Flex and Bison Calculator Implementations

Calculator 1: +, *, SIN, COS, ASIN, ACOS

Flex (calculator1.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
[+\-*/] { return *yytext; }
sin { return SIN; }
cos { return COS; }
asin { return ASIN; }
acos { return ACOS; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

Bison (calculator1.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
double num;
}

%token <num> NUMBER
%token SIN COS ASIN ACOS
%left '+'
%left '*'
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
```

```
expression:
NUMBER { $$ = $1; }
| expression '+' expression { $$ = $1 + $3; }
| expression '*' expression { $$ = $1 * $3; }
| SIN expression { $$ = sin($2); }
| COS expression { $$ = cos($2); }
| ASIN expression { $$ = asin($2); }
| ACOS expression { $$ = acos($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```

Calculator 2: /, -, POW, EXP

Flex (calculator2.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
[+\-*/] { return *yytext; }
pow { return POW; }
exp { return EXP; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

Bison (calculator2.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
double num;
}
%token <num> NUMBER
%token POW EXP
%left '+' '-'
%left '*' '/'
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
expression:
NUMBER { $$ = $1; }
| expression '/' expression { $$ = $1 / $3; }
| expression '-' expression { $$ = $1 - $3; }
| POW expression expression { $$ = pow($2, $3); }
| EXP expression { $$ = exp($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```

Calculator 3: +, -, SINH, COSH, ASIN, ACOS

Flex (calculator3.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
[+\-] { return *yytext; }
sinh { return SINH; }
cosh { return COSH; }
```

```
asin { return ASIN; }
acos { return ACOS; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

Bison (calculator3.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
double num;
}
%token <num> NUMBER
%token SINH COSH ASIN ACOS
%left '+' '-'
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
expression:
NUMBER { $$ = $1; }
| expression '+' expression { $$ = $1 + $3; }
| expression '-' expression { $$ = $1 - $3; }
| SINH expression { $$ = sinh($2); }
| COSH expression { $$ = cosh($2); }
| ASIN expression { $$ = asin($2); }
| ACOS expression { $$ = acos($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```

Calculator 4: /, *, TAN, ATAN, TANH

Flex (calculator4.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
[*/] { return *yytext; }
tan { return TAN; }
atan { return ATAN; }
tanh { return TANH; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

Bison (calculator4.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
double num;
}
%token <num> NUMBER
%token TAN ATAN TANH
%left '*' '/'
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
expression:
NUMBER { $$ = $1; }
| expression '/' expression { $$ = $1 / $3; }
| expression '*' expression { $$ = $1 * $3; }
| TAN expression { $$ = tan($2); }
| ATAN expression { $$ = atan($2); }
| TANH expression { $$ = tanh($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```

Calculator 5: +, *, -, SQRT, CBRT

Flex (calculator5.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
[+\-*] { return *yytext; }
sqrt { return SQRT; }
cbrt { return CBRT; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

Bison (calculator5.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
double num;
}
%token <num> NUMBER
%token SQRT CBRT
```

```
%left '+' '-'
%left '*'
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
expression:
NUMBER { $$ = $1; }
| expression '+' expression { $$ = $1 + $3; }
| expression '*' expression { $$ = $1 * $3; }
| expression '-' expression { $$ = $1 - $3; }
| SQRT expression { $$ = sqrt($2); }
| CBRT expression { $$ = cbrt($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```

Calculator 6: /, -, LOG, LOG10

Flex (calculator6.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
[/-] { return *yytext; }
log { return LOG; }
log10 { return LOG10; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

Bison (calculator6.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
double num;
}
%token <num> NUMBER
%token LOG LOG10
%left '-'
%left '/'
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
expression:
NUMBER { $$ = $1; }
| expression '/' expression { $$ = $1 / $3; }
| expression '-' expression { $$ = $1 - $3; }
| LOG expression { $$ = log($2); }
| LOG10 expression { $$ = log10($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```

Calculator 7: LOG10, LOG, LOG2, ABS

Flex (calculator7.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
log10 { return LOG10; }
log { return LOG; }
log2 { return LOG2; }
abs { return ABS; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

Bison (calculator7.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
double num;
}
%token <num> NUMBER
%token LOG10 LOG LOG2 ABS
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
expression:
NUMBER { $$ = $1; }
| LOG10 expression { $$ = log10($2); }
| LOG expression { $$ = log($2); }
| LOG2 expression { $$ = log2($2); }
| ABS expression { $$ = fabs($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```

Calculator 8: -, /, CEIL, FLOOR

Flex (calculator8.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
[-/] { return *yytext; }
ceil { return CEIL; }
floor { return FLOOR; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

Bison (calculator8.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
double num;
}
%token <num> NUMBER
%token CEIL FLOOR
%left '-'
%left '/'
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
expression:
```

```
NUMBER { $$ = $1; }
| expression '-' expression { $$ = $1 - $3; }
| expression '/' expression { $$ = $1 / $3; }
| CEIL expression { $$ = ceil($2); }
| FLOOR expression { $$ = floor($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```

Calculator 9: *, ABS, FABS, COT

Flex (calculator9.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
[*] { return *yytext; }
abs { return ABS; }
fabs { return FABS; }
cot { return COT; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() { return 1; }
```

Bison (calculator9.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
```

```
double num;
}
%token <num> NUMBER
%token ABS FABS COT
%left '*'
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
expression:
NUMBER { $$ = $1; }
| expression '*' expression { $$ = $1 * $3; }
| ABS expression { $$ = abs((int)$2); }
| FABS expression { $$ = fabs($2); }
| COT expression { $$ = 1.0 / tan($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```

Calculator 10: LOG5, LOG3, SIN, ACOS

Flex (calculator10.l)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }
log5 { return LOG5; }
log3 { return LOG3; }
sin { return SIN; }
acos { return ACOS; }
[ \t\n] { /* Ignore whitespace */ }
. { printf("Invalid character: %s\n", yytext); }
```

```
%%
int yywrap() { return 1; }
```

Bison (calculator10.y)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void yyerror(const char *s);
int yylex();
%}
%union {
double num;
}
%token <num> NUMBER
%token LOG5 LOG3 SIN ACOS
%type <num> expression
%%
calculation:
calculation expression '\n' { printf("Result: %lf\n", $2); }
| /* empty */
;
expression:
NUMBER { $$ = $1; }
| LOG5 expression { $$ = log($2) / log(5); }
| LOG3 expression { $$ = log($2) / log(3); }
| SIN expression { $$ = sin($2); }
| ACOS expression { $$ = acos($2); }
;
%%
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}
int main() {
printf("Enter expressions (Ctrl+D to exit):\n");
yyparse();
return 0;
}
```