COMP 3031 Assignment 2
Flex and Bison Programming
Fall 2014

Due: 5:00pm on Nov. 13, 2014 (Thursday)

Instructions

- Put your *name, ITSC account name, and student ID* (using C style comment /* comment */) on the first line of all your files: *parseconfig.lex*, *parseconfig.y*.
- Zip the following three files into a single package using *exactly* the following command (case-sensitive):

  ```
  zip parseconfig.zip Makefile parseconfig.lex parseconfig.y
  ```

- Submit your zipped file *parseconfig.zip* through the course assignment submission system (CASS) of COMP3031 (do not submit to other courses) before the deadline: `https://course.cse.ust.hk/cass/submit.html`.
- Instructions on using CASS are available at: `http://cssystem.cse.ust.hk/home.php?docbase=UGuides/cass&req_url=UGuides/cass/student.html`.
- *No* late submissions will be accepted.
- Your submission will be compiled and run on a lab 2 machine by the following commands (we will test your program by many different configuration files):

  ```
  unzip parseconfig.zip
  make
  ./parseconfig ...
  ```

- *Make sure your submission can be ran using the above commands. If it can not be compiled or ran properly, you may get 0 marks for this assignment!*

1 Problem Description

Write flex and bison programs to implement a firewall configuration file parser. Your task is to maintain an IP address pool and output the final IP addresses after the parsing process. A typical configuration file is written in this form:

```
interface eth0
{
    pool 0.0.18.0/30;
    pool 0.0.1.100-0.0.1.105;
}
whitelist
{
    0.0.1.2, 0.0.1.201, 0.0.3.4;
}
blacklist
{
    0.0.18.35, 0.0.1.103, 0.0.18.3;
}
whitelist
{
    0.0.18.20;
}
```

The output of your program for this configuration file should be:

```
eth0:
0.0.1.2
0.0.1.100
0.0.1.101
0.0.1.102
0.0.1.104
0.0.1.105
0.0.1.201
0.0.3.4
0.0.18.1
0.0.18.2
0.0.18.20
```

The configuration file consists of three sections: interface, whitelist and blacklist. The interface section defines the initial IP address pools. The whitelist section defines the additional IP addresses which should be added to the interface pools. The blacklist section defines the IP addresses which should be removed from the interface pools.

The interface section appears at the *beginning* of the file and appears *only once*. There can be *zero or more* whitelist and blacklist sections in the file.

2 Interface Section

The interface section may have *many* IP address pools. The IP address pool can be defined in one of the following two formats:

```
pool minaddress-maxaddress;
pool address/prefix;
```

Then, the interface section is defined as:

```
interface INTERFACENAME
{
    pool minaddress-maxaddress;
    pool address/prefix;
    ...
}
```

The interface section is *required* and appears *exactly once at the beginning of the file*. It can contain *zero or more* pool definitions.

IP Address Definition

An IP address[1] is a 32-bit number and is canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots, e.g., 0.0.35.1 (binary format: `00000000.00000000.00100011.00000001`, unsigned int format: 8961). In the binary format, each part represents a group of 8 bits of the address.

IP Slash Notation

As aforementioned, there is a total of 32 bits in IP address space. A network numbered "0.0.1.0/24" is a network with 24 bits of prefix, denoted by "/24" after the slash. In this assignment, assume the prefix length is l, the valid range of an IP slash notation is from 1 to $2^{32-l} - 1$ (inclusive). Table 1 provides examples for IP slash notation and equivalent IP range.

IP Slash Notation	Binary Format	Equivalent IP Range
`0.0.1.0/24`	`00000000 00000000 00000001 ********`	`0.0.1.1-0.0.1.255`
`0.0.18.0/30`	`00000000 00000000 00010010 000000**`	`0.0.18.1-0.0.18.3`

Table 1: IP Slash Notation Example

In the IP slash notation, the min address and max address can be obtained as follows: in the wildcard bits, set the lowest bit to "1" and all the other bits to "0" and you get the min address; set all the wildcard bits to "1" and you get the max address. For instance, for the first example in Table 1, the min and max address are obtained by `00000000 00000000 00000001 00000001` and `00000000 00000000 00000001 11111111`, respectively.

Actions for the Interface Section

We store the interface pools in an int array "ipmap":

```
int ipmap[MAXIPNUM];
```

If an IP is in the pool, the corresponding element (unsigned int format of this IP) of "ipmap" is set to 1; otherwise, the corresponding element is set to 0.

For each pool definition, you should set the corresponding elements of the "ipmap" to 1 in your parser program. You can do this by the provided two C functions:

```
void addsubnet(const char *ipstr, int prefixlen);
void addrange(const char *ipminstr, const char *ipmaxstr);
```

[1]Here we only discuss IPv4 addresses.

3 Whitelist Section

The whitelist section is composed of an IP address list. It is defined as:

```
whitelist
{
    IP, IP, ... IP;
}
```

There is *at least one* IP in this section.

Actions for the Whitelist Section

For each IP in the whitelist: if the IP is in one of the interface pools, set the corresponding element of "ipmap" to 1; otherwise, no actions needed. You can do this by the provided function:

```
void addip(const char *ipstr);
```

4 Blacklist Section

The blacklist section is composed of an IP address list. It is defined as:

```
blacklist
{
    IP, IP, ... IP;
}
```

There is *at least one* IP in this section.

Actions for the Blacklist Section

For each IP in the blacklist: if the IP is in one of the interface pools, set the corresponding element of "ipmap" to 0; otherwise, no actions needed. You can do this by the provided function:

```
void deleteip(const char *ipstr);
```

5 Output

The output format of the parser program is (the IP list should be printed in ascending order):

```
INTERFACENAME:
IP1
IP2
.
.
.
```

You can output the IP list (*no INTERFACENAME*) by the provided function:

```
void outputiplist();
```

6 Your Work

We provide a code skeleton for you. The functions we provided are self-explanatory by their names.

Makefile is also given. You can compile this project on our server by a simple command "make". You can clean the project by typing "make clean".

Three sample configuration files are provided. After unzip and compilation, you can run the program by typing "./parseconfig config.txt". The output looks like:

```
zsuab@ras1:~/test$ unzip skeleton.zip
Archive:  skeleton.zip
  inflating: config1.txt
  inflating: config2.txt
  inflating: config.txt
  inflating: Makefile
  inflating: parseconfig.lex
  inflating: parseconfig.y
zsuab@ras1:~/test$ make
bison -d parseconfig.y
parseconfig.y: conflicts: 1 shift/reduce
parseconfig.y:42.8: warning: rule never reduced because of conflicts: command: /* empty
flex -oparseconfig.c parseconfig.lex
make: Warning: File 'parseconfig.tab.c' has modification time 1.4 s in the future
gcc -o parseconfig parseconfig.c parseconfig.tab.c
make: warning:  Clock skew detected.  Your build may be incomplete.
zsuab@ras1:~/test$ ./parseconfig config.txt

^1: syntax error at interface #105
```

Your tasks are listed below:

1. Add missing flex definitions in *parseconfig.lex*.
2. Add missing flex rules in *parseconfig.lex*.
3. Add necessary tokens in *parseconfig.y*.
4. Add remaining grammar rules in *parseconfig.y*.

We have marked these missing parts by:

```
/************* Start: ... */
/* End: ...*/
```

7 Remarks

1. Assume there is no IP address overlap between any pair of pools in the interface section.
2. Assume all test configuration files are syntactically correct.
3. To simplify the problem, assume the leftmost 16 bits of all IP address are "0", namely the IPs will be in this form: "0.0.x.x". As a result, there are at most $2^{16} = 65536$ IP addresses.

4. There may be redundant whitespace, tab and newline between tokens in the file, your program should be able to skip all these spaces properly to avoid parsing error. You can find such examples in the sample configuration files provided (*config.txt*, *config1.txt*, *config2.txt*).

5. You *must* output exactly the same format as defined in the output section as we will test your program by *diff* the output file and standard answer.

6. Do not modify *Makefile*.

8 BNF of the Configuration File

$$< S >::= interface < NAME >< POOLBLOCK >< LIST >$$
$$< POOLBLOCK >::= \{< POOLS >\}$$
$$< POOLS >::= < empty > | < POOLS >< POOLSLASH > | < POOLS >< POOLRANGE >$$
$$< POOLSLASH >::= pool < IP > / < PREFIX >;$$
$$< POOLRANGE >::= pool < IP > - < IP >;$$
$$< LIST >::= < empty > | < LIST >< WHITELIST > | < LIST >< BLACKLIST >$$
$$< WHITELIST >::= whitelist\{< WIPLIST >< IP >;\}$$
$$< WIPLIST >::= < empty > | < WIPLIST >< IP >,$$
$$< BLACKLIST >::= blacklist\{< BIPLIST >< IP >;\}$$
$$< BIPLIST >::= < empty > | < BIPLIST >< IP >,$$
$$< NAME >::= < LETTER >< LETTERORDIGIT >$$
$$< LETTERORDIGIT >::= < empty > | < LETTERORDIGIT >< LETTER > | < LETTERORDIGIT >< D >$$
$$< LETTER >::= a|b|c \ldots z|A|B|C \ldots Z$$
$$< PREFIX >::= < D > |1 < D > |2 < D > |30|31|32$$
$$< IP >::= < I > . < I > . < I > . < I >$$
$$< I >::= < D > | < N >< D > |1 < D >< D > |2 < F >< D > |25 < V >$$
$$< D >::= 0|1|2|3|4|5|6|7|8|9$$
$$< N >::= 1|2|3|4|5|6|7|8|9$$
$$< F >::= 0|1|2|3|4$$
$$< V >::= 0|1|2|3|4|5$$