Taking Strings Apart
(and putting them together)
Lecture 11
Hartmut Kaiser
hkaiser@cct.lsu.edu
http://www.cct.lsu.edu/~hkaiser/spring_2015/csc1254.html
Programming Principle of the Day

• Write Code for the Maintainer - Almost any code that is worth writing is worth maintaining in the future, either by you or by someone else. The future you who has to maintain code often remembers as much of the code, as a complete stranger, so you might as well always write for someone else. A memorable way to remember this is “Always code as if the person who ends up maintaining your code is a violent psychopath who knows where you live.”

Abstract

- This lecture will go back to our initial example of framing a string. So far, we have dealt with strings as a whole only. In this lecture we will look at the characters of a string separately: strings are apparently special containers. Many techniques we know from vectors are applicable to strings as well.
Splitting a Line into Words

• We’ll write a function which takes a whole line of input and returns a vector of strings holding the single words of that line:

    `vector<string> split(string const& s);`

• Strings support indexing in the same way as vectors:
  • `s[0]`: refers to the first character in the string ‘s’
  • `s[s.size()-1]`: refers to the last character in a string

• Our function will find indices ‘i’ and ‘j’ delimiting each of the words, where the range of characters `[i, j)` constitutes the word
Splitting a Line into Words

- This looks like:

```
  Some text read into a line
```

- Words are split at whitespace characters
  - Very similar to the processing during stream input into a string
Splitting a Line into Words

```cpp
vector<string> split(const string& s)
{
    vector<string> words;
    vector<string>::size_type i = 0;

    // invariant: we have processed characters [original value of i, i)
    while (i != s.size()) {
        // ignore leading blanks, find begin of word

        // find end of next word

        // if we found some non-whitespace characters, store the word
        return words;
    }
}
```
Splitting a Line into Words

• Ignore leading blanks
  // invariant: characters in range [original i, current i)
  // are all spaces
  while (i != s.size() && isspace(s[i]))  // short-circuiting
    ++i;

• Find end of next word
  // find end of next word
  auto j = i;

  // invariant: none of the characters in range
  // [original j, current j) is a space
  while (j != s.size() && !isspace(s[j]))  // short-circuiting
    ++j;
Splitting a Line into Words

- Store the word if any

```cpp
// if we found some non-whitespace characters
if (i != j) {
    // copy from s starting at i and having j - i characters
    words.push_back(s.substr(i, j - i));
    i = j;
}
```
Test our new Function

• Read a line of input and invoke our function
• Write content of returned vector
• Results should be the same as when using an input stream to read strings
  • Remember, this separates words as well
• Run both tests on same input data and compare results
Test our new Function

- Simple test program (C++11, gcc 4.6):

```cpp
int main() {
    string in;
    // read and split each line of input
    while (getline(cin, in)) {
        vector<string> v = split(in);
        // write each word in v
        for (auto const& s: v)
            cout << s << endl;
    }
    return 0;
}
```
Test our new Function

• Test word splitting using input streams:

```cpp
int main()
{
    string s;
    while (cin >> s)
    {
        cout << s << endl;
    }
    return 0;
}
```
int main()
{
    string input;
    while (getline(cin, input)) {
        stringstream sstrm = input;

        vector<string> v;
        string str;
        while (sstrm >> str)
            v.push_back(str);

        for_each(v.begin(), v.end(), [](string const& s) {
            cout << s << endl;
        });
    }
    return 0;
}
Putting Strings Together

• Earlier we wrote a program to frame a string
  • Never created the output in a string
  • Rather printed the parts separately

• Let’s assume vector<string> is a ‘picture’ (ASCII art), each string is one line

• Now, we will build a program framing such a picture
  • Puts together the whole picture in a data structure before printing it
Putting Strings Together

• Write all strings in a vector<string> on one line each and surround it by a border

• For example, a vector<string> holding “this is an”, “example”, “to illustrate”, “framing” will result in:

```
*************************
* this is an  *
* example      *
* to illustrate *
* framing      *
*************************
```
Putting Strings Together

- Box is rectangular not ragged as single strings
- Find the longest string

```cpp
string::size_type width(vector<string> const& v) {
  string::size_type maxlen = 0;
  for (auto const& s: v)
    maxlen = std::max(maxlen, s.size());
  return maxlen;
}
```

- Exercise: use standard algorithm (accumulate)
Framing Pictures

• What should the interface look like?
  • Let’s consider vector<string> to represent a ‘picture’
  • Function frame() will take a ‘picture’ and should return a new ‘picture’:

```c++
vector<string> frame(vector<string> const& v)
{
    // find longest string
    // create top line and append to result
    // append each line from v to result after adding '*'
    // create bottom line and append to result
}
```
Framing Pictures

```cpp
vector<string> frame(vector<string> const& v)
{
    vector<string> ret;
    auto maxlen = width(v); // find longest string

    // create top line and append to result
    string border(maxlen + 4, '*');
    ret.push_back(border);

    // append each line from v to result after adding '*'
    for (auto const& s : v)
    {
        ret.push_back("* " + s + string(maxlen - s.size(), ' ') + " *");
    }

    ret.push_back(border); // write the bottom border
    return ret;
}
```
Vertical Concatenation

• What else can we do to ‘pictures’?
  • Concatenation! Vertical and horizontal

• ‘Pictures’ are represented by vector<string>
  • Vertical concatenation is simple: just concatenate the two vectors
  • ‘Pictures’ will line up along their left margin
  • No predefined concatenation of vectors, let’s define one:

    ```cpp
    vector<string> vcat(
        vector<string> const& top,
        vector<string> const& bottom);
    ```
Vertical Concatenation

```cpp
vector<string> vcat(
    vector<string> const& top,
    vector<string> const& bottom)
{
    // copy the top picture
    vector<string> ret = top;

    // copy entire bottom picture
    for (auto const& s: bottom)
    {
        ret.push_back(s);
    }

    return ret;
}
```
Vertical Concatenation

```cpp
vector<string> vcat(
    vector<string> const& top,
    vector<string> const& bottom)
{
    // copy the top picture
    vector<string> ret = top;

    // copy entire bottom picture, use iterators
    for (auto it = bottom.begin(); it != bottom.end(); ++it)
    {
        ret.push_back(*it);
    }

    return ret;
}
```
Vertical Concatenation

```cpp
vector<string> vcat(
    vector<string> const& top,
    vector<string> const& bottom)
{
    // copy the top picture
    vector<string> ret = top;

    // copy entire bottom picture, use vector facilities
    ret.insert(ret.end(), bottom.begin(), bottom.end());

    return ret;
}
```
Horizontal Concatenation

• For example:

```
this is an **************
example   * this is an *
to        * example   *
illustrate * to        *
framing   * illustrate *
        * framing   *
**************
```
Horizontal Concatenation

- If left hand side picture is shorter than right hand side, we need padding
- Otherwise just copy the picture
- Interface similar to vcat:

```cpp
vector<string> hcat(  
    vector<string> const& left, vector<string> const& right)  
{
    // get width of left, add 1 to leave a space between pictures
    // handle all rows from both pictures, line by line
    // copy a row from the left-hand side, if there is one
    // pad the line to full width of left + 1
    // copy row from the right-hand side, if there is one
    // append overall line to result vector

    }
```
Horizontal Concatenation

```cpp
vector<string> hcat(vector<string> const& left, vector<string> const& right)
{
    vector<string> ret;
    // add 1 to leave a space between pictures
    auto width1 = width(left) + 1;
    auto i = 0, j = 0;

    // continue until we've seen all rows from both pictures
    while (i != left.size() || j != right.size()) {
        string s; // construct new string to hold characters from both pictures
        // copy a row from the left-hand side, if there is one
        if (i != left.size())
            s = left[i++];
        s += string(width1 - s.size(), ' '); // pad to full width

        // copy a row from the right-hand side, if there is one
        if (j != right.size())
            s += right[j++];
        ret.push_back(s); // add s to the picture
    }
    return ret;
}
```
Vertically Flip a Picture

• For example:

this is an  flipping
to  to
illustrate  example
flipping  this is an
Vertically Flip a Picture

- **Straight forward solution:**

  ```cpp
  vector<string> vflip(vector<string> const& v)
  {
    vector<string> ret;
    for (auto it = v.rbegin(); it != v.rend(); ++it)
      ret.push_back(*it);
    return ret;
  }
  ```

- **Using Standard algorithm**

  ```cpp
  vector<string> vflip(vector<string> const& v)
  {
    vector<string> ret;
    reverse_copy(ret.begin(), ret.end(), back_inserter(ret));
    return ret;
  }
  ```
Rotate a Picture

• For example:

  this is an example to illustrate rotation
  n e a t an se ro il ti
  p st sm ua ia lt
  hxolo tetir
Rotate a Picture

```cpp
vector<string> rotate_left(vector<string> const& v)
{
    vector<string> ret;

    // take a letter from each string, starting at the end
    auto maxlen = width(v);
    for (auto i = maxlen; i != 0; --i)
    {
        string line;       // new line
        // for all lines in the input image
        for (auto const& current: v)
        {
            if (current.size() < i)
                line += ' ';  
            else
                line += current[i-1];
        }
        // store the new line in the result picture
        ret.push_back(line);
    }

    return ret;
}
```