情報科学プロジェクト実験（コンピュータシステム研究室）

前回までにsqlite3を使ったデータベース操作のプログラムの作成方法を学んだ。
今回はこれをWebサーバ側のCGIプログラムとすることを目標とする。
CGIプログラムはいろいろなプログラミング言語で記述することができる。
この実験ではC++を使ってCGIプログラムを作成する。

CGIとは：
    Common Gateway InterfaceはWebサーバから動的なコンテンツを
    提供するための仕組みである。もっとも基本的なWebサーバは
    クライアントからのリクエストにより、あらかじめ用意された
    ファイルを送り返す（これを静的なコンテンツと呼ぶ）。
      [クライアント] ----リクエスト　　--->Webサーバ（ファイルアクセス)
      [クライアント] <---静的なコンテンツ----Webサーバ
    CGIを利用すると、Webサーバはリクエストによりプログラム実行し、
    その実行結果の出力をクライアントに返すことができる。
      [クライアント] ----リクエスト　　--->Webサーバ--->CGIプログラム
      [クライアント] <---動的なコンテンツ----Webサーバ<---CGIプログラム
    これによりリクエストのたびに結果の異なる動的なコンテンツを
    生成することが可能となる。
    CGIプログラムはperlなどのインタプリタ形式のスクリプト言語で書かれる
    ことが多いが、C++言語で書くことも可能である。

CGIプログラムへの入力：
    クライアントからのリクエストはHTTPにそって行われる。
    その際に、メソッドとしてGETまたはPOSTが利用される。
      GETメソッドの場合には、リクエストのURLの後ろに256文字
    までのデータを付加することができる。例えば、
       http://localhost/~foo/cgi-bin/test.cgi?abcdefg
    というWebブラウザでの指定は、以下のようなGETメソッドとなる。
       GET /~foo/cgi-bin/test.cgi?abcdefg HTTP/1.1
    ここで?文字以降の文字列abcdefgが付加されたデータである。
    この文字列は、CGIプログラムを実行する際の環境変数QUERY_STRINGに
    格納される（環境変数については後で学ぶ）。
      一方で、POSTメソッドの場合には、URLとは別にデータを指定するので
    文字数の制限がなく、CGIプログラムは標準入力からデータを読み出す。
      CGIプログラムのなかでメソッドがGETであるかPOSTであるかを
    判断するには、環境変数REQUEST_METHODを参照すれば良い。
    この環境変数にはGETまたはPOSTの文字列が設定される。

POSTメソッドの例から見ていく
```
-------------------------------------- test.cpp
// CGIプログラムとして働くC++プログラム
// これを動作させるには、
//    ・実行ファイルの拡張子は .cgiとする (例えば test.cgi)
//    ・実行ファイルを~/public_html/cgi-bin にコピーする
//    ・public_html/, cgi-bin/ test.cgi のパーミッション 755
//        % chmod 755 ~/public_html
//        % chmod 755 ~/public_html/cgi-bin/
//        % chmod 755 ~/public_html/cgi-bin/test.cgi
//    ・ホームディレクトリのパーミッション 711
//        % chmod 711 ~
#include <iostream>
using namespace std;

int main()
{
        cout << "Content-type: text/plain\r\n\r\n"; // ヘッダ情報

        char ch;
        while (cin >> noskipws >> ch)
                cout << ch;
        cout << endl;

        return 0;
}
--------------------------------------
```
このプログラムをコンパイルして、test.cgiという実行ファイルを作り
~/public_html/cgi-bin/にコピーする

次にtelnetを使って、Webサーバ経由で、CGIプログラムとしてこれを実行する。

```
---------  HTTP/1.0で送信してみる
$ telnet 133.220.114.232 80
Trying 133.220.114.232...
Connected to 133.220.114.232.
Escape character is '^]'.
POST /~okam/cgi-bin/test.cgi HTTP/1.0
Content-Length: 5

abcde
----------

---------  結果が戻る
HTTP/1.1 200 OK
Date: Fri, 08 Nov 2013 03:04:14 GMT
Server: Apache/2.4.6 (Fedora)
Connection: close
Content-Type: text/plain; charset=UTF-8

abcde
Connection closed by foreign host.
---------

---------  HTTP/1.1で送信してみる
$ telnet 133.220.114.232 80
Trying 133.220.114.232...
Connected to 133.220.114.232.
Escape character is '^]'.
POST /~okam/cgi-bin/test.cgi HTTP/1.1
Host: csweb
Content-Length: 5

abcde
----------

---------  結果が戻る
HTTP/1.1 200 OK
Date: Fri, 08 Nov 2013 03:07:06 GMT
Server: Apache/2.4.6 (Fedora)
Transfer-Encoding: chunked                    <<<--- chunked形式になった
Content-Type: text/plain; charset=UTF-8

1
a
1
b
1
c
1
d
1
e
1


0

Connection closed by foreign host.
---------
```

プログラムを以下の様に変更してみる
```
------------------------------------- test1.cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
        string s = "Content-type: text/plain\r\n\r\n"; // ヘッダ情報

        char ch;
        while (cin >> noskipws >> ch)
                s += ch;
        cout << s;

        return 0;
}
---------------------------------------

---------  再度HTTP/1.1で送信してみる
$ telnet 133.220.114.232 80
Trying 133.220.114.232...
Connected to 133.220.114.232.
Escape character is '^]'.
POST /~okam/cgi-bin/test1.cgi HTTP/1.1
Host: csweb
Content-Length: 5

abcde
---------

---------  結果が戻る
HTTP/1.1 200 OK
Date: Fri, 08 Nov 2013 06:37:00 GMT
Server: Apache/2.4.6 (Fedora)
Transfer-Encoding: chunked
Content-Type: text/plain; charset=UTF-8

5
abcde
0

Connection closed by foreign host.
---------
```

解説：
・HTTPではヘッダ情報と本体情報の2つを送る
  クライアントからのリクエストとサーバからの応答の双方とも
・その際ヘッダと本体は空行で分ける
  改行コードには"\r\n"を使う。
  従って、ヘッダと本体は "\r\n\r\n"という文字列で区切られる
  (telnetはデフォルトで改行に"\r\n"を使ってくれる)
・HTTP/1.1の場合クライアント側ではHost:情報が必須
  サーバ側で複数の仮想ホストをサポートする場合の分類に使う。
  いい加減な文字列を指定すればデフォルトのホストが使われる。
・クライアント側でPOSTメソッドを指定する場合には、
  Content-Length:を指定し、送りたい情報を本体に入れる。
・Content-Length:の指定は本体情報のバイト数(10進数で指定)
・Content-Lengthを指定しないで、Transfer-Encoding: chunked を
  指定すると、本体を分割して送ることができる。
  これは徐々に内容を送るときに便利。
・chunkedを指定した場合には、本体に
  転送バイト数の行(16進数)とそのバイト数分の情報を
  繰り返し指定する。
  転送バイト数 0 の指定は本体情報の終わりを意味する。
・Content-LengthとchunkedのどちらのHTTP指定も
  クライアントとサーバの双方で利用できる。

GETメソッドの場合にはCGIプログラムが自分の環境変数を
調べなければならない。

環境変数：
  プログラムを実行する際の共通の設定条件などを納めておく変数を
  環境変数という。abc=xyz という形式をしており、abcという変数の値が
  xyzであることを示す。プログラム中から環境変数を読み込むことで、
  OSやユーザごとの設定を複数のプログラム間で共有することができる。
  また、コマンド引数の情報の様に、あるプログラムが実行を開始した
  その瞬間の状況を、プログラムの中で判断するのにも使用できる。

  CGIプログラムの実行では、
      REQUEST_METHOD=GET
  または
      REQUEST_METHOD=POST
  という環境変数が設定される。
  さらに、この値がGETで、追加のパラメータがある場合には、
      QUERY_STRING=....
  と環境変数が設定される。

  Unixプログラムでは、getenv()というライブラリ関数で環境変数の値を
  調べることができる。詳細は man getenv としてみよ。
  この関数はgetenv("abc")とすると、環境変数 abc の値となる文字列の
  先頭アドレスを返す。abcが設定されていない場合には、NULLを返す。

```
------------ test1env.cpp
#include <iostream>
#include <string>
#include <cstdlib> // for getenv()
using namespace std;

int main()
{
   string s = "Content-type: text/plain\r\n\r\n"; // ヘッダ情報

   if (const char* p = getenv("REQUEST_METHOD")) {
      string method = p;
      s += method;
      if (method == "GET") {
         s += ": ";
         if (const char* p2 = getenv("QUERY_STRING"))
            s += p2;
      }
   }

   cout << s << endl;;
   return 0;
}
-----------

---------  リクエスト
 telnet 133.220.114.232 80
Trying 133.220.114.232...
Connected to 133.220.114.232.
Escape character is '^]'.
GET /~okam/cgi-bin/test1env.cgi?abcdef HTTP/1.0

---------

---------  応答
HTTP/1.1 200 OK
Date: Fri, 08 Nov 2013 05:25:15 GMT
Server: Apache/2.4.6 (Fedora)
Connection: close
Content-Type: text/plain; charset=UTF-8

GET: abcdef
Connection closed by foreign host.
---------
```

HTMLフォームからの入力：
  Webブラウザでユーザからの入力を受けとるにはフォームが使われる。ユーザが
  フォームのsubmitボタンをクリックすると、指定されたGETまたはPOSTメソッドで
  リクエストとともに入力内容がWebサーバに送られる。
------------------------------------- test.html
```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>フォームのテスト</title>
</head>
<body>

<h1>フォームのテスト</h1>
<form action="cgi-bin/test.cgi" method="post">
<p>メッセージ1: <input name="message1" size="60" /> </p>
<p>メッセージ2: <input name="message2" size="60" /> </p>
<input type="submit" name="OK" value="OKay" />
</form>

</body>
</html>
```

このHTMLファイルが以下の場所に置かれている場合、
  ホスト：133.220.114.232
  ディレクトリ：/home/okam/public_html/
Webブラウザからは
  http://133.220.114.232/~okam/test.html
を指定してアクセスすることになる。
public_htmlがokamユーザの公開ディレクトリなので、
URLにはpublic_htmlの文字列が含まれないことに注意。

これはHTTPでは、133.220.114.232に接続したあとに、
  GET /~okam/test.html HTTP/1.1
というリクエストとなる。

フォームのアクションとなるCGIは相対パスで指定されているので、
/~okam/が起点のディレクトリとなり、相対的に
  POST /~okam/cgi-bin/test.cgi HTTP/1.1
というリクエストとなる。
アクションに絶対パス、action="/cgi-bin/test.cgi" と書くと、
  POST /cgi-bin/test.cgi HTTP/1.1
となってしまうことに注意

Webブラウザに表示される結果は以下の様になる。
```
----------
message1=abc&message2=xyz&OK=OKay
----------
```

解説：
・Webブラウザはinputタグのname属性と入力された文字列（value属性）を
        name1=value1
  の形式にする。
・複数のinputタグがある場合には、& 文字でそれぞれを区切る。
・特定の文字は次のルールにより他の文字に置き換えられる。
        スペースは + 文字に置き換えられる。
        &,+,=を含むほとんどの記号文字や漢字コードは、1バイトごとの
        %NN の形式に置き換えられる。NNは16進数2桁の数字である。
        つまり8ビット分の情報を3バイト（3文字）で表現する。

これから推測されるように、
Webブラウザは以下のようなHTTPリクエストを送ったのと同様である。

```
--------- Webブラウザと同じ結果を得るリクエスト
$ telnet 133.220.114.232 80
Trying 133.220.114.232...
Connected to 133.220.114.232.
Escape character is '^]'.
POST /~okam/cgi-bin/test.cgi HTTP/1.1
Host: csweb
Content-Length: 33

message1=abc&message2=xyz&OK=OKay
---------
```

補足： HTTP persistent connection (または HTTP Keep Alive)
telnetを使ってHTTP/1.0のリクエストを送るとWebサーバは応答とともに即座にTCPの接続を切る。一方で、HTTP/1.1の場合にはすぐには接続を切らない。通常、1つのHTMLファイルには図などが入っており、それを表示するためには複数のHTTPリクエストが必要である。HTTP/1.0ではあるクライアントが複数のHTTPリクエストを出す場合には、なんどもTCP接続を行わなければならなかった。複数の同時TCP接続はサーバのCPUの負荷/メモリ使用量の増大とともに応答時間の増大も招いてしまう。多数の同時ユーザに対処するWebサーバではそれらの増大は深刻な問題である。そこで、同一クライアントからの複数リクエストを単一のTCP接続で処理させるようになった。

あとはこれらの知識をプログラムとしてまとめれば良い。
例えば、以下のようなプログラムが考えられる。

```
//--------- 参考程度に見よ ----------------------------------------
// CGIプログラムとして働くC++プログラム
// フォームデータを解析して簡単に使えるようにする
#include <iostream>
#include <string>
#include <map>       // map<string, string>用
#include <cctype>    // getnum内で呼び出す関数用
using namespace std;

// 16進数表記の2文字の列c1c2を対応する数値に変換する
int getnum(char c1, char c2)
{
   int a = isdigit(c1) ? (c1 - '0')      :
           isupper(c1) ? (c1 - 'A' + 10) :
           islower(c1) ? (c1 - 'a' + 10) : 0;
   int b = isdigit(c2) ? (c2 - '0')      :
           isupper(c2) ? (c2 - 'A' + 10) :
           islower(c2) ? (c2 - 'a' + 10) : 0;
   return 16*a + b;
}

// 文字列中の特殊文字を変換する
string decode(string s)
{
   string x;
   for (size_t i = 0; i < s.size(); i++) {
      switch (s[i]) {
      case '+':
         x += " ";
         break;
      case '%':
         if (i+2 < s.size()) {
            x += getnum(s[i+1],s[i+2]);
            i += 2;
            break;
         }
      default:  x += s[i];
      }
   }
   return x;
}
```

```
// 標準入力からの文字列を解析して、フォームデータを
// map型の変数に入れる
map<string, string> read_and_parse()
{
   map<string, string> env;
   string line;
   while (getline(cin, line, '&')) {
      if (line.size()==0) continue;
      size_t p = line.find('=');
      if (p == string::npos) {
         env[line] = "";        // 登録
         continue;
      }
      string key   = decode(line.substr(0,p));
      string value = decode(line.substr(p+1));
      env[key] = value;        // 登録
    }
    return env;
}

int main()
{
    map<string, string> fmd = read_and_parse();

    string s =
         "Content-type: text/html\r\n\r\n"       // HTMLを返す時のタイプ
         "<!doctype html><html>"
         "<head><meta charset=\"utf-8\"></head>"
         "<body><h1>cgi test</h1>";
    s += "message1 ->" + fmd["message1"] + "<br>";
    s += "message2 ->" + fmd["message2"] + "<br>";
    s += "</body></html>";
    cout << s;
    return 0;
}
---------
```

今回もお手軽クラスを用意した。
<CGIinput.hpp>をインクルードすると以下のようなプログラムになる。
これを使うと、メソッドがGETでもPOSTでも同一のプログラムにできる。

```
-----------------------------------------------------------------
//--------- とても重要 ----------------------------------------
-----------------------------------------------------------------
// CGIプログラムとして働くC++プログラム
// フォームデータを解析して簡単に使えるようにする
// GETメソッド/POSTメソッドのどちらでも対応する
#include <CGIinput.hpp>
using namespace std;

int
main()
{
    CGIinput fmd; // 宣言とともに入力を終える

    // 結果を出力
    string s =
         "Content-type: text/html\r\n\r\n"       // HTMLを返す時のタイプ
         "<!doctype html><html>"
         "<head><meta charset=\"utf-8\"></head>"
         "<body><h1>cgi test</h1>";
    s += "message1 ->" + fmd["message1"] + "<br>";
    s += "message2 ->" + fmd["message2"] + "<br>";
    s += "</body></html>";
    cout << s;
    return 0;
}
-----------------------------------------------------------------
```

実験準備：
csweb2(133.220.114.232)にloginする

~/public_html　ディレクトリに html形式のファイルをおいて公開するには
・ホームディレクトリのパーミッション　rwx--x--x (711)とする
　% chmod 711 ~
・公開ディレクトリ(~/public_html)のパーミッションを rwxr-xr-x (755)とする
　% chmod 755 ~/public_html

C++プログラムをCGIとして動作させるには
・CGIプログラムを置くディレクトリの(~/public_html/cgi-bin)のパーミッションを
　rwxr-xr-x (755)とする
　% chmod 755 ~/public_html/cgi-bin
・実行ファイルの拡張子は .cgiとする (例えば test.cgi)
・実行ファイルを~/public_html/cgi-bin にコピーする
・実行ファイルのパーミッションを rwxr-xr-x (755)とする。
　これだけはコンパイルするたびに行う必要がある
　% chmod 755 ~/public_html/cgi-bin/test.cgi

CGIinput.hppは
　csweb2上の/usr/local/include ディレクトリにあるので
　コンパイルする際にg++に -I/usr/local/include オプションを加える

ブラウザ側の注意
・研究室のWebサーバを使うので、web proxyの設定を確認する

課題

前回までに作った郵便番号検索C++プログラムを
CGIプログラムに修正して、
Webのインタフェースで使えるように変更する