1 Code

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define LINE_SZ 256
#define MASK 0xf

char scanner (void);
int  mask     (int num);

int E   (void);
int EE (int subtotal);
int A   (void);
int AA (int subtotal);
int B   (void);
int BB (int subtotal);
int C   (void);

char token = '\n';

int main(int argc, char ** argv) {
        if (argc == 2 && argv[1][0] == 'v') {
                #define VERBOSE
        }
        while (scanner() != '\0') {
                int result = E();
                printf("Ans: 0x%x\n\n", result);
                // printf("%x\n", result);
        }
        return 0;
}

char scanner(void) {
        static char * line = NULL;
        static int pos;
        if (line == NULL) { // Initialize line
                line = (char *) malloc(LINE_SZ * sizeof(char));
        }
        if (token == '\n' || token == '\0') { // Reached end of line
                pos = 0;
                do {
                        char * l = fgets(line, LINE_SZ, stdin); // Read new
                            line
                        if (l == NULL) { // If EOF return with null
                            terminator
                                free(line);
                                puts("EOF");
                                return '\0';
                        }
                } while (line[pos] == '\n');
                char last_char = line[strlen(line) - 1];
                printf("Exp: %s%s", line, last_char == '\n' ? "" : "\n");
                token = line[pos];
        } else { // Read next token
                pos++;
                token = line[pos];
        }
        return token;
}

int mask(int num) {
        return MASK & num;
}

int E(void) {
        int subtotal = A();
        int value = EE(subtotal);
        return value;
}

int EE(int subtotal) {
        int value;
        if (token == '|') {
                scanner(); // Consume bar
                int st = subtotal | A();
                value = EE(st);
        } else {
                value = subtotal;
        }
        return mask(value);
}

int A(void) {
        int subtotal = B();
        int value = AA(subtotal);
        return value;
}

int AA(int subtotal) {
        int value;
        if (token == '^') {
                scanner(); // Consume caret
                int st = subtotal ^ B();
                value = AA(st);
        } else {
                value = subtotal;
        }
        return mask(value);
}

int B(void) {
        int subtotal = C();
        int value = BB(subtotal);
        return value;
}
```

```c

int BB(int subtotal) {
        int value;
        if (token == '&') {
                scanner(); // Consume ampersand
                int st = subtotal & C();
                value = BB(st);
        } else {
                value = subtotal;
        }
        return mask(value);
}

int C(void) {
        int value;
        char my_token = token; // Store current token
        scanner(); // Consume next token for all subcalls
        switch (my_token) {
                case '<':
                        value = C() << 1;
                        break;
                case '>':
                        value = C() >> 1;
                        break;
                case '~':
                        value = ~ C();
                        break;
                case '(': {
                        value = E();
                        if (token != ')') { // Test for closing parenthesis
                                puts("Error: Missing parenthesis.");
                                exit(1);
                        }
                        scanner(); // Consume right parenthesis
                        break;
                }
                default: {
                        char buff[2] = {my_token, 0};
                        value = strtol(buff, NULL, 16);
                        break;
                }
        }
        return mask(value);
}
```

2 Makefile

```
.PHONY: parser run tar

parser: parser.c
	gcc -static -Wall -Werror -o parser parser.c

run:
	./parser < data.txt

tar: parser
	tar -cvf landaverdeea03.tar README.txt parser.c parser data.txt
	   output.txt makefile
```

3 Input

```
f&a
b|3
f^1
~0
>>f
<1
3|6&c
(3|6)&c
(3|c)&6
~~f
f^>f
c&3&f
<3|3
~(e^7)
>>>>~(a^c)
~(>1|>2|>4|>8)^~5
(d^2|1)&(<<2|c)
((f&>9)|(~3^8)|(~c|b))
>f|<f&1
(>(<1&>f)|8|9)^(~3&7)
~(><8|<>1)
```

4 Output

```
./parser < data.txt
Exp: f&a
Ans: 0xa

Exp: b|3
Ans: 0xb

Exp: f^1
Ans: 0xe

Exp: ~0
Ans: 0xf

Exp: >>f
Ans: 0x3

Exp: <1
Ans: 0x2

Exp: 3|6&c
Ans: 0x7

Exp: (3|6)&c
Ans: 0x4

Exp: (3|c)&6
Ans: 0x6

Exp: ~~f
Ans: 0xf

Exp: f^>f
Ans: 0x8

Exp: c&3&f
Ans: 0x0

Exp: <3|3
Ans: 0x7

Exp: ~(e^7)
Ans: 0x6
```

```
Exp: >>>>~(a^c)
Ans: 0x0

Exp: ~(>1|>2|>4|>8)^~5
Ans: 0x2

Exp: (d^2|1)&(<<2|c)
Ans: 0xc

Exp: ((f&>9)|(~3^8)|(~c|b))
Ans: 0xf

Exp: >f|<f&1
Ans: 0x7

Exp: (>(<1&>f)|8|9)^(~3&7)
Ans: 0xd

Exp: ~(><8|<>1)
Ans: 0xf

EOF
```

5 README

ELmer Landaverde ID: 9054-91691

How to invoke the program: To compile the program run the command ”make”. This will compile and write the executable to a file named ”parse”.

To run the program run the command ”./parse”. This will run the program and read input from the console. Use input redirection to feed inpu from a file (e.g. ”./parse ¡ file.txt”). Alternatively you can run ”make run” which will run the programan and feed it the information in the ”data.txt” file.

What works and what doesn't The parser expects correct expressions, so it does not handle invalid syntax. It reads one line at a time until it reaches the end of file. Each line must end with either a new line or an EOF.