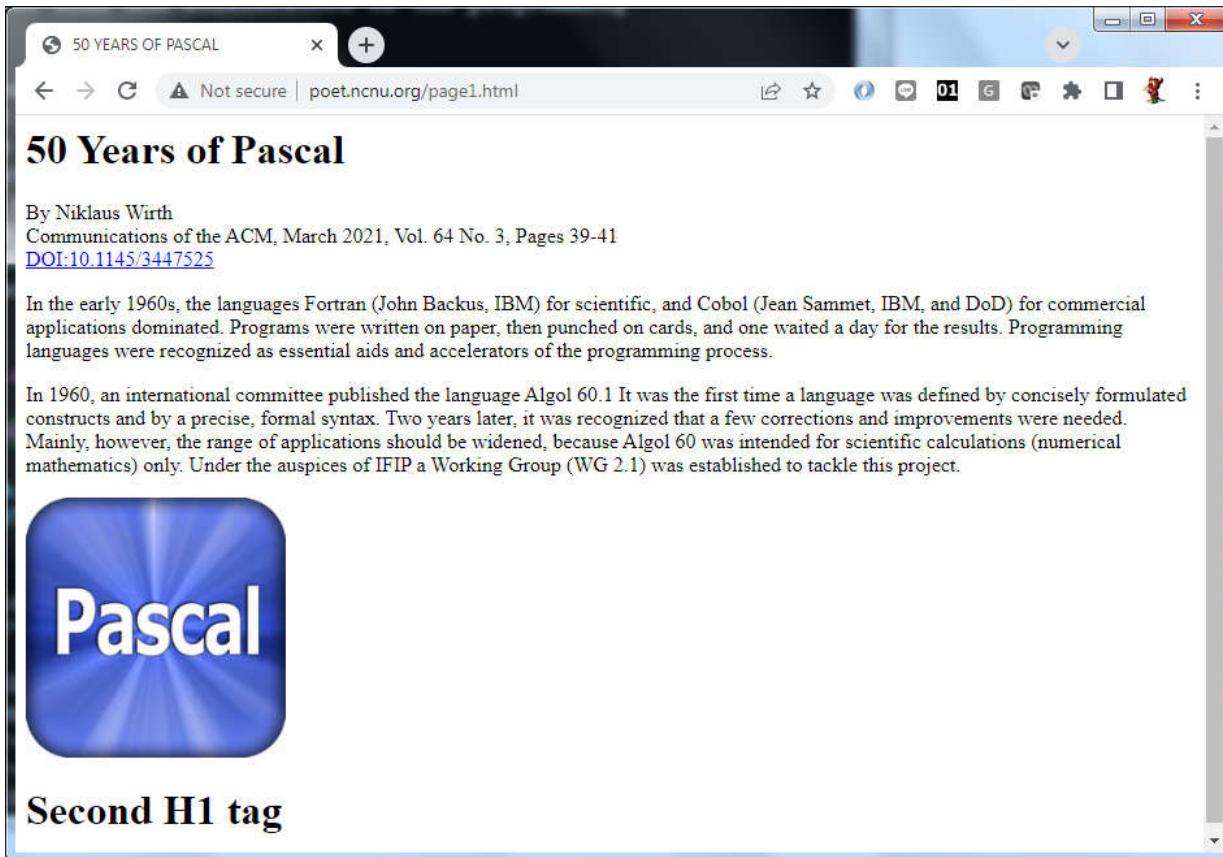




# cURL

## and the Road to Object-Oriented

# Webpages you see in a browser ...



The screenshot shows a web browser window with the title "50 YEARS OF PASCAL". The address bar indicates the page is "Not secure | poet.ncnu.org/page1.html". The main content area features a large blue button with the word "Pascal" in white. Below the button, the text "Second H1 tag" is displayed.

50 YEARS OF PASCAL

Not secure | poet.ncnu.org/page1.html

## 50 Years of Pascal

By Niklaus Wirth  
Communications of the ACM, March 2021, Vol. 64 No. 3, Pages 39-41  
[DOI:10.1145/3447525](https://doi.org/10.1145/3447525)

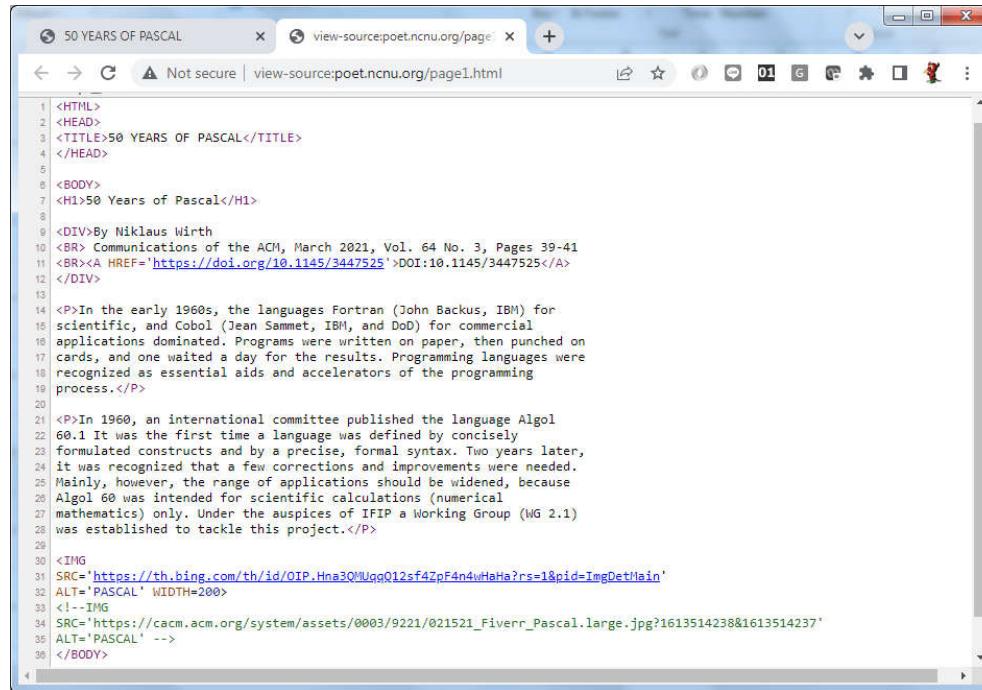
In the early 1960s, the languages Fortran (John Backus, IBM) for scientific, and Cobol (Jean Sammet, IBM, and DoD) for commercial applications dominated. Programs were written on paper, then punched on cards, and one waited a day for the results. Programming languages were recognized as essential aids and accelerators of the programming process.

In 1960, an international committee published the language Algol 60.1. It was the first time a language was defined by concisely formulated constructs and by a precise, formal syntax. Two years later, it was recognized that a few corrections and improvements were needed. Mainly, however, the range of applications should be widened, because Algol 60 was intended for scientific calculations (numerical mathematics) only. Under the auspices of IFIP a Working Group (WG 2.1) was established to tackle this project.

Pascal

Second H1 tag

is actually an **HTML** file  
interpreted and formatted by software

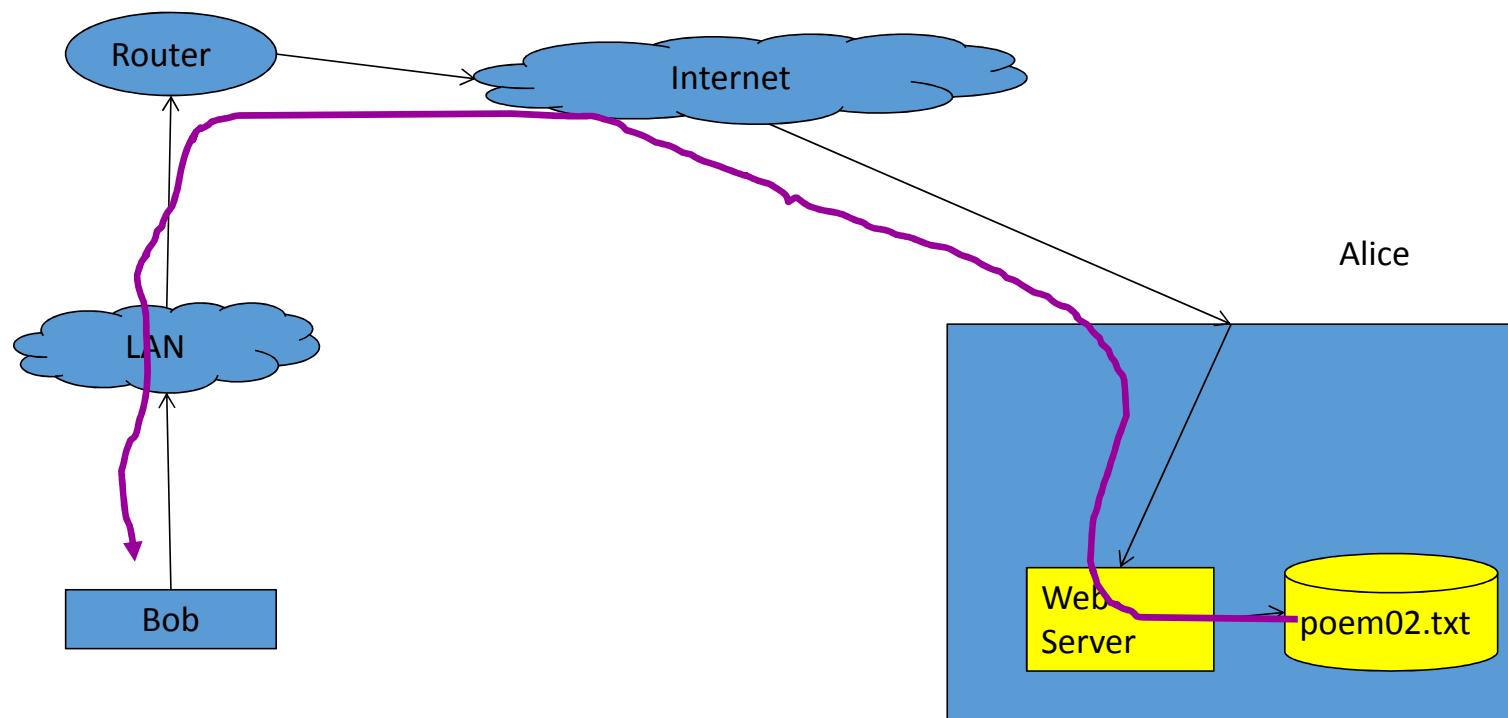
A screenshot of a Microsoft Internet Explorer browser window. The title bar says "50 YEARS OF PASCAL" and the address bar says "view-source:poet.ncnu.org/page1.html". The main content area shows the raw HTML code for the page. The code includes standard HTML tags like <HTML>, <HEAD>, <TITLE>, <BODY>, and <DIV>. It also contains some text content and a few image tags (<IMG>) with their SRC attributes pointing to external URLs.

```
1 <HTML>
2 <HEAD>
3 <TITLE>50 YEARS OF PASCAL</TITLE>
4 </HEAD>
5
6 <BODY>
7 <H1>50 Years of Pascal</H1>
8
9 <DIV>By Niklaus Wirth
10 <BR> Communications of the ACM, March 2021, Vol. 64 No. 3, Pages 39-41
11 <BR><A HREF='https://doi.org/10.1145/3447525'>DOI:10.1145/3447525</A>
12 </DIV>
13
14 <P>In the early 1960s, the languages Fortran (John Backus, IBM) for
15 scientific, and Cobol (Jean Sammet, IBM, and DoD) for commercial
16 applications dominated. Programs were written on paper, then punched on
17 cards, and one waited a day for the results. Programming languages were
18 recognized as essential aids and accelerators of the programming
19 process.</P>
20
21 <P>In 1960, an international committee published the language Algol
22 60.1 It was the first time a language was defined by concisely
23 formulated constructs and by a precise, formal syntax. Two years later,
24 it was recognized that a few corrections and improvements were needed.
25 Mainly, however, the range of applications should be widened, because
26 Algol 60 was intended for scientific calculations (numerical
27 mathematics) only. Under the auspices of IFIP a Working Group (WG 2.1)
28 was established to tackle this project.</P>
29
30 <IMG
31 SRC="https://th.bing.com/th/id/OIP.Hna3QIUgg012sf4ZpF4n4wHaHa?rs=1&pid=ImgDetMain"
32 ALT="PASCAL" WIDTH=200>
33 <!--IMG
34 SRC="https://cacm.acm.org/system/assets/0003/9221/021521_Fiverr_Pascal.large.jpg?1613514238&1613514237"
35 ALT="PASCAL" -->
36 </BODY>
```

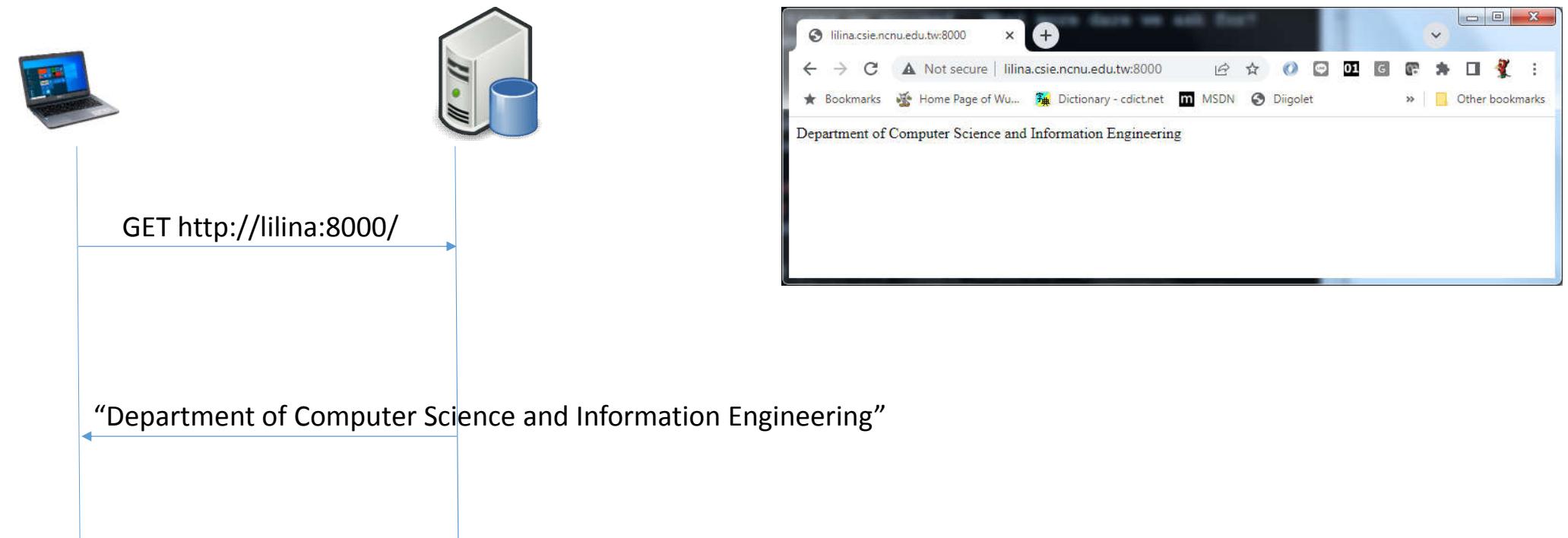
**Exercise:**

1. Visit a webpage, e.g., <http://poet.ncnu.org/page1.html>
2. View its HTML source (by Ctrl-U)

# Sharing Data on the Internet



# Getting Data from a Webpage



You can also do this from the command line.

- curl http://lilina:8000/

```
$ curl http://lilina:8000/  
Department of Computer Science and Information Engineering
```

# You can do this with a Python program

- `from urllib.request import urlopen`
- `url = 'http://lilina:8000/'`
- `html = urlopen(url).read().decode()`
- `print(html)`

```
/* Open a Connection */
#include <stdio.h>
#include <curl/curl.h>
#define URL      "http://lilina:8000/"

int main(void)
{
    CURL *curl;
    CURLcode res;

    curl = curl_easy_init();
    if(curl) {
        curl_easy_setopt(curl, CURLOPT_URL, URL);
        res = curl_easy_perform(curl);
        if(res != CURLE_OK)
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
                    curl_easy_strerror(res));
        curl_easy_cleanup(curl);
    }
    return 0;
}
```

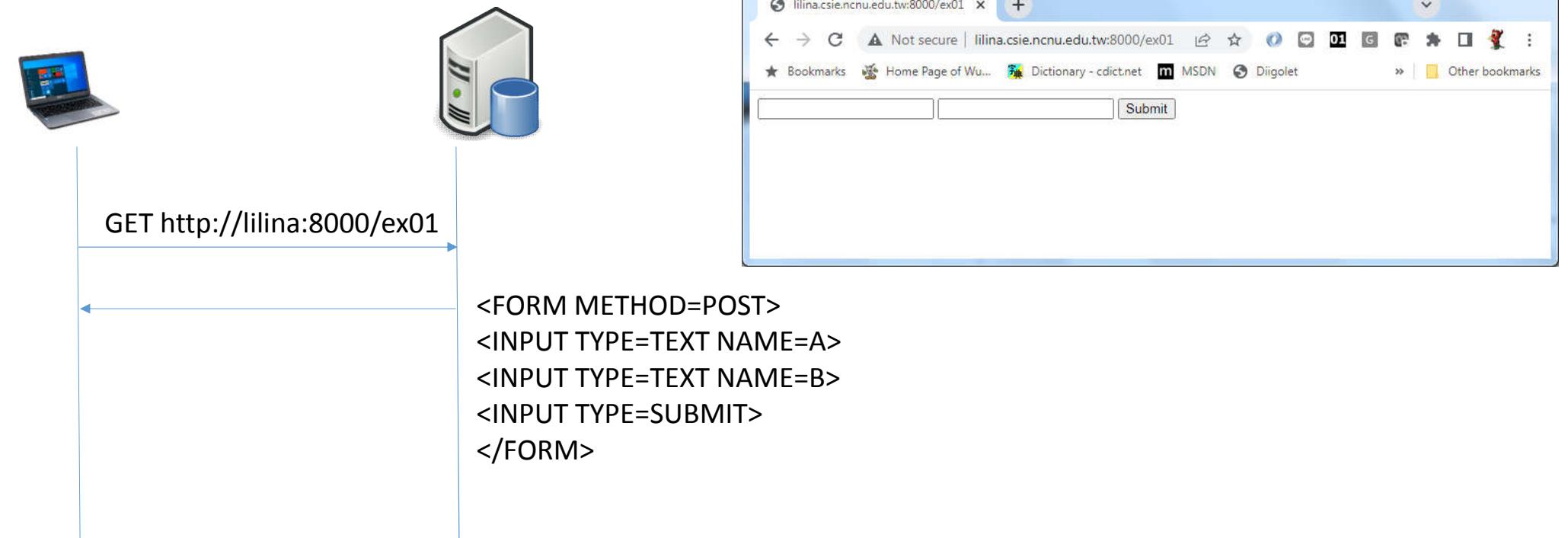
You can also do this  
with a C program

Compile with the option “-lcurl”

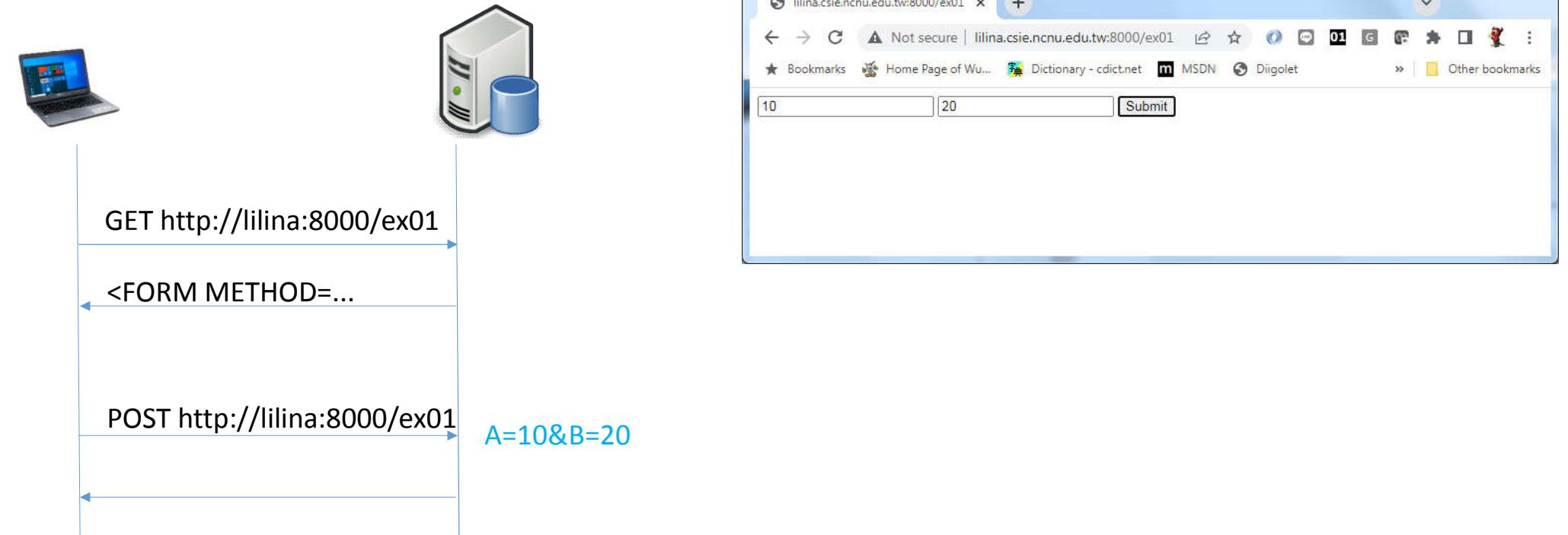
## curl-2.c

- The previous program simply retrieves a webpage, and prints out the HTML contents.
- This just mimics the behavior of the “curl” command.
- If you want to store the contents for later usage, you may (dynamically) allocate a memory to store that.
  - See “curl-2.c”

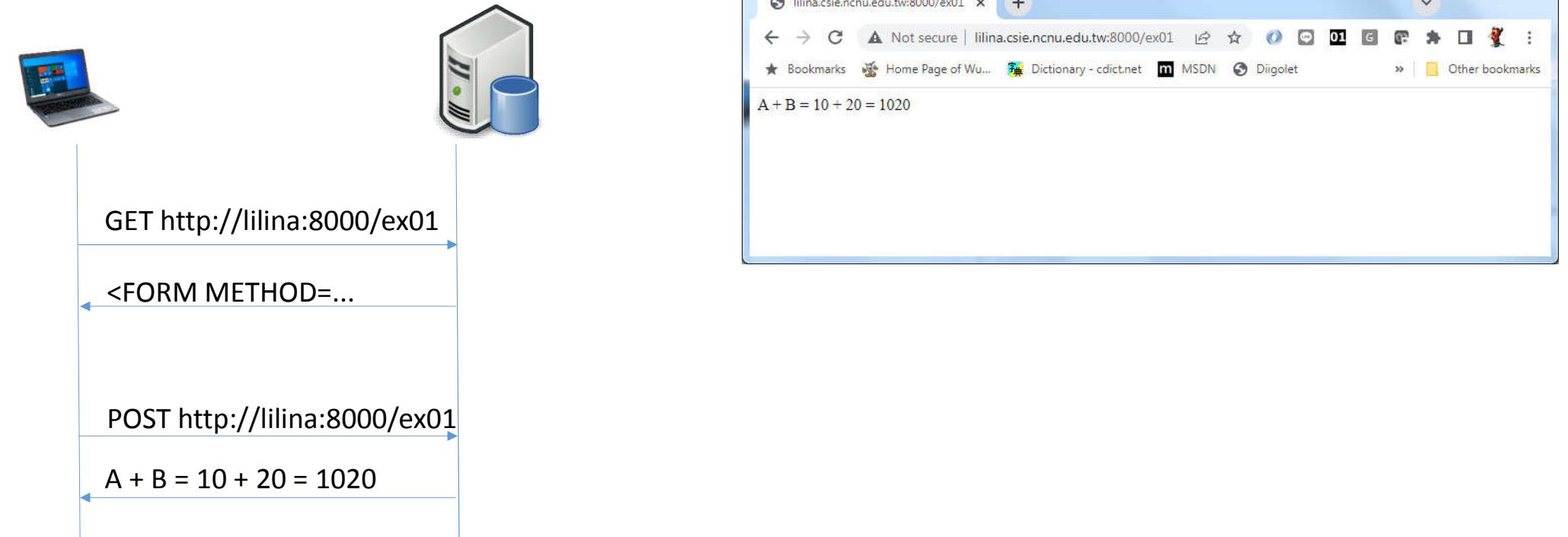
# Posting Data to a WWW Server



# Posting Data to a WWW Server



# The WWW Server responds with a string



# You can do this from a command line

- curl http://lilina:8000/ex01

```
<FORM METHOD=POST>
<INPUT TYPE=TEXT NAME=A>
<INPUT TYPE=TEXT NAME=B>
<INPUT TYPE=SUBMIT>
</FORM>
```

- curl -X POST -F "A=10" -F "B=20" http://lilina:8000/ex01

```
A + B = 10 + 20 = 1020
```

# You can do this from a Python program

```
from urllib.request import urlopen  
url = 'http://lilina:8000/ex01'  
post_str = 'A=10&B=20'.encode()  
html = urlopen(url, post_str).read().decode()  
print(html)
```

You can also do this  
with a C program

```
curl = curl_easy_init();
if(curl) {
    curl_easy_setopt(curl, CURLOPT_URL, WWW);
    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, "A=10&B=20");

    /* Perform the request, res will get the return code */
    res = curl_easy_perform(curl);

    /* Check for errors */
    if(res != CURLE_OK)
        fprintf(stderr, "curl_easy_perform() failed: %s\n",
                curl_easy_strerror(res));

    /* Always cleanup */
    curl_easy_cleanup(curl);
}
```

Compile with the option “-lcurl”

## curl-4.c

- The previous program simply post data to a WWW server, and prints out the response.
- This just mimics the behavior of the “curl” command.
- If you want to store the response for later usage, you may (dynamically) allocate a memory to store that.
  - See “curl-4.c”

# Why do people prefer Python in Webscraping?

- Sir, I know C++ is powerful, but it is just too complicated.
- There are 88 lines of C code in “curl-2.c” to retrieve a webpage.
- With Python, it takes only 4 lines!
- Maybe C program runs faster, but it is faster to write a Python program.
- Not necessarily. That's why you need to learn Object-Oriented Programming.

# Encapsulate curl-4.c as an Object

- class CRequest {
- private:
- string url;
- string response;
- public:
- CRequest& urlopen(string s, string post\_str = "");
- string read();
- };

## Some details to re-write curl-4.c

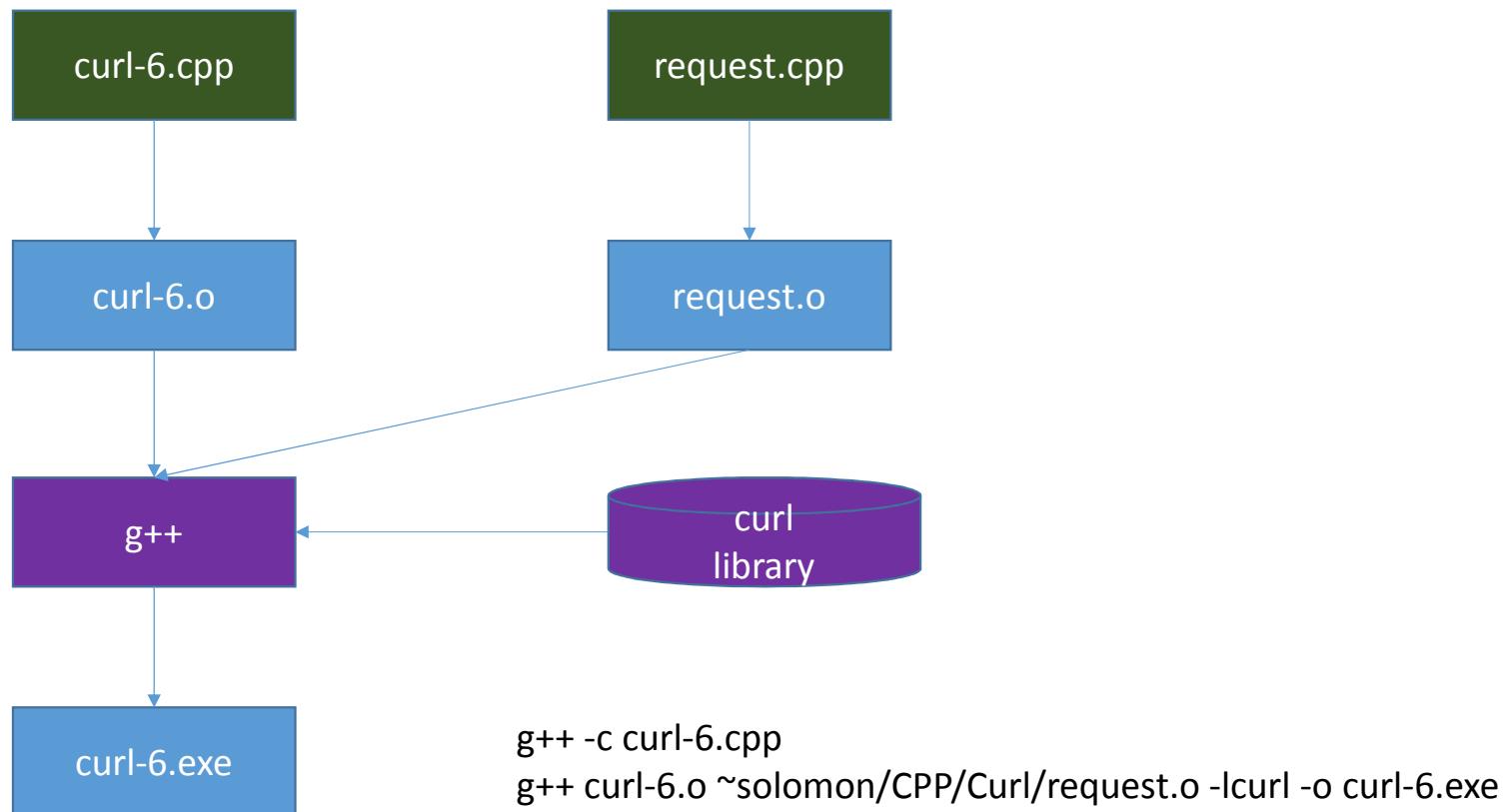
- `realloc()` returns `void*`, so it must be `reinterpret_cast` to `char*`.
- The `main()` stores the retrieved webpage in a dynamically allocated memory which is pointed by `chunk.memory`, so we construct a string with that and return the string as the response.

# Now I can use it in my main program

- int main() {
- CRequest request;
- string html;
- html = request.urlopen(URL).read();
- cout << html << endl;
- html = request.urlopen("http://lilina.csie.ncnu.edu.tw:8000/ex01", "A=5&B=3").read();
- cout << html << endl;
- return 0;
- }

```
g++ curl-5.cpp -lcurl -o curl-5.exe
```

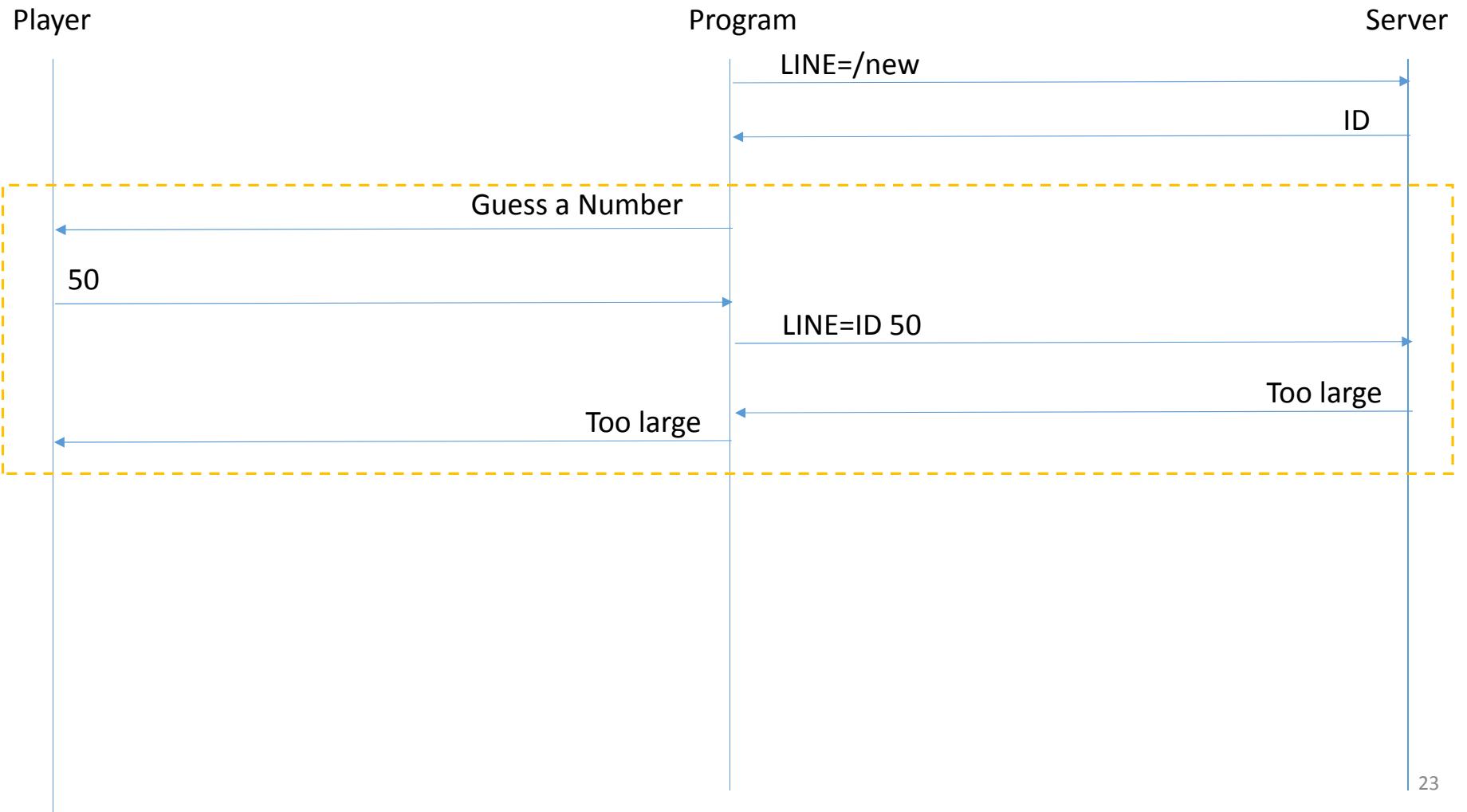
# Separate the Class Implementation



## Hands-On: Login

- Write a C++ program to post a string “STUID=111321xxx” to <http://lilina:8000/login>.
- You can see the result at <http://lilina:8000/list>

# Exercise: Guess a Number



# Exercise: Guess a Number

