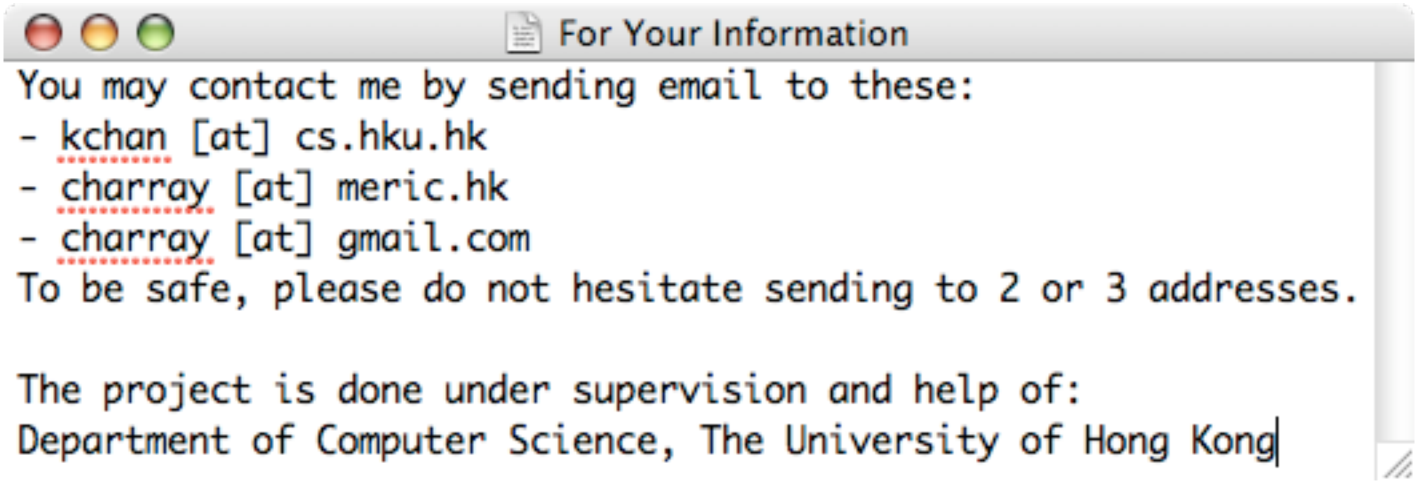
S++ Final Year Project Report

Member: Kinson Chan
Supervisor: A.T.C. Tam
2nd Examiner: K.W. Chong



```
You may contact me by sending email to these:
- kchan [at] cs.hku.hk
- charray [at] meric.hk
- charray [at] gmail.com
To be safe, please do not hesitate sending to 2 or 3 addresses.

The project is done under supervision and help of:
Department of Computer Science, The University of Hong Kong
```

Table of Contents

1. Introduction

The idea to implement S++ originated from painful experience using scripting solutions for dynamic web pages. Automatic conversion in scripting solutions comes in handy. However, in large applications, if strict correctness is required, automatic conversion increases maintenance effort and cost.

On the other extra extreme, one may apply *strong typing* solution such as Java Servlet. However, applying the language is not strict forward as the language is not designed for Internet operations. Moreover, a lot of servlet solutions requires knowledge and experience in object oriented programming and even pointers handling, which are in fact more than enough.

Being a dream replacement for the tools above, S++ aims to improve efficiency in coding and strength against malicious input, with the help of strong typing property on C++.

S++ can be thought as a C++ program with blanks. Programmers can fill in the blanks on their own, compile, and obtain an executable. Although coding in this way is complicated, expert programmers may like the full power of control.

Meanwhile, S++ provides an interesting way to fill in the blanks - let the *code generator* in S++ to do the job.

The code generator improves efficiency in coding by simplifying the code required for common tasks in a web application. Examples include user input conversion and file access. In the following context, we call variables related to these operations *parameters.*

The generator improves security also by simplifying the code required for *parameter checking.* Because of simplified checking, programmers are encouraged to write more restrictive codes.

Parameters are marked by some special tokens called *modifiers.* Instead of requiring programmers to tell how to do the checking, progrmmers are simply required to tell how they would like the parameters be.

While not everyone likes S++ (probably due to C++'s nature), somebody may find this tool useful.

2. Dynamic Page Technologies

In this part, we discuss some of the existing dynamic web page technologies. We learn where they are not attractive enough and where they are useful.

2.1 PHP as Script

PHP is one of the most popular scripting solutions for dynamic pages. Unlike other solutions, PHP is specially designed around serving dynamic web pages.

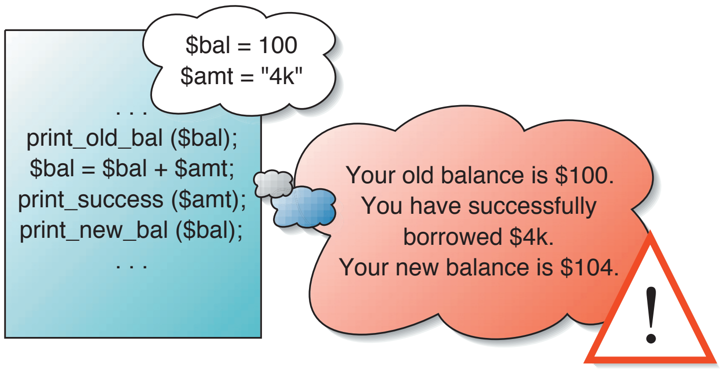
2.1.1 Language

PHP is a *weak typing* language. Programmers can use variables without prior definition and initialization. Strings and numeric values can be implicitly converted. The same type of variable is also capable to be an array or hash.

Internet applications involve a lot of conversions. Examples include communicating with user and database as string, and calculating as number internally, for the same variable. Weak typing allows all these conversions implicitly done.

The downside of this design is the impossibility to check the type of a parameter automatically. According to the PHP manual, although it is possible to give hints about type of an object, giving hints for primitive types is not supported. If a programmer wish to have type checking, it is required to check it manually in the code.

Failure to do checking usually leads to logic error, as shown in the picture. “4k” is treated as 4 for calculation and “4k” again for user output. A wrong message is therefore generated.



```
$bal = 100
$amt = "4k"
```

```
. . .
print_old_bal ($bal);
$bal = $bal + $amt;
print_success ($amt);
print_new_bal ($bal);
. . .
```

```
Your old balance is $100.
You have successfully
borrowed $4k.
Your new balance is $104.
```

Unfortunately, it is not straight forward to ensure the checking is well done in all parts of the code. Having big PHP-based software means high cost of code audit and maintenance. For example, one of the most popular open-source PHP forum software, phpBB (www.phpbb.com), has received half a decade of patching for 2.0.x without another major release.

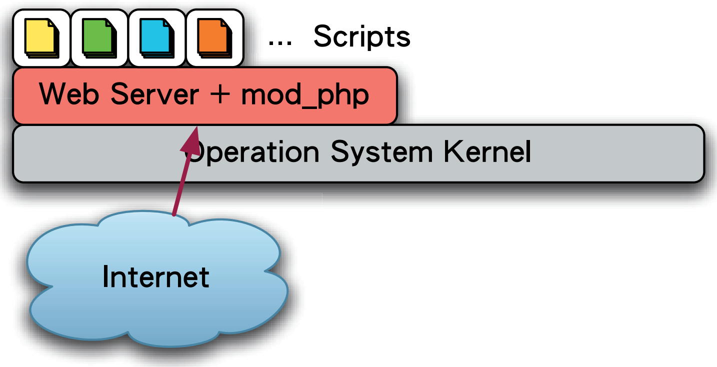
Moreover, doing this checking everywhere is computationally expensive, further hampering performance. We discuss a new way to do these checking in “New Ideas”.

One may question if type inference helps to do checking without runtime penalty. Type inference determines if there are any type mismatch, given the operation on the variables. Mismatches are reported in compile time.

The answer to this possibility is no. Values of variables usually come from user input and database. It is not possible to predict what users enter in runtime. It is neither possible to tell what type a variable is expected from function calls. For example, `mysql_escape()` accepts all the strings. Moreover, it is questionable whether programmers are willing to apply type inference checking, provided the one without works well apparently.

2.1.2 Infrastructure

While it is possible to run PHP alone in command line interface, it is a common practice to install PHP as a module inside web server. It is required to have modules built according to each of the web server. The choice of these modules and choice between web servers are limited.



2.1.3 Security

When PHP is running as a module, it shares the same security setting of the web server. Without proper setting, a PHP script can touch contents of a web site that it does not belong to, or even some system settings. In order to rectify the problem, a lot of security settings (e.g. safe mode) are introduced. However, compared to how an operation system partition the processes, PHP's approach is not natural.

2.1.4 Deployment

PHP usually exist as a module in web server. Programmers simply need to copy the PHP scripts onto some locations readable by the web server process, and expect the code to run.

Although we have a single organization managing PHP, PHP language has been changed a lot. It is questionable whether the code is portable. Properties of object oriented programming is one of the well known issues. In PHP4, copying object is done by assignment operator (=). In PHP5, the same instruction is interpreted as copying the reference. These differences silently make applications fail.

PHP setting is also one of the source of incompatibilities. PHP interpreter's behavior can be changed drastically simply because of different configuration. For example, the `register_global` option can be switched on and off easily and this affects how parameters are saved - as global variable or as an element inside an predefined array. With all these settings under control of administrators, it is difficult to make sure programs run seamlessly.

2.2 Java Servlet as Secure Solution

Java is one-language-fit-all solution from Sun Microsystems. The language is designed to be object oriented (except primitive types). Java servlet is one of its application.

2.2.1 Language

Java is a *strong typing* language aiming straightforward reading and writing. Although it is from C++, it intentionally drop some of the features for clarity and security. Examples include pointers and operator overloading. Without pointers, a lot of forceful type conversions in C and dangerous situations are abolished.

Java is successful in improving sustainability of programs. In a C program, failure to access to a pointer is fatal. In Java, the failure usually causes terminal of a thread only.

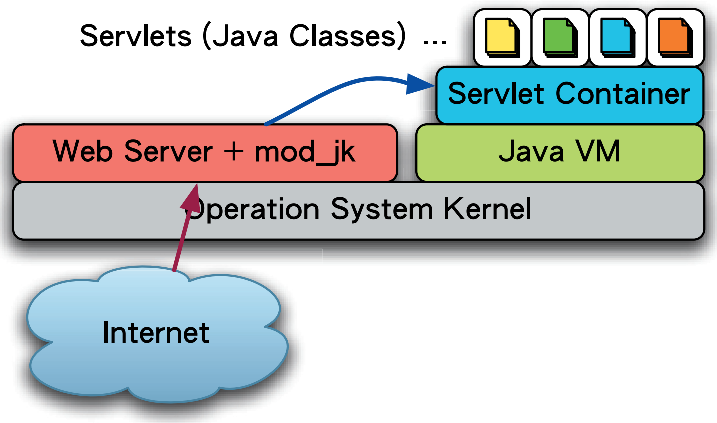
Java is also successful in making sure parameter types agree with the methods in most of the cases. On the other hand, this strictness is not useful for the front end of web applications. The conversions between types have to be done manually. Moreover, this kind of conversion is not exception-safe, meaning a lot of codes have to be written for error handling.

```
try {
    int Num1 = Integer.parseInt
                   ((String)request.getParameter(“Num1”));
} catch (Exception e) {
    // error processing...
}
```

2.2.2 Infrastructure

Unlike PHP, design and implementation of Java Servlet container is separated. We have a few implementations of Java Servlet containers.

Java Servlet containers are usually capable of running as a standalone server, or get connected to web servers through modules. Examples include mod_jk, which links Apache HTTP server with Apache Tomcat. This makes the structure relatively more complicated. The web server has to communicate with the container through socket operations.



As the name suggest, Java Servlet is written in Java and compiled in Java byte-code. We need Java runtime and even Java SDK to run the containers.

2.2.3 Security

Java is built for secure computing. On the other hand, servlet container is a very complicated piece of program. It runs under a single Unix user permission. Failure of partitioning may damage the web application. However, given that a few containers can run independently, the area of damage is smaller.

2.2.4 Deployment

Deploying a Java Servlet is more demanding than a PHP file. In order to run Java Servlet, we need to have Java virtual runtime, Java SDK and a Java Servlet Container ready. Servlets are not executing on their own. They have to be put inside a web application.

In order to have an web application ready, one has to provide some configuration files about the application. Examples include mappings between URL to Servlets and environment variables for the application. Programmers can rely on their favorite integrated development environment to prepare the files.

In order to compensate the complexity, the whole web application, including the configuration files, can be packaged into a .war file for easier handling. This eliminates the risk of having a specific file missing.

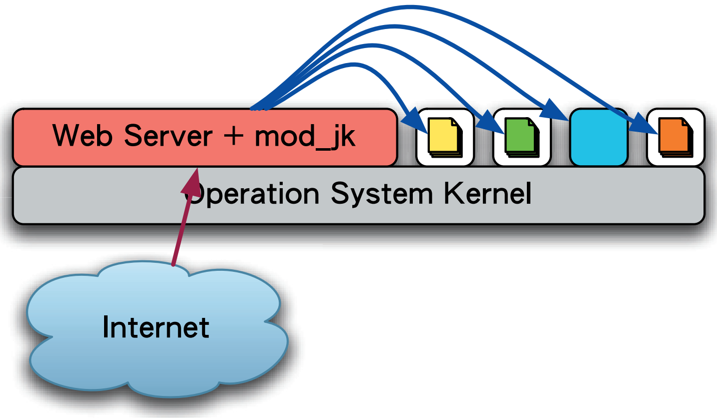
One interesting point to note is, the infrastructure software for running Java Servlet is relatively more demanding to the computer system. For example, Java Servlets as well as the containers are usually in the form of interpretation rather than native code. Relative more CPU resource is required to do the same task in PHP or CGI. It is not always possible to be run on underpowered embedded system.

2.3 Common Gateway Interface as Traditional Answer

Common Gateway Interface (CGI) is a unix style answer to dynamic web page technology. It is a specification how a web server can interact with an external program for dynamic web pages, and does not bound programmers to any languages.

2.3.1 Infrastructure

When a request for dynamic web page is received, the web server initiates a new process and execute the CGI executable. Request's information (e.g. query string, upload content) are passed through environment variables and standard output. Web server delegates the output from process's standard output to the user.



Although CGI is the oldest technology among the three discussed, it has the least software dependency with the languages and web servers. Implementing according to CGI standard is the least difficult because it involves piping input and output only. While we may not want to apply CGI anymore, it will be wise for us to consider the beauty of its dependency-avoidance.

2.3.2 Security

CGI had been well known for the bad security. Web servers was used to initiate the CGI executables as root or a centralized Unix account. Processes can step on each other's data. Besides, processes of the same application run without central coordination, processes of the same executable may overlap execution so close that race condition takes place.

The potential to be secure on CGI is, however, high. The security problems can be solved easily by having different Unix accounts for different executables, as well as caution on race conditions. The processes are partitioned by the operation system. The partitioning is natural (without extra programming for partitioning) and effective.

2.3.3 Deployment

In order to deploy a CGI program, it has to be copied to a folder accessible by the web server, and mark the program (or the folder) to be executable. Whenever there is a request, the server instantiates the program once for result.

This design does not restrict users to choose either compiled programs or scripts. With correct header (i.e. the interpreter's absolute path), scripts can be executed as if they are executable. Other programs can be compiled manually and be put inside.

Although this deployment is simple, the execution cost is not low. Whenever there is a request, a process is instantiated. Instantiating a process is a high cost operation, especially when the CGI program involved (as well as the interpreter) is large.

3. Problems with C++ for Dynamic Pages

3.1 Data Conversion

Unlike scripting solutions, converting from string to number, or from number to string in C++ is not trivial. It is in fact even more troublesome than Java. The following code segments show the two ways to convert a string into an integer.

In fact, it requires extra checking to make sure the conversion is correct. The following examples treat "4k" as 4, just like other scripting solutions.

```
// C++ style.
istringstream input(str_value);
input >> int_value;

// C-style, not thread-safe.
int_value = atoi(str_value.c_str());
```

3.2 Memory Faults and Leakage

It is not uncommon to have dynamically allocated memory. In C and C++, doing this can be very dangerous because the memory locations are represented in pointers. Pointers are dangerous because their value can be illegally changed, pointing to invalid locations. It is also possible to have memory leakage because allocated memory does not go out of scope with the pointers.

A badly-written C++ program may contain a few above-mentioned bugs. They may hide at the beginning and suddenly appears, turning off the software or even destroying data.

3.3 System Call Incompatibility

While web applications are related to simple calculations only, it is inevitable to have access to system calls. Examples include network and file handling. However, these functions' return value do not agree. For example, `fopen(3)` returns zero for failure and `read(2)` returns negative number for failure.

Programmers can get confused when they have to handle these on their own.

4. S++ as New Solution

4.1 Language

S++ allows programmers to create dynamic pages in C++. C++ is a *strong typing* language. Meanwhile, it has flexibilities for interesting and useful effects.

One of the examples is operator overloading. It is possible to define new types and the operations on them. For example, we can define a date structure and a time structure, and allow addition of a date instance and a time instance for another date. Without overloading, one would have to implement all these through function calls, making the program less readable.

S++ has applied the operator overloading (type conversion) to allow semi automatic variable conversion. A data type, var, is created. It is possible to use var instances to store numbers and strings.

```
var x = “123”;
cout << (int)x + 123 << endl; // 246
cout << (string)x + “123” << endl; // “123123”
```

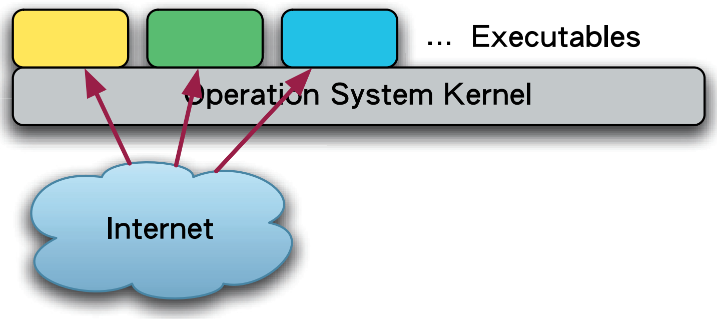
This improves the readability of code as the code is made shorter. Programmers can convert variables from HTTP request to any type that they want at the beginning, and continue to enjoy strong typing feature in the rest of the code.

Security and correctness are also enhanced because programmers are no longer tempted to think of their own ways for the conversion. For example, using thread-unsafe `atoi()` for conversion may bring problems.

S++ intentionally left operations on itself undefined (which is different from other weak typing languages). This forces programmers to choose what data type they would use for the operation. This looks more troublesome than weak-typed languages however it ensures there are not any ambiguities.

4.2 Infrastructure

S++ servlets run as normal Unix processes. They can be individually switched on and off at administrators' command. Unlike CGI, however, individual page requests are treated as threads instead of processes. When a request arrives, S++ creates a thread. The thread is responsible serving the user by receiving and sending HTTP messages. The thread terminates when the user has finished the connection.



Thread is chosen over process partitioning because this allows better data sharing. This saves the cost for interprocess communication. Besides, starting a thread is less expensive than starting a new process because threads completely share the address space. Starting a new process involves some memory page copying at some time (even if copy-on-write is applied, the cost could not be totally eliminated).

4.3 Security

S++ servlets do not interfere each other. They run as individual Unix processes and they are partitioned by the operation systems.

However, thread is a poor mean to partition different requests. Badly written servlets may have requests interfering each others. This should be rectified in the new designs and improved versions.

4.4 Deployment

In order to deploy a S++ program, the program has to be compiled for the platform. Meanwhile, thanks to the compiler, some potential mismatches and grammar errors are filtered.

After the compilation, the program can be started immediately. There is no need for web servers or runtimes. The total cost for having a S++ program running is smaller.

5. Usage of S++

5.1 Choice 1: Complete the Code

If a programmer chooses to start from choice 1, he / she can start from the folder `example-empty`, and start modifying main.cc inside.

5.1.1 Configuration - sxx_config()

Configuration can be done by filling in contents in the `sxx_config()`. Initially, this function is empty. This implies the default settings are used. Programmers can change the settings by altering the following global variables:

Variable	*Meaning*	*Default*
`string sxx_hostname`	Hostname (or IP) of the interface to listen to.	`localhost`
`string sxx_port`	Port number for servlet to listen.	`8080`
`int sxx_backlog`	Number of unserved connections to hold.	`10`
`int sxx_max_threads`	Maximum number of threads at the same time.	`100`
`int sxx_receive_timeout`	Number of seconds to wait for a read.	`10`
`int sxx_transfer_timeout`	Number of seconds to spend for an output.	`60`

5.1.2 Initialize and Destroy - sxx_init() & sxx_destroy()

After calling the configuration, the server attempts to to start listening the socket. If it is successful, before the first request is served, initialization procedure is called. Programmers can specify the procedure in `sxx_init()`.

Before the server shuts down, it calls the destroy procedure. Programmers can specify the procedure in `sxx_destroy()`. A server can be shut down through sending interrupt signal to it.

5.1.3 Main Dish - sxx_handler()

Programmers can define what to do while handling a page request. sxx_handler() is a function with four parameters: `const request& input, response& output, var& page` and `var& session`. Operations on these data structure is discussed in the following sections.

5.1.4 Var

In order to provide type conversion, a var class is designed. A var instance can handle number, string, array or a map. It is also one of the most important class in this project, transferring data through different code sections.

5.1.4.1 Scalar - Number and String

In order to assign a number or string into a var instance, one simply need to do a simple assignment. It is possible to extract the value in another type, therefore providing the effect of conversion.

We do not mean a simple conversion in scripts. We wish to make sure the source string is in correct format. That is, we do not want to treat “4k” as 4.

```
string source = “123”;
var convert = source;
int result = convert;
```

Alternatively, it is possible to use cast-like syntax and make it shorter.

```
string source = “123”;
int result = (var)source;
```

The equivalent C++ and PHP code are as follows.

```
string source = “123”;
int result = 0;
istringstream input(source);
// Throw exception, instead of continue, when anything fails.
input.exceptions(istringstream::failbit);
input >> result >> ws;
// some more checking to ensure there is no trailing characters.

$source = “123”;
if (is_numeric($source)) {
    $result = intval($source);
} else {
    // error procedure.
}
```

5.1.4.2 Vector - Map

A `var` instance can also be an array or a map. Conversion of array or map to any data type yields an exception. Accessing array or map contents of a scalar (number or string) `var` instance is also an exception.

An array and a map is different that array accepts unsigned integer as index and map accepts string.

The index of an array should not get out of bound by more than one, or an exception would be thrown. If it is just exceeded by one, a new element is created.

When an instance is made `const`, it would yield an exception when an element not defined is called. This eliminates the need for programmers check if a variable exist before proceeding.

```
var x;
x[0] = 123;
x[1] = 234;
x[x.size()] = 345; // x.size() is 2 - x[0] and x[1]

const var& x2 = x;
x[100]; // exception thrown!

var y;
y[“a”] = 1;
y[“b”] = 2;

const var& y2 = y;
y[“c”]; // exception thrown!
```

5.1.5 Request Class

Request information from the user is kept in a class called `request`. It does not have any member functions or private data members. It is totally transparent and allows programmers to handle it as if a simple abstract data type. The data members are as follow:

Member	*Meaning*
`string remote_host`	The IP the user is connecting from.
`string remote_port`	The port number the user is connecting from.
`string method`	Method of the HTTP (e.g. GET, POST).
`string host`	Host name given by the Host: header.
`string path`	Path name requested by the web server.
`string protocol`	Protocol reported by the browser.
`map<string, string> headers`	HTTP headers.
`var get`	Parameters sent through query string
`var post`	Parameters sent as POST attachment.*
`var cookie`	Parameters sent as cookie.

* Support for multipart attachment is not implemented.

5.1.6 Response Class

Response information, similar to request information, is transferred through a transparent class. The data members are as follow:

Member	*Meaning*
`int status`	Status code (e.g. 200 for OK, 404 for File Not Found).
`bool connection_close`	Whether to shut down the client after serving this response.
`string location`	URL to redirect, if any.
`string content_type`	Content type of the file to return.
`map<string, string> headers`	Other headers to include in response.
`map<string, var> expire_cookie`	Cookie entries that browser should drop.*
`map<string, var> session_cookie`	Cookie entries that browser should keep as session.*
`map<string, var> persistent_cookie`	Cookie entries that browser should keep for an year.*

* Although var is used, nested structure of `var` (e.g. array of array) is not supported.

5.1.7 Page & Session

`page` and `session` are two extra place holders for state information. Information stored in `page` persists for a single page request and those in `session` persists until the user drops the session id store by a cookie.

5.2 Choice 2: Code Generation

If a programmer chooses to start from choice 2, he / she can start from the folder `example-gen`. The main. cc of `example-gen` is slightly different than the one in `example-empty`. The function `sxx_handler()` is filled with a few lines of code. This is done to allow the code generated in the below procedure gets involved with the final compilation.

The following sections tell the format and usage of the .page file in the pages folder. The remaining part is similar to the methods stated in choice 1.

5.2.1 Syntax

A *page* file can be considered as three sections: *header*, *parameters*, *body*. The partitions are partitioned by "%%" on its own line.

Page files should be put into the *pages* folder, with *.page* as the filename extension. Similarly, *static* content should be put into the *static* folder. There should not be any folders inside the two folders.

5.2.2 Header

Header allows programmer to include header files and define functions. When the intermediate code is generated, the lines are put on the top most, with least modifications.

Meanwhile, it is not necessarily to include the `sxx` library header, nor common C++ libraries (`string`, `map`, `vector`). They are automatically included and they are required to make the intermediate code compilable.

5.2.3 Parameters and Modifiers

Parameters are the variables obtained through different means of communication and storage. Besides specifying priority of the source, programmers can also define how a parameter looks like. These hints and requirements are represented in *modifiers*.

In order to define a parameter, programmers can write in the following format:
<*type*> <*name*> [= <*init_value*>]; <*modifiers*>

5.2.4 Variety of Modifiers

There are three types of modifiers - source, and two levels of restrictions, as well as an optional hint for error:

	Modifier	*Function*
Source	`GET`	Retrive value passed by GET method (query string)
	`POST`	Retrive value passed by POST method
	`ECOOKIE`	Retrive value from cookie. Discard the cookie after this page.
	`SCOOKIE`	Retrive value from cookie and declare it to persist in session.
	`PCOOKIE`	Retrive value from cookie and declare it to persist for 1 year.
	`PAGE*`	Tempoary value for this page request.
	`SESSION`	Tempoary value for this session.
	`READ(filename)`	Retrive value from a file.
	`WRITE(filename)`	Save value to a file after this page.
1st Level Restriction	`MINL(len)`	Minimun length of the input.
	`MAXL(len)`	Maximum length of the input.
	`MINV(val)`	Minimum value of the input.
	`MAXV(val)`	Maximum value of the input.
	`REGEX(exp)`	Regular expression for the input.
	`ASSERT(exp)*`	Assert the expression returns true.
2nd Level Restriction	`MATCH(exp)`	The variable must match with the given expression.
	`MISMATCH(exp)*`	The variable must mismatch with the given expression.
	`ERROR(exp)`	Hint for error string. (Default: name of variable).

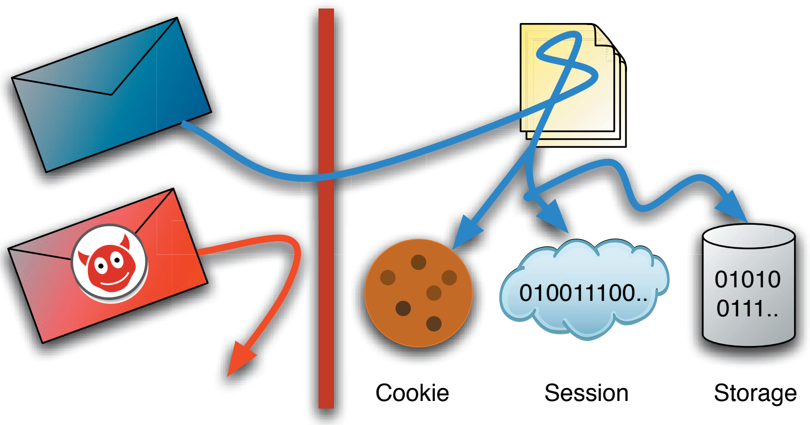
* These modifiers are proposed. However they are not implemented at the time of writing.

`READ()` and `WRITE()` provides file access in the `data` folder. Programmers do not need to worry about the performance of repetitive reading and writing because S++ does have cache for these operations. In future, it is possible to replace it with some efficient means of storage.

`PAGE` is meant to be persistent within a single page request only. `E-`, `S-` and `PCOOKIE` represent expire cookie, session cookie and persistent cookie (1 year) respectively. The three cookies are the same except that they have different expiry time.

`MINL()`, `MAXL()`, `MINV()`, `MAXV()` represents minimum and maximum length and value.

`REGEX()` is used to ensure a parameter value fits into a pattern. Understanding regular expression is not well known for some people, it is suggested to have `ASSERT()`, where programmers can pass the value to an external function for checking. Error is raised if the returned value is false.



5.2.5 Execution of Modifiers

When a parameter is specified, each of the sources is tried. A try is considered successful if and only if all the *first level* restrictions are fulfilled. If not, initialization value is taken, if present, or an exception is thrown.

After determining the value of a variable, it is checked against the *second level* of restrictions. If any of the restrictions fail, an exception is thrown.

When the code segment throws an exception, content in the `ERROR()` modifier is thrown. If this modifier is absent, the name of the variable is thrown.

Thrown exceptions are handled by the `sxx_main()` program. A default field error page is shown with the error shown.

5.2.6 Example

The following is an example about how to apply the checking - password check.

```
%%

string username; GET POST REGEX("^[a-z][a-zA-Z0-9]*$")
string internal_password; READ(username + string(".password"))
string password; GET POST MINL(1) MATCH(internal_password)

%%
```

The equivalent but longer PHP script is as follow:

```
<?php

$username = $_GET['username'];
if (!preg_match("^[a-z][a-zA-Z0-9]*$", $username)) {
    $username = $_POST['username'];
    if (!preg_match("^[a-z][a-zA-Z0-9]*$", $username)) {
        die ("Invalid field: username");
    }
}

$internal_password = @file_get_contents($username . ".password")
                         or die ("Invalid field: internal_password");

$password = $_GET['password'];
if (strlen($password) < 1) {
    $password = $_GET['password'];
    if (strlen($password) < 1) {
        die ("Invalid field: Password Mismatch");
    }
}

if ($password != $internal_password) {
    die ("Invalid field: Password Mismatch");
}

?>
```

Username is first determined. It is checked that the username to be alphanumerical only, with a mandatory small letter at the front.

Internal password is second obtained from the file. Since the above line has made sure there is no evil value in username, it is save to take the username to obtain the internal password.

Finally, user provided password is obtained. It is checked that the password has to match with the internal password. `MINL(1)` is used to determine if a variable is intentionally filled. (It is normal for web browsers to submit null values, if there are unfilled fields.)

In order to shield the error of failure (as a good practice for system security), `ERROR("Password Mismatch")` can be placed on each of the lines in parameter section. No matter what the reason is, the user gets the given string as error.

Variables mapped with `E-`, `S-`, `PCOOKIE`, `PAGE`, `SESSION`, `WRITE()` modifiers are updated when the code finish without exception thrown. For instance, the following shows how to implement a session counter.

```
%%

int count = 0; SESSION

%%

output.content << "Count = " << ++count << endl;
```

5.2.7 Body

The body can be considered as the one we find in a function. The function header is filled by the generator. Some of the macros functions are available.

Macro	*Function*
`sxx_forward(exp)`	Silently forward the execution to another page.
`sxx_include(exp)*`	Silently include another page and continue execution.
`sxx_redirect(exp)`	Put the new URL into Location: field, and return immediately.
`sxx_read(filename)`	Read the given file name in data folder.
`sxx_write(filename, exp)`	Write to the given file name in data folder.

* Proposed but not implemented.

5.3 Compile and Execution

To compile, issue a "`make`" command at the base folder of the project (e.g. `example-empty`). If there is no error, an executable `servlet` is created.

If method 2 is employed for development, it is necessarily to have the `data` and `static` folder writable by the servlet.

The executable can be started by issuing "`./servlet`". It blocks the standard input and output normally. It can get gracefully terminated when it receives interrupt (Ctrl-C). Some other signals, for example, hang-up signal, are ignored.

When the server terminates, it wait for all the threads to finish and the file cache are written back to the persistent storage.

6. Code Summary

6.1 Classes

In order to reduce software dependency, class is applied to partition the codes. This make the code more manageable. Each class has its special duty to handle. Some of them are built in order to represent information while some others are façade classes (i.e. encapsulating system calls).

However, applying class does not mean to be object oriented programming. For example, inheritance and polymorphism are not applied through out the code. Unlike Java Servlet, S++ programmers do not need to understand these theories.

The following is the list for the classes built.

Class	*Function*
`var`	Represent common data types (e.g. string, numbers) as well as array.
`request`	Represent HTTP request from web user.
`response`	Represent HTTP response from the servlet.
`client_socket`	Façade class for active socket (Socket in Java).
`server_socket`	Façade class for passive socket (ServerSocket in Java).
`request_parser`	Interact with client_socket and return request instances.
`response_generator`	Express response instances back to the client_socket.
`file_manager`	File input, output and caching. (Singleton)
`session_manager`	Session creation and timeout handling. (Singleton)

6.2 Execution Flow

Similar to C++ programs, S++ starts with the `main()` function in `sxx_main.cc`. The `main()` function is responsible to glue the classes together to work.

`main()` runs programmer-provided `sxx_config()` for configuration information. This includes IP and port number to listen to, number of seconds to wait for a read operation, etc.

After getting the configuration information, main() instantiates server_socket and calls the member function `accept_client()` in order to start listening to the given IP and port number.

Whenever there is a new connection, `accept_client()` returns with a `client_socket` instance. `main()` creates a thread, executes `sxx_client_handler()` to serve the connection.

The client thread instantiates a `request_parser` to accept the HTTP requests. Corresponding session information is provided by `session_manager`. Obtained requests and session information are passed to programmer-provided `sxx_handler()` for processing. Afterwards, the result (in form of response instance) is passed to `response_generator` for HTTP message.

It is possible that exceptions are thrown, uncaught by the `sxx_handler()` and the stack unwinds to `sxx_client_handler()`. In this case, the function generates a small error page, reporting what is caught.

Some signals are caught by the functions in `sxx_main.cc`. For example, interrupt signals are caught and the program gracefully terminates after receiving the code. Hang-up signals are caught and ignored, however. Segmentation faults are also caught and the offending thread gets terminated.

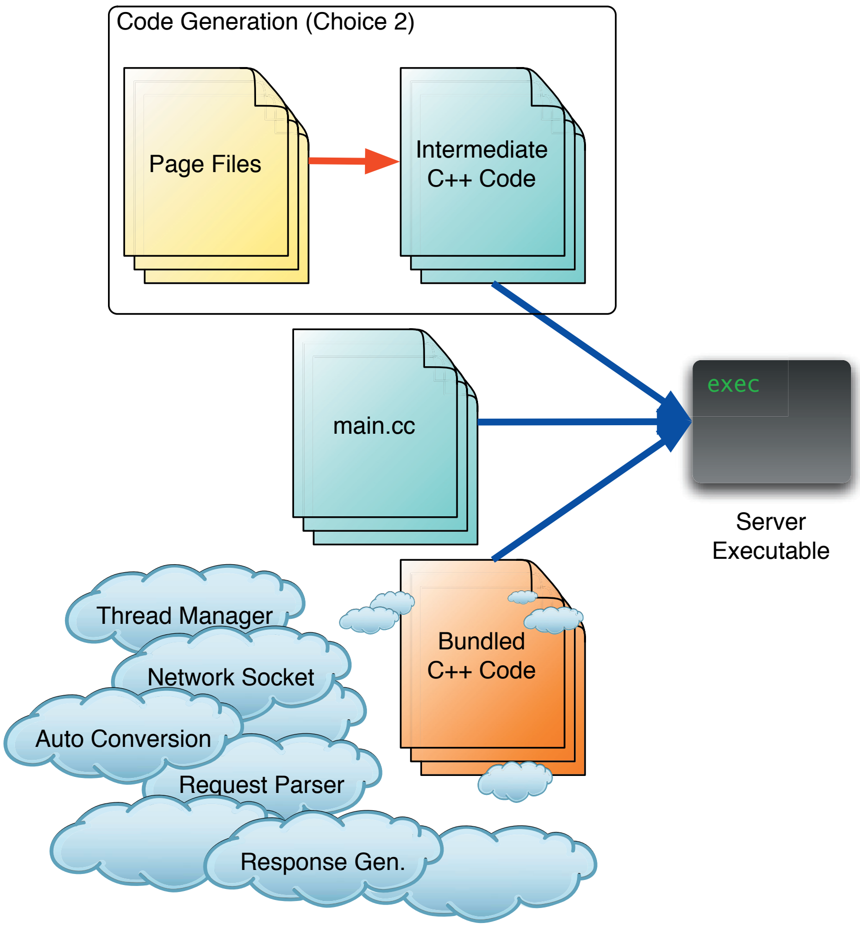
6.3 Compilation

Compilation of the program is done with the help of `Makefile`, read by `make`. `make` is a traditional tool to coordinate generation of various files. It understands the dependency of files and decide which files to be updated. The update process can be specified in `Makefile`.

When `make` is issued in a web application folder, it first makes sure the `sxx` (bundled code) is up to date by issuing another make command in the folder.

If choice 2 is selected, `make` reads `.page` files in `pages` and generate another `Makefile` in the folder. By issuing another `make` in the folder, it generates intermediate C++ code and then compiled parts. Besides, a few default pages in the web application folder is compiled.

Finally but not the least, the `main.cc` is compiled with all the above-mentioned parts joined.



7. Possible Applications

7.1 Embedded Systems

Designers can apply S++ without worrying if their systems are strong enough to support sophisticated software - S++ simply does not require any of them as companion.

7.2 Quick Programming

Since parameters can be checked and obtained easily in S++, it can be applied for quick programming without losing security against malicious input.

7.3 Programming Assignments

A number of computer science students learn C++ as their first language. S++ can be taken as a choice to apply what they have learnt directly.

8. Review of the Solution

In this section, we review if problems related to dynamic web page in C++ are solved, as well as related difficulties and findings.

8.1 Automatic Conversion

S++ succeed providing automatic conversion. With the `var` class, it is possible to convert string to number and vice versa. On the other hand, it fails to provide automatic conversion.

Conversion to any data type is believed to be possible because C++ supports templates. Templates allow parameter of functions to be expressed in generic form, and have the concrete functions implicitly created on demand. Whenever there is a new type to convert, a new function can be silently generated.

On the other hand, we met the wall. For example, it is not possible to have an above-mentioned converted to string. Thanks to the variety of string constructors, the compiler gets confused because there are multiple pathways for conversion:

```
var -> int -> string
var -> char* -> string
var -> string -> string
```

The procedure to disable some of the pathway is not known. Therefore we fall back to using explicitly defined conversion to string and integers.

8.2 Program Safety

Writing web program with S++ can be safer than other solutions.

First of all, S++ is designed to use C++ style programing, instead of C-style. The interface of S++ does not involve pointers. (The internal involves a few.) For example, we use `string` instead of `char*`, `vector<>` instead of array. The danger of having dangling pointers and memory leakage is much reduced.

Secondly, S++ provides some encapsulation of system calls. The wrapped interfaces, once again, do not have pointers directly involved. Programmers can exchange data with the class through safer mechanism.

On the other hand, S++ fails to make the environment completely safe.

S++ does not provide enough partitioning between the connections. The connections are served by threads of the same process. Threads within a single process are not partitioned. Although S++ can shut down the offending thread if memory access violation is detected, it does not ensure data in the process is intact, nor telling where the problem is.

Even worse, S++ does not disallow programmers to use their own dangerous C code. Although some people consider this as a flexible design, this can be also viewed as dangerous.

To ensure safety, we need to consider more about process partitioning and even defining a new language. They will be discussed in the following sections.

8.3 System Call Encapsulation

Except for socket, S++ does not provide a lot of usable encapsulation of system calls. It is because S++ is a user-space program with limited area of usage.

One of the regret, however, is that we lack façade class for locking. Locking is essential to protect critical region of a program, improving the correctness and stability.

The reason not to implement locking is, failure to unlock after a thread gets killed can lock the whole system. Besides, some programmers may get confused with locking mechanism and get into trouble, such as dead lock and starvation.

8.4 Ease of Use

S++ simplifies some of the task while complicating others.

Parsing and converting user input is made simpler. Programmers do not need to think of their own way to do the capturing and conversion. The tasks are handled by the *bundled code*.

On the other hand, the compilation process is made very difficult. In order to prepare a S++ program, programmers need to put their source files at the correct location, change the makefile (in order to get extra source files compiled) and get messed with lines of make output.

8.5 Choice C++ Language

Writing web program in C++ is a strange (or novel) idea, as questioned by a lot of people.

C++ allows better flexibility. For example, its operator overloading is superior and there are not any counterparts in modern languages. Operator overloading allows creation of a `var` class with ease, and the saved effort was spent on other issues and interesting designs. If other languages had been chosen, we would have spent time thinking how to compensate the loss.

Writing program in C++ is challenging. After working for this year, I insist that writing a project in C++ is a fruitful experience. C and C++ are difficult to handle because there are a lot of pointers involved. Getting these done correctly is a very good mental exercise.

However, it is no longer necessary to rely on C++ in the future design.

One of the examples is parameter checking. Although it is implemented in C++ for the time being, there are not any restrictions for it to be applied on other languages, from scripts to compiled. All the complications of the target language can be hidden by the *code generator*.

Operator overloading is no longer a valid reason to stick with C++. With the help of code generation, we can emulate operator overloading with function overloading. This allows us to choose other languages for implementation.

9. Possible Variations

9.1 Objects, Services and Templates

In the previous sections, it is shown how convenient it can get to represent a *page*. We can extend the idea to *objects*, *functions* and *templates*.

Object is used to store data. They can be used for grouping a few basic data pieces together. Advantages of using generated object over original C++ class is, programmers do not need to specify the code for getter, setter, and other simple elements. The generator can also generate code to serialize the instances for linear storage.

Services are essentially non-accessible pages. Using services instead of pages ensures user cannot access the page directly and cause unexpected result. Functions can be used for repetitively called procedures among a lot of pages. Examples include password checking, including the common header for output, etc.

Templates are another type of pages. They are different in the sense that they focus on output, instead of computation. In a template file, anything recorded in the body are output directly.

These three extra things look simple. However, we have to note that parsing the modifiers with original C++ code is not trivial. This is one of the reasons why only page is implemented.

9.2 Process Partitioning

In the current design, each of the connections is handled by a thread. There is not much partitioning and protection between threads except they use their own function stack. The security is not strong. This is a compromise to enable users to share global variables across different processes without knowing how to make interprocess calls.

When the idea of parameter mapping appears, the need to share global variables become less. With the help of modifiers, we can mark some variables to be global, or map it to a common file. It is then S++'s responsibility to make interprocess calls whenever sharing is required.

Considering this fact, it is now feasible to change this unsafe threading mechanism to a process-based partitioning without forcing programmers to learn interprocess communications.

This design is more fault-tolerant than the current design. The processes are partitioned by the operation system. If a computation of a single web page shows problem, the part of computation can be dropped neatly by killing the process. If we collect all the update till the end of serving a page, the status can be rolled back easily.

9.3 Persistent Storage

In the current design, it is possible to map a variable to a file only. This is design very inefficient and inconvenient. For example, in order to represent scores of 100 students in 4 assignments, we need 400 files. Operation system allocates 4 kilobyte to each of the files, wasting a lot of disk spaces. It is also troublesome to issue 400 read requests just for reading the table.

It is proposed to have a tabular method of data access. Instead mapping a variable to a file, we map a variable to a cell of a CSV table. Programmers can represent this kind of operations by providing modifiers. An example is as follow.

```
%%

string student; GET POST REGEX(“^[a-z][a-zA-Z0-9]*$”)
int row_number; LOCATE_ROW(“score_list”, 1, student) // search col 1 for student
int quiz1; TABLE(“score_list”, row_number, 2)
int quiz2; TABLE(“score_list”, row_number, 3)
...

%%
```

Similar to mapping variable to a file, a table can be cached and error conditions can be automatically handled. Besides, according to the access pattern to a table, it is possible to make decision and build some dynamic indices for better searching.

9.4 New Data Types

In C++, size of primitive data type is different on different machines. This complicates the amount of work to check in compiling, and hampers the portability of code.

We can avoid this problem on S++ servlets by providing a new set of data types. We may include replacement for C++ primitive types, as well as some types related to web programming (e.g. date). Thanks to operator overloading function, we can define operators for the classes.

Besides, defining new types allows better arithmetic security. C++ does not generate exceptions when there are arithmetic errors such as overflow and underflow. By redefining the operations, we can put this checking back.

Last but not the least, we can implement a new string class. The new class can be designed to support UTF-8 and regular expression operations. Examples include counting amount of characters instead of bytes, and allows partial replacement of a string.

However, we have to note that besides the five operations (+, -, *, /, %), there are also math functions in `math.h`, as well as string functions. Careful planning and investigation is required to obtain an optimal set of operations and functions.

10. New ideas

After working for half an year, some new ideas are collected. Some of the ideas involve fundamental changes to the project. Due to the time constraint, it is not possible to change the design and implement them.

10.1 Application of Unix Tools

10.1.1 lex and yacc

In the current S++ design, a “hand-built” parser is applied for HTTP messages.

Building (and probably maintaining) of the parser is a difficult task. HTTP messages is not a simple combination of read line processes.

The parser is not efficient in the sense that it reads byte by byte. On the other hand, if we chose to build a buffer, we may end up further complicating the parser design.

If we redesign the parser with the help of lex and yacc, the parser source code will be more readable. The product may also be faster because these tools have been built for decades and they are optimized to do their job well.

10.1.2 nc

The parser relies on the specific socket façade class. This design has high software dependency and looks unnecessary given we have `nc` for piping network signals to pipes.

The challenging part is that `nc` is cannot serve more than one connections at a time. We may need to redesign it to suit our need.

The benefits of applying `nc` is, we can choose not to get pipe from `nc`, and reuse the software in other means. This includes running as CGI (pipe from web server).

10.2 New Language for Programmers

After working for half an year with C++, I have accumulated experience with the language. However, I have also noticed redesigning a new language is a good idea. It masks dangerous properties of C++ and allows more possibilities for better web programming.

10.2.1 Elimination of Pointers

One of the most criticized and dangerous features of C and C++ is pointer. Ability to point at anywhere is a dangerous source of runtime error.

In the new language, parameters can be only passed as value and reference.

In order to improve efficiency of pass by value, value assignments, and returning from functions, copy-on-write can be applied.

10.2.2 New Definition of "Type"

In traditional programming, a type is identified if it is a string, a numeric value, or array of something.

This design is not satisfactory for the Internet world because everything is string and we need better differentiation between the types. For example, date and English name are both strings. However it is a logical error to mix them up.

Inspired by the parameter checking in S++, a new idea comes into mind. We can first make a very brief separation - scalar and array, and provide flexible checking mechanism on top of them.

A scalar type can be represented in regular expression. Programmers are allowed to define operations with types (e.g. addition, multiplication) and possible conversions (e.g. integer -> float).

A scalar variable is defined as string at the beginning. In order to facilitate different operations, its type can be changed on the run time through checking. After variable passes checking for a type, the positive result is cached until the variable is changed. This design allows repetitive checking of variable types without huge penalty.

After getting the type changed, the behavior of operations on the variable changes. When there is type mismatch at any part of code, exception is thrown.

```
var str1 = “123”;
var str2 = “234”;
var str3 = str1 + str2; // “123234”
str1(NUMBER); // str1 is now a number.
str2(NUMBER); // str2 is now a number.
var str4 = str1 + str2; // “358”
```

Besides scalar string, we need an expandable array (with index). Both the indices and contents of the array can be checked with the fore-mentioned mechanism.

```
var ar1;
ar1[0] = “1”;
ar1[1] = “2”;
ar1[] = 3; // ar[2] = 3;
ar1(ARRAY(NUMBER)); // everything inside ar1 is now treated as number.
```

10.2.3 Numeric Operations

Numbers can be represented as string, instead of binary. This feature exists on some languages such as Perl. This feature avoids number overflow and underflow. Processor speed increases faster than harddisk and network media. The bottleneck is not on the processor. This design allows better use of the extra processor cycles. Moreover, string conversion from web and to database becomes easier.

11. Conclusion

In this project, a framework for developing dynamic web server is developed.

S++ is easier to be used in some aspects. For example, the code for doing type conversion is shortened while the strictness of conversion is increased. This allows programmers to express their commonly used operations in a faster way.

Meanwhile, S++ has a lot of space to improve. Examples include better type conversion, better process partitioning and better programmer experience.

Although I am not confident with the code quality and I do not believe S++ can be applied directly, I believe the idea obtained behind is useful for others. I am also optimistic in improving the software in the future.