CIS 330:

Univ and C/C++

C++ strings, potpourri, & review
Standard Template Library

• Standard Template Library: STL
• Many, many templated types
• Can ease your programming
• Can also hide significant performance issues
  – And you use C/C++ for performance
• My recommendation: use with caution for code that needs to be performant
Vector

#include <vector>

using std::vector;

int main()
{
    vector<int> intArray(2);
    intArray[0] = 0;
    intArray[1] = 1;
    intArray.push_back(1);
    intArray.push_back(2);
    intArray.push_back(3);
    intArray.push_back(5);
    cout << "Size is " << intArray.size() << endl;
    cout << "Last val of Fib is " << intArray[5] << endl;
}

C02LN00GFD58:330 hank$ g++ vector.C
C02LN00GFD58:330 hank$ ./a.out
Size is 6
Last val of Fib is 5
std::vector

- Always has the amount of memory you need
- Many STL algorithms work on it
- Memory allocation:
  - If you don’t have enough, double it
    - (can be a big overestimation)
- Overhead in access
  - Maybe not a big deal if data-intensive?
#include <map>
#include <string>

using std::map;
using std::string;

int main()
{
    map<string, int> ageLookup;
    ageLookup["Hank"] = 37;
    ageLookup["Charlotte"] = 11;
    ageLookup["William"] = 9;

    cout << "Hank's age is " << ageLookup["Hank"] << endl;
    cout << "Carissa's age is " << ageLookup["Carissa"] << endl;
}

C02LN00GFD58:330 hank$ g++ map.C
C02LN00GFD58:330 hank$ ./a.out
Hank's age is 37
Carissa's age is 0
Outline

• Announcements
• Special Topics
• Review
• Potpourri from Lec 17
• Templates
• Standard Template Library
• C++ Strings
• Bonus Slides
(not a template thing): String

- C++ string class is very useful
- Great implementation of a class that encapsulates a string

```cpp
#include <string>

using std::string;

int main()
{
    string str = "Hello";
    str += " world";
    cout << str << endl;
}
```

C02LN00GFD58:330 hank$ g++ string.C
C02LN00GFD58:330 hank$ ./a.out
Hello world
## String methods

### Iterators:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>begin</code></td>
<td>Return iterator to beginning</td>
<td>public member function</td>
</tr>
<tr>
<td><code>end</code></td>
<td>Return iterator to end</td>
<td>public member function</td>
</tr>
<tr>
<td><code>rbegin</code></td>
<td>Return reverse iterator to reverse beginning</td>
<td>public member function</td>
</tr>
<tr>
<td><code>rend</code></td>
<td>Return reverse iterator to reverse end</td>
<td>public member function</td>
</tr>
<tr>
<td><code>cbegin</code></td>
<td>Return const_iterator to beginning</td>
<td>public member function</td>
</tr>
<tr>
<td><code>cend</code></td>
<td>Return const_iterator to end</td>
<td>public member function</td>
</tr>
<tr>
<td><code>crbegin</code></td>
<td>Return const_reverse_iterator to reverse beginning</td>
<td>public member function</td>
</tr>
<tr>
<td><code>crend</code></td>
<td>Return const_reverse_iterator to reverse end</td>
<td>public member function</td>
</tr>
</tbody>
</table>

### Capacity:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size</code></td>
<td>Return length of string</td>
<td>public member function</td>
</tr>
<tr>
<td><code>length</code></td>
<td>Return length of string</td>
<td>public member function</td>
</tr>
<tr>
<td><code>max_size</code></td>
<td>Return maximum size of string</td>
<td>public member function</td>
</tr>
<tr>
<td><code>resize</code></td>
<td>Resize string</td>
<td>public member function</td>
</tr>
<tr>
<td><code>capacity</code></td>
<td>Return size of allocated storage</td>
<td>public member function</td>
</tr>
<tr>
<td><code>reserve</code></td>
<td>Request a change in capacity</td>
<td>public member function</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>Clear string</td>
<td>public member function</td>
</tr>
<tr>
<td><code>empty</code></td>
<td>Test if string is empty</td>
<td>public member function</td>
</tr>
<tr>
<td><code>shrink_to_fit</code></td>
<td>Shrink to fit</td>
<td>public member function</td>
</tr>
</tbody>
</table>
# String methods

## Element access:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operator[]</code></td>
<td>Get character of string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>at</code></td>
<td>Get character in string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>back</code></td>
<td>Access last character</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>front</code></td>
<td>Access first character</td>
<td>(public member function)</td>
</tr>
</tbody>
</table>

## Modifiers:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operator+=</code></td>
<td>Append to string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>append</code></td>
<td>Append to string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>push_back</code></td>
<td>Append character to string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>assign</code></td>
<td>Assign content to string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>Insert into string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>erase</code></td>
<td>Erase characters from string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>Replace portion of string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>swap</code></td>
<td>Swap string values</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>pop_back</code></td>
<td>Delete last character</td>
<td>(public member function)</td>
</tr>
</tbody>
</table>

## String operations:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c_str</code></td>
<td>Get C string equivalent</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>data</code></td>
<td>Get string data</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>get_allocator</code></td>
<td>Get allocator</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>copy</code></td>
<td>Copy sequence of characters from string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>find</code></td>
<td>Find content in string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>rfind</code></td>
<td>Find last occurrence of content in string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>find_first_of</code></td>
<td>Find character in string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>find_last_of</code></td>
<td>Find character in string from the end</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>find_first_not_of</code></td>
<td>Find absence of character in string</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>find_last_not_of</code></td>
<td>Find non-matching character in string from the end</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>substr</code></td>
<td>Generate substring</td>
<td>(public member function)</td>
</tr>
<tr>
<td><code>compare</code></td>
<td>Compare strings</td>
<td>(public member function)</td>
</tr>
</tbody>
</table>
Outline

- Announcements
- Special Topics
- Review
- Potpourri from Lec 17
- Templates
- Standard Template Library
- C++ Strings
- Bonus Slides
Backgrounding

• “&”: tell shell to run a job in the background
  – Background means that the shell acts as normal, but the command you invoke is running at the same time.

• “sleep 60” vs “sleep 60 &”

When would backgrounding be useful?
Suspending Jobs

• You can suspend a job that is running
  Press “Ctrl-Z”
• The OS will then stop job from running and not schedule it to run.
• You can then:
  – make the job run in the background.
    • Type “bg”
  – make the job run in the foreground.
    • Type “fg”
      – like you never suspended it at all!!
printf

• Print to stdout
  – printf(“hello world\n”);
  – printf(“Integers are like this %d\n”, 6);
  – printf(“Two floats: %f, %f”, 3.5, 7.0);

Have you ever wondered how printf takes a variable number of arguments?
Variable-Length Argument Lists

```c
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>

int SumIntList(int X, ...) {
    va_list ap;  /* points to each unnamed arg in turn */
    int sum = 0;
    int ival;
    int i;

    va_start(ap, X); /* make ap point to 1st unnamed arg */
    for (i = 0; i < X; i++)
    {
        ival = va_arg(ap, int);
        sum += ival;
    }
    va_end(ap);

    return sum;
}

int main()
{
    printf("List sum = %d\n", SumIntList(3, 13, 17, 22));
    printf("List sum = %d\n", SumIntList(5, 1, 2, 3, 4, 5));
}
```

Adapted from Kernigan & Ritchie C book
Review
Final

• Some will find the Final trivial, some will find the Final hard
• I will ask questions that test how much you really understand
• Most of you will have ~70 points going into the Final
  – Those who understand very well will get an A
  – Those who understand less will get a B
Questions I will ask

- “What does this print?”
- “Write some code”
- True/false or multiple choice
My recommended list

• (Note: Final has not been written yet)
• Basic Unix commands
• Symbols, gcc, linking, etc.
• Pointers
• General knowledge of how a compiler interprets things
• Memory allocation, memory errors
• Inheritance / virtual functions
• How to read code and look for errors