Makefile.am

```
# ---------------------------------------------------------------------------
#
# RuleWall: A Firewall Configuration Parser
# Copyright (C) 2006 Benjamin Gaillard
#
# ---------------------------------------------------------------------------
#
#        File: src/Makefile.am
#
# Description: Automake Makefile
#
# ---------------------------------------------------------------------------
#
# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the Free
# Software Foundation; either version 2 of the License, or (at your option)
# any later version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
# more details.
#
# You should have received a copy of the GNU General Public License along
# with this program; if not, write to the Free Software Foundation, Inc., 59
# Temple Place - Suite 330, Boston, MA 02111-1307, USA.
#
# ---------------------------------------------------------------------------


# Flags
AM_CPPFLAGS = -D_POSIX_SOURCE -D_BSD_SOURCE
AM_LFLAGS   = -p -p -s
AM_YFLAGS   = -d

# Source files
bin_PROGRAMS = rulewall
rulewall_SOURCES = \
    main.c \
    memory.c \
    memory.h \
    parser.y \
    lexer.l \
    structs.c \
    structs.h \
    iptables.c \
    iptables.h

# Extra files to include in the distribution archive
EXTRA_DIST = Unimakefile.mk

# End of File
```

main.c

```
/* ---------------------------------------------------------------------------
 *
 * RuleWall: A Firewall Configuration Parser
 * Copyright (C) 2006 Benjamin Gaillard
 *
 * ---------------------------------------------------------------------------
 *
 *        File: src/main.c
 *
 * Description: Main Function
 *
 * ---------------------------------------------------------------------------
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program; if not, write to the Free Software Foundation, Inc., 59
 * Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * ---------------------------------------------------------------------------
 */


/*****************************************************************************
 *
 * Headers
 *
 */

/* System headers */
#include <stdlib.h> /* NULL, malloc(), free()               */
#include <stdio.h>  /* puts(), fputs(), printf(), fprintf() */
#include <string.h> /* strcmp()                             */

/* Configuration */
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif /* HAVE_CONFIG_H */

/* Local headers */
#include "structs.h"
#include "iptables.h"
#include "memory.h"


/*****************************************************************************
 *
 * Local Functions
 *
```

```c
 */

/* Version string */
#ifndef PACKAGE_VERSION
#define PACKAGE_VERSION "<unknown>"
#endif

/* Prototypes */
static void usage(const char *exe);

/*
 * Display the program usage help message
 */
static void usage(const char *const exe)
{
    /* Cut in two, because ISO C compilers are required to accept strings of
     * only 509 bytes at least */
    printf("Syntax:␣%s␣[options...]␣[files...]\n"
           "\n"
           "Available␣options:\n"
           "␣␣␣␣-c/--color:␣␣␣␣␣␣␣␣␣use␣colors␣for␣the␣dump\n"
           "␣␣␣␣-d/--dump:␣␣␣␣␣␣␣␣␣␣dump␣the␣configuration␣structures\n"
           "␣␣␣␣-e/--exe:␣␣␣␣␣␣␣␣␣␣␣IPTables␣executable␣name␣(\"iptables\"␣by"
                   "␣default)\n"
           "␣␣␣␣-h/--help:␣␣␣␣␣␣␣␣␣␣display␣this␣help␣message\n"
           "␣␣␣␣-i/--iptables:␣␣␣␣␣␣generate␣an␣IPTables␣shellscript\n"
           "␣␣␣␣-n/--no-color:␣␣␣␣␣␣don’t␣use␣colors␣for␣the␣dump\n"
           "␣␣␣␣-o/--output␣<file>:␣output␣filename\n"
           "␣␣␣␣-v/--version:␣␣␣␣␣␣␣display␣the␣program␣version\n"
           "\n", exe);
    puts("You␣can␣specify␣any␣number␣of␣files␣in␣the␣command␣line,␣Use␣"
                 "\"-\"␣for␣the\n"
         "standard␣input␣as␣long␣as␣chain␣names␣are␣all␣different.␣␣If␣no␣"
                 "filename␣is\n"
         "given,␣the␣standard␣input␣is␣read.\n"
         "\n"
         "If␣an␣option␣is␣given␣more␣than␣once,␣the␣last␣one␣takes␣"
                 "precedence.\n"
         "\n"
         "Note␣about␣colors:␣by␣default,␣colors␣are␣used␣if␣the␣IPTables␣"
                 "script␣isn’t\n"
         "generated.␣␣This␣is␣not␣to␣\"corrupt\"␣the␣script␣if␣it␣is␣"
                 "edited.\n"
         "\n"
         "Thank␣you␣for␣using␣RuleWall!");
}


/*****************************************************************************
 *
 * Global Functions
 *
 */

/* Defined in parser.y */
extern struct chain *parse_config(const char *const filename);

/*
 * Main function
 */
```

```c
int main(const int argc, const char *const *const argv)
{
    /* Files and generated config */
    const char **const files
            = malloc(sizeof(char *) * (argc > 1 ? argc - 1 : 1));
    unsigned nb_files = 0;
    const char *out_file = NULL;
    FILE *output;
    struct chain *config, *last;
    const char *exe = "iptables";

    /* Command line options */
    enum {
        COLORS_DEFAULT, COLORS_FALSE, COLORS_TRUE
    } use_colors = COLORS_DEFAULT;
    enum bool do_dump = FALSE, do_iptables = FALSE, do_usage = FALSE;
    enum bool do_version = FALSE, do_output = FALSE, do_exe = FALSE;

    /* Counters */
    unsigned i, j;

    if (files == NULL) {
        fputs("Not␣enough␣memory!␣Aborting.\n", stderr);
        return 10;
    }

    /* Parse options */
    for (i = 1; i < (unsigned) argc; i++) {
        if (do_output == TRUE) {
            do_output = FALSE;
            out_file = argv[i];
        } else if (do_exe == TRUE) {
            do_exe = FALSE;
            exe = argv[i];
        } else if (argv[i][0] == '-') {
            if (argv[i][1] == '-') {
                if (strcmp(argv[i] + 2, "color") == 0)
                    use_colors = COLORS_TRUE;
                else if (strcmp(argv[i] + 2, "dump") == 0)
                    do_dump = COLORS_TRUE;
                else if (strcmp(argv[i] + 2, "exe") == 0)
                    do_exe = TRUE;
                else if (strcmp(argv[i] + 2, "help") == 0)
                    do_usage = TRUE;
                else if (strcmp(argv[i] + 2, "iptables") == 0)
                    do_iptables = TRUE;
                else if (strcmp(argv[i] + 2, "no-color") == 0)
                    use_colors = COLORS_FALSE;
                else if (strcmp(argv[i] + 2, "output") == 0)
                    do_output = TRUE;
                else if (strcmp(argv[i] + 2, "version") == 0)
                    do_version = TRUE;
                else {
                    fprintf(stderr, "Error:␣invalid␣option␣\"%s\".\n"
                            "Use␣-h␣or␣--help␣for␣a␣full␣list.\n",
                            argv[i]);
                    return 1;
                }
            } else {
                for (j = 1; argv[i][j] != '\0'; j++)
                switch (argv[i][j]) {
                case 'c':
                    use_colors = COLORS_TRUE;
                    break;

                case 'd':
                    do_dump = TRUE;
                    break;

                case 'e':
                    if (do_output == TRUE) {
                        fputs("Error:␣cannot␣use␣\"-e\"␣and␣\"-o\"␣at␣the"
                              "␣same␣time.\n", stderr);
                        return 2;
                    }
                    do_exe = TRUE;
                    break;

                case 'h':
                    do_usage = TRUE;
                    break;

                case 'i':
                    do_iptables = TRUE;
                    break;

                case 'n':
                    use_colors = COLORS_FALSE;
                    break;

                case 'o':
                    if (do_exe == TRUE) {
                        fputs("Error:␣cannot␣use␣\"-e\"␣and␣\"-o\"␣at␣the"
                              "␣same␣time.\n", stderr);
                        return 2;
                    }
                    do_output = TRUE;
                    break;

                case 'v':
                    do_version = TRUE;
                    break;

                default:
                    fprintf(stderr, "Error:␣invalid␣option␣\"-%c\".\n"
                            "Use␣-h␣or␣--help␣for␣a␣full␣list.\n",
                            argv[i][j]);
                    return 1;
                }
        }
    } else
        files[nb_files++] = argv[i];
}

/* If help message is requested */
if (do_usage == TRUE) {
    usage(argv[0]);
    return 0;
}
```

```c
/* If version is requested */
if (do_version == TRUE) {
    puts("RuleWall␣version␣" PACKAGE_VERSION "\n"
         "Copyright␣(C)␣2006␣Benjamin␣Gaillard\n"
         "This␣program␣is␣covered␣by␣the␣GPL␣licence␣version␣2");
    return 0;
}

/* Check is at least one action has been given */
if (do_dump == FALSE && do_iptables == FALSE) {
    fputs("Error:␣no␣action␣selected.␣␣Use␣-d/--dump␣and/or␣"
          "-i/--iptables,␣or␣-h/--help\nfor␣a␣full␣list␣of␣options.\n",
          stderr);
    return 2;
}

/* Check for output file */
if (do_output == TRUE) {
    fputs("Error:␣-o/--output␣option␣used,␣but␣no␣file␣specified.\n",
          stderr);
    return 2;
}
if (out_file == NULL || (out_file[0] == '-' && out_file[1] == '\0'))
    output = stdout;
else if ((output = fopen(out_file, "w")) == NULL) {
    fprintf(stderr, "Error:␣cannot␣write␣to␣file␣\"%s\":␣", out_file);
    perror(NULL);
    return 3;
}

/* Enable colors if desired */
if (use_colors == COLORS_DEFAULT)
    use_colors = do_iptables ? COLORS_FALSE : COLORS_TRUE;

/* Parse files */
if (nb_files == 0) {
    /* If no file was given, use standard input */
    files[0] = "-";
    nb_files = 1;
}
if ((last = config = parse_config(files[0])) == NULL)
    return 4;
for (i = 1; i < nb_files; i++) {
    /* Look for the last chain */
    while (last->next != NULL)
        last = last->next;

    /* Append the new read chains to the list */
    if ((last->next = parse_config(files[i])) == NULL)
        return 4;
}

/* Free some memory */
free(files);

/* Header, for IPTables script */
if (do_iptables == TRUE) {
    fputs("#!/bin/sh\n\n"
          "#␣This␣script␣has␣been␣generated␣by␣RuleWall.\n\n", output);
    if (do_dump == TRUE) {
            fputs("#␣Here␣is␣a␣dump␣of␣the␣full␣configuration:\n#\n", output);
        }
    }

    /* Dump */
    if (do_dump == TRUE)
        dump_config(config, output, do_iptables == TRUE ? "#␣" : NULL,
                    !do_iptables, use_colors == COLORS_TRUE ? TRUE : FALSE);

    /* Create IPTables script */
    if (do_iptables == TRUE)
        ipt_config(config, exe, output);

    /* Close files and free all this stuff */
    fclose(output);
    free_chain(config);

    /* Check memory allocation */
    if (mem_get_count() != 0)
        fprintf(stderr, "Warning:␣%u␣remaining␣memory␣areas␣(not␣freed)!\n",
                mem_get_count());

    /* Finally, it's done! */
    return 0;
}

/* End of File */
```

memory.c

```
/* ---------------------------------------------------------------------------
 *
 * RuleWall: A Firewall Configuration Parser
 * Copyright (C) 2006 Benjamin Gaillard
 *
 * ---------------------------------------------------------------------------
 *
 *       File: src/memory.c
 *
 * Description: Memory Management Functions
 *
 * ---------------------------------------------------------------------------
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program; if not, write to the Free Software Foundation, Inc., 59
 * Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * ---------------------------------------------------------------------------
 */


/*****************************************************************************
 *
 * Headers
 *
 */

/* System headers */
#include <stdlib.h> /* NULL, malloc(), free() */
#include <stddef.h> /* size_t                 */
#include <string.h> /* strlen(), strcpy()     */

/* Local headers */
#include "memory.h"


/*****************************************************************************
 *
 * Local Datatypes and Variables
 *
 */

/* Memory area structure (back and forward linked list) */
struct mem_area {
    struct mem_area *prev, *next; /* Previous and next element in list */
};

/* First element of the memory area linked list */
static struct mem_area *first = NULL;

/* Memory area count */
static unsigned count = 0;


/*****************************************************************************
 *
 * Flobal Functions
 *
 */

/*
 * Allocate memory and register the area in the linked list
 */
void *mem_alloc(const size_t size)
{
    /* Allocate memory */
    struct mem_area *const mem = malloc(sizeof(struct mem_area) + size);

    /* No more memory... */
    if (mem == NULL)
        return NULL;

    /* Initialize and link with the preceding one */
    if (first != NULL)
        first->prev = mem;
```

```c
    mem->next = first;
    mem->prev = NULL;
    first = mem;

    /* Count it and return the actual reserved dataspace */
    count++;
    return mem + 1;
}

/*
 * Free a previously allocated dataspace
 */
void mem_free(void *const pointer)
{
    /* Pointer to actual memory area */
    struct mem_area *const mem = (struct mem_area *) pointer - 1;

    /* Unlink from the list */
    if (mem->prev != NULL)
        mem->prev->next = mem->next;
    else
        first = mem->next;
    if (mem->next != NULL)
        mem->next->prev = mem->prev;

    /* Free the area and retire it from the counter */
    free(mem);
    count--;
}

/*
 * Free all allocated memory
 */
void mem_free_all(void)
{
    struct mem_area *cur;

    /* Walk throuth the linked list and free all memory */
    for (cur = first; cur != NULL; cur = cur->next) {
        free(cur);
        count--;
    }

    /* Reinitialize list head */
    first = NULL;
}

/*
 * Get the count of the remaining allocated memory areas
 */
unsigned mem_get_count(void)
{
    return count;
}

/*
 * Duplicate a string by allocating space for it and copying it
 */
char *mem_strdup(const char *const string)
{
    /* Allocate space */
    char *mem = mem_alloc(strlen(string) + 1);

    /* Copy string */
    if (mem != NULL)
        strcpy(mem, string);
    return mem;
}

/* End of File */
```

memory.h

```c
/* ------------------------------------------------------------------------
 *
 * RuleWall: A Firewall Configuration Parser
 * Copyright (C) 2006 Benjamin Gaillard
 *
 * ------------------------------------------------------------------------
 *
 *        File: src/memory.h
 *
 * Description: Memory Management Functions Header
 *
 * ------------------------------------------------------------------------
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program; if not, write to the Free Software Foundation, Inc., 59
 * Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * ------------------------------------------------------------------------
 */


/* Process only once */
#ifndef MEMORY_H
#define MEMORY_H

/* C++ protection */
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/* System headers */
#include <stddef.h> /* size_t */
/* Memory management functions */
void *mem_alloc(size_t size);
void mem_free(void *pointer);
void mem_free_all(void);
unsigned mem_get_count(void);
char *mem_strdup(const char *string);

/* C++ protection */
#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* !MEMORY_H */

/* End of File */
```

parser.y

```
/* ---------------------------------------------------------------------------
 *
 * RuleWall: A Firewall Configuration Parser
 * Copyright (C) 2006 Benjamin Gaillard
 *
 * ---------------------------------------------------------------------------
 *
 *        File: src/parser.y
 *
 * Description: Yacc Parser
 *
 * ---------------------------------------------------------------------------
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program; if not, write to the Free Software Foundation, Inc., 59
 * Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * ---------------------------------------------------------------------------
 */


%{

/*****************************************************************************
 *
 * Headers
 *
 */

#include <stdlib.h> /* NULL, malloc(), free() */
#include <stdio.h>

#include "structs.h"
#include "memory.h"

/* The first defined chain */
struct chain *config;


/*****************************************************************************
 *
 * Yacc Functions
 *
 */

/* Yacc needs yylex() to be defined */
extern int yylex(void);

/* Yacc needs this... */
#ifdef __GNUC__
#define UNUSED __attribute__((__unused__))
#else
#define UNUSED
#endif
static void yyerror(char *string UNUSED)
{}

%}

/* The union used to return values from symbols */
%union {
    struct chain       *chain_val;     /* Chain          */
    const struct chain *chain_cval;    /* Constant chain */
    struct action      *action_val;    /* Action         */
    struct test        *test_val;      /* Test           */
    struct expr        *expr_val;      /* Expression     */
    struct condition   *condition_val; /* Condition      */
    struct addr        *addrs_val;     /* Addresses      */
    struct port        *ports_val;     /* Ports          */
    enum final          final_val;     /* Final action   */
    enum proto          proto_val;     /* Protocol       */
    enum direction      dir_val;       /* Direction      */
    struct one_port     port_val;      /* Just one port  */
    char               *string;        /* Simple string  */
}

/* Non terminal symbols */
%type <chain_val> configuration chain
%type <action_val> action
%type <test_val> test
%type <expr_val> expr
%type <condition_val> condition
%type <dir_val> direction
%type <addrs_val> addrs addrlist addr
%type <ports_val> ports portlist port

/* Chain definition symbols */
```

```
%token CHAINSEP
%token ASSIGN
%token <final_val> FINAL
%token <string> NEWCHAIN
%token <chain_cval> USERCHAIN

/* if/then/else keywords */
%token IF THEN ELSE

/* Condition operands */
%token PAR_OPEN PAR_CLOSE
%left OP_OR
%left OP_AND
%nonassoc OP_NOT

/* Protocol-related tokens */
%token <proto_val> IP PROTO
%token <dir_val> DIRECTION

/* List operators */
%token LIST_BEGIN LIST_END
%token LIST_SEP

/* Port-related tokens */
%token <port_val> PORT
%token <string> PORTNAME

/* A simple malloc()'ed string, returned by several terminal symbols */
%token <string> ADDR

/* Invalid token */
%token INVALID

/* Start symbol */
%start configuration

%%


/*****************************************************************************
 *
 * Yacc Rules
 *
 */

/* A configuration: a chain ensemble */
configuration:
    chain configuration {
        $1->next = $2;
        $$ = $1;
    } | chain {
        $$ = $1;
    };

/* A chain definition */
chain:
    NEWCHAIN ASSIGN action CHAINSEP {
        if (($$ = mem_alloc(sizeof(struct chain))) != NULL) {
            $$->next = NULL;
            $$->name = $1;
            $$->action = $3;

            if (config == NULL)
                config = $$;
        }
    };

/* An action : final, user or conditional (test) */
action:
    FINAL {
        /* Pre-defined final action chain */
        if (($$ = mem_alloc(sizeof(struct action))) != NULL) {
            $$->type = TARGET_FINAL;
            $$->action.final = $1;
        }
    } | USERCHAIN { /* Extension */
        /* User-defined action chain */
        if (($$ = mem_alloc(sizeof(struct action))) != NULL) {
            $$->type = TARGET_USER;
            $$->action.user = $1;
        }
    } | test {
        /* Conditional actions */
        if (($$ = mem_alloc(sizeof(struct action))) != NULL) {
            $$->type = TARGET_TEST;
            $$->action.test = $1;
        }
    };

/* An if/then/else test */
test:
    IF expr test_then action ELSE action {
        if (($$ = mem_alloc(sizeof(struct test))) != NULL) {
            $$->expr = $2;
            $$->act_then = $4;
            $$->act_else = $6;
        }
    };
test_then: THEN | ;

/* A test expression */
expr:
    OP_NOT expr {
        $2->not = !$2->not;
        $$ = $2;
    } | PAR_OPEN expr PAR_CLOSE {
        $$ = $2;
    } | expr OP_AND expr {
        if (($$ = mem_alloc(sizeof(struct expr))) != NULL) {
            $$->not = FALSE;
            $$->type = EXPR_AND;
            $$->sub.expr.left = $1;
            $$->sub.expr.right = $3;
        }
    } | expr OP_OR expr {
        if (($$ = mem_alloc(sizeof(struct expr))) != NULL) {
            $$->not = FALSE;
            $$->type = EXPR_OR;
            $$->sub.expr.left = $1;
            $$->sub.expr.right = $3;
        }
    } | condition {
        if (($$ = mem_alloc(sizeof(struct expr))) != NULL) {
            $$->not = FALSE;
            $$->type = EXPR_COND;
            $$->sub.cond = $1;
        }
    };

/* A simple condition */
condition:
    IP direction addrs {
        if (($$ = mem_alloc(sizeof(struct condition))) != NULL) {
            $$->type = COND_ADDR;
            $$->proto = $1;
            $$->dir = $2;
            $$->cond.addr = $3;
        }
    } | PROTO direction ports {
        if (($$ = mem_alloc(sizeof(struct condition))) != NULL) {
            $$->type = COND_PORT;
            $$->proto = $1;
            $$->dir = $2;
            $$->cond.port = $3;
        }
    };

/* A packet direction */
direction:
    DIRECTION {
        $$ = $1;
    } | {
        $$ = DIR_BOTH;
    };

/* Either a single address or an address list */
addrs:
    addr {
        $1->next = NULL;
        $$ = $1;
    } | LIST_BEGIN addrlist LIST_END {
        $$ = $2;
    };

/* An address list */
addrlist:
    addr LIST_SEP addrlist {
        $1->next = $3;
        $$ = $1;
    } | addr {
        $1->next = NULL;
        $$ = $1;
    };

/* One address */
addr:
    ADDR {
        if (($$ = mem_alloc(sizeof(struct addr))) != NULL) {
            $$->string = $1;
        }
    };

/* Either a single port or a port list */
ports:
    port {
        $1->next = NULL;
        $$ = $1;
    } | LIST_BEGIN portlist LIST_END {
        $$ = $2;
    };

/* A port list */
portlist:
    port LIST_SEP portlist {
        $1->next = $3;
        $$ = $1;
    } | port {
        $1->next = NULL;
        $$ = $1;
    };

/* One port */
port:
    PORT {
        if (($$ = mem_alloc(sizeof(struct port))) != NULL) {
            $$->type = PORT_NUMERIC;
            $$->port.range = $1;
        }
    } | PORTNAME {
        if (($$ = mem_alloc(sizeof(struct port))) != NULL) {
            $$->type = PORT_NAME;
            $$->port.name = $1;
        }
    };

%%


/******************************************************************************
 *
 * Additional Functions
 *
 */

/* Lex text buffer, defined in lexer.l */
extern char *yytext;

/* Extern functions defined in lexer.l */
extern enum bool begin_file(const char *name);
extern const char *get_file(void);
extern unsigned get_line(void);

/*
 * Parse a configuration file, or standard input if filename is NULL
 */
struct chain *parse_config(const char *const filename)
{
    if (begin_file(filename) == FALSE)
        return NULL;
    /* Initialize variables */
    config = NULL;

    /* Parse input/file */
    if (yyparse() != 0) {
        fprintf(stderr,
                "Parsing error: file \"%s\", line %d, near \"%s\".\n",
                get_file(), get_line(), yytext);
        mem_free_all();
        return NULL;
    }

    return config;
}

/* End of File */
```

lexer.l

```
/* ---------------------------------------------------------------------------
 *
 * RuleWall: A Firewall Configuration Parser
 * Copyright (C) 2006 Benjamin Gaillard
 *
 * ---------------------------------------------------------------------------
 *
 *        File: src/lexer.l
 *
 * Description: Lex/Flex Lexer
 *
 * ---------------------------------------------------------------------------
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program; if not, write to the Free Software Foundation, Inc., 59
 * Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * ---------------------------------------------------------------------------
 */


%{

/*****************************************************************************
 *
 * Headers
 *
 */

/* ./configure result */
#ifdef HAVE_CONFIG_H
# include <config.h>
#endif /* HAVE_CONFIG_H */

/* System headers */
#include <stdlib.h> /* NULL, malloc(), free(), atoi() */
#include <stdio.h>  /* fopen(), fclose(), printf() */
#include <string.h> /* strlen(), strdup(), strrchr(), strcpy(), memcpy() */
#if CHECK_PORT_NAMES
#include <netdb.h> /* getservbyname() */
#endif /* CHECK_PORT_NAMES */

/* Local headers */
#include "structs.h"
#include "memory.h"
#include "parser.h"


/*****************************************************************************
 *
 * Local Variables
 *
 */

/* State values */
static enum bool is_list;       /* Wether a host/port belongs to a list    */
static enum proto cur_proto;    /* Current protocol (ip, tcp, udp)         */
static enum direction is_dired; /* Wether the direction has beed specified */

/* Current state */
static enum {
    STATE_INIT, STATE_INCL, STATE_CHAIN, STATE_HOSTS, STATE_PORTS
} state;

/* File context structure */
struct context {
    struct context *prev;   /* Previous context    */
    FILE *file;             /* File descriptor     */
    char *name;             /* Filename            */
    int cur_line;           /* Current line number */
    YY_BUFFER_STATE buffer; /* Lex buffer          */
};

/* Current file context */
static struct context *cur_context = NULL;


/*****************************************************************************
 *
 * Local functions
 *
 */

/* Functions defined at the end of this file */
extern enum bool begin_file(const char *name);
extern enum bool end_file(void);
```

```
/* The first element of the chain linked list */
extern struct chain *config; /* Defined in parser.y */

/*
 * Find a chain structure corresponding to its associated name
 */
static const struct chain *find_chain(const char *const name)
{
    const struct chain *chain;

    /* Here we search throughout the list; we could have implemented a hash
     * table or something like that, but it’s time-consuming and it goes far
     * beyond the scope of this project. */
    for (chain = config; chain != NULL; chain = chain->next)
        if (strcmp(chain->name, name) == 0)
            return chain;

    /* Not found */
    return NULL;
}

%}


/******************************************************************************
 *
 * Lex/Flex Options and Subexpressions
 *
 */

/* Flex-specific options */
%option nointeractive
%option noyywrap noinput nounput
%option noyy_push_state noyy_pop_state noyy_top_state
%option noyy_scan_buffer noyy_scan_bytes noyy_scan_string


/*
 * Additional states
 */

/* C-style comment */
%x COMMENT

/* Chain definition and include */
%x CHAIN INCL

/* Host, host list, port and port list */
%x HOSTS PORTS


/*
 * Subexpressions (for simpler expressions below)
 */

/* Whitespace */
SPACE [ \t\n\r]

/* Decimal byte (0-255) */
BYTE 25[0-5]|2[0-4][0-9]|([01]?[0-9])?[0-9]

/* IPv4 address */
IPV4 ({BYTE}\.){3}{BYTE}

/* IPv4 address mask */
MASK \/({IPV4}|3[0-2]|[0-2]?[0-9])

/* IPv6 address */
IPV6 ([0-9A-Fa-f]{1,4}:){7}[0-9A-Fa-f]{1,4}

%%


 /******************************************************************************
  *
  * Symbols Definitions
  *
  */

 /*
  * Comments
  */

 /* C++- and shell-style comments */
<*>("//"|#).*\n cur_context->cur_line++;

 /* C-style comments */

 /* State-changing start of comment */
"/*"        { state = STATE_INIT;  BEGIN(COMMENT); }
<INCL>"/*"  { state = STATE_INCL;  BEGIN(COMMENT); }
<CHAIN>"/*" { state = STATE_CHAIN; BEGIN(COMMENT); }
<HOSTS>"/*" { state = STATE_HOSTS; BEGIN(COMMENT); }
<PORTS>"/*" { state = STATE_PORTS; BEGIN(COMMENT); }
<COMMENT>{
    [^*]*|\*+[^*/]* { /* Eat up the content of a comment */
        int i;
        for (i = 0; yytext[i] != ’\0’; i++)
            if (yytext[i] == ’\n’)
                cur_context->cur_line++;
    }

    \*+\/ { /* End of comment */
        switch (state) {
        case STATE_INCL:
            BEGIN(INCL);
            break;

        case STATE_CHAIN:
            BEGIN(CHAIN);
            break;

        case STATE_HOSTS:
            BEGIN(HOSTS);
            break;

        case STATE_PORTS:
            BEGIN(PORTS);
            break;
        default:
            BEGIN(INITIAL);
        }
    }
}


 /*
  * Include State
  */

 /* Include keyword */
include/{SPACE} BEGIN(INCL);

 /* Filename */
<INCL>{
    \"[^"\n]*(\\\"[^"\n]*)*\" {
        yytext[strlen(yytext) - 1] = ’\0’;
        begin_file(yytext + 1);
        BEGIN(INITIAL);
    }

    [^ \t\n\r]+ {
        begin_file(yytext);
        BEGIN(INITIAL);
    }
}


 /*
  * Initial (Global) State
  */

 /* Chain operators */
<CHAIN>{
    ; { BEGIN(INITIAL); return CHAINSEP; } /* Chain separator   */
    =                   return ASSIGN;     /* Chain assignation */
}

 /* Condition operands */
<CHAIN>{
    !    return OP_NOT;
    &&   return OP_AND;
    "||" return OP_OR;
    \(   return PAR_OPEN;
    \)   return PAR_CLOSE;
}

 /* Final (predefined) chains: ACCEPT, DROP, REJECT */
<CHAIN>{
    (ACCEPT|accept)/[ \t\n\r;] { yylval.final_val = FINAL_ACCEPT;
                                 return FINAL; }
    (DROP|drop)/[ \t\n\r;]     { yylval.final_val = FINAL_DROP;
                                 return FINAL; }
    (REJECT|reject)/[ \t\n\r;] { yylval.final_val = FINAL_REJECT;
                                 return FINAL; }
}

 /* if/then/else keywords */
<CHAIN>{
    if/[ \t\n\r!(] return IF;
    then/{SPACE}   return THEN;
    else/{SPACE}   return ELSE;
}

 /* Network (IP) and transport (TCP, UDP) protocols identifiers */
<CHAIN>{
    ip/{SPACE}   { BEGIN(HOSTS); is_list = FALSE; is_dired = FALSE;
                   cur_proto = yylval.proto_val = PROTO_IP;   return IP;    }
    ipv4/{SPACE} { BEGIN(HOSTS); is_list = FALSE; is_dired = FALSE;
                   cur_proto = yylval.proto_val = PROTO_IPV4; return IP;    }
    ipv6/{SPACE} { BEGIN(HOSTS); is_list = FALSE; is_dired = FALSE;
                   cur_proto = yylval.proto_val = PROTO_IPV6; return IP;    }
    port/{SPACE} { BEGIN(PORTS); is_list = FALSE; is_dired = FALSE;
                   cur_proto = yylval.proto_val = PROTO_PORT; return PROTO; }
    udp/{SPACE}  { BEGIN(PORTS); is_list = FALSE; is_dired = FALSE;
                   cur_proto = yylval.proto_val = PROTO_UDP;  return PROTO; }
    tcp/{SPACE}  { BEGIN(PORTS); is_list = FALSE; is_dired = FALSE;
                   cur_proto = yylval.proto_val = PROTO_TCP;  return PROTO; }
}

 /* Chain identifier */
<INITIAL,CHAIN>[A-Za-z_-][A-Za-z0-9_-]* {
    char *name;
    const struct chain *chain;

    /* Names beginning with "__" are reserved for internal usage */
    if (yytext[0] == ’_’ && yytext[1] == ’_’)
        return INVALID;

    /* Save name in memory */
    name = mem_strdup(yytext);

    /* Check if the chain already exists */
    if ((chain = find_chain(name)) != NULL) {
        yylval.chain_cval = chain;
        return USERCHAIN;
    }

    BEGIN(CHAIN);
    yylval.string = name;
    return NEWCHAIN;
}


 /*
  * Host and Port State
  */

 /* Direction: source or destination */
<HOSTS,PORTS>{
    (source|src)/[ \t\n\r{] { /* Source */
#define MAKE_DIR(dir)               \
    if (is_dired) {                 \
        if (cur_proto < PROTO_PORT) \
            goto jump_host;         \
        goto jump_port;             \
    }                               \
                                    \
    is_dired = TRUE;                \
    yylval.dir_val = dir;           \
    return DIRECTION;

        MAKE_DIR(DIR_SRC)
    }

    (destination|dst)/[ \t\n\r{] { /* Destination */
        MAKE_DIR(DIR_DST)
    }

    both/[ \t\n\r{] { /* Both ways */
        MAKE_DIR(DIR_BOTH)
    }
}

 /* List operators */
<HOSTS,PORTS>{
    \{ { is_dired = TRUE; is_list = TRUE; return LIST_BEGIN; } /* Beginning */
    \} { is_dired = TRUE; BEGIN(CHAIN);   return LIST_END;   } /* End       */
    ,  { is_dired = TRUE;                 return LIST_SEP;   } /* Separator */
}

 /* Numeric IPv4 address or machine name, with possible mask */
<HOSTS>({IPV4}|[A-Za-z0-9-]+(\.[A-Za-z0-9-]+)*){MASK}? {
jump_host:
    is_dired = TRUE;

    /* Save name in memory */
    if (is_list == FALSE)
        BEGIN(CHAIN);
    yylval.string = mem_strdup(yytext);
    return ADDR;
}

<PORTS>{
    [0-9]{1,5}(-[0-9]{1,5})? { /* Numeric port number/range */
        char *second = strchr(yytext, '-');
        int port;

        is_dired = TRUE;

        /* Separe the two numbers in case of a range */
        if (second != NULL)
            *second++ = '\0';

        /* Get the first number and check it */
        port = atoi(yytext);
        if ((port & ~0xFFFF) != 0)
            return INVALID;
        yylval.port_val.from = (unsigned short) port;

        /* Get the second number and check it */
        if (second == NULL)
            /* Same as first number */
            yylval.port_val.to = (unsigned short) port;
        else {
            /* Second part of the string */
            port = atoi(second);
            if ((port & ~0xFFFF) != 0)
                return INVALID;
            yylval.port_val.to = (unsigned short) port;
        }

        /* If not in a list, it's done */
        if (is_list == FALSE)
            BEGIN(CHAIN);
        return PORT;
    }

    [A-Za-z0-9_-]+ { /* Port service name */
    jump_port:
        is_dired = TRUE;

#if CHECK_PORT_NAMES
        /* Verify port name for existence */

        /* Note: it isn't specified in the manual page wether this function
         * returns a dynamically allocated (malloc()'ed) structure; I suppose
         * it doesn't, hence there's no free()... */
        if (getservbyname(yytext, proto_name[cur_proto]) == NULL)
            return INVALID;
#endif

        /* If not in a list, it's done */
        if (is_list == FALSE)
            BEGIN(CHAIN);
        yylval.string = mem_strdup(yytext);
        return PORTNAME;
    }
}


 /*
  * Whitespace and Invalid Characters
  */

 /* Count end of lines for line numbering facility */
<*>\r\n?|\n\r? cur_context->cur_line++;

 /* Ignore space characters */
<*>[ \t]+ ;

 /* Everything not catched yet is considered invalid */
<*>[A-Za-z0-9_.-]+|. return INVALID;

 /* End of file */
<<EOF>> {
    if (end_file() == FALSE)
        return EOF;
}

%%


/******************************************************************************
 *
 * Local Functions
 *
 */
```

```c
/*
 * Build a full filename relative to a reference
 */
static char *make_rel_name(const char *const ref, const char *const name)
{
    const char *file;
    unsigned dirlen;
    char *res;

    /* If absolute or reference has no directory part, it remains the same */
    if (name[0] == '/' || (file = strrchr(ref, '/')) == NULL)
        return strdup(name);

    /* Get the directory length */
    dirlen = (unsigned) ((unsigned long) file - (unsigned long) ref);

    /* Make the filename */
    if ((res = malloc(dirlen + strlen(name) + 2)) != NULL) {
        memcpy(res, ref, dirlen);
        res[dirlen] = '/';
        strcpy(res + dirlen + 1, name);
    }

    return res;
}

/*
 * Begin the processing of a new (included) file
 */
enum bool begin_file(const char *name)
{
    struct context *cont;

    if ((cont = malloc(sizeof(struct context))) == NULL)
        return FALSE;
    if (name == NULL || (name[0] == '-' && name[1] == '\0')) {
        /* Standard input */
        cont->file = stdin;
        cont->buffer = YY_CURRENT_BUFFER;
        cont->name = NULL;
    } else {
        /* Get a correct path */
        if (cur_context != NULL && cur_context->name != NULL)
            cont->name = make_rel_name(cur_context->name, name);
        else
            cont->name = strdup(name);

        /* Given file */
        if ((cont->file = fopen(cont->name, "r")) == NULL) {
            fprintf(stderr, "Error: could not open \"%s\": ", cont->name);
            perror(NULL);
            free(cont);
            return FALSE;
        }
        if ((cont->buffer = yy_create_buffer(cont->file, YY_BUF_SIZE))
                    == NULL) {
            fclose(cont->file);
            free(cont);
            return FALSE;
        }
        yy_switch_to_buffer(cont->buffer);
    }

    /* Initialize structure */
    cont->prev = cur_context;
    cont->cur_line = 1;
    cur_context = cont;

    return TRUE;
}

/*
 * End the processing of the current file and return back to the previous one
 */
enum bool end_file(void)
{
    struct context *prev;

    /* Check if it isn't already the last one */
    if (cur_context == NULL)
        return FALSE;
    prev = cur_context->prev;

    /* Switch back to the previous buffer */
    if (prev != NULL && prev->buffer != NULL)
        yy_switch_to_buffer(prev->buffer);
    if (cur_context->buffer != NULL)
        yy_delete_buffer(cur_context->buffer);

    /* Free file and memory */
    fclose(cur_context->file);
    free(cur_context->name);
    free(cur_context);

    /* Update the current pointer */
    if ((cur_context = prev) == NULL)
        return FALSE;
    return TRUE;
}

/*
 * Get the current line number
 */
unsigned get_line(void)
{
    return cur_context->cur_line;
}

/*
 * Get the current file name
 */
const char *get_file(void)
{
    return cur_context->name;
}

/* Suppress a warning about an unused Flex function */
#ifdef FLEX_SCANNER
void *(*const _disable_warning)(void *ptr, yy_size_t size) = yy_flex_realloc;
#endif
```

```
/* End of File */
```

structs.c

```
/* ---------------------------------------------------------------------------
 *
 * RuleWall: A Firewall Configuration Parser
 * Copyright (C) 2006 Benjamin Gaillard
 *
 * ---------------------------------------------------------------------------
 *
 *        File: src/structs.c
 *
 * Description: Structure Dunping and Freeing Functions
 *
 * ---------------------------------------------------------------------------
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program; if not, write to the Free Software Foundation, Inc., 59
 * Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * ---------------------------------------------------------------------------
 */


/*****************************************************************************
 *
 * Headers
 *
 */

/* System headers */
#include <stdlib.h> /* NULL, free()               */
#include <string.h> /* strcmp()                   */
#include <stdio.h>  /* putc(), fputs(), fprintf() */

/* Local headers */
#include "memory.h"
#include "structs.h"


/*****************************************************************************
 *
 * Constants and Macros
 *
 */

/* Number of spaces used for the indentation in the dumps */
#define INDENT_SPACES 2

/* Macros used to make a color string */
#define MAKE_STRING(string)  MAKE_STRING2(string)
#define MAKE_STRING2(string) #string
#define MAKE_COLOR(number)   "\033[" MAKE_STRING(number) ";1m"

/* Color strings */
#define COLOR_RESET   "\033[0m"
#define COLOR_SILVER  "\033[37m"
#define COLOR_GRAY    MAKE_COLOR(30)
#define COLOR_RED     MAKE_COLOR(31)
#define COLOR_GREEN   MAKE_COLOR(32)
#define COLOR_YELLOW  MAKE_COLOR(33)
#define COLOR_BLUE    MAKE_COLOR(34)
#define COLOR_MAGENTA MAKE_COLOR(35)
#define COLOR_CYAN    MAKE_COLOR(36)
#define COLOR_WHITE   MAKE_COLOR(37)

/* Colors for language elements */
#define COLOR_COMMENT  COLOR_SILVER
#define COLOR_CHAIN    COLOR_CYAN
#define COLOR_OPERATOR COLOR_WHITE
#define COLOR_KEYWORD  COLOR_YELLOW
#define COLOR_FINAL    COLOR_MAGENTA
#define COLOR_PROTO    COLOR_GREEN
#define COLOR_DIR      COLOR_RED
#define COLOR_HOST     COLOR_BLUE
#define COLOR_PORT     COLOR_BLUE

/* Macro to simplify conditional use of colors ("CD" for "Color Display") */
#define CD(with, without) (use_colors ? (with) : (without))


/*****************************************************************************
 *
 * Freeing Functions
 *
 */

/*
 * Free a chain structure
 */
void free_chain(struct chain *chain)
{
    if (chain == NULL)
        return;

    free_chain(chain->next);
    mem_free(chain->name);
    free_action(chain->action);

    mem_free(chain);
}

/*
```

```
 * Free an action structure
 */
void free_action(struct action *action)
{
    if (action == NULL)
        return;

    if (action->type == TARGET_TEST)
        free_test(action->action.test);

    mem_free(action);
}

/*
 * Free a test structure
 */
void free_test(struct test *test)
{
    if (test == NULL)
        return;

    free_expr(test->expr);
    free_action(test->act_then);
    free_action(test->act_else);

    mem_free(test);
}

/*
 * Free an expr structure
 */
void free_expr(struct expr *expr)
{
    if (expr == NULL)
        return;

    if (expr->type == EXPR_COND) {
        free_condition(expr->sub.cond);
    } else {
        free_expr(expr->sub.expr.left);
        free_expr(expr->sub.expr.right);
    }

    mem_free(expr);
}

/*
 * Free a condition structure
 */
void free_condition(struct condition *condition)
{
    if (condition == NULL)
        return;

    switch (condition->type) {
    case COND_ADDR:
        free_addr(condition->cond.addr);
        break;

    case COND_PORT:
        free_port(condition->cond.port);
    }

    mem_free(condition);
}

/*
 * Free an addr structure
 */
void free_addr(struct addr *addr)
{
    if (addr == NULL)
        return;

    free_addr(addr->next);
    mem_free(addr->string);

    mem_free(addr);
}

/*
 * Free a port structure
 */
void free_port(struct port *port)
{
    if (port == NULL)
        return;

    free_port(port->next);
    if (port->type == PORT_NAME)
        mem_free(port->port.name);

    mem_free(port);
}


/*****************************************************************************
 *
 * Dumping Functions
 *
 */

/* Local functions */
static void dump_chain(const struct chain *chain, unsigned depth);
static void dump_action(const struct action *action, unsigned depth);
static void dump_test(const struct test *test, unsigned depth);
static void dump_test_2(const struct test *test, unsigned depth);
static void dump_expr(const struct expr *expr, unsigned depth);
static void dump_condition(const struct condition *condition);
static void dump_addr(const struct addr *addr);
static void dump_one_addr(const struct addr *addr);
static void dump_port(const struct port *port);
static void dump_one_port(const struct port *port);

/* Local variables */
static FILE *out_file;          /* The file where tu output the dump    */
static const char *line_prefix; /* What to display in front of lines    */
static enum bool use_colors;    /* Wether to display the dump in colors */

/*
```

```
 * Prefix and indent an output line
 */
static void indent(unsigned depth)
{
    fputs(line_prefix, out_file);

    depth *= INDENT_SPACES;
    while (depth-- != 0)
        putc('␣', out_file);
}

/*
 * Dump a full configuration
 */
void dump_config(const struct chain *chain, FILE *const file,
                 const char *const prefix, const enum bool comment,
                 const enum bool colors)
{
    out_file = file == NULL ? stdout : file;
    line_prefix = prefix == NULL ? "" : prefix;
    use_colors = colors;

    if (comment) {
        indent(0U);
        fputs(CD(COLOR_COMMENT "//␣Configuration␣dump␣generated␣by␣RuleWall"
                 COLOR_RESET "\n", "//␣Configuration␣dump␣generated␣by␣"
                 "RuleWall\n"), out_file);
        indent(0U);
        putc('\n', out_file);
    }

    dump_chain(chain, 0U);
    while ((chain = chain->next) != NULL) {
        indent(0U);
        putc('\n', out_file);
        dump_chain(chain, 0U);
    }

    out_file = stdout;
    line_prefix = "";
    use_colors = FALSE;
}

/*
 * Dump a chain structure
 */
static void dump_chain(const struct chain *const chain, const unsigned depth)
{
    indent(depth);
    fprintf(out_file, CD(COLOR_CHAIN "%s" COLOR_RESET "␣"
                         COLOR_OPERATOR "=" COLOR_RESET "\n", "%s␣=\n"),
            chain->name);

    dump_action(chain->action, depth + 1);
    indent(depth);
    fputs(CD(COLOR_OPERATOR ";" COLOR_RESET "\n", ";\n"), out_file);
}

/*
 * Dump an action structure
 */
static void dump_action(const struct action *const action,
                        const unsigned depth)
{
    switch (action->type) {
    case TARGET_FINAL:
        indent(depth);
        switch (action->action.final) {
        case FINAL_ACCEPT:
            fputs(CD(COLOR_FINAL "accept" COLOR_RESET "\n", "accept\n"),
                  out_file);
            break;

        case FINAL_DROP:
            fputs(CD(COLOR_FINAL "drop" COLOR_RESET "\n", "drop\n"),
                  out_file);
            break;

        case FINAL_REJECT:
            fputs(CD(COLOR_FINAL "reject" COLOR_RESET "\n", "reject\n"),
                  out_file);
        }
        break;

    case TARGET_USER:
        indent(depth);
        fprintf(out_file, CD(COLOR_CHAIN "%s" COLOR_RESET "\n", "%s\n"),
                action->action.user->name);
        break;

    case TARGET_TEST:
        dump_test(action->action.test, depth);
    }
}

/*
 * Dump a test structure
 */
static void dump_test(const struct test *const test, const unsigned depth)
{
    indent(depth);
    dump_test_2(test, depth);
}

/*
 * Dump a test structure, without indenting the first "if" (used to dieplay
 * "else if" on a single line)
 */
static void dump_test_2(const struct test *const test, const unsigned depth)
{
    fputs(CD(COLOR_KEYWORD "if" COLOR_RESET "\n", "if\n"), out_file);
    dump_expr(test->expr, depth + 1);

    indent(depth);
    fputs(CD(COLOR_KEYWORD "then" COLOR_RESET "\n", "then\n"), out_file);
    dump_action(test->act_then, depth + 1);

    indent(depth);
    fputs(CD(COLOR_KEYWORD "else" COLOR_RESET, "else"), out_file);
    if (test->act_else->type == TARGET_TEST) {
        putc('␣', out_file);
        dump_test_2(test->act_else->action.test, depth);
    } else {
        putc('\n', out_file);
        dump_action(test->act_else, depth + 1);
    }
}

/*
 * Dump an expr structure
 */
static void dump_expr(const struct expr *const expr, const unsigned depth)
{
    indent(depth);
    if (expr->not == TRUE)
        fputs(CD(COLOR_OPERATOR "!" COLOR_RESET "␣", "!␣"), out_file);

    if (expr->type == EXPR_COND)
        dump_condition(expr->sub.cond);
    else {
        fputs(CD(COLOR_OPERATOR "(" COLOR_RESET "\n", "(\n"), out_file);

        dump_expr(expr->sub.expr.left, depth + 1);

        indent(depth);
        switch (expr->type) {
        case EXPR_AND:
            fputs(CD(COLOR_OPERATOR "&&" COLOR_RESET "\n", "&&\n"), out_file);
            break;

        case EXPR_OR:
            fputs(CD(COLOR_OPERATOR "||" COLOR_RESET "\n", "||\n"), out_file);

        default:
            break;
        }

        dump_expr(expr->sub.expr.right, depth + 1);

        indent(depth);
        fputs(CD(COLOR_OPERATOR ")" COLOR_RESET "\n", ")\n"), out_file);
    }
}

/*
 * Dump a condition structure
 */
static void dump_condition(const struct condition *const condition)
{
    const char *dir;
    static const char *const protos[] =
            {"ip", "ipv4", "ipv6", "port", "tcp", "udp"};

    switch (condition->dir) {
    case DIR_BOTH:
        dir = "";
        break;

    case DIR_SRC:
        dir = CD("␣" COLOR_DIR "source" COLOR_RESET, "␣source");
        break;

    case DIR_DST:
        dir = CD("␣" COLOR_DIR "destination" COLOR_RESET, "␣destination");
        break;

    default:
        dir = "";
    }

    fprintf(out_file, CD(COLOR_PROTO "%s" COLOR_RESET "%s␣", "%s%s␣"),
            protos[condition->proto], dir);

    switch (condition->type) {
    case COND_ADDR:
        dump_addr(condition->cond.addr);
        break;

    case COND_PORT:
        dump_port(condition->cond.port);
        break;
    }

    putc('\n', out_file);
}

/*
 * Dump an addr structure
 */
static void dump_addr(const struct addr *addr)
{
    if (addr->next == NULL)
        dump_one_addr(addr);
    else {
        fputs(CD(COLOR_OPERATOR "{" COLOR_RESET "␣", "{␣"), out_file);
        dump_one_addr(addr);
        while ((addr = addr->next) != NULL) {
            fputs(CD(COLOR_OPERATOR "," COLOR_RESET "␣", ",␣"), out_file);
            dump_one_addr(addr);
        }
        fputs(CD("␣" COLOR_OPERATOR "}" COLOR_RESET, "␣}"), out_file);
    }
}

/*
 * Dump a one_addr structure
 */
static void dump_one_addr(const struct addr *const addr)
{
    fprintf(out_file, CD(COLOR_HOST "%s" COLOR_RESET, "%s"), addr->string);
}

/*
 * Dump a port structure
 */
static void dump_port(const struct port *port)
{
    if (port->next == NULL)
        dump_one_port(port);
    else {
        fputs(CD(COLOR_OPERATOR "{" COLOR_RESET "␣", "{␣"), out_file);
        dump_one_port(port);
        while ((port = port->next) != NULL) {
            fputs(CD(COLOR_OPERATOR "," COLOR_RESET "␣", ",␣"), out_file);
            dump_one_port(port);
        }
        fputs(CD("␣" COLOR_OPERATOR "}" COLOR_RESET, "␣}"), out_file);
    }
}

/*
 * Dump a one_port structure
 */
static void dump_one_port(const struct port *const port)
{
    fputs(CD(COLOR_PORT, ""), out_file);
    switch (port->type) {
    case PORT_NUMERIC:
        fprintf(out_file, "%u", (unsigned) port->port.range.from);
        if (port->port.range.to != port->port.range.from)
            fprintf(out_file, "-%u", (unsigned) port->port.range.to);
        break;

    case PORT_NAME:
        fputs(port->port.name, out_file);
    }
    fputs(CD(COLOR_RESET, ""), out_file);
}

/* End of File */
```

structs.h

```
/* ------------------------------------------------------------------------
 *
 * RuleWall: A Firewall Configuration Parser
 * Copyright (C) 2006 Benjamin Gaillard
 *
 * ------------------------------------------------------------------------
 *
 *        File: src/structs.h
 *
 * Description: Structures Header File
 *
 * ------------------------------------------------------------------------
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program; if not, write to the Free Software Foundation, Inc., 59
 * Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * ------------------------------------------------------------------------
 */


/* Process only once */
#ifndef STRUCT_H
#define STRUCT_H

/* C++ protection */
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/* System headers */
#include <stdio.h> /* FILE * */


/*
 * Custom types: enumerations
 */

/* Boolean value */
enum bool { FALSE = 0, TRUE = 1 };

/* Final target */
enum final { FINAL_ACCEPT, FINAL_DROP, FINAL_REJECT };

/* Expression type */
enum expr_type { EXPR_COND, EXPR_AND, EXPR_OR };

/* Type of condition */
enum cond_type { COND_ADDR, COND_PORT };

/* Protocols */
enum proto { PROTO_IP = 0, PROTO_IPV4, PROTO_IPV6,
             PROTO_PORT, PROTO_TCP, PROTO_UDP };

/* Packet direction */
enum direction { DIR_BOTH, DIR_SRC, DIR_DST };

/* A port range */
struct one_port { unsigned short from, to; };


/*
 * Custom types: structures
 */

/* Chain */
struct chain {
    struct chain *next;    /* Next chain (linked list) */
    char *name;            /* Chain name               */
    struct action *action; /* Associated action        */
};

/* Action */
struct action {
    enum { TARGET_FINAL, TARGET_USER, TARGET_TEST } type; /* Action type */
    union {
        enum final          final; /* Final target       */
        const struct chain *user;  /* User-defined chain */
        struct test        *test;  /* Test (conditions)  */
    } action;
};

/* Test */
struct test {
    struct expr *expr;                  /* Associated test expression */
    struct action *act_then, *act_else; /* Taken actions              */
};

/* Expression */
struct expr {
    enum bool not;       /* Wether to negate test ("!" operator) */
    enum expr_type type; /* Expression type                      */
    union {
        struct {
            struct expr *left, *right; /* Left and right operands */
        } expr;
        struct condition *cond; /* Condition */
    } sub;
};

/* Condition */
struct condition {
    enum cond_type type; /* Condition type     */
    enum direction dir;  /* Packet direction   */
    enum proto proto;    /* Concerned protocol */
    union {
        struct addr *addr; /* Host address    */
        struct port *port; /* Port (TCP, UDP) */
    } cond;
};

/* Host address */
struct addr {
    struct addr *next; /* Next address (linked list) */
    char *string;      /* Corresponding string       */
};

/* Port number/range/name */
struct port {
    struct port *next;                     /* Next port range (linked list) */
    enum { PORT_NUMERIC, PORT_NAME } type; /* Port type                     */
    union {
        struct one_port range; /* Port range */
        char *name;            /* Port name  */
    } port;
};

/* Freeing functions */
extern void free_chain(struct chain *chain);
extern void free_action(struct action *action);
extern void free_test(struct test *test);
extern void free_expr(struct expr *expr);
extern void free_condition(struct condition *condition);
extern void free_addr(struct addr *addr);
extern void free_port(struct port *port);

/* Dumping functions */
extern void dump_config(const struct chain *chain, FILE *file,
                        const char *prefix, enum bool comment,
                        enum bool colors);

/* C++ protection */
#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* !STRUCT_H */

/* End of File */
```

iptables.c

```
/* ------------------------------------------------------------------------
 *
 * RuleWall: A Firewall Configuration Parser
 * Copyright (C) 2006 Benjamin Gaillard
 *
 * ------------------------------------------------------------------------
 *
 *        File: src/iptables.c
 *
 * Description: IPTables Rules Generation Functions
 *
 * ------------------------------------------------------------------------
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program; if not, write to the Free Software Foundation, Inc., 59
 * Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * ------------------------------------------------------------------------
 */


/*****************************************************************************
 *
 * Headers
 *
 */
```

```c
/* System headers */
#include <stdlib.h> /* NULL, malloc(), free() */
#include <stdio.h> /* printf() */
#include <string.h> /* strdup(), strcmp() */

/* Local headers */
#include "structs.h"
#include "iptables.h"


/******************************************************************************
 *
 * Prototypes and Local Variables
 *
 */

/* Default IPTables program name */
#define DEFAULT_IPT_EXE "iptables"

/* Auxiliary functions */
static void ipt_out_create(const char *table);
static char *ipt_new_table(const struct action *action);
static void ipt_out_jump(const char *table, const char *target);
static const char *make_port(const struct port *port);

/* Local functions */
static void ipt_chain(const struct chain *chain);
static void ipt_action(const char *table,
                       const struct action *action);
static void ipt_test(const char *table, const struct test *test);
static void ipt_expr(const char *table,
                     const char *tbl_then, const char *tbl_else,
                     const struct expr *expr);
static void ipt_cond(const char *table,
                     const char *tbl_then, const char *tbl_else,
                     const struct condition *cond);

/* Local variables */
static const char *const default_ipt_exe = "iptables";
static const char *ipt_exe;
static const char *cur_chain;
static FILE *out_file;


/******************************************************************************
 *
 * Global Functions
 *
 */

void ipt_config(const struct chain *config, const char *const exe,
                FILE *const out)
{
    ipt_exe = exe == NULL ? default_ipt_exe : exe;
    out_file = out == NULL ? stdout : out;

    while (config != NULL) {
        putc('\n', out_file);
        ipt_chain(config);
        config = config->next;
    }

    ipt_exe = default_ipt_exe;
    out_file = stdout;
}


/******************************************************************************
 *
 * Auxiliary Functions
 *
 */

/*
 * Output an IPTables table creation command
 */
static void ipt_out_create(const char *const table)
{
    fprintf(out_file, "%s␣-N␣%s\n", ipt_exe, table);
}

/*
 * Create an IPTables table for later use
 */
static char *ipt_new_table(const struct action *const action)
{
    static unsigned count = 0;
    unsigned digits = count, nb = 0;
    char *res;

    if (action != NULL)
        switch (action->type) {
        case TARGET_FINAL:
            switch (action->action.final) {
            case FINAL_ACCEPT:
                return strdup("ACCEPT");
            case FINAL_DROP:
                return strdup("DROP");
            case FINAL_REJECT:
                return strdup("REJECT");
            }
            break;

        case TARGET_USER:
            return strdup(action->action.user->name);

        default:
            break;
        }

    /* Count the number of digits */
    do {
        digits /= 10;
        nb++;
    } while (digits != 0);

    /* Allocate memory and build string */
    if ((res = malloc(nb + 5)) != NULL)
        sprintf(res, "__RW%u", count++);
    /* Output and return the result */
    ipt_out_create(res);
    return res;
}

/*
 * Output an IPTables jump rule
 */
static void ipt_out_jump(const char *const table, const char *const target)
{
    if ((table[0] == ’_’ && table[1] == ’_’) || strcmp(table, cur_chain) == 0)
        fprintf(out_file, "%s␣-A␣%s␣-j␣%s\n", ipt_exe, table, target);
}

/*
 * Make a string from the given port structure
 */
static const char *make_port(const struct port *const port)
{
    static char res[12];

    switch (port->type) {
    case PORT_NUMERIC:
        if (port->port.range.from == port->port.range.to)
            sprintf(res, "%d", port->port.range.from);
        else
            sprintf(res, "%d:%d", port->port.range.from, port->port.range.to);
        break;

    case PORT_NAME:
        return port->port.name;
    }

    return res;
}


/*****************************************************************************
 *
 * Local Functions
 *
 */

/*
 * Process a chain
 */
static void ipt_chain(const struct chain *const chain)
{
    cur_chain = chain->name;
    ipt_out_create(chain->name);
    ipt_action(chain->name, chain->action);
}

/*
 * Process an action
 */
static void ipt_action(const char *const table,
                       const struct action *const action)
{
```

```
    switch (action->type) {
    case TARGET_FINAL:
        switch (action->action.final) {
        case FINAL_ACCEPT:
            ipt_out_jump(table, "ACCEPT");
            break;

        case FINAL_DROP:
            ipt_out_jump(table, "DROP");
            break;

        case FINAL_REJECT:
            ipt_out_jump(table, "REJECT");
        }
        break;

    case TARGET_USER:
        ipt_out_jump(table, action->action.user->name);
        break;

    case TARGET_TEST:
        ipt_test(table, action->action.test);
        break;
    }
}

/*
 * Process a test
 */
static void ipt_test(const char *const table, const struct test *const test)
{
    char *const tbl_then = ipt_new_table(test->act_then);
    char *const tbl_else = ipt_new_table(test->act_else);

    ipt_expr(table, tbl_then, tbl_else, test->expr);
    ipt_action(tbl_then, test->act_then);
    ipt_action(tbl_else, test->act_else);

    free(tbl_else);
}

/*
 * Process an expression
 */
static void ipt_expr(const char *const table,
                     const char *tbl_then, const char *tbl_else,
                     const struct expr *const expr)
{
    char *inter;

    if (expr->not) {
        const char *const tmp = tbl_then;
        tbl_then = tbl_else;
        tbl_else = tmp;
    }

    if (expr->type == EXPR_COND)
        ipt_cond(table, tbl_then, tbl_else, expr->sub.cond);
    else {
        inter = ipt_new_table(NULL);
        switch (expr->type) {
        case EXPR_AND:
            ipt_expr(table, inter, tbl_else, expr->sub.expr.left);
            break;

        case EXPR_OR:
            ipt_expr(table, tbl_then, inter, expr->sub.expr.left);

        case EXPR_COND:
            break;
        }
        ipt_expr(inter, tbl_then, tbl_else, expr->sub.expr.right);

        free(inter);
    }
}

/*
 * Process a condition
 */
static void ipt_cond(const char *const table,
                     const char *const tbl_then, const char *const tbl_else,
                     const struct condition *const cond)
{
    const struct addr *addr;
    const struct port *port;

    switch (cond->type) {
    case COND_ADDR:
        for (addr = cond->cond.addr; addr != NULL; addr = addr->next) {
            if (cond->dir == DIR_BOTH || cond->dir == DIR_SRC)
                fprintf(out_file, "%s␣-A␣%s␣-s␣%s␣-j␣%s\n",
                        ipt_exe, table, addr->string, tbl_then);
            if (cond->dir == DIR_BOTH || cond->dir == DIR_DST)
                fprintf(out_file, "%s␣-A␣%s␣-d␣%s␣-j␣%s\n",
                        ipt_exe, table, addr->string, tbl_then);
        }
        break;

    case COND_PORT:
        for (port = cond->cond.port; port != NULL; port = port->next) {
            if (cond->proto == PROTO_PORT || cond->proto == PROTO_TCP) {
                if (cond->dir == DIR_BOTH || cond->dir == DIR_SRC)
                    fprintf(out_file, "%s␣-A␣%s␣-p␣tcp␣--sport␣%s␣-j␣%s\n",
                            ipt_exe, table, make_port(port), tbl_then);
                if (cond->dir == DIR_BOTH || cond->dir == DIR_DST)
                    fprintf(out_file, "%s␣-A␣%s␣-p␣tcp␣--dport␣%s␣-j␣%s\n",
                            ipt_exe, table, make_port(port), tbl_then);
            }
            if (cond->proto == PROTO_PORT || cond->proto == PROTO_UDP) {
                if (cond->dir == DIR_BOTH || cond->dir == DIR_SRC)
                    fprintf(out_file, "%s␣-A␣%s␣-p␣udp␣--sport␣%s␣-j␣%s\n",
                            ipt_exe, table, make_port(port), tbl_then);
                if (cond->dir == DIR_BOTH || cond->dir == DIR_DST)
                    fprintf(out_file, "%s␣-A␣%s␣-p␣udp␣--dport␣%s␣-j␣%s\n",
                            ipt_exe, table, make_port(port), tbl_then);
            }
        }
    }
```

```
    ipt_out_jump(table, tbl_else);
}

/* End of File */
```

iptables.h

```
/* ---------------------------------------------------------------------------
 *
 * RuleWall: A Firewall Configuration Parser
 * Copyright (C) 2006 Benjamin Gaillard
 *
 * ---------------------------------------------------------------------------
 *
 *        File: src/iptables.h
 *
 * Description: IPTables Rules Generation Functions Header
 *
 * ---------------------------------------------------------------------------
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by the Free
 * Software Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 * more details.
 *
 * You should have received a copy of the GNU General Public License along
 * with this program; if not, write to the Free Software Foundation, Inc., 59
 * Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * ---------------------------------------------------------------------------
 */


/* Process only once */
#ifndef IPTABLES_H
#define IPTABLES_H

/* C++ protection */
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/* System headers */
#include <stdio.h> /* FILE * */

/* IPTables-related functions */
void ipt_config(const struct chain *config, const char *exe, FILE *out);

/* C++ protection */
#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* !IPTABLES_H */

/* End of File */
```