY

Empty Member Function

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

```
vector<int> v1 = {1,2,3,4,5,6,7,8,9,10};
vector<int> v2;

if( v1.empty() ) // no output
    cout << "V1 is empty" << endl;
if( v2.empty() ) // output
    cout << "V2 is empty" << endl;
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Max Size and Capacity Member Functions

- max_size, returns an integer representing how large the vector can become.
- capacity, returns an integer representing the number of elements the vector can hold be for it needs to resize.

```
cout << "Max Size: " << v1.max_size() << endl;
cout << "Capacity: " << v1.capacity() << endl;
// max_size: 1073741823  capacity:  10
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

[] and at Member functions

- Both allow access to individual elements.
- Neither will increase the size of the vector
- At function will do boundary checking. Safely exits program.

```
vector<int> v1= {1,2,3,4,5,6,7,8,9,10};
v1[1] = 42;           v1.at(1) = 42;
cout << v1[1];        cout << v1.at(1);
v1[10] = 11;          v1.at(10) = 11;
```

Front and Back Member Functions

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

```
vector<int> v1 = {1,2,3,4,5,6,7,8,9,10};

    v1.front() = 42;              v1.back() = 42;
// v1 = 42,2,3,4,5,6,7,8,9,42
    cout << v1.front();           cout << v1.back();
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Data Member Function

- Data() returns a pointer to the first element in the vector.
- This pointer can be use to access the data within the vector directly.
- This pointer can be passed to a function written for an array.

Data Example

```
vector<int> v1= {1,2,3,4,5,6,7,8,9,10};
int *p;

p = v1.data();
for( i=0; i<v1.size(); i++)
    p[i] = 1;           // changes every element with v1 to 1

for( i=0; i<v1.size(); i++)
    cout << v1[i] << " ";
cout << endl;           // outputs 1 1 1 1 1 1 1 1 1 1
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Iterator Functions

- begin() and end() forward iterators

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

- rbegin() and rend() reverse iterators

```
vector<int>::reverse_iterator irit;
vector<double>::reverse_iterator drit;
```

Passing vectors to functions – By Value

- Function Prototypes

```
void func1( vector<int> v );
void func2( vector<double> v);
```

- Function Calls

```
vector<int> v1;
vector<double> v2;
func1(v1);
func2(v2);
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Passing Vectors to Functions – By Reference

- Function Prototypes

```
void func1( vector<int> &v );
void func2( vector<double> &v);
```

- Function Calls

```
vector<int> v1;
vector<double> v2;
func1(v1);
func2(v2);
```

SOUTH DAKOTA
SCHOOL OF MINES & TECHNOLOGY