Strings

CSci 588: Data Structures, Algorithms and Software Design

http://www.cplusplus.com/reference/string/string/

Overview

- Introduction to strings
- Creating strings
- Iterating over strings
- Appending to strings
- Substrings
- Comparing Strings

Introduction to Strings

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring = "This is a string";
  cout << mystring;
  return 0;
}
```

C++ provides a standard `string` library. strings are not fundamental types, but behave very similarly. `strings` are sequences of multiple characters.

To use strings, include the `string` header file and use the std namespace.

Introduction to Strings

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring = "This is a string";
  cout << mystring;
  return 0;
}
```

This program will output:

```
This is a string
```

Introduction to Strings

```
string mystring = "This is a string";
string mystring ("This is a string");
```

Strings are initialized in the same way that variables are initialized. These two statements are equivalent.

Introduction to Strings

```cpp
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is the initial string content";
  cout << mystring << endl;
  mystring = "This is a different string content";
  cout << mystring << endl;
  return 0;
}
```

Just like variables, strings can be declared and assigned later, and then reassigned.

Introduction to Strings

```cpp
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is the initial string content";
  cout << mystring << endl;
  mystring = "This is a different string content";
  cout << mystring << endl;
  return 0;
}
```

This program will output:

```
This is the initial string content
This is a different string content
```

Strings

Strings are extremely similar to vectors (which we'll discuss in depth later), except they are specialized -- every element within a string is a character.

They also provide functions common to dealing with strings.

Creating Strings

Creating Strings

```
// string constructor
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string s0 ("Initial string");

  // constructors used in the same order as described above:
  string s1;                                         // create an empty string
  string s2 (s0);                                    // copy s0 into s2
  string s3 (s0, 8, 3);                              // copy the 8th to 11th characters
  string s4 ("A character sequence", 6);             // copy those first 6 characters
  string s5 ("Another character sequence");          // make a copy of that string
  string s6 (10, 'x');                               // make the string 10 ‘x’s
  string s7a (10, 42);
  string s7b (s0.begin(), s0.begin()+7);

  cout << "s1: " << s1 << "\ns2: " << s2 << "\ns3: " << s3;
  cout << "\ns4: " << s4 << "\ns5: " << s5 << "\ns6: " << s6;
  cout << "\ns7a: " << s7a << "\ns7b: " << s7b << endl;
  return 0;
}
```

Size, Capacity, Max Size Reserving Space for and Resizing Strings

Size, Capacity and max size of a String

```
// comparing size, length, capacity and max_size
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str ("Test string");
  cout << "size: " << str.size() << "\n";
  cout << "length: " << str.length() << "\n";
  cout << "capacity: " << str.capacity() << "\n";
  cout << "max_size: " << str.max_size() << "\n";
  return 0;
}
```

Checking to see if a String is Empty

```
// string::empty
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string content;
  string line;
  cout << "Please introduce a text. Enter an empty line to finish:\n";
  do {
    getline(cin,line);
    content += line + '\n';
  } while (!line.empty());
  cout << "The text you introduced was:\n" << content;
  return 0;
}
```

Resizing a String

```cpp
// resizing string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  size_t sz;
  string str ("I like to code in C");
  cout << str << endl;

  sz=str.size();

  str.resize (sz+2,'+');
  cout << str << endl;

  str.resize (14);
  cout << str << endl;
  return 0;
}
```

Reserving more space for a String

```
// string::reserve
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main ()
{
  string str;
  size_t filesize;

  ifstream file ("test.txt",ios::in|ios::end);
  filesize=file.tellg();

  str.reserve(filesize);

  file.seekg(0);
  while (!file.eof())
  {
    str += file.get();
  }
  cout << str;
  return 0;
}
```

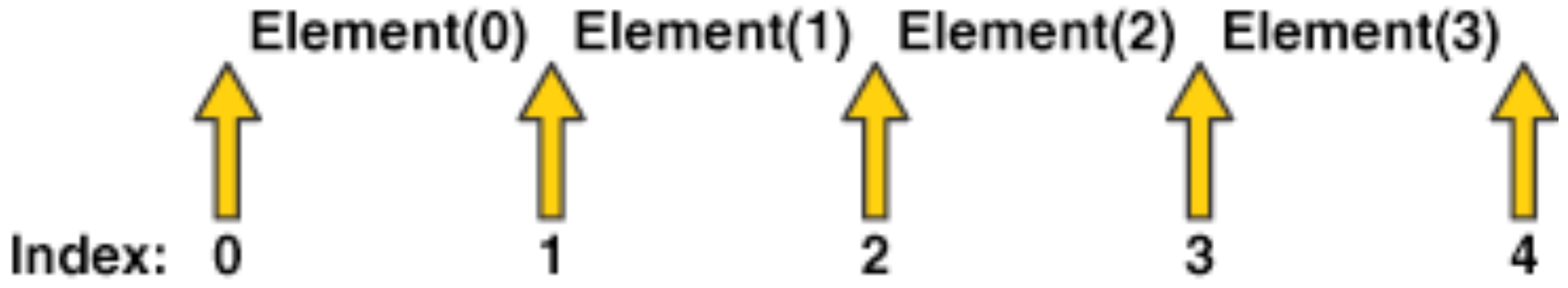
Passing Strings to Functions

Passing Strings to Functions

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

void call_by_value_test(string s) {
    s = “elephant!”;
}
void modify_contents_test(string s) {
    s[12] = “?”;
}
void call_by_reference_test(string &s) {
    s = “hippo”;
}

int main() {
    string s1 = “preposterous!”

    cout << "initial string -- " << s1 << endl;

    call_by_value_test(s1);
    cout << "after call by value test -- " << s1 << endl;

    modify_contents_test(s1);
    cout << "after modify contents test -- " << s1 << endl;

    call_by_reference_test(s1);
    cout << "after call by reference test -- " << s1 << endl;
}
```

Iterating over Strings

Possible Iterator Positions



Iterating over Strings

```cpp
// string::begin and string::end
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str ("Test string");
  string::iterator it;
  for ( it=str.begin() ; it < str.end(); it++ )
    cout << *it;
  return 0;
}
```

reverse iteration over strings

```
// string::rbegin and string::rend
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str ("now step live...");
  string::reverse_iterator rit;
  for ( rit=str.rbegin() ; rit < str.rend(); rit++ )
    cout << *rit;
  return 0;
}
```

inserting at an iterator

```
// inserting into a string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str="to be question";
  string str2="the ";
  string str3="or not to be";
  string::iterator it;

  // used in the same order as described above:
  str.insert(6,str2);                 // to be (the )question
  str.insert(6,str3,3,4);             // to be (not )the question
  str.insert(10,"that is cool",8);    // to be not (that is )the question
  str.insert(10,"to be ");            // to be not (to be )that is the question
  str.insert(15,1,':');               // to be not to be(:) that is the question
  it = str.insert(str.begin()+5,','); // to be(,) not to be: that is the question
  str.insert (str.end(),3,'.');       // to be, not to be: that is the 
question(...)
  str.insert (it+2,str3.begin(),str3.begin()+3); // (or )

  cout << str << endl;
  return 0;
}
```

erasing at an iterator

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

int main ()
{
  string str ("This is an example phrase.");
  string::iterator it;

  // erase used in the same order as described above:
  str.erase (10,8);
  cout << str << endl;          // "This is an phrase."

  it=str.begin()+9;
  str.erase (it);
  cout << str << endl;          // "This is a phrase."

  str.erase (str.begin()+5, str.end()-7);
  cout << str << endl;          // "This phrase."
  return 0;
}
```

string.at(n) vs string[n]

at vs []

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

int main() {
    string s = “something witty”;

    s[16] = ‘!’;    // this may or may not crash; if
                           // it does it will not tell you why
    s.at(16) = ‘!’; // this will crash with an error
                           // saying you tried to access something
                           // outside of the bounds of the vector
}
```

at is much safer than using [], but it is a little slower. In general, use at unless you have fully debugged your code and have a need for as much performance as possible.

Appending to strings

appending characters

```
// string::push_back
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main ()
{
  string str;
  ifstream file ("test.txt",ios::in);
  while (!file.eof())
  {
    str.push_back(file.get());
  }
  cout << str;
  return 0;
}
```

NOTE: strings to **NOT** implement pop_back, and front, unlike vectors.

appending to a string

```
// string::operator+=
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string name ("John");
  string family ("Smith");
  name += " K. ";          // c-string
  name += family;          // string
  name += '\n';            // character

  cout << name;
  return 0;
}
```

appending to a string 2

```
// appending to string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str;
  string str2="Writing ";
  string str3="print 10 and then 5 more";

  // used in the same order as described above:
  str.append(str2);                       // "Writing "
  str.append(str3,6,3);                   // "10 "
  str.append("dots are cool",5);          // "dots "
  str.append("here: ");                   // "here: "
  str.append(10,'.');                     // ".........."
  str.append(str3.begin()+8,str3.end());  // " and then 5 more"
  str.append<int>(5,0x2E);                // "....."

  cout << str << endl;
  return 0;
}
```

Clearing and Assigning Strings

Clearing a String

```
// string::clear
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str;
  char c;
  cout << "Please type some lines of text. Enter a period to finish:\n";
  do {
    c=cin.get();
    str += c;
    if (c=='\n')
    {
       cout << str;
       str.clear();
    }
  } while (c!='.');
  return 0;
}
```

Assigning a String

```
// string::assign
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str;
  string base="The quick brown fox jumps over a lazy dog.";

  // used in the same order as described above:

  str.assign(base);              // assign the base string to str
  cout << str << endl;

  str.assign(base,10,9);         // assign the 10th through 19th characters in base to str
  cout << str << endl;           // "brown fox"

  str.assign("pangrams are cool",7); // assign the first 7 characters of “pangrams are cool” to str
  cout << str << endl;           // "pangram"

  str.assign("c-string");        // assign "c-string" to str
  cout << str << endl;           // "c-string"

  str.assign(10,'*');            // assign ten *s to str
  cout << str << endl;           // "**********"

  str.assign<int>(10,0x2D);      // assign 10 of the int representation of character 0x2D to str
  cout << str << endl;           // "----------"

  str.assign(base.begin()+16,base.end()-12); // assign from the 16th character of base to the
                                             // 12th from last character of base to str
  cout << str << endl;           // "fox jumps over"

  return 0;
}
```

Substrings

Substrings

```
// string::substr
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str="We think in generalities, but we live in details.";
                             // quoting Alfred N. Whitehead
  string str2, str3;
  size_t pos;

  str2 = str.substr (12,12); // "generalities"

  pos = str.find("live");    // position of "live" in str
  str3 = str.substr (pos);   // get from "live" to the end

  cout << str2 << ' ' << str3 << endl;

  return 0;
}
```

Comparing Strings

Comparing Strings

```
// comparing apples with apples
#include <iostream>
#include <string>

int main ()
{
  std::string str1 ("green apple");
  std::string str2 ("red apple");

  if (str1.compare(str2) != 0)
    std::cout << str1 << " is not " << str2 << '\n';

  if (str1.compare(6,5,"apple") == 0)
    std::cout << "still, " << str1 << " is an apple\n";

  if (str2.compare(str2.size()-5,5,"apple") == 0)
    std::cout << "and " << str2 << " is also an apple\n";

  if (str1.compare(6,5,str2,4,5) == 0)
    std::cout << "therefore, both are apples\n";

  return 0;
}
```