Arrays/Strings/Vectors

Prof. Seokin Hong

Agenda

- Introduction to Array
- Arrays in Function
- Multidimensional Arrays
- Array type for strings
- string class
- vector

Introduction to Array

Introduction to Arrays

- An array is used to process a collection of data of the same type
- Declaring an Array

  int score[5]; → This is like declaring 5 variables of type int:

  score[0], score[1], … , score[4]

  index

- The variables making up the array are referred to as
  - Indexed variables
  - Elements of the array

Arrays in a Loop

- **for-loops are commonly used to step through arrays**
  - Example:

First index is 0

Last index is (size – 1)

```
for (i = 0; i < 5; i++)
{
      cout << score[i] << endl;
}
```

Constants and Arrays

- **Use constants to declare the size of an array**
  - Using a constant allows your code to be easily altered for use on a smaller or larger set of data
    - **Example**:

```
const int  NUMBER_OF_STUDENTS = 50;
int score [NUMBER_OF_STUDENTS];
                              …
for ( i = 0; i < NUMBER_OF_STUDENTS; i++)
        cout << score[i] << endl;
```

Variables and Declarations

- Some compilers do not allow the use of a variable to declare the size of an array

```
cout << "Enter number of students: ";
cin >> number;
int score[number];
```

- This code is illegal on many compilers

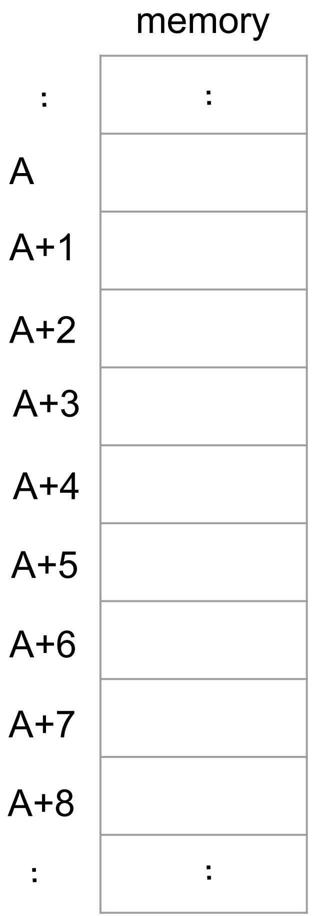
Computer Memory

- **Computer memory consists of bytes**

- **A simple variable is stored in consecutive bytes**
  - The number of bytes depends on the variable's type

- **A variable's address is the address of its first byte**
  - Big-endian
  - Little-endian

0x59654148

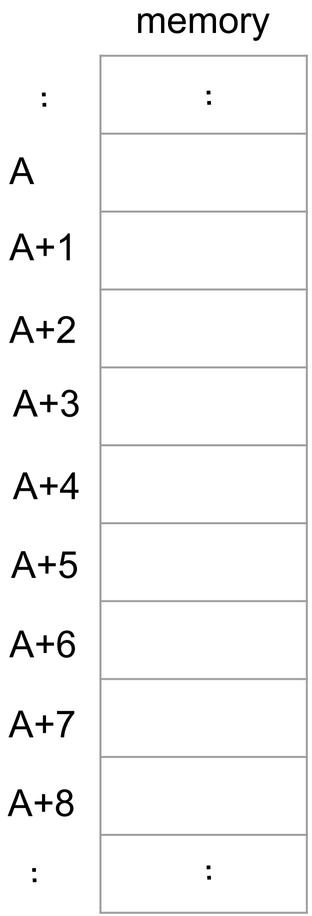


memory

:
A
A+1
A+2
A+3
A+4
A+5
A+6
A+7
A+8
:

Arrays and Memory

- **Declaring the array** **int a[2]**
  - Reserves memory for two variables of type int (4 bytes)
  - The variables are stored one after another
  - The address of **a[0]** is remembered
  - To determine the address of **a[1]**
    - Start at a[0]
    - Count past enough memory for two integers to find a[1]



Array Index Out of Range

- **A common error is using a nonexistent index**
  - Index values for **int a[6]** are the values 0 through 5
  - An index value not allowed by the array declaration is out of range
  - Using an out of range index value doe not produce an error message!

- **Example**
  - If an array is declared as: int a[6];
    and an integer is declared as: int i = 7;

Initializing Arrays

- **Initializing an array when it is declared**
  - Example: int children[3] = { 2, 12, 1 };

    Is equivalent to:

    ```
    int children[3];
    children[0] = 2;
    children[1] = 12;
    children[2] = 1;
    ```

- **Default Values**
  - If too few values are listed in an initialization statement
    - The listed values are used to initialize the first of the indexed variables
    - The remaining indexed variables are initialized to a zero of the base type
    - Example: int a[10] = {5, 5};

      initializes a[0] and a[1] to 5, and **a[2] through a[9] to 0**

DO NOT DEPEND ON THIS!

Range-Based For Loops

- C++11 includes a new type of for loop, the range-based for loop, that simplifies iteration over every element in an array. The syntax is shown below:

```
for (datatype varname : array)
{
    // varname is successively set to each
    // element in the array
}
```

```
int arr[ ] = {2, 4, 6, 8};
for (int x : arr)
            cout << x;
```

```
int arr[ ] = {2, 4, 6, 8};
for (auto x : arr)
            cout << x;
```

```
int arr[ ] = {2, 4, 6, 8};
for (int& x : arr)
            x++;
```

```
int arr[ ] = {2, 4, 6, 8};
for (auto& x : arr)
            x++;
```

Arrays in Function

Function Calls With Arrays

- If function fillUp() is declared in this way:

```
void fillUp(int a[ ], int size);
```

?

and array score is declared this way:

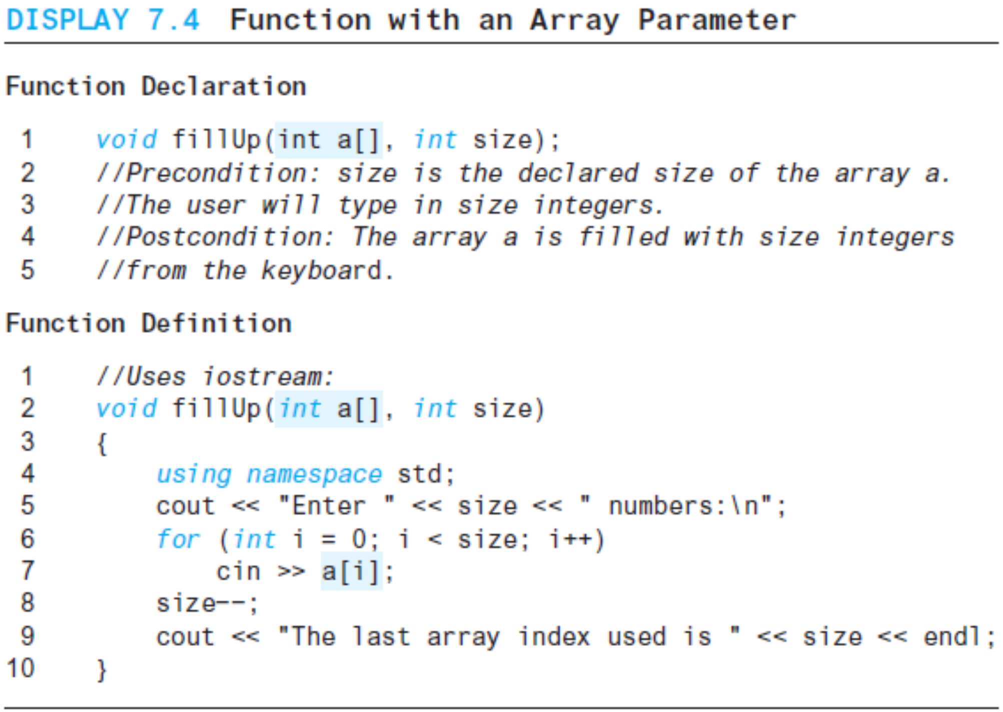
```
int score[5], numberOfScores;
```

fillUp() is called in this way:

```
fillUp(score, numberOfScores);
```

- The values of the indexed variables can be changed by the function

Function Calls With Arrays - Example



DISPLAY 7.4 Function with an Array Parameter

Function Declaration

```
void fillUp(int a[], int size);
//Precondition: size is the declared size of the array a.
//The user will type in size integers.
//Postcondition: The array a is filled with size integers
//from the keyboard.
```

Function Definition

```
//Uses iostream:
void fillUp(int a[], int size)
{
    using namespace std;
    cout << "Enter " << size << " numbers:\n";
    for (int i = 0; i < size; i++)
        cin >> a[i];
    size--;
    cout << "The last array index used is " << size << endl;
}
```

const Modifier

- Array parameters allow a function to change the values stored in the array argument
- If a function should not change the values of the array argument, use the modifier const
  - Example:

```
void showTheWorld(const int a[ ], int size);
```

- The compiler will issue an error if you write code that changes the values stored in the array parameter

const Modifier - Example

- **double computeAverage(int a[ ], int size);**

```
void showDifference(const int a[ ], int size)
{
      double average = computeAverage(a, size);
       …
}
```

- **computeAverage has no constant array parameter**
- **This code generates an error message because computeAverage could change the array parameter**

Multidimensional Arrays

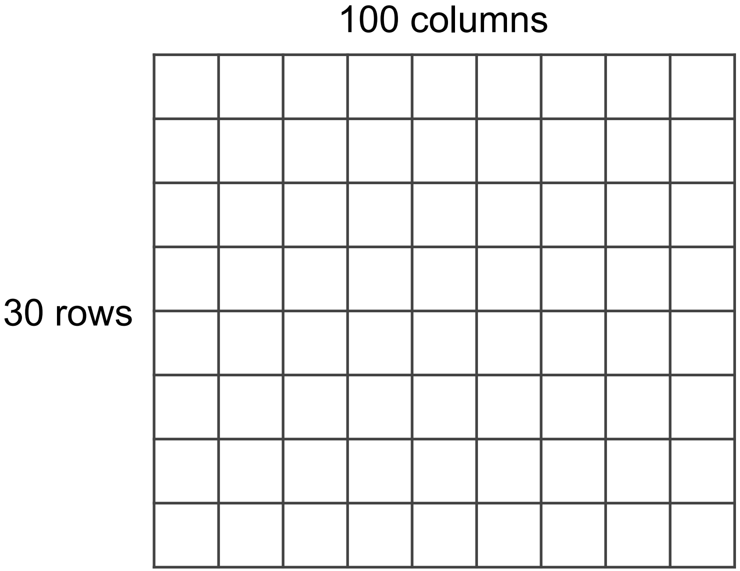
Multi-Dimensional Arrays

- **C++ allows arrays with multiple index values**
  - char page [30] [100];
    → an array of 30 rows and 100 columns (Two-dimensional array)

- **The indexed variables for array page are**
  page[0][0], page[0][1], …, page[0][99]
  page[1][0], page[1][1], …, page[1][99]

  …

  page[29][0], page[29][1], … , page[29][99]



- **page is actually an array of size 30**
  - page's base type is an array of 100 characters

Multi-Dimensional Array Parameters

- **Recall that the size of an array is not needed when declaring a formal parameter:**

```
void displayLine(const char a[ ], int size);
```

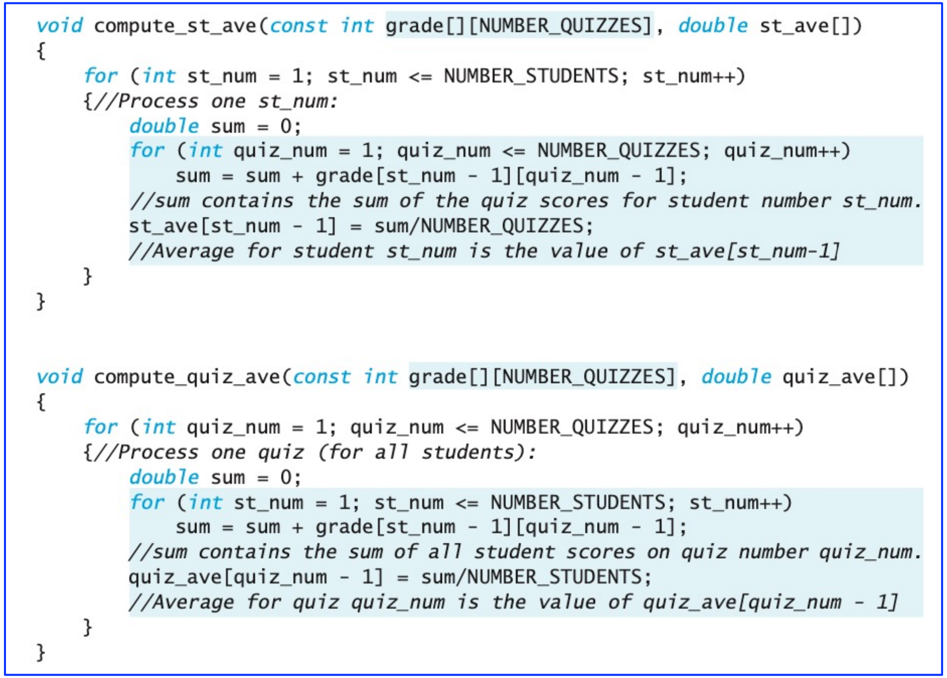
- **For a multidimensional array parameter, the size of the first dimension is not given, but the remaining dimension sizes must be given**

```
void displayPage(const char page[ ] [100], int sizeDimension1);
```

Program Example

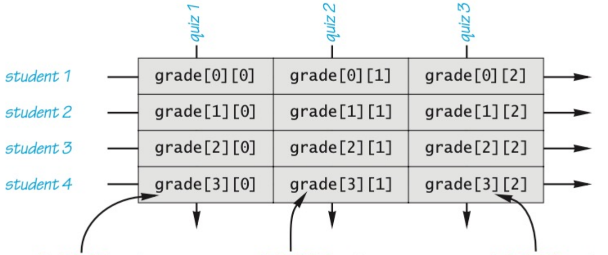
```
const int NUMBER_STUDENTS = 4, NUMBER_QUIZZES = 3;

int main( ) {
 using namespace std;
 int grade[NUMBER_STUDENTS][NUMBER_QUIZZES];
 double st_ave[NUMBER_STUDENTS];
 double quiz_ave[NUMBER_QUIZZES];
  compute_st_ave(grade, st_ave);
  compute_quiz_ave(grade, quiz_ave);
  display(grade, st_ave, quiz_ave);
  return 0;
}
```



```
void compute_st_ave(const int grade[][NUMBER_QUIZZES], double st_ave[])
{
    for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
    {//Process one st_num:
        double sum = 0;
        for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
            sum = sum + grade[st_num - 1][quiz_num - 1];
        //sum contains the sum of the quiz scores for student number st_num.
        st_ave[st_num - 1] = sum/NUMBER_QUIZZES;
        //Average for student st_num is the value of st_ave[st_num-1]
    }
}


void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double quiz_ave[])
{
    for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
    {//Process one quiz (for all students):
        double sum = 0;
        for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
            sum = sum + grade[st_num - 1][quiz_num - 1];
        //sum contains the sum of all student scores on quiz number quiz_num.
        quiz_ave[quiz_num - 1] = sum/NUMBER_STUDENTS;
        //Average for quiz quiz_num is the value of quiz_ave[quiz_num - 1]
    }
}
```



`grade[3][0]` *is the grade that student 4 received on quiz 1.*

`grade[3][1]` *is the grade that student 4 received on quiz 2.*

`grade[3][2]` *is the grade that student 4 received on quiz 3.*

An Array Type for Strings

An Array Type for Strings

- **C-strings can be used to represent strings of characters**
  - C-strings are stored as arrays of characters
  - C-strings use the null character '\0' to end a string
  - To declare a C-string variable, declare an array of characters:

```
char s[11];
```

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
H	i		M	o	m	!	\0	?	?

Declaring and Initializing a C-string

- **To declare a C-string variable, use the syntax:**
  char Array_name[Maximum_cString_Size + 1];
  - + 1 reserves the additional character needed by '\0'

- **To initialize a C-string during declaration:**
  char myMessage[20] = "Hi there.";
  - The null character '\0' is added for you

- **Another alternative:**

```
        char shortString[ ] = "abc";
```

but not this:

```
         char shortString[ ] = {'a', 'b', 'c'};
```

Assignment With C-strings

- **A common method to assign a value to a C-string variable is to use strcpy(), defined in the cstring library**
  - Example:

```
#include <cstring>
        …
char aString[ 11];
strcpy (aString, "Hello");
```

- **This statement is illegal:**

```
aString = "Hello";
```

  - The assignment operator does not work with C-strings!!

== Alternative for C-strings

- **The == operator does not work with C-strings**
- **strcmp() is used to compare C-string variables**

  - Example:

```
#include <cstring>
                    …
if (strcmp(cString1, cString2))
        cout << "Strings are not the same.";
else
        cout << "String are the same.";
```

strcmp()'s logic

- **strcmp() compares the numeric codes of elements in the C-strings a character at a time**
  - If the two C-strings are the same, strcmp returns 0
  - As soon as the characters do not match
    - strcmp() returns a negative value if the numeric code in the first parameter is less
    - strcmp() returns a positive value if the numeric code in the second parameter is less

More C-string Functions

- **The cstring library includes other functions**
  - **strlen()** returns the number of characters in a string

```
int x = strlen( aString);
```

  - **strcat ()** concatenates two C-strings
    - The second argument is added to the end of the first
    - The result is placed in the first argument
    - Example:

```
char stringVar[20] = "The rain";
strcat(stringVar, "in Spain");     \\ Now stringVar contains "The rainin Spain"
```

C-string Output

- **C-strings can be output with the insertion operator**
  - Example:

    ```
    char news[ ] = "C-strings";
    cout << news << " Wow."
         << endl;
    ```

C-string Input

- **The extraction operator >> can fill a C-string**
  - Whitespace ends reading of data
  - Example:

```
char a[80], b[80];
cout << "Enter input: " << endl;
cin >> a  >> b;
cout << a << b << "End of Output";
```

```
Enter input:
    Do be do to you!
    DobeEnd of Output
```

Reading an Entire Line

- **getline() can read an entire line, including spaces**
  - getline() is a member of all input streams (**istream**)
  - getline() has two arguments

    ```
    cin.getline(String_Var, Max_Characters + 1);
    ```

    - The first is a C-string variable to receive input
    - The second is an integer, usually the size of the first argument specifying the maximum number of elements to fill
      - Max_Characters + 1 reserves one element for the null character
    - **cin** can be replaced by **any input stream**

Using getline()

- **The following code is used to read an entire line including spaces into a single C-string variable**

```
char a[80];
cout << "Enter input:\n";
cin.getline(a, 80);
cout << a << End Of Output\n";
```

and could produce:

```
Enter some input:
Do be do to you!
Do be do to you!End of Output
```

getline() and Files

- C-string input and output work the same way with file streams

```
std::ifstream inStream;
std::ofstream out_stream;
inStream.open(“infile.dat”);
out_stream.open(“outfile.dat”);


inStream >> cString;
inStream.getline(cString, 80);

out_stream << cString;
```

C-strings to Numbers

#include <cstdlib>

- **C-strings to Integers**
  - Read input as characters into a C-string, removing unwanted characters
  - Use the predefined function **atoi()** to convert the C-string to an int value
    - Example:
      - atoi("1234") returns the integer 1234
      - atoi("#123") returns 0 because # is not a digit
- **C-strings to long**
  - **atol()**
- **C-strings to double**
  - **atof()**

atof("9.99") returns 9.99
atof("$9.99") returns 0.0 because the $ is not a digit

The Standard string Class

The Standard string Class

- The **string class** allows the **programmer to treat strings as a basic data type**
- **The string class is defined in the string library and the names are in the standard namespace (std)**

```
#include <string>
  std::string s1, s2, s3;
```

OR

```
  using namespace std;
  string s1, s2, s3;
```

string Constructors

- **The default string constructor initializes the string to the empty string**
- **Another string constructor takes a C-string argument**
  - Example:

```
string phrase;          // empty string
string noun("ants");    // a string version of "ants"
```

Assignment of Strings

- **Variables of type string can be assigned with the = operator**
  - Example:

```
string s1, s2, s3;
        …
s3 = s2;
```

- **Quoted strings are type cast to type string**
  - Example:

```
string s1 = "Hello Mom!";
```

Using + With strings

- **Variables of type string can be concatenated with the + operator**
  - Example:

```
string s1, s2, s3;
            …
 s3 = s1 + s2;
```

  - If s3 is not large enough to contain s1 + s2, more space is allocated

I/O With Class string

- **The insertion operator << is used to output objects of type string**
  - Example:

```
string s = "Hello Mom!";
cout << s;
```

- **The extraction operator >> can be used to input data for objects of type string**
  - Example:

```
string s1;
cin >> s1;
```

std::getline() and Type string

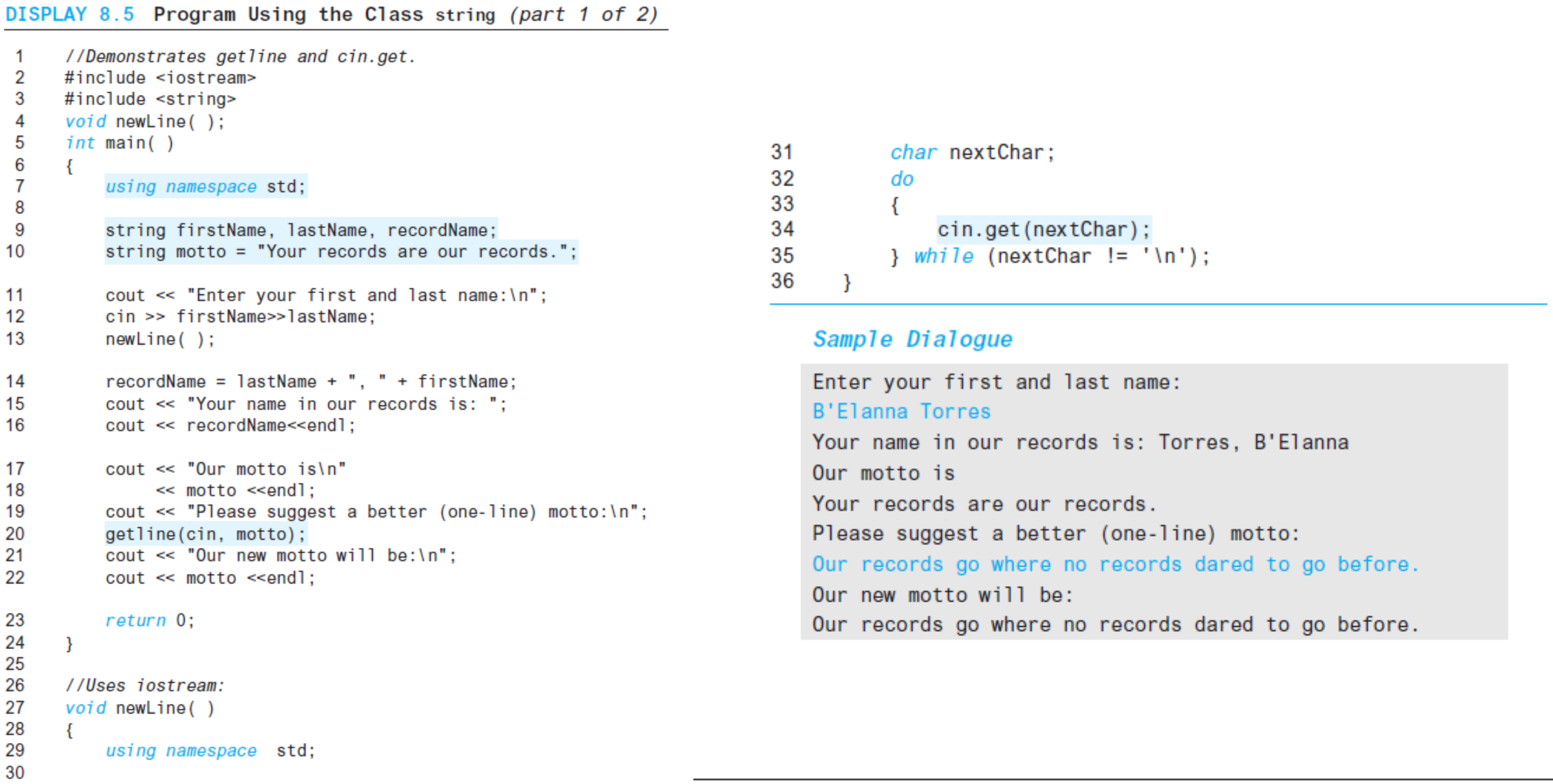
- **A std::getline() function exists to read entire lines into a string variable**
  - This version of getline is not a member of the istream class, it is a non-member function
  - Syntax for using this getline() is different than that used with cin: cin.getline(…)
- **Syntax for using getline with string objects:**

  getline(Istream_Object, String_Object);

- **Example**

```
std::string line;
std::cout << "Enter a line of input:\n";
std::getline(std::cin, line);
std::cout << line << "END OF OUTPUT\n";
```

std::getline() and Type string – example



DISPLAY 8.5 Program Using the Class string *(part 1 of 2)*

```cpp
//Demonstrates getline and cin.get.
#include <iostream>
#include <string>
void newLine( );
int main( )
{
    using namespace std;

    string firstName, lastName, recordName;
    string motto = "Your records are our records.";

    cout << "Enter your first and last name:\n";
    cin >> firstName>>lastName;
    newLine( );

    recordName = lastName + ", " + firstName;
    cout << "Your name in our records is: ";
    cout << recordName<<endl;

    cout << "Our motto is\n"
         << motto <<endl;
    cout << "Please suggest a better (one-line) motto:\n";
    getline(cin, motto);
    cout << "Our new motto will be:\n";
    cout << motto <<endl;

    return 0;
}

//Uses iostream:
void newLine( )
{
    using namespace  std;

    char nextChar;
    do
    {
        cin.get(nextChar);
    } while (nextChar != '\n');
}
```

Sample Dialogue

```
Enter your first and last name:
B'Elanna Torres
Your name in our records is: Torres, B'Elanna
Our motto is
Your records are our records.
Please suggest a better (one-line) motto:
Our records go where no records dared to go before.
Our new motto will be:
Our records go where no records dared to go before.
```

Another Version of std::getline()

- **The versions of std::getline() we have seen, stop reading at the end of line marker '\n'**
- **getline can stop reading at a character specified in the argument list**
  - This code stops reading when a '?' is read

```
std::string line;
std::cout <<"Enter some input: \n";
std::getline(cin, line, '?');
```

getline() Declarations

- **These are the declarations of the versions of getline for string objects we have seen**
  - istream& getline(istream& ins, string& strVar);
  - istream& getline(istream& ins, string& strVar, char delimiter);

Mixing cin >> and std::getline()

- **Recall cin >> n skips whitespace to find what it is to read then stops reading when whitespace is found**

- **cin >> leaves the '\n' character in the input stream**
  - **Example**

```
int n;
std::string line;
std::cin >> n;
std::getline(std::cin, line);
```

leaves the '\n' which immediately ends getline’s reading…

line is set equal to the empty string

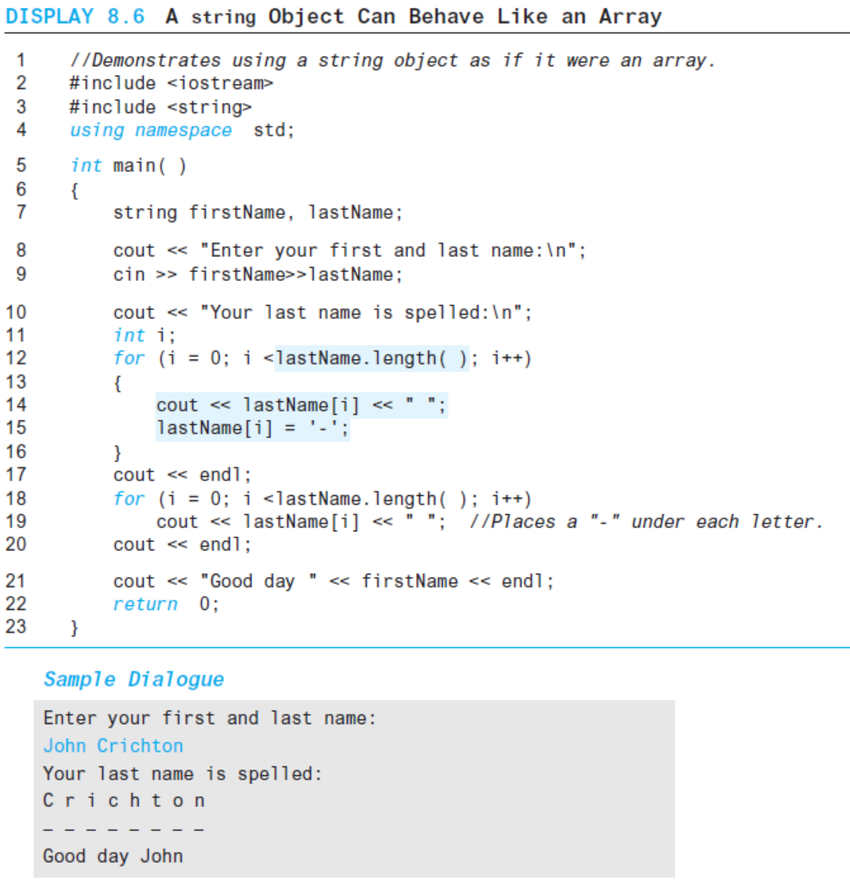
ignore()

- **ignore() is a member of the istream class**
- **ignore() can be used to read and discard all the characters, including '\n’**
- **ignore() takes two arguments**
    - First, the maximum number of characters to discard
    - Second, the character that stops reading and discarding
  - Example:

```
std::string line;
std::cin>>n;
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n’); \\reads up to max() characters or to '\n’
std::getline(std::cin, line);
```

String Processing

- **The string class allows the same operations we used with C-strings**
  - accessed as if they are in an array
    - ***last_name[i]*** provides access to a single character as in an array
    - It does not check for valid index value
      - For invalid index, operator [] brings undefined behavior
- **length()**
  - returns the number of characters in the string object
  - Example: int n = stringVar.length( );



DISPLAY 8.6 A string Object Can Behave Like an Array

```
//Demonstrates using a string object as if it were an array.
#include <iostream>
#include <string>
using namespace  std;

int main( )
{
    string firstName, lastName;

    cout << "Enter your first and last name:\n";
    cin >> firstName>>lastName;

    cout << "Your last name is spelled:\n";
    int i;
    for (i = 0; i <lastName.length( ); i++)
    {
        cout << lastName[i] << " ";
        lastName[i] = '-';
    }
    cout << endl;
    for (i = 0; i <lastName.length( ); i++)
        cout << lastName[i] << " ";  //Places a "-" under each letter.
    cout << endl;

    cout << "Good day " << firstName << endl;
    return  0;
}
```

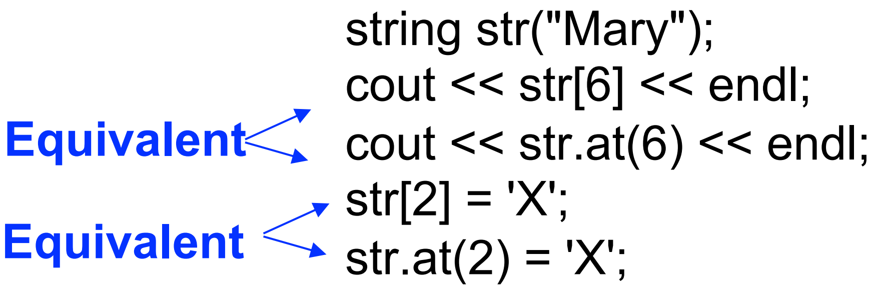
Sample Dialogue

```
Enter your first and last name:
John Crichton
Your last name is spelled:
C r i c h t o n
- - - - - - - -
Good day John
```

String Processing (Cont.)

- **at()**
  - an alternative to using [ ]'s to access characters in a string.
    - **at()** checks for valid index values
      - If the index is not valid, through an exception.

    - **Example:**

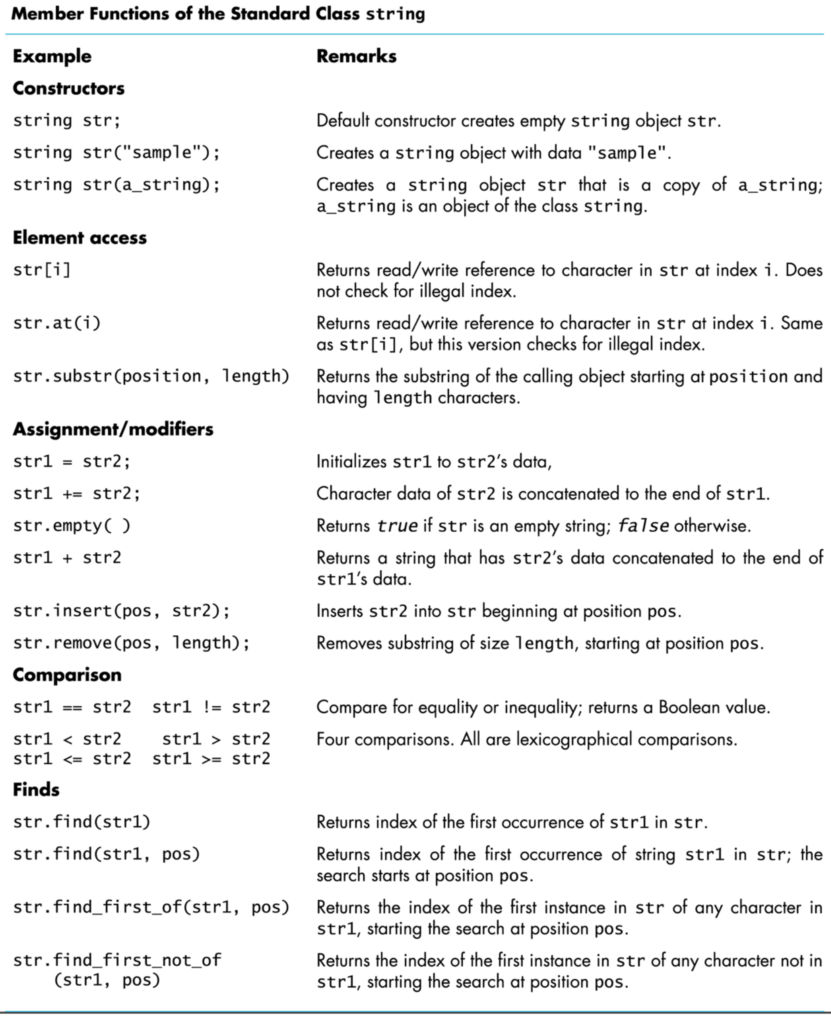


```
string str("Mary");
cout << str[6] << endl;
cout << str.at(6) << endl;
str[2] = 'X';
str.at(2) = 'X';
```

Equivalent

Equivalent

More member functions



Member Functions of the Standard Class string

Example	Remarks
Constructors	
`string str;`	Default constructor creates empty `string` object `str`.
`string str("sample");`	Creates a `string` object with data `"sample"`.
`string str(a_string);`	Creates a `string` object `str` that is a copy of `a_string`; `a_string` is an object of the class `string`.
Element access	
`str[i]`	Returns read/write reference to character in `str` at index `i`. Does not check for illegal index.
`str.at(i)`	Returns read/write reference to character in `str` at index `i`. Same as `str[i]`, but this version checks for illegal index.
`str.substr(position, length)`	Returns the substring of the calling object starting at `position` and having `length` characters.
Assignment/modifiers	
`str1 = str2;`	Initializes `str1` to `str2`'s data,
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`.
`str.empty( )`	Returns *`true`* if `str` is an empty string; *`false`* otherwise.
`str1 + str2`	Returns a string that has `str2`'s data concatenated to the end of `str1`'s data.
`str.insert(pos, str2);`	Inserts `str2` into `str` beginning at position pos.
`str.remove(pos, length);`	Removes substring of size `length`, starting at position pos.
Comparison	
`str1 == str2  str1 != str2`	Compare for equality or inequality; returns a Boolean value.

`str1 < str2    str1 > str2`
`str1 <= str2  str1 >= str2`
	Four comparisons. All are lexicographical comparisons.

Finds	
`str.find(str1)`	Returns index of the first occurrence of `str1` in `str`.
`str.find(str1, pos)`	Returns index of the first occurrence of string `str1` in `str`; the search starts at position pos.
`str.find_first_of(str1, pos)`	Returns the index of the first instance in `str` of any character in `str1`, starting the search at position pos.

`str.find_first_not_of`
`(str1, pos)`
	Returns the index of the first instance in `str` of any character not in `str1`, starting the search at position pos.

string class to numbers

- **C++11 has new functions to convert a string class object to a number**
  - std::stoi(), std::stod(), std::stol(), std::stof()

```
int i;
double d;
string s1 = ”35”; string s2=“2.5”
i = std::stoi(s1);  // Converts the string "35" to an integer 35
d = std::stod(s2); // Converts the string "2.5" to the double 2.5
```

- **to_string()**
  - Convert a numeric type to a string

```
string s = std::to_string(1.2*2);  // “2.4” stored in s
```

Comparison of strings

- **Comparison operators work with string objects**
  - Objects are compared using lexicographic order (Alphabetical ordering using the order of symbols in the ASCII character set.)
  - **== returns true** if two string objects contain the same characters in the same order
  - <, >, <=, >= can be used to compare string objects

Converting strings to C-strings

- **The string class member function c_str() returns the C-string version of a string object**
  - **Example:**

    ```
    strcpy(aCString, stringVariable.c_str( ) );
    ```

- **This line is still illegal**

  ```
  aCString = stringVariable.c_str( ) ;
  ```

Vectors

Vectors

- Vectors are like arrays that can change size (i.e., dynamic array, array list)

- To declare an empty vector with base type int:

  ```
  vector<int> v;
  ```

  - <int> identifies vector as a template class

- You can use any base type in a template class:

  ```
  vector<string> v;
  ```

Member functions

- **push_back()**
  - Elements are added to the next available position of a vector
  - Example:

```
vector<int> sample;
sample.push_back(10);

sample.push_back(20);

sample.push_back(30);
```

10	20	30

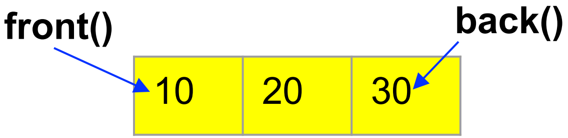
Member functions

- **front()**
  - returns a reference to the first element
- **back()**
  - returns a reference to the last element

```
std::cout<<sample.front()<<std::endl;
std::cout<<sample.back()<<std::endl;
```

Output:

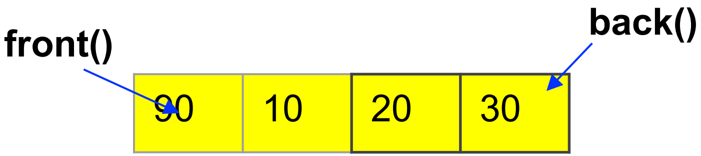
```
10
30
```



Member functions

- **insert()**
  - Elements are added before a specific position of the vector
    - sample.insert(sample.begin(), 90);   //add 90 at the beginning of a vector

iterator



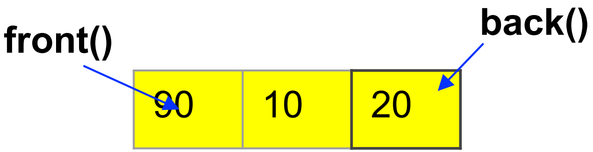
front()

back()

90	10	20	30

Member functions

- **pop_back()**
  - The last element is deleted



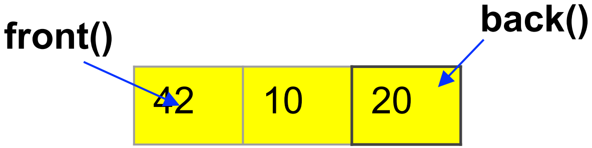
More functions: https://www.cplusplus.com/reference/vector/vector/

Accessing vector Elements

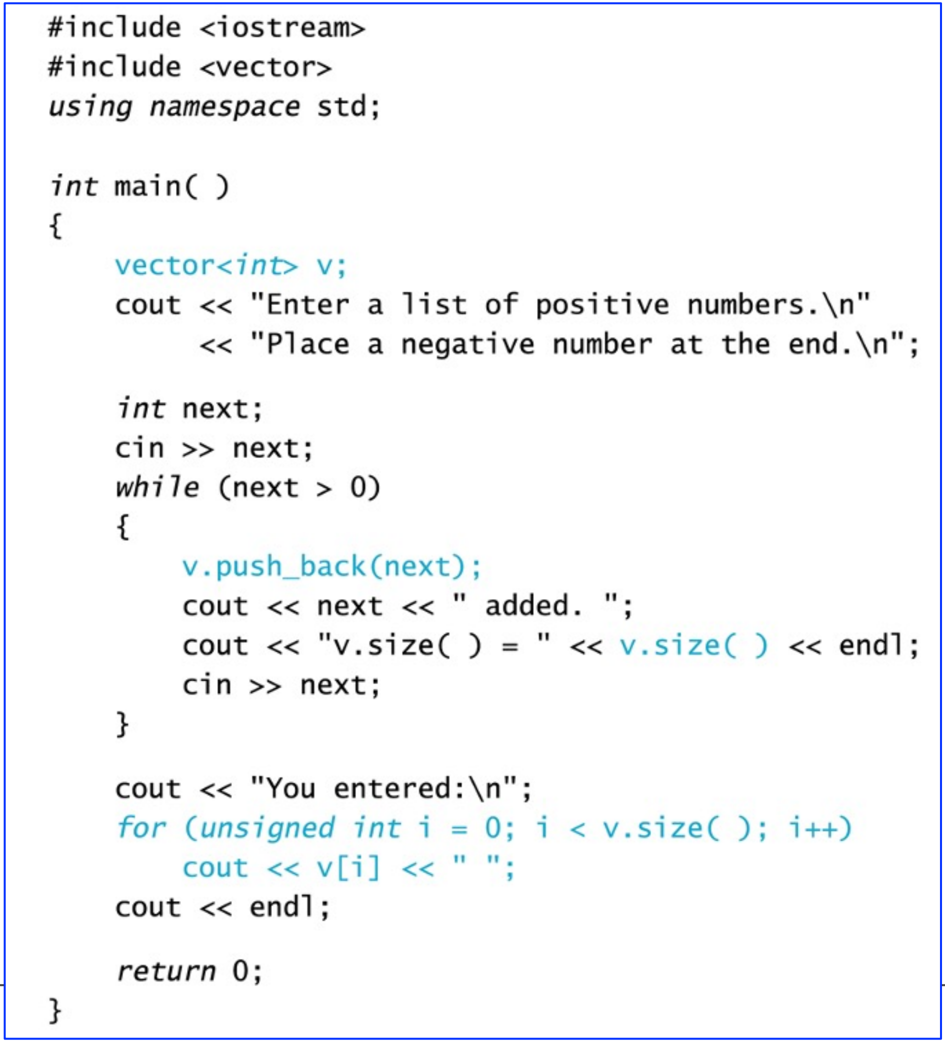
- **Vectors elements are indexed starting with 0**
  - [ ]'s are used to read or change the value of an item:

```
sample[0] = 42;

cout << sample[1];
```



Example



```
#include <iostream>
#include <vector>
using namespace std;

int main( )
{
    vector<int> v;
    cout << "Enter a list of positive numbers.\n"
         << "Place a negative number at the end.\n";

    int next;
    cin >> next;
    while (next > 0)
    {
        v.push_back(next);
        cout << next << " added. ";
        cout << "v.size( ) = " << v.size( ) << endl;
        cin >> next;
    }

    cout << "You entered:\n";
    for (unsigned int i = 0; i < v.size( ); i++)
        cout << v[i] << " ";
    cout << endl;

    return 0;
}
```

Iterator and Iteration

- **front()**
  - returns a reference to the first element
- **back()**
  - returns a reference to the last element
- **begin()**
  - returns an iterator pointing to the first element of the vector
- **end()**
  - Returns an iterator pointing to the *past-the-end* element of the vector
    - *past-the-end* element: element that follow the last element
    - this does not point to an element in a vector (dereferencing does not work)

Iterator and Iteration

- **Iterators**
  - Objects that act like pointers
    - Increment, decrement, dereference
  - Use to cycle through vector(container)'s elements
  - Declare using iterator type defined by container
    - Example: vector<int>::iterator iter = …..;
- **Iteration with iterator**

```
    for (vector<int>::iterator it = sample.begin(); it != sample.end(); it++)
            cout<< *it << endl;
```

OR

```
    for (auto it = sample.begin(); it != sample.end(); it++)
            cout<< *it << endl;
```

OR

```
    for (auto& it : sample)
            cout<< it << endl;
```

The size Of A vector

- **The member function size() returns the number of elements in a vector**
  - **Example:**

    To print each element of a vector given the previous vector initialization:

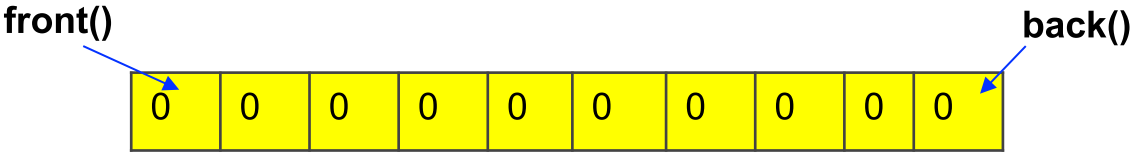
```
for (int i= 0; i < sample.size( ); i++)
            cout << sample[i] << endl;
```

Alternate vector Initialization

- **A vector constructor exists that takes an integer argument and initializes that number of elements**
  - Example:

    ```
    vector<int> v(10);
    ```

    - initializes the first 10 elements to 0, v.size( ) would return 10
    - [ ]'s can now be used to assign elements 0 through 9
    - push_back() is used to assign elements greater than 9



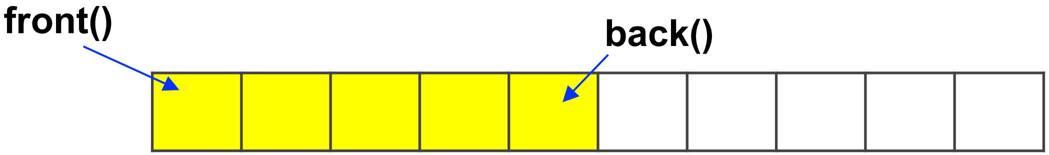
front()

back()

0	0	0	0	0	0	0	0	0	0

vector Efficiency

- **capacity( ) is the number of elements allocated in memory**
  - member function returns the capacity of the vector
- **size() is the number of elements initialized**
- **When a vector runs out of space, the capacity is automatically increased**
  - A common scheme is to double the size of a vector → Consume runtime of program



Capacity: 10
Size: 5

Controlling vector Capacity

- **reserve()**
  - increase the capacity of a vector
    - **Example**:

```
v.reserve(32); // at least 32 elements
 v.reserve(v.size( ) + 10);  // at least 10 more
```

https://en.cppreference.com/w/cpp/container/vector/reserve

- **resize()**
  - increase or decrease the size of a vector
  - If the new size is smaller than current size, then extra elements are destroyed
    - **Example**:

```
v.resize(24);     //at least 24 elements
v.resize(24, 0);     //at least 24 elements, new elements are initialized with 0
```

Increasing size is a very cheap operation, but increasing capacity is expensive