Introduction to CGICC

Aleksandr Zaks zaks@cs.nyu.edu
some material is taken from
http://www.cgiicc.org/doc/

New York University, February, 2005
Installation

- Download ftp://ftp.gnu.org/gnu/cgicc/cgicc-3.2.3.tar.gz
- tar -xvzf cgicc-3.2.3.tar.gz
- cd cgicc-3.2.3
- ./configure
- make

If everything is ok, you should have libcgicc.a in cgicc-3.2.3/cgicc/.libs/

Note that at NYU you can find libcgicc.a in /usr/local/lib
(include files are in /usr/local/include/cgicc)
Creating a CGI program

- To link, run `g++ -o test.cgi test.o -lcgicc`
- To install/host a cgi script follow directions from http://cims.nyu.edu/systems/resources/webhosting/index.html
- Do not forget to run `chmod 555 test.cgi`
Basics of CGI Programming

- CGI stands for Common Gateway Interface.
- A CGI program is a regular executable. Yes, even when you use a scripting language such as perl.
- All the information about CGI can be found at http://www.w3.org/CGI/

Input:

- Environment variables. For example, GET submissions, use the environment variable QUERY_STRING.
- Command line arguments, used by ISINDEX queries.
- Standard input. POST submissions use stdin. The exact number of bytes to read is in the environment variable CONTENT_LENGTH.

Output:

- Standard Output.
A Test Script

Compare:


CGI script source:

- http://www.cs.nyu.edu/ zaks/test.cpp
CGICC

CGICC is a C++ library that provides the following basic functions:

- Parses CGI input.

  bucks=120&time=9999&thoughts=I\%27don\%27t+have+any.&hungry=on

- Helps to create an HTTP response.

To produce the output:

```html
<head><title>Title</title></head>
```

**USE**

```cpp
cout << head(title("Title")) << endl;
```

**OR**

```cpp
cout<<head();
cout<<title("Title");
cout<<head()<<endl;
```
Parsing the Input

- The class `cgicc::Cgicc` is used for retrieving information on the submitted form elements.
- The class `cgicc::CgiEnvironment` is used to retrieve information on environment variables passed from the HTTP server.
- The class `cgicc::FormEntry` is used to extract various types of data from the submitted form elements.
A **FormEntry** is essentially a name/value pair, where the name is the name of the form element as specified in the HTML form itself, and the value is the user-entered or user-selected value.

```cpp
inline std::string operator* () const { return getValue(); }

long cgicc::FormEntry::getIntegerValue(long min,
                                        long max) const
{
    long value = std::atol(fValue.c_str());

    if(value > max)
        value = max;
    else if(value < min)
        value = min;

    return value;
}
```
**CgiEnvironment**

**CgiEnvironment** encapsulates the data passed from the HTTP server to the CGI application. This includes all environment variables set by the HTTP server specified in the CGI standard.

```cpp
const std::vector<HTTPCookie> & getCookieList() const
```

An HTTP cookie is a way to maintain state between stateless HTTP requests. HTTP cookies consist of name/value pairs, with optional comments, domains, and expiration dates. Usually, you will add one or more **HTTPCookie** objects to the HTTP headers your script is returning. For example, to set a cookie called count to 1 in a normal HTML document:

```cpp
cout << HTTPHTMLHeader().setCookie(HTTPCookie("count","1"));
```
Cgicc

Cgicc is the main class of the cgicc library. It is used to retrieve information on specific HTML form elements (such as checkboxes, radio buttons, and text fields), on uploaded files, and to save, restore, and retrieve information on the CGI environment. The default constructor of Cgicc essentially parses all the information from the web server and stores all the form data in the vector of FormEntry.

inline const std::vector<FormEntry>& getElements() const { return fFormData; }

bool cgicc::Cgicc::queryCheckbox(const std::string& elementName) const {
    const_form_iterator iter = getElement(elementName);
    return (iter != fFormData.end()) &&
           stringsAreEqual( (*iter).getValue(), "on");
}
void Cgicc::save(const std::string& filename) const;

void Cgicc::restore(const std::string& filename);

Note that all the information from a web server is saved in CgiEnvironment. Indeed, save and restore actually call corresponding protected members of CgiEnvironment (Cgicc is a friend class).
using namespace cgicc;

try {
    Cgicc cgi;
    ...
}
catch(exception& e) {
    // Caught a standard library exception
}
Replace ... with:

```cpp
// Send HTTP header
cout << HTTPHTMLHeader() << endl;

// Set up the HTML document
cout << html() << head(title("Cgicc example")) << endl;
cout << body() << endl;

// Print out the submitted element
form_iterator name = cgi.getElement("name");
if (name != cgi.getElements().end()) {
    cout << "Your name: " << **name << endl;
}

// Close the HTML document
cout << body() << html();
```
HTML Formatting Classes

ATOMIC_ELEMENT (br, "br"); // line break
BOOLEAN_ELEMENT (head, "head"); // document body

#define TAG(name, tag) class name##Tag { public: inline static const char* getName() { return tag; } }

#define ATOMIC_ELEMENT(name, tag) TAG(name, tag);
typedef HTMLAtomicElement<name##Tag> name

#define BOOLEAN_ELEMENT(name, tag) TAG(name, tag);
typedef HTMLBooleanElement<name##Tag> name
Declaration of HTMLBooleanElement

template<class Tag>
class HTMLBooleanElement : public HTMLElement

Using Tag:

virtual inline const char* getName() const
{
  return Tag::getName();
}
Generating Matching Opening/Closing HTML Tags

virtual inline bool getState() const { return sState; }
virtual inline void swapState() const { sState = ! sState; }
static inline void reset() { sState = false; }

private:
    static bool sState;

Allocating space for sState

template<class Tag>
    bool cgicc::HTMLBooleanElement<Tag>::sState = false;

Output is created via the parent class MStreamable

std::ostream& cgicc::operator<<(
    std::ostream&out, const MStreamable& obj) {
    obj.render(out);
    return out;
}
**HTMLElement::render**

```cpp
void cgicc::HTMLElement::render(std::ostream& out) const {
    if(getType() == eBoolean && dataSpecified() == false) {
        /* no embedded elements */
        if(getEmbedded() == 0) {
            swapState();
            /* getState() == true ===> element is active */
            if(getState() == true) {
                out << '<' << getName();
                /* render attributes, if present */
                if(getAttributes() != 0) {
                    out << ' ';
                    getAttributes()->render(out);
                } 
                else 
                    out << '</' << getName() << '>'; 
            } 
            } 
        }
    else 
        out << "</" << getName() << '>'; 
}
```
More HTML Formatting (1/2)

This tag has four attributes: src, width, height, and alt, with the values file.jpg, 100, 100, and description, respectively. Attributes in HTML tags are represented by the class HTMLAttribute, which essentially is a name/value pair. To build an HTMLElement containing HTMLAttribute objects, use the set method on HTMLElement. To generate the <img> tag given above:

```cpp
cout << img().set("src","file.jpg").set("width","100").set("height","100") .set("alt", "description") << endl;
```

In a similar way, multiple HTMLElement objects may be embedded at the same level inside another HTMLElement. To build an HTMLElement containing multiple embedded HTMLElement objects, use the add method on HTMLElement:

```cpp
cout << tr().add(td("0")).add(td("1")).add(td("2")) << endl;
```

This produces as output

```html
<tr><td>0</td><td>1</td><td>2</td></tr>
```
More HTML Formatting (2/2)

Embedded HTML elements:

```cpp
inline const HTMLElementList* getEmbedded() const
{ return fEmbedded; }

void setEmbedded(const HTMLElementList& embedded);

HTMLElement& add(const HTMLElement& element);
HTMLElement& add(HTMLElement* element);
```

Attributes:

```cpp
inline const HTMLAttributeList* getAttributes() const
{ return fAttributes; }

void setAttributes(const HTMLAttributeList& attributes);

HTMLElement& set(const std::string& name,
                 const std::string& value);
```