Programming in C++
Zyante

Chapter 9: Streams

String streams

File input/output

Command-line Arguments and Files

M. Yau, CHC

STRING STREAMS

- What stream data type can be created to be associated with a string rather than with the keyboard (standard input)?
- What library must be included?

Figure 9.4.1: Reading a string as an input stream.

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

int main() {
   string  myString = "Amy Smith 19";
   istringstream inSS(myString);
   string  fname, lname;
   int age;

   inSS >> fname;
   inSS >> lname;
   inSS >> age;

   cout << "First name: " << fname << endl;
   cout << "Last  name: " << lname << endl;
   cout << "Age: "        << age << endl;

   return 0;
}
```

Initialize buffer to myString

```
First name: Amy
Last  name: Smith
Age: 19
```

Figure 9.4.2: Using a string stream to process a line of input text.

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main() {
   istringstream inSS;    // Input string stream
   string  lineString;    // Holds line of text
   string  fname, lname;
   int     age = 0;
   bool    done = false;


   cout << "Enter \"firstname lastname age\" on each line\n";
   cout << "(\"Exit\" as firstname exits).\n" << endl;

   done = false;
```

```
   while (!done) {
      getline(cin, lineString); // Entire line into lineString

      inSS.clear();
      inSS.str(lineString);     // Copies to inSS's string buffer

      // Now process the line
      inSS >> fname;

      if (fname == "Exit") {
         cout << "   Exiting." << endl;

         done = true;
      }
      else {
         inSS >> lname;
         inSS >> age;

         cout << "   First name: " << fname << "\n";
         cout << "   Last  name: " << lname << "\n";
         cout << "   Age:        " << age   << "\n";
         cout << endl;
      }
   }

   return 0;
}
```

```
Enter "firstname lastname age" on each line
("Exit" as firstname exits).

Mary Jones 22
   First name: Mary
   Last  name: Jones
   Age:        22

Mike Smith 24
   First name: Mike
   Last  name: Smith
   Age:        24

Exit
   Exiting.
```

?

1. Define an istringstream variable named inSS that creates an input string stream using the String variable myString.
2. Write a statement that initializes the buffer of the input string stream called inSS to a different string called myString2.

?

What is the output string stream data type for writing to a string?

Figure 9.4.3: Output string stream example.

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main() {
   ostringstream fullnameOSS;
   ostringstream ageOSS;
   string  fname, lname, fullname, agestr;
   int age=0;

   cout << "\nEnter \"firstname lastname age\": \n   ";
   cin >> fname;
   cin >> lname;
   cin >> age;

   fullnameOSS << lname << ", " << fname; // Writes to buffer
   fullname = fullnameOSS.str(); // Copies from buffer

   cout << "\n   Full name: " << fullname << endl;

   ageOSS << age; // Writes int age as chars to buffer

   if (age < 21) {
      ageOSS << " (minor)";
   }

   agestr = ageOSS.str();
   cout << "   Age: " << agestr << endl;

   return 0;
}
```

```
Enter "firstname lastname age":
   Mary Jones 22

   Full name: Jones, Mary
   Age: 22

...

Enter "firstname lastname age":
   Sally Smith 14

   Full name: Smith, Sally
   Age: 14 (minor)
```

copies buffer to a string variable

?

1. Given an ostringstream variable called outSS, write a statement that writes the value of the double variable mpg to the stream's buffer.
2. Write a statement that copies the contents of an output string stream to a string variable called myStr. Assume the ostringstream variable is called outSS.

FILE INPUT/OUTPUT

File I/O

1. Open file
2. if file is available, perform read / write
3. close the file

Figure 9.5.1: Input from a file.

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

int main() {
   ifstream inFS;
   int num1, num2;
   cout << "\nOpening file myfile.txt." << endl;

   inFS.open("myfile.txt"); // Try to open file
   if (!inFS.is_open()) {
      cout << "Could not open file myfile.txt.\n";
      return 1; // 1 indicates error
   }
   // Can now use inFS stream like cin stream
   // myfile.txt should contain two integers, else problems

   cout << "Reading two integers." << endl;
   inFS >> num1;
   inFS >> num2;
   cout << "Closing file myfile.txt.\n" << endl;
   inFS.close(); // Done with file, so close it

   cout << "num1: " << num1 << endl;
   cout << "num2: " << num2 << endl;
   cout << "num1+num2: " << (num1 + num2) << endl;

   return 0;
}
```

enables use of the file stream class

input file stream variable

checks if file is opened successfully

```
Opening file myfile.txt.
Reading two integers.
Closing file myfile.txt.

num1: 5
num2: 10
num1+num2: 15
```

Figure 9.5.3: Using c_str() to convert a C++ string to a C string before passing the string to the file open function.

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

int main() {
   ifstream inFS;
   string   filename;

   cin >> filename;
   inFS.open(filename.c_str());

   // rest of program...
}
```

```
int main() {
   ifstream inFS;
   string   filename = "myfile.txt";

   inFS.open(filename);

   // rest of program...
}
```

Figure 9.5.4: Program that reads data from myfile.txt into a vector.

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

int main() {
   ifstream inFS;
   vector<int> nums; // User numbers; memory allocated later
   int N=0; // User-specified number of numbers
   int i=0; // Loop index

   inFS.open("myfile.txt"); //*** Try to open file

   if (!inFS.is_open()) {    //***
      cout << "Could not open file myfile.txt.\n";

      return 1; // 1 indicates error
   }
```

```
// Can now use inFS stream just like cin stream
inFS >> N; // Get number of numbers (must be first item)
nums.resize(N); // Allocate enough memory for nums

// Get N numbers. If too few, may encounter problems
i = 1;
while (i <= N) {
   inFS >> nums.at(i-1);

   i = i + 1;
}

inFS.close();            //*** Done with file, so close it
```

```
    // Print numbers
    cout << "\nNumbers: ";

    i = 0;
    while (i < N) {
        cout << nums.at(i) << " ";

        ++i;
    }

    cout << endl;

    return 0;
}
```

```
Numbers: 10 20 40 80 1
```

?

What function returns true if the previous stream operation reached the end of file?

(It is used with a loop to read varying amounts of data until the end of file has been reached)

Figure 9.5.5: Reading a varying amount of data from a file.

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

int main() {
   ifstream inFS;
   int num;

   cout << "\nOpening file myfile.txt." << endl;

   inFS.open("myfile.txt");

   if (!inFS.is_open()) {
      cout << "Could not open file myfile.txt.\n";

      return 1;
   }

   // Print read numbers to output
   cout << "Reading and printing numbers." << endl;

   inFS >> num;
   while (!inFS.eof()) {
     cout << "num: " << num << endl;

     inFS >> num;
   }

   cout << "Closing file myfile.txt.\n" << endl;

   inFS.close(); // Done with file, so close it

   return 0;
}
```

```
Opening file myfile.txt.
Reading and printing numbers.
num: 111
num: 222
num: 333
num: 444
num: 555
Closing file myfile.txt.
```

?

What stream variable type do we use for writing output to a file?

Figure 9.5.6: Sample code for writing to a file.

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

int main() {
   ofstream outFS;

   outFS.open("myoutfile.txt");

   if (!outFS.is_open()) {
      cout << "Could not open file myoutfile.txt.\n";
      return 1;
   }

   outFS << "Hello" << endl;
   outFS << "1 2 3" << endl;

   // Rest of program

   outFS.close(); // Done with file, so close it

   return 0;
}
```

?

- What is the error in the following code?

```
ifstream inFS;
vector<int> v;
int N=0;
int i=0;
inFS >> N;
v.resize(N);
```

?

- Given:

```
ofstream outFS;
outFS.open("outfile.txt");
```

Write a statement that writes the string “oranges" to file outfile.txt.

?

Given:

```
string myfile = "outfile.txt";
ofstream outFS;
```

Write a statement that opens the file “outfile.txt”.

COMMAND-LINE ARGUMENTS AND FILES

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

int main(int argc, char* argv[]) {
   ifstream inFS;
   int num1, num2;

   if (argc != 2) {
      cout << "\nUsage: myprog.exe inputFileName" << endl;
      return 1; // 1 indicates error
   }

   cout << "\nOpening file " << argv[1] << "." << endl;

   inFS.open(argv[1]); // Try to open file
   if (!inFS.is_open()) {
      cout << "Could not open file " << argv[1] << ".\n";
      return 1; // 1 indicates error
   }
   // Can now use inFS stream like cin stream
   // myfile.txt should contain two integers, else problems

   cout << "Reading two integers." << endl;
   inFS >> num1;
   inFS >> num2;
   cout << "Closing file myfile.txt.\n" << endl;
   inFS.close(); // Done with file, so close it

   cout << "num1: " << num1 << endl;
   cout << "num2: " << num2 << endl;
   cout << "num1+num2: " << (num1 + num2) << endl;

   return 0;
}
```

Figure 9.6.1: Using command-line arguments to specify the name of an input file.

Review

- Arrays (size, adding, printing)
- Vectors (size, adding, removing, printing)
- User defined Functions
  - Data types
  - passing by ref and val
  - overloading
- Streams and Files (opening and reading)

Array Basics

- Named constants are commonly used as size declarators.

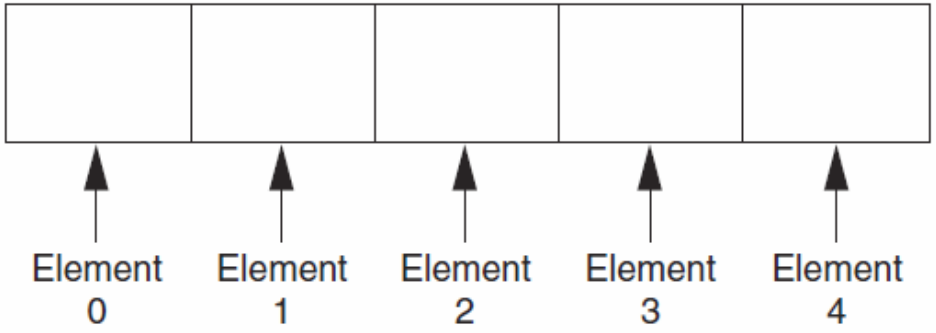
```
const int SIZE = 5;
int tests[SIZE];
```

- This eases program maintenance when the size of the array needs to be changed.

Array Basics

- The index / subscript always starts at 0

```
const int SIZE = 5;
int numbers[SIZE];
```



Array Basics

- Arrays can be initialized with an <u>initialization list</u>:

```
const int SIZE = 5;
int tests[SIZE] = {79,82,91,77,84};
```

- The values are stored in the array in the order in which they appear in the list.
- The initialization list cannot exceed the array size.
- No bounds checking

Array Basics

- Using a loop to step through an array

```
// Create an array to hold three integers.
const int SIZE = 3;
int myValues[SIZE];

// Assign 99 to each array element.
for (int index = 0; index < SIZE; index++)
{
    myValues[index] = 99;
}
```

Vector Basics

- A ***vector*** is an ordered list of items (called ***elements***) of a given data type.
- To use a vector, a program must first include the following: #include <vector>.

```
vector<dataType> name(size);
```

- A vector's elements may be initialized in the variable definition.
- The definition

```
vector<int> myVector(3, -1);
```

creates myVector with three elements each with value -1, i.e., -1, -1, -1.

Accessing vector elments using .at(i)

Figure 5.2.1: Vector's i^{th} element can be directly accessed using ".at(i)": Oldest people program.

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

int main() {
   vector<int> oldestPeople(5); // Source: Wikipedia.org
   int n=1; // User input, Nth oldest person

   oldestPeople.at(0) = 122; // Died 1997 in France
   oldestPeople.at(1) = 119; // Died 1999 in U.S.
   oldestPeople.at(2) = 117; // Died 1993 in U.S.
   oldestPeople.at(3) = 117; // Died 1998 in Canada
   oldestPeople.at(4) = 116; // Died 2006 in Ecuador

   cout << "Enter N (1-5): ";
   cin >> n;

   if ((n >= 1) && (n <= 5)) {
      cout << "The " << n << "th oldest person lived ";
      cout << oldestPeople.at(n-1) << " years.\n\n";
   }

   return 0;
}
```

```
Enter N (1-5): 1
The 1th oldest person lived 122 years.
...
Enter N (1-5): 4
The 4th oldest person lived 117 years.
...
Enter N (1-5): 9
...
Enter N (1-5): 0
...
Enter N (1-5): 5
The 5th oldest person lived 116 years.
```

Iterating

Figure 5.4.3: Iterating through a vector example: Program that finds the sum of a vector's eleme

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

int main() {
   const int N=8;
   vector<int> nums(N); // User numbers
   int i=0; // Loop index
   int sum=0; // For computing sum
   cout << "\nEnter " << N << " numbers...\n";
   for (i = 0; i < N; ++i) {
      cout << i+1 << ": ";
      cin >> nums.at(i);
   }
   // Determine sum
   sum = 0;
   for (i = 0; i < N; ++i) {
      sum = sum + nums.at(i);
   }
   cout << "sum: " << sum << "\n";

   return 0;
}
```

```
Enter 8 numbers...
1: 3
2: 5
3: 234
4: 346
5: 234
6: 73
7: 26
8: -1
sum: 920
...
Enter 8 numbers...
1: 3
2: 5
3: 234
4: 346
5: 234
6: 73
7: 26
8: 1
sum: 922
```

- ***vctr.push_back(x)*** creates a new element at the end of vector vctr and assigns the value of x to that element.
- push_back() thus increases a vector's size by 1.

Two related functions are:

- ***vctr.back()*** -- returns the value of vctr's last element, leaving the element in the vector.
  - Given vctr has elements 8, 9, 12, vctr.back() returns 12.
- ***vctr.pop_back()*** -- removes vctr's last element, thus decreasing vctr's size by 1. vctr.pop_back() returns nothing (void).
  - Given vctr has elements 8, 9, 12, vctr.pop_back() changes vctr to 8, 9, reducing vctr's size to 2.

Functions Basics

- A function definition includes:
  - Function header
    - return type: The data type of the value that function returns to the part of the program that called it. If the word `void` is used, the function does not return a value.
    - name: name of the function. Function names follow same rules as variables
    - parameter list: variables containing values passed to the function
  - Function body: statements that perform the function's task, enclosed in `{}`

Return type

Function name

Parentheses

No semicolon here!

```
void displayMessage()
{
    cout << "This is the displayMessage function." << endl;
}
```

- A **local variable** is declared inside a function and cannot be accessed by statements that are outside the function. Remember scopes!

Animation 6.1.1: A function example.

Start

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

void PrintFace() {
   char c = 'o', nose = '>';
   cout << "\n   " << c << "   " << c << "\n"; // Eyes
   cout << "     " << nose << "\n"; // Nose
   cout << "   " << c << c << c << c << c << "\n\n"; // Mouth
   return;
}

int main() {
   PrintFace();
   return 0;
}
```

```
o   o
  >
ooooo
```

- Commonly a programmer wishes to influence a function's behavior by providing input values to the function.
- An **argument** is any piece of data that is passed into a function when the function is called.
- A **parameter** is a variable that receives an argument that is passed into a function.

```
int main()
{
   cout << "The sum of 12 and 45 is:" << endl;
   showSum(12, 45);
   return 0;
}

void showSum(int num1, int num2)
{
   int result = num1 + num2;
   cout << result << endl;
}
```

Passing arguments

- ***pass by value***: a local variable is created for the parameter and the value of the function call's argument is copied to that local variable as an initial value.
- ***pass by reference:*** a local variable is not created, but instead the parameter name refers to the argument's memory location.
- A parameter's ***default value*** is the value used in the absence of an argument in the function call.

Animation 6.11.2: "Pass by reference" parameters allow a function to update arguments.

Start

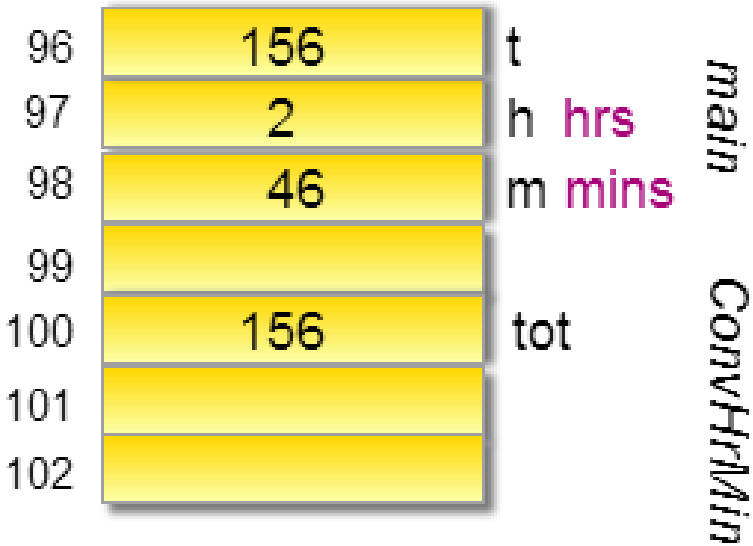
```
#include <iostream>
using namespace std;

void ConvHrMin
      (int tot, int& hrs, int& mins) {
   hrs  = tot / 60;
   mins = tot % 60;
   return;
}

int main() {
   int t=0, h=0, m=0;

   cout << "Enter tot minutes: ";
   cin >> t;
   ConvHrMin(t, h, m);
   cout << "Equals: ";
   cout << h << " hrs ";
   cout << m << " mins\n";

   return 0;
}
```



Succeeds: hrs/min refer to h/m, so h/m get updated.

```
Enter tot minutes:    156
Equals 2 hrs 36 mins.
```

Function Declaration / Prototype

- specifies the function's return type, name, and parameters, ending with a semicolon where the opening brace would have gone
- The function declaration gives the compiler enough information to recognize valid calls to the function. So by placing function declarations at the top of a file, the main function can then appear next, with actual function definitions appearing later in the file.

What does function overloading mean?

Figure 6.15.1: Overloaded function name.

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

void DatePrint(int day, int month, int year) {
   cout << month << "/" << day << "/" << year;
   return;
}

void DatePrint(int day, string month, int year) {
   cout << month << " " << day << ", " << year;
   return;
}

int main() {
   DatePrint(30, 7, 2012); cout << endl;
   DatePrint(30, "July", 2012); cout << endl;
   return 0;
}
```

```
7/30/2012
July 30, 2012
```

- Must use different numbers of formal parameters or have one or more parameter of different types

```
int average(int n1, int n2);

int average(int n1, int n2, int n3);

int average(double n1, double n2);
```

Questions?