Working with Batches of Data

Lecture 4
Hartmut Kaiser
hkaiser@cct.lsu.edu
http://www.cct.lsu.edu/~hkaiser/spring_2015/csc1254.html
Abstract

• So far we looked at simple ‘read a string – print a string’ problems. Now we will look at more complex problems involving multiple pieces of similar data.
Computing Student Grades

- Calculate the overall grade of a student (20% midterm, 40% final exam, 40% homework)
  - Read values interactively
  - Arbitrary number of homework grades
Computing Student Grades

// include directives

// using directives

int main()
{
    // ask for and read the student's name

    // ask and read the midterm and final grades

    // ask for the homework grades

    // write the result

    return 0;
}
#include and using directives

// #include directives
#include <iomanip>    // std::setprecision
#include <ios>        // std::streamsize
#include <iostream>
#include <string>

// using directives
using std::cin;        using std::setprecision;    using std::cout;
using std::string;      using std::endl;           using std::streamsize;

int main()
{
    // ask for and read the student's name
    cout << "Please enter your first name: ";
    string name;
    cin >> name;
    cout << "Hello, " << name << "!" << endl;

    // ...
    return 0;
}
Ask and Read the Grades

```cpp
int main()
{
    // ...
    // ask and read the midterm and final grades
    cout << "Please enter your midterm and final grades: ";
    double midterm, final;
    cin >> midterm >> final;

    // ask for the homework grades
    cout << "Enter all your homework grades, "
         << "followed by end-of-file: ";

    int count = 0;  // the number and sum of grades read so far
    double sum = 0, x;  // a variable to read the grades into

    // invariant:
    // we have read 'count' number of grades so far
    // 'sum' is the sum of the first 'count' grades
    while (cin >> x) {
        ++count;
        sum += x;
    }

    // process the final grades
    // and perform whatever calculations are needed
    // ...
}
```
Computing Student Grades

• Calculate and output overall grade

//...

// write the result
streamsize prec = cout.precision();
cout << name << " , your final grade is: " << setprecision(3)
   << 0.2 * midterm + 0.4 * final + 0.4 * sum / count
   << setprecision(prec) << endl;

return 0;
}
Testing for end of Input

• We have seen:
  
  ```cpp
  while (cin >> x) { /*...*/ }
  ```

• Generally istreams can be used as conditions:
  
  ```cpp
  if (cin >> x) { /*...*/ }
  ```

• Which is equivalent to:
  
  ```cpp
  cin >> x;
  if (cin) { /*...*/ }
  ```

• Detects:
  
  • Operation reached end of input
  • Input characters are not compatible with expected type
  • System has detected hardware failure
Calculating the Median Value

• So far we throw away the values right after having read them
  • Fine for averages, doesn’t work for medians

• Median:
  • Sort all values and pick the middle one (or average of two values nearest the middle)
  • We must store all numbers now!
Data for Iteration - Vector

• To do just about anything of interest, we need a collection of data to
work on. We can store this data in a vector. For example:

```cpp
// read some grades into a std::vector:

int main()
{
    std::vector<double> grades; // declare a vector of type double to
    // store grades - like 62.4
    double x; // a single grade value
    while (std::cin >> x) // cin reads a value and stores it in x
        grades.push_back(x); // store the value of x in the vector

    // ... do something ...
}

// std::cin >> x will return true until we reach the end of
// file or encounter something that isn’t a double: like the word “end”
```
Data for Iteration - Vector

- Vector is a container holding a collection of values
- All values in the same vector have same type, but different vectors can hold objects of different types
- The type of the objects held by a vector needs to be specified: `vector<double>"
Vector holds Sequences of Data

- Vector is the most useful standard library data type
  - A vector<T> holds a sequence of values of type T
  - Think of a vector this way:
    - A vector named v contains 5 elements: {1, 4, 2, 3, 5}:
Vector Interface

template <class T>
class vector { // read "for all types T" (just like in math)
    // ... Internal members
    public:
        vector();                      // default constructor
        explicit vector(int s);      // constructor
        vector(const vector&);       // copy constructor
        vector& operator=(const vector&);  // copy assignment
        ~vector();                   // destructor

        T& operator[](int n);         // access: return reference
        int size() const;            // the current size
        bool empty() const;          // test, whether vector is empty

        void push_back(T d);         // add element at the end
    // ... more functions here
};
Vector grows if needed

```cpp
std::vector<int> v; // start off empty
v.push_back(1);    // add an element with the value 1
v.push_back(4);    // add an element with the value 4 at end ("back")
v.push_back(3);    // add an element with the value 3 at end ("back")
```

![Diagram of vector operations]
Vectors

- Once you get your data into a vector you can easily manipulate it:

```cpp
// compute median grade:
int main()
{
    std::vector<double> grades;  // grades, e.g. 64.6
    // invariant: grades contains all the homework grades read so far
    double x;
    while (std::cin >> x) // read next value and put into vector
        grades.push_back(x); // maintains loop invariant!

    // sort values stored in the vector
    std::sort(grades.begin(), grades.end());
    std::cout << "Median grade: " << grades[grades.size()/2] << std::endl;
}
```
Calculating Median properly

- If number of elements is odd, pick middle:

\[
\text{auto mid} = \text{grades.size()} / 2;
\]
\[
\text{median} = \text{grades[mid]};
\]
Calculating Median properly

• If number is even, pick average of 2 values nearest the middle:

```cpp
auto mid = grades.size() / 2;
median = (grades[mid] + grades[mid-1]) / 2;
```
Calculating Median properly (C++11)

// calculating the median grade
auto size = grades.size();
auto mid = size / 2;
double median = size % 2 == 0 ? (grades[mid] + grades[mid-1]) / 2
  : grades[mid];

// fully equivalent to
double median;
if (size % 2 == 0)
    median = (grades[mid] + grades[mid-1]) / 2;
else
    median = grades[mid];
Combining Language Features

• You can write many new programs by combining language features, built-in types, and user-defined types in new and interesting ways.
  • So far, we have
    • Variables and literals of types bool, char, int, double
    • string, vector, find(), [ ] (subscripting)
    • !=, ==, =, +, -, +=, <, &&, | |, !
    • transform(), sort(), std::cin >, std::cout <<
    • if, for, while
  • You can write a lot of different programs with these language features! Let’s try to use them in a slightly different way...
Example – Word List (C++11)

// "boilerplate" left out

std::vector<std::string> words;
std::string s;

while (std::cin >> s && s != "quit") // && means AND
    words.push_back(s);

std::sort(words.begin(), words.end()); // sort the words we read

std::for_each(
    words.begin(), words.end(), // print all words
    [](std::string s) {
        std::cout << s << "\n";
    });

// read a bunch of strings into a vector of strings, sort
// them into lexicographical order (alphabetical order),
// and print the strings from the vector to see what we have.
Word list – Eliminate Duplicates

// Note that duplicate words were printed multiple times. For
// example "the the the". That's tedious, let's eliminate duplicates:

std::vector<std::string> words;
std::string s;
while (std::cin >> s && s != "quit")
    words.push_back(s);

std::sort(words.begin(), words.end());

for (auto i = 1; i != words.size(); ++i)
    if (words[i-1] == words[i])
        // get rid of words[i]  // (pseudocode)

std::for_each(
    words.begin(), words.end(), // print all words
    [](std::string s) {
        std::cout << s << "\n";
    });

// there are many ways to “get rid of words[i]”; many of them are messy
// (that’s typical). Our job as programmers is to choose a simple clean
// solution – given constraints – time, run-time, memory.
Example (cont.) Eliminate Words!

// Eliminate the duplicate words by copying only unique words:
std::vector<std::string> words;
string s;

while (std::cin >> s && s != "quit")
    words.push_back(s);
std::sort(words.begin(), words.end());

vector<string> w2;
if (0 != words.size()) {   // Note style { }
    w2.push_back(words[0]);
    for (auto i = 1; i != words.size(); ++i)
        if (words[i-1] != words[i])
            w2.push_back(words[i]);
}

std::cout << "found " << words.size()-w2.size() << " duplicates\n";
std::for_each(    // print all remaining words
    w2.begin(), w2.end(), [](std::string s) {
        std::cout << s << "\n";
    });
C++11 Loop Alternatives

- The common way of iterating over all elements of a vector:
  ```cpp
  for (vector<string>::size_type i = 0; i != words.size(); ++i)
  cout << words[i] << "\n";
  ```

- C++11 gives simpler alternatives:
  - Using auto for loop variable (requires g++ 4.4, vs2010)
    ```cpp
    for (auto i = 0; i != words.size(); ++i)
    cout << words[i] << "\n";
    ```
  - Range based for loop (requires g++ 4.6, vs2012)
    ```cpp
    for (string w: words)
    cout << w << "\n";
    ```
  - Using lambda functions (requires g++ 4.4, vs2010)
    ```cpp
    std::for_each(words.begin(), words.end(),
    [](string s) {
    cout << s << "\n";
    });
    ```
Algorithm

• We just used a simple algorithm

• An algorithm is (from Google search)
  • “a logical arithmetical or computational procedure that, if correctly applied, ensures the solution of a problem.” – Harper Collins
  • “a set of rules for solving a problem in a finite number of steps, as for finding the greatest common divisor.” – Random House
  • “a detailed sequence of actions to perform or accomplish some task. Named after an Iranian mathematician, Al-Khawarizmi. Technically, an algorithm must reach a result after a finite number of steps, ...The term is also used loosely for any sequence of actions (which may or may not terminate).” – Webster’s

• We eliminated the duplicates by first sorting the vector (so that duplicates are adjacent), and then copying only strings that differ from their predecessor into another vector.
Ideal

• Basic language features and libraries should be usable in essentially arbitrary combinations.
  • We are not too far from that ideal.
  • If a combination of features and types make sense, it will probably work.
    • The compiler helps by rejecting some absurdities.