EXPERIMENT NO 9
EPSILON CLOSURE

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 09/10/2024

CODE:

```
#include<stdio.h>
#define MAX_STATES 100
#define MAX_SYMBOLS 100
 typedef struct {

int transition[MAX_STATES][MAX_SYMBOLS][MAX_STATES];

int epsilon[MAX_STATES][MAX_STATES];

int numStates;

int numSymbols;

} NFA;


void addTransition(NFA *nfa, int from, int to, int symbol) {

nfa->transition[from][symbol][to] = 1;

}


void addEpsilonTransition(NFA *nfa, int from, int to) {

nfa->epsilon[from][to] = 1;

}



void epsilonClosure(NFA *nfa, int state, int *closure, int *closureSize) {

closure[*closureSize] = state;

(*closureSize)++;

// Check for epsilon transitions
```

```
for (int i = 0; i < nfa->numStates; i++) {
if (nfa->epsilon[state][i]) {
// Check if state is already in closure
int alreadyInClosure = 0;
for (int j = 0; j < *closureSize; j++) {
if (closure[j] == i) {
alreadyInClosure = 1;
break;
}
}
if (!alreadyInClosure) {
epsilonClosure(nfa, i, closure, closureSize);
}
}
}
}

int main() {
NFA nfa;
int epsilonClosureResult[MAX_STATES];
printf("Enter the number of states: ");
scanf("%d", &nfa.numStates);
printf("Enter the number of input symbols: ");
scanf("%d", &nfa.numSymbols);

for (int i = 0; i < nfa.numStates; i++) {
for (int j = 0; j < nfa.numSymbols; j++) {
```

```
for (int k = 0; k < nfa.numStates; k++) {
nfa.transition[i][j][k] = 0; // No transitions
}
}
for (int k = 0; k < nfa.numStates; k++) {
nfa.epsilon[i][k] = 0; // No epsilon transitions
}
}

for (int i = 0; i < nfa.numStates; i++) {
printf("\nEnter the transitions for state %d\n",i);
for (int j = 0; j < nfa.numSymbols; j++) {
printf("\t for symbol %c: ", j==nfa.numSymbols-1? 35 : j+97);
int toState;
while (1) {
scanf("%d", &toState);
if(toState == -1) break;
if(j==nfa.numSymbols-1) //checking whether epsilon transition or normal input transition
addEpsilonTransition(&nfa, i, toState);
else
addTransition(&nfa, i, toState, j);
}
}
}

for (int i = 0; i < nfa.numStates; i++) {
```

```
int closure[MAX_STATES];

int closureSize = 0;

epsilonClosure(&nfa, i, closure, &closureSize);

printf("Epsilon closure of state %d: { ", i);

for (int j = 0; j < closureSize; j++) {

printf("%d ", closure[j]);

}

printf("}\n");

}

return 0;

}
```

OUTPUT

```
Enter the number of states: 8
Enter the number of input symbols: 3

Enter the transitions for state 0
        for symbol a: -1
        for symbol b: -1
        for symbol #: 1 7 -1

Enter the transitions for state 1
        for symbol a: -1
        for symbol b: -1
        for symbol #: 2 4 -1

Enter the transitions for state 2
        for symbol a: 3 -1
        for symbol b: -1
        for symbol #: -1

Enter the transitions for state 3
        for symbol a: -1
        for symbol b: -1
        for symbol #: 6 -1

Enter the transitions for state 4
        for symbol a: -1
        for symbol b: 5 -1
        for symbol #: -1
```

```
Enter the transitions for state 5
        for symbol a: -1
        for symbol b: -1
        for symbol #: 6 -1

Enter the transitions for state 6
        for symbol a: -1
        for symbol b: -1
        for symbol #: 1 7 -1

Enter the transitions for state 7
        for symbol a: -1
        for symbol b: -1
        for symbol #: -1
Epsilon closure of state 0: { 0 1 2 4 7 }
Epsilon closure of state 1: { 1 2 4 }
Epsilon closure of state 2: { 2 }
Epsilon closure of state 3: { 3 6 1 2 4 7 }
Epsilon closure of state 4: { 4 }
Epsilon closure of state 5: { 5 6 1 2 4 7 }
Epsilon closure of state 6: { 6 1 2 4 7 }
Epsilon closure of state 7: { 7 }
```

EXPERIMENT NO 10
E-NFA TO DFA

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 16/10/2024

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include<stdbool.h>

char states[100],terms[100],stack[50];
int isfinal[100],resfinal[100];
int ind=0,stacklen=0;
char grams[20];
char rhs[20][50];
int count=0,rescount=0,resind=0;
int sn=0,tn=0;

char resStates[100][20],restable[100][20][10];
```

```
char table[50][50][10];
char hasEps[100];
int eind=0;

char eclsr[50];
int clsind=0;

char sstr[30];
int sortString()
{
	int l=strlen(sstr);
	for(int i=0;i<l;i++)
	{
		for(int j=0;j<l-1-i;j++)
		{
			if(sstr[j]>sstr[j+1])
			{
				char tem=sstr[j];
				sstr[j]=sstr[j+1];
				sstr[j+1]=tem;
			}
		}
	}
}

int eclfnl=0;

int eClosure(char c)
{
	for(int i=0;i<sn;i++)
	{
		if(states[i]==c)
		{	eclfnl=eclfnl|isfinal[i];
			if(strcmp(table[i][tn],"-")!=0)
			{
				strcat(eclsr,table[i][tn]);
				clsind+=strlen(table[i][tn]);
				for(int j=0;j<strlen(table[i][tn]);j++)
				{
					eClosure(table[i][tn][j]);
				}
			}
			break;
		}
	}
	return 0;
}


int main()
{
	printf("Number of states\t");
```

```
scanf("%d",&sn);
printf("Number of terminals\t");
scanf("%d",&tn);
getchar();
printf("Enter States\t");
for(int i=0;i<sn;i++)
{
	states[i]=getchar();
	getchar();

}
printf("Enter terminals\t");
for(int i=0;i<tn;i++)
{
	terms[i]=getchar();
	getchar();
}
printf("Transition table\n");
for(int i=0;i<tn;i++)
{
	printf("\t%c",terms[i]);
}
printf("\tepsln\tFinal?\n");
for(int i=0;i<sn;i++)
{
	printf("%c\t",states[i]);
	for(int j=0;j<tn+2;j++)
	{
		scanf("%s",table[i][j]);
		if(j==tn&&strcmp(table[i][j],"-")!=0)
		{
			hasEps[eind]=states[i];
			eind++;
		}
		if(j==tn+1)
		{
			if(strcmp(table[i][j],"y")==0)
			{
			isfinal[i]=1;
			}
			else
			isfinal[i]=0;
		}
	}
}

for(int i=0;i<eind;i++)
{
	eclsr[0]='\0';
	clsind=0;
	eClosure(hasEps[i]);
	for(int j=0;j<sn;j++)
```

```
{
	for(int k=0;k<tn;k++)
	{
		for(int l=0;l<strlen(table[j][k]);l++)
		{
			if(table[j][k][l]==hasEps[i])
			{
				for(int p=0;p<clsind;p++)
				{
					int flag=1;
					for(int o=0;o<strlen(table[j][k]);o++)
					{
						if(table[j][k][o]==eclsr[p])
						{
							flag=0;
							break;
						}
					}
					if(flag)
					{
						table[j][k][strlen(table[j][k])]=eclsr[p];
					}
				}
				break;
			}
		}
	}
}
int thisind=0,fin=0;
for(int k=0;k<sn;k++)
{
	for(int l=0;l<clsind;l++)
	{
		for(int p=0;p<sn;p++)
		{
			if(states[p]==eclsr[l]&&isfinal[p]==1)
			{
				fin=1;
				break;
			}
		}
		if(fin==1)
		break;
	}
	if(states[k]==hasEps[i])
	thisind=k;
}
for(int j=0;j<clsind;j++)
{
	for(int k=0;k<sn;k++)
	{
		if(eclsr[j]==states[k])
```

```
                                {
                                        for(int l=0;l<tn;l++)
                                        {
                                                for(int o=0;o<strlen(table[k][l]);o++)
                                                {
                                                        int ffll=0;
                                                        for(int p=0;p<strlen(table[thisind][l]);p++)
                                                        {
                                                                if(table[thisind][l][o]==table[k][l][p])
                                                                {
                                                                ffll=1;
                                                                break;
                                                                }
                                                        }
                                                        if(ffll==0&&table[k][l][o]!='-')
                                                        {
                                                                if(table[thisind][l][0]=='-')
                                                                table[thisind][l][0]=table[k][l][o];
                                                                else
                                                                table[thisind][l][strlen(table[thisind]
[l])]=table[k][l][o];
                                                        }
                                                }
                                        }
                                }
                        }
                }
                isfinal[thisind]=fin|isfinal[thisind];
        }
        printf("\nThe equivalent DFA\n");
        eclsr[0]='\0';
        clsind=0;
        eclfnl=0;
        if(strcmp(table[0][tn],"-")==0)
        {
                eclsr[0]=states[0];
                eclsr[1]='\0';
                resfinal[0]=isfinal[0];
        }
        else
        {
                eClosure(states[0]);
                resfinal[0]=eclfnl;
        }
        strcpy(resStates[0],eclsr);
        for(int i=0;i<tn;i++)
        {
                strcpy(restable[0][i],table[0][i]);
        }
        rescount=1;
        resind=0;
        char curres[50];
```

```
int curlen=0;
while(rescount>resind)
{
	for(int j=0;j<tn;j++)
	{
		int flgg=0;
		int ri=0;
		flgg=-1;
		curlen=0;
		for(int k=0;k<strlen(resStates[resind]);k++)
		{
			for(int l=0;l<sn;l++)
			{
				if(states[l]==resStates[resind][k])
				{
					resfinal[resind]=resfinal[resind]|isfinal[l];
					flgg=l;
					break;
				}
			}
			if(flgg>=0)
			{	for(int p=0;p<strlen(table[flgg][j]);p++)
				{
					int fff=1;
					for(int l=0;l<curlen;l++)
					{

						if(curres[l]==table[flgg][j][p])
						{
							fff=0;
							break;
						}
					}
					if(fff==1&&table[flgg][j][p]!='-')
					{
						curres[curlen]=table[flgg][j][p];
						curlen++;
						resfinal[resind]=isfinal[flgg];

					}
				}
			}
		}
		if(curlen==0)
		continue;
		flgg=1;
		strcpy(sstr,curres);
		sortString();
		strcpy(curres,sstr);
		for(int k=0;k<rescount;k++)
		{
			curres[curlen]='\0';
```

```
                        strcpy(sstr,resStates[k]);
                        sortString();
                        strcpy(resStates[k],sstr);
                        if(strcmp(resStates[k],curres)==0)
                        {
                                flgg=0;
                        }
                }
                if(flgg==1)
                {
                        strcpy(resStates[rescount],curres);
                        rescount++;
                }
                strcpy(restable[resind][j],curres);
        }
        resind++;
}
for(int i=0;i<tn;i++)
{
        printf("\t%c",terms[i]);
}
printf("\tFinal?\n");
for(int i=0;i<rescount;i++)
{
        printf("%c\t",('A'+i));
        for(int j=0;j<tn+1;j++)
        {
                if(j==tn)
                {
                        if(resfinal[i])
                        {
                                printf("YES\n");
                        }
                        else
                                printf("N0\n");
                }
                else if(strcmp(restable[i][j],"")==0||strcmp(restable[i][j],"-")==0)
                        printf("NULL\t");
                else
                {
                        for(int k=0;k<rescount;k++)
                        {
                                if(strcmp(resStates[k],restable[i][j])==0)
                                {
                                        printf("%c\t",('A'+k));
                                        break;
                                }
                        }
                }
        }
}
printf("\n");
```

}

OUTPUT

Number of states 11
Number of terminals 2
Enter States A B C D E F G H I J K
Enter terminals 0 1
Transition table

	0	1	epsln	Final?
A	-	-	BH	n
B	-	-	CE	n
C	D	-	-	n
D	-	-	G	n
E	-	F	-	n
F	-	-	G	n
G	-	-	BH	n
H	I	-	-	n
I	-	J	-	n
J	-	K	-	n
K	-	-	-	y

The equivalent DFA

	0	1	final?
A	B	C	No
B	B	D	No
C	B	C	No
D	B	E	No
E	B	C	Yes

EXPERIMENT NO 11
LEXICAL ANALYZER USING LEX TOOL

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 09/10/2024

lex.l

```
%{
#include <stdio.h>
	int no=1;
%}
lite [0-9][0-9]*
inv [0-9][a-zA-Z0-9]*
ident [a-zA-Z][a-zA-Z0-9]*
lite2 \"[a-zA-Z][a-zA-Z0-9]*\"
rel <=|<|>=|>
key int|void|char|if
%%
{lite} {printf("%s\t%i\tLiteral\n",yytext,no);}
{lite2} { char aa[10],bb[10];
	strcpy(aa,yytext);
	for(int i=1;i<strlen(aa)-1;i++){
		bb[i-1]=aa[i];
	}
	printf("%s\t%i\tLiteral\n",bb,no);

	}
{inv} {printf("Invalid:\t%s\n",yytext);}
{rel} {
	printf("%s\t%i\tRelational Operator, \t",yytext,no);
	if(strcmp(yytext,"<=")==0){printf("LE");}
	if(strcmp(yytext,"<")==0){printf("LT");}
	if(strcmp(yytext,">=")==0){printf("LE");}
	if(strcmp(yytext,">")==0){printf("LT");}
	printf("\n");
    }
"+"|"-"|"*"|"/" {printf("%s\t%i\tArithmetic opeator, ",yytext,no);
		if(strcmp(yytext,"+")==0){printf("ADD");}
		if(strcmp(yytext,"-")==0){printf("SUB");}
		if(strcmp(yytext,"/")==0){printf("DIV");}
		if(strcmp(yytext,"*")==0){printf("MUL");}
		printf("\n");
		}
"=" {printf("%s\t%i\tAssignment operator, EQ\n",yytext,no);}
{key} {printf("%s\t%i\tkeyword\n",yytext,no);}
{ident} { printf("%s\t%i\tIdentifier\n",yytext,no);}
```

```
"\n" {no++;}
. {}
%%
int main()
{
        yyin=fopen("input.txt","r");
        printf("Lexeme\tLine\tToken\n");
        yylex();
}
```

lex.yy.c

```
#line 3 "lex.yy.c"

#define  YY_INT_ALIGNED short int

/* A lexical scanner generated by flex */

#define FLEX_SCANNER
#define YY_FLEX_MAJOR_VERSION 2
#define YY_FLEX_MINOR_VERSION 6
#define YY_FLEX_SUBMINOR_VERSION 4
#if YY_FLEX_SUBMINOR_VERSION > 0
#define FLEX_BETA
#endif

/* First, we deal with  platform-specific or compiler-specific issues. */

/* begin standard C headers. */
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>

/* end standard C headers. */

/* flex integer type definitions */

#ifndef FLEXINT_H
#define FLEXINT_H

/* C99 systems have <inttypes.h>. Non-C99 systems may or may not. */

#if defined (__STDC_VERSION__) && __STDC_VERSION__ >= 199901L

/* C99 says to define __STDC_LIMIT_MACROS before including stdint.h,
 * if you want the limit (max/min) macros for int types.
 */
#ifndef __STDC_LIMIT_MACROS
```

```
#define __STDC_LIMIT_MACROS 1
#endif

#include <inttypes.h>
typedef int8_t flex_int8_t;
typedef uint8_t flex_uint8_t;
typedef int16_t flex_int16_t;
typedef uint16_t flex_uint16_t;
typedef int32_t flex_int32_t;
typedef uint32_t flex_uint32_t;
#else
typedef signed char flex_int8_t;
typedef short int flex_int16_t;
typedef int flex_int32_t;
typedef unsigned char flex_uint8_t;
typedef unsigned short int flex_uint16_t;
typedef unsigned int flex_uint32_t;

/* Limits of integral types. */
#ifndef INT8_MIN
#define INT8_MIN               (-128)
#endif
#ifndef INT16_MIN
#define INT16_MIN              (-32767-1)
#endif
#ifndef INT32_MIN
#define INT32_MIN              (-2147483647-1)
#endif
#ifndef INT8_MAX
#define INT8_MAX               (127)
#endif
#ifndef INT16_MAX
#define INT16_MAX              (32767)
#endif
#ifndef INT32_MAX
#define INT32_MAX              (2147483647)
#endif
#ifndef UINT8_MAX
#define UINT8_MAX              (255U)
#endif
#ifndef UINT16_MAX
#define UINT16_MAX             (65535U)
#endif
#ifndef UINT32_MAX
#define UINT32_MAX             (4294967295U)
#endif

#ifndef SIZE_MAX
#define SIZE_MAX               (~(size_t)0)
#endif

#endif /* ! C99 */
```

```
#endif /* ! FLEXINT_H */

/* begin standard C++ headers. */

/* TODO: this is always defined, so inline it */
#define yyconst const

#if defined(__GNUC__) && __GNUC__ >= 3
#define yynoreturn __attribute__((__noreturn__))
#else
#define yynoreturn
#endif

/* Returned upon end-of-file. */
#define YY_NULL 0

/* Promotes a possibly negative, possibly signed char to an
 *   integer in range [0..255] for use as an array index.
 */
#define YY_SC_TO_UI(c) ((YY_CHAR) (c))

/* Enter a start condition.  This macro really ought to take a parameter,
 * but we do it the disgusting crufty way forced on us by the ()-less
 * definition of BEGIN.
 */
#define BEGIN (yy_start) = 1 + 2 *
/* Translate the current start state into a value that can be later handed
 * to BEGIN to return to the state.  The YYSTATE alias is for lex
 * compatibility.
 */
#define YY_START (((yy_start) - 1) / 2)
#define YYSTATE YY_START
/* Action number for EOF rule of a given start state. */
#define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
/* Special action meaning "start processing a new file". */
#define YY_NEW_FILE yyrestart( yyin  )
#define YY_END_OF_BUFFER_CHAR 0

/* Size of default input buffer. */
#ifndef YY_BUF_SIZE
#ifdef __ia64__
/* On IA-64, the buffer size is 16k, not 8k.
 * Moreover, YY_BUF_SIZE is 2*YY_READ_BUF_SIZE in the general case.
 * Ditto for the __ia64__ case accordingly.
 */
#define YY_BUF_SIZE 32768
#else
#define YY_BUF_SIZE 16384
#endif /* __ia64__ */
#endif
```

```
/* The state buf must be large enough to hold one state per character in the main buffer.
 */
#define YY_STATE_BUF_SIZE   ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))

#ifndef YY_TYPEDEF_YY_BUFFER_STATE
#define YY_TYPEDEF_YY_BUFFER_STATE
typedef struct yy_buffer_state *YY_BUFFER_STATE;
#endif

#ifndef YY_TYPEDEF_YY_SIZE_T
#define YY_TYPEDEF_YY_SIZE_T
typedef size_t yy_size_t;
#endif

extern int yyleng;

extern FILE *yyin, *yyout;

#define EOB_ACT_CONTINUE_SCAN 0
#define EOB_ACT_END_OF_FILE 1
#define EOB_ACT_LAST_MATCH 2

    #define YY_LESS_LINENO(n)
    #define YY_LINENO_REWIND_TO(ptr)

/* Return all but the first "n" matched characters back to the input stream. */
#define yyless(n) \
        do \
                { \
                /* Undo effects of setting up yytext. */ \
      int yyless_macro_arg = (n); \
      YY_LESS_LINENO(yyless_macro_arg);\
                *yy_cp = (yy_hold_char); \
                YY_RESTORE_YY_MORE_OFFSET \
                (yy_c_buf_p) = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ; \
                YY_DO_BEFORE_ACTION; /* set up yytext again */ \
                } \
        while ( 0 )
#define unput(c) yyunput( c, (yytext_ptr)  )

#ifndef YY_STRUCT_YY_BUFFER_STATE
#define YY_STRUCT_YY_BUFFER_STATE
struct yy_buffer_state
        {
        FILE *yy_input_file;

        char *yy_ch_buf;            /* input buffer */
        char *yy_buf_pos;           /* current position in input buffer */

        /* Size of input buffer in bytes, not including room for EOB
         * characters.
         */
```

```
		int yy_buf_size;

		/* Number of characters read into yy_ch_buf, not including EOB
		 * characters.
		 */
		int yy_n_chars;

		/* Whether we "own" the buffer - i.e., we know we created it,
		 * and can realloc() it to grow it, and should free() it to
		 * delete it.
		 */
		int yy_is_our_buffer;

		/* Whether this is an "interactive" input source; if so, and
		 * if we're using stdio for input, then we want to use getc()
		 * instead of fread(), to make sure we stop fetching input after
		 * each newline.
		 */
		int yy_is_interactive;

		/* Whether we're considered to be at the beginning of a line.
		 * If so, '^' rules will be active on the next match, otherwise
		 * not.
		 */
		int yy_at_bol;

    int yy_bs_lineno; /**< The line count. */
    int yy_bs_column; /**< The column count. */

		/* Whether to try to fill the input buffer when we reach the
		 * end of it.
		 */
		int yy_fill_buffer;

		int yy_buffer_status;

#define YY_BUFFER_NEW 0
#define YY_BUFFER_NORMAL 1
		/* When an EOF's been seen but there's still some text to process
		 * then we mark the buffer as YY_EOF_PENDING, to indicate that we
		 * shouldn't try reading from the input source any more.  We might
		 * still have a bunch of tokens to match, though, because of
		 * possible backing-up.
		 *
		 * When we actually see the EOF, we change the status to "new"
		 * (via yyrestart()), so that the user can continue scanning by
		 * just pointing yyin at a new input file.
		 */
#define YY_BUFFER_EOF_PENDING 2

		};
#endif /* !YY_STRUCT_YY_BUFFER_STATE */
```

```
/* Stack of input buffers. */
static size_t yy_buffer_stack_top = 0; /**< index of top of stack. */
static size_t yy_buffer_stack_max = 0; /**< capacity of stack. */
static YY_BUFFER_STATE * yy_buffer_stack = NULL; /**< Stack as an array. */

/* We provide macros for accessing buffer states in case in the
 * future we want to put the buffer states in a more general
 * "scanner state".
 *
 * Returns the top of the stack, or NULL.
 */
#define YY_CURRENT_BUFFER ( (yy_buffer_stack) \
                          ? (yy_buffer_stack)[(yy_buffer_stack_top)] \
                          : NULL)
/* Same as previous macro, but useful when we know that the buffer stack is not
 * NULL or when we need an lvalue. For internal use only.
 */
#define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]

/* yy_hold_char holds the character lost when yytext is formed. */
static char yy_hold_char;
static int yy_n_chars;		/* number of characters read into yy_ch_buf */
int yyleng;

/* Points to current character in buffer. */
static char *yy_c_buf_p = NULL;
static int yy_init = 0;		/* whether we need to initialize */
static int yy_start = 0; /* start state number */

/* Flag which is used to allow yywrap()'s to do buffer switches
 * instead of setting up a fresh yyin.  A bit of a hack ...
 */
static int yy_did_buffer_switch_on_eof;

void yyrestart ( FILE *input_file  );
void yy_switch_to_buffer ( YY_BUFFER_STATE new_buffer  );
YY_BUFFER_STATE yy_create_buffer ( FILE *file, int size  );
void yy_delete_buffer ( YY_BUFFER_STATE b  );
void yy_flush_buffer ( YY_BUFFER_STATE b  );
void yypush_buffer_state ( YY_BUFFER_STATE new_buffer  );
void yypop_buffer_state ( void );

static void yyensure_buffer_stack ( void );
static void yy_load_buffer_state ( void );
static void yy_init_buffer ( YY_BUFFER_STATE b, FILE *file  );
#define YY_FLUSH_BUFFER yy_flush_buffer( YY_CURRENT_BUFFER )

YY_BUFFER_STATE yy_scan_buffer ( char *base, yy_size_t size  );
YY_BUFFER_STATE yy_scan_string ( const char *yy_str  );
YY_BUFFER_STATE yy_scan_bytes ( const char *bytes, int len  );
```

```
void *yyalloc ( yy_size_t  );
void *yyrealloc ( void *, yy_size_t  );
void yyfree ( void *  );

#define yy_new_buffer yy_create_buffer
#define yy_set_interactive(is_interactive) \
        { \
        if ( ! YY_CURRENT_BUFFER ){ \
      yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
        yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
        }
#define yy_set_bol(at_bol) \
        { \
        if ( ! YY_CURRENT_BUFFER ){\
      yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
        yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
        }
#define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)

/* Begin user sect3 */
typedef flex_uint8_t YY_CHAR;

FILE *yyin = NULL, *yyout = NULL;

typedef int yy_state_type;

extern int yylineno;
int yylineno = 1;

extern char *yytext;
#ifdef yytext_ptr
#undef yytext_ptr
#endif
#define yytext_ptr yytext

static yy_state_type yy_get_previous_state ( void );
static yy_state_type yy_try_NUL_trans ( yy_state_type current_state  );
static int yy_get_next_buffer ( void );
static void yynoreturn yy_fatal_error ( const char* msg  );

/* Done after the current pattern has been matched and before the
 * corresponding action - sets up yytext.
 */
#define YY_DO_BEFORE_ACTION \
        (yytext_ptr) = yy_bp; \
        yyleng = (int) (yy_cp - yy_bp); \
```

```
	(yy_hold_char) = *yy_cp; \
	*yy_cp = '\0'; \
	(yy_c_buf_p) = yy_cp;
#define YY_NUM_RULES 11
#define YY_END_OF_BUFFER 12
/* This struct is not used in this scanner,
   but its presence is necessary. */
struct yy_trans_info
	{
	flex_int32_t yy_verify;
	flex_int32_t yy_nxt;
	};
static const flex_int16_t yy_accept[30] =
    {   0,
        0,    0,   12,   10,    9,   10,    5,    1,    4,    6,
        4,    8,    8,    8,    8,    0,    1,    3,    4,    8,
        8,    7,    8,    8,    2,    0,    8,    8,    0
    } ;

static const YY_CHAR yy_ec[256] =
    {   0,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    2,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    3,    1,    1,    1,    1,    1,    1,
        1,    4,    5,    1,    6,    1,    7,    8,    8,    8,
        8,    8,    8,    8,    8,    8,    8,    8,    1,    1,    9,
       10,   11,    1,    1,   12,   12,   12,   12,   12,   12,
       12,   12,   12,   12,   12,   12,   12,   12,   12,   12,
       12,   12,   12,   12,   12,   12,   12,   12,   12,   12,
        1,    1,    1,    1,    1,    1,   13,   12,   14,   15,

       12,   16,   12,   17,   18,   12,   12,   12,   12,   19,
       20,   12,   12,   21,   12,   22,   12,   23,   12,   12,
       12,   12,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1
    } ;

static const YY_CHAR yy_meta[24] =
```

```
    {   0,
        1,    1,    2,    1,    1,    1,    1,    3,    1,    1,
        1,    4,    4,    4,    4,    4,    4,    4,    4,    4,
        4,    4,    4
    } ;

static const flex_int16_t yy_base[34] =
    {   0,
        0,    0,   47,   48,   48,    0,   48,   38,   35,   48,
       34,    0,   26,    8,   22,   38,   32,    0,   48,    0,
       26,    0,   16,   19,   48,   33,   14,   11,   48,   21,
       25,   27,   30
    } ;

static const flex_int16_t yy_def[34] =
    {   0,
       29,    1,   29,   29,   29,   30,   29,   31,   29,   29,
       29,   32,   32,   32,   32,   33,   31,   31,   29,   32,
       32,   32,   32,   32,   29,   33,   32,   32,    0,   29,
       29,   29,   29
    } ;

static const flex_int16_t yy_nxt[72] =
    {   0,
        4,    5,    6,    7,    7,    7,    7,    8,    9,   10,
       11,   12,   12,   13,   12,   12,   12,   14,   12,   12,
       12,   12,   15,   22,   16,   22,   23,   18,   18,   20,
       20,   26,   26,   26,   22,   25,   28,   22,   27,   17,
       25,   24,   21,   19,   19,   17,   29,    3,   29,   29,
       29,   29,   29,   29,   29,   29,   29,   29,   29,   29,
       29,   29,   29,   29,   29,   29,   29,   29,   29,   29,
       29
    } ;

static const flex_int16_t yy_chk[72] =
    {   0,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,   14,   30,   28,   14,   31,   31,   32,
       32,   33,   33,   33,   27,   26,   24,   23,   21,   17,
       16,   15,   13,   11,    9,    8,    3,   29,   29,   29,
       29,   29,   29,   29,   29,   29,   29,   29,   29,   29,
       29,   29,   29,   29,   29,   29,   29,   29,   29,   29,
       29
    } ;

static yy_state_type yy_last_accepting_state;
static char *yy_last_accepting_cpos;

extern int yy_flex_debug;
int yy_flex_debug = 0;
```

```
/* The intent behind this definition is that it'll catch
 * any uses of REJECT which flex missed.
 */
#define REJECT reject_used_but_not_detected
#define yymore() yymore_used_but_not_detected
#define YY_MORE_ADJ 0
#define YY_RESTORE_YY_MORE_OFFSET
char *yytext;
#line 1 "pgm.l"
#line 2 "pgm.l"
#include <stdio.h>
        int no=1;
#line 468 "lex.yy.c"
#line 469 "lex.yy.c"

#define INITIAL 0

#ifndef YY_NO_UNISTD_H
/* Special case for "unistd.h", since it is non-ANSI. We include it way
 * down here because we want the user's section 1 to have been scanned first.
 * The user has a chance to override it with an option.
 */
#include <unistd.h>
#endif

#ifndef YY_EXTRA_TYPE
#define YY_EXTRA_TYPE void *
#endif

static int yy_init_globals ( void );

/* Accessor methods to globals.
   These are made visible to non-reentrant scanners for convenience. */

int yylex_destroy ( void );

int yyget_debug ( void );

void yyset_debug ( int debug_flag  );

YY_EXTRA_TYPE yyget_extra ( void );

void yyset_extra ( YY_EXTRA_TYPE user_defined  );

FILE *yyget_in ( void );

void yyset_in  ( FILE * _in_str  );

FILE *yyget_out ( void );

void yyset_out  ( FILE * _out_str  );
```

```
                        int yyget_leng ( void );

char *yyget_text ( void );

int yyget_lineno ( void );

void yyset_lineno ( int _line_number  );

/* Macros after this point can all be overridden by user definitions in
 * section 1.
 */

#ifndef YY_SKIP_YYWRAP
#ifdef __cplusplus
extern "C" int yywrap ( void );
#else
extern int yywrap ( void );
#endif
#endif

#ifndef YY_NO_UNPUT

    static void yyunput ( int c, char *buf_ptr  );

#endif

#ifndef yytext_ptr
static void yy_flex_strncpy ( char *, const char *, int );
#endif

#ifdef YY_NEED_STRLEN
static int yy_flex_strlen ( const char * );
#endif

#ifndef YY_NO_INPUT
#ifdef __cplusplus
static int yyinput ( void );
#else
static int input ( void );
#endif

#endif

/* Amount of stuff to slurp up with each read. */
#ifndef YY_READ_BUF_SIZE
#ifdef __ia64__
/* On IA-64, the buffer size is 16k, not 8k */
#define YY_READ_BUF_SIZE 16384
#else
#define YY_READ_BUF_SIZE 8192
#endif /* __ia64__ */
#endif
```

```
/* Copy whatever the last rule matched to the standard output. */
#ifndef ECHO
/* This used to be an fputs(), but since the string might contain NUL's,
 * we now use fwrite().
 */
#define ECHO do { if (fwrite( yytext, (size_t) yyleng, 1, yyout )) {} } while (0)
#endif

/* Gets input and stuffs it into "buf".  number of characters read, or YY_NULL,
 * is returned in "result".
 */
#ifndef YY_INPUT
#define YY_INPUT(buf,result,max_size) \
	if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
		{ \
		int c = '*'; \
		int n; \
		for ( n = 0; n < max_size && \
			     (c = getc( yyin )) != EOF && c != '\n'; ++n ) \
			buf[n] = (char) c; \
		if ( c == '\n' ) \
			buf[n++] = (char) c; \
		if ( c == EOF && ferror( yyin ) ) \
			YY_FATAL_ERROR( "input in flex scanner failed" ); \
		result = n; \
		} \
	else \
		{ \
		errno=0; \
		while ( (result = (int) fread(buf, 1, (yy_size_t) max_size, yyin)) == 0 &&
ferror(yyin)) \
			{ \
			if( errno != EINTR) \
				{ \
				YY_FATAL_ERROR( "input in flex scanner failed" ); \
				break; \
				} \
			errno=0; \
			clearerr(yyin); \
			} \
		}\
\

#endif

/* No semi-colon after return; correct usage is to write "yyterminate();" -
 * we don't want an extra ';' after the "return" because that will cause
 * some compilers to complain about unreachable statements.
 */
#ifndef yyterminate
#define yyterminate() return YY_NULL
```

```
#endif

/* Number of entries by which start-condition stack grows. */
#ifndef YY_START_STACK_INCR
#define YY_START_STACK_INCR 25
#endif

/* Report a fatal error. */
#ifndef YY_FATAL_ERROR
#define YY_FATAL_ERROR(msg) yy_fatal_error( msg )
#endif

/* end tables serialization structures and prototypes */

/* Default declaration of generated scanner - a define so the user can
 * easily add parameters.
 */
#ifndef YY_DECL
#define YY_DECL_IS_OURS 1

extern int yylex (void);

#define YY_DECL int yylex (void)
#endif /* !YY_DECL */

/* Code executed at the beginning of each rule, after yytext and yyleng
 * have been set up.
 */
#ifndef YY_USER_ACTION
#define YY_USER_ACTION
#endif

/* Code executed at the end of each rule. */
#ifndef YY_BREAK
#define YY_BREAK /*LINTED*/break;
#endif

#define YY_RULE_SETUP \
        YY_USER_ACTION

/** The main scanner function which does all the work.
 */
YY_DECL
{
        yy_state_type yy_current_state;
        char *yy_cp, *yy_bp;
        int yy_act;

        if ( !(yy_init) )
                {
                (yy_init) = 1;
```

```
#ifdef YY_USER_INIT
		YY_USER_INIT;
#endif

		if ( ! (yy_start) )
			(yy_start) = 1; /* first start state */

		if ( ! yyin )
			yyin = stdin;

		if ( ! yyout )
			yyout = stdout;

		if ( ! YY_CURRENT_BUFFER ) {
			yyensure_buffer_stack ();
			YY_CURRENT_BUFFER_LVALUE =
				yy_create_buffer( yyin, YY_BUF_SIZE );
		}

		yy_load_buffer_state(  );
		}

	{
#line 11 "pgm.l"

#line 688 "lex.yy.c"

	while ( /*CONSTCOND*/1 )		/* loops until end-of-file is reached */
		{
		yy_cp = (yy_c_buf_p);

		/* Support of yytext. */
		*yy_cp = (yy_hold_char);

		/* yy_bp points to the position in yy_ch_buf of the start of
		 * the current run.
		 */
		yy_bp = yy_cp;

		yy_current_state = (yy_start);
yy_match:
		do
			{
			YY_CHAR yy_c = yy_ec[YY_SC_TO_UI(*yy_cp)] ;
			if ( yy_accept[yy_current_state] )
				{
				(yy_last_accepting_state) = yy_current_state;
				(yy_last_accepting_cpos) = yy_cp;
				}
			while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
				{
				yy_current_state = (int) yy_def[yy_current_state];
```

```
                        if ( yy_current_state >= 30 )
                                yy_c = yy_meta[yy_c];
                        }
                yy_current_state = yy_nxt[yy_base[yy_current_state] + yy_c];
                ++yy_cp;
                }
        while ( yy_base[yy_current_state] != 48 );

yy_find_action:
        yy_act = yy_accept[yy_current_state];
        if ( yy_act == 0 )
                { /* have to back up */
                yy_cp = (yy_last_accepting_cpos);
                yy_current_state = (yy_last_accepting_state);
                yy_act = yy_accept[yy_current_state];
                }

        YY_DO_BEFORE_ACTION;

do_action:      /* This label is used only to access EOF actions. */

        switch ( yy_act )
        { /* beginning of action switch */
                case 0: /* must back up */
                /* undo the effects of YY_DO_BEFORE_ACTION */
                *yy_cp = (yy_hold_char);
                yy_cp = (yy_last_accepting_cpos);
                yy_current_state = (yy_last_accepting_state);
                goto yy_find_action;

case 1:
YY_RULE_SETUP
#line 12 "pgm.l"
{printf("%s\t%i\tLiteral\n",yytext,no);}
        YY_BREAK
case 2:
YY_RULE_SETUP
#line 13 "pgm.l"
{ char aa[10],bb[10];
        strcpy(aa,yytext);
        for(int i=1;i<strlen(aa)-1;i++){
                bb[i-1]=aa[i];
        }
        printf("%s\t%i\tLiteral\n",bb,no);

        }
        YY_BREAK
case 3:
YY_RULE_SETUP
#line 21 "pgm.l"
{printf("Invalid:\t%s\n",yytext);}
        YY_BREAK
```

```
case 4:
YY_RULE_SETUP
#line 22 "pgm.l"
{
        printf("%s\t%i\tRelational Operator, \t",yytext,no);
        if(strcmp(yytext,"<=")==0){printf("LE");}
        if(strcmp(yytext,"<")==0){printf("LT");}
        if(strcmp(yytext,">=")==0){printf("LE");}
        if(strcmp(yytext,">")==0){printf("LT");}
        printf("\n");
    }
        YY_BREAK
case 5:
YY_RULE_SETUP
#line 30 "pgm.l"
{printf("%s\t%i\tArithmetic opeator, ",yytext,no);
                if(strcmp(yytext,"+")==0){printf("ADD");}
                if(strcmp(yytext,"-")==0){printf("SUB");}
                if(strcmp(yytext,"/")==0){printf("DIV");}
                if(strcmp(yytext,"*")==0){printf("MUL");}
                printf("\n");
                }
        YY_BREAK
case 6:
YY_RULE_SETUP
#line 37 "pgm.l"
{printf("%s\t%i\tAssignment operator, EQ\n",yytext,no);}
        YY_BREAK
case 7:
YY_RULE_SETUP
#line 38 "pgm.l"
{printf("%s\t%i\tkeyword\n",yytext,no);}
        YY_BREAK
case 8:
YY_RULE_SETUP
#line 39 "pgm.l"
{ printf("%s\t%i\tIdentifier\n",yytext,no);}
        YY_BREAK
case 9:
/* rule 9 can match eol */
YY_RULE_SETUP
#line 40 "pgm.l"
{no++;}
        YY_BREAK
case 10:
YY_RULE_SETUP
#line 41 "pgm.l"
{}
        YY_BREAK
case 11:
YY_RULE_SETUP
#line 42 "pgm.l"
```

```
ECHO;
	YY_BREAK
#line 821 "lex.yy.c"
case YY_STATE_EOF(INITIAL):
	yyterminate();

	case YY_END_OF_BUFFER:
		{
		/* Amount of text matched not including the EOB char. */
		int yy_amount_of_matched_text = (int) (yy_cp - (yytext_ptr)) - 1;

		/* Undo the effects of YY_DO_BEFORE_ACTION. */
		*yy_cp = (yy_hold_char);
		YY_RESTORE_YY_MORE_OFFSET

		if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status == YY_BUFFER_NEW
)
			{
			/* We're scanning a new file or input source.  It's
			 * possible that this happened because the user
			 * just pointed yyin at a new source and called
			 * yylex().  If so, then we have to assure
			 * consistency between YY_CURRENT_BUFFER and our
			 * globals.  Here is the right place to do so, because
			 * this is the first action (other than possibly a
			 * back-up) that will match for the new input source.
			 */
			(yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
			YY_CURRENT_BUFFER_LVALUE->yy_input_file = yyin;
			YY_CURRENT_BUFFER_LVALUE->yy_buffer_status =
YY_BUFFER_NORMAL;
			}

		/* Note that here we test for yy_c_buf_p "<=" to the position
		 * of the first EOB in the buffer, since yy_c_buf_p will
		 * already have been incremented past the NUL character
		 * (since all states make transitions on EOB to the
		 * end-of-buffer state).  Contrast this with the test
		 * in input().
		 */
		if ( (yy_c_buf_p) <= &YY_CURRENT_BUFFER_LVALUE-
>yy_ch_buf[(yy_n_chars)] )
			{ /* This was really a NUL. */
			yy_state_type yy_next_state;

			(yy_c_buf_p) = (yytext_ptr) + yy_amount_of_matched_text;

			yy_current_state = yy_get_previous_state(  );

			/* Okay, we're now positioned to make the NUL
			 * transition.  We couldn't have
			 * yy_get_previous_state() go ahead and do it
```

```
		 * for us because it doesn't know how to deal
		 * with the possibility of jamming (and we don't
		 * want to build jamming into it because then it
		 * will run more slowly).
		 */

		yy_next_state = yy_try_NUL_trans( yy_current_state );

		yy_bp = (yytext_ptr) + YY_MORE_ADJ;

		if ( yy_next_state )
			{
			/* Consume the NUL. */
			yy_cp = ++(yy_c_buf_p);
			yy_current_state = yy_next_state;
			goto yy_match;
			}

		else
			{
			yy_cp = (yy_c_buf_p);
			goto yy_find_action;
			}
		}

	else switch ( yy_get_next_buffer(  ) )
		{
		case EOB_ACT_END_OF_FILE:
			{
			(yy_did_buffer_switch_on_eof) = 0;

			if ( yywrap(  ) )
				{
				/* Note: because we've taken care in
				 * yy_get_next_buffer() to have set up
				 * yytext, we can now set up
				 * yy_c_buf_p so that if some total
				 * hoser (like flex itself) wants to
				 * call the scanner after we return the
				 * YY_NULL, it'll still work - another
				 * YY_NULL will get returned.
				 */
				(yy_c_buf_p) = (yytext_ptr) + YY_MORE_ADJ;

				yy_act = YY_STATE_EOF(YY_START);
				goto do_action;
				}

			else
				{
				if ( ! (yy_did_buffer_switch_on_eof) )
					YY_NEW_FILE;
```

```
                        }
                    break;
                    }

                case EOB_ACT_CONTINUE_SCAN:
                    (yy_c_buf_p) =
                        (yytext_ptr) + yy_amount_of_matched_text;

                    yy_current_state = yy_get_previous_state(  );

                    yy_cp = (yy_c_buf_p);
                    yy_bp = (yytext_ptr) + YY_MORE_ADJ;
                    goto yy_match;

                case EOB_ACT_LAST_MATCH:
                    (yy_c_buf_p) =
                    &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)];

                    yy_current_state = yy_get_previous_state(  );

                    yy_cp = (yy_c_buf_p);
                    yy_bp = (yytext_ptr) + YY_MORE_ADJ;
                    goto yy_find_action;
                }
            break;
            }

        default:
            YY_FATAL_ERROR(
                "fatal flex scanner internal error--no action found" );
        } /* end of action switch */
            } /* end of scanning one token */
        } /* end of user's declarations */
} /* end of yylex */

/* yy_get_next_buffer - try to read in a new buffer
 *
 * Returns a code representing an action:
 *      EOB_ACT_LAST_MATCH -
 *      EOB_ACT_CONTINUE_SCAN - continue scanning from current position
 *      EOB_ACT_END_OF_FILE - end of file
 */
static int yy_get_next_buffer (void)
{
        char *dest = YY_CURRENT_BUFFER_LVALUE->yy_ch_buf;
        char *source = (yytext_ptr);
        int number_to_move, i;
        int ret_val;

        if ( (yy_c_buf_p) > &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars) + 1] )
            YY_FATAL_ERROR(
            "fatal flex scanner internal error--end of buffer missed" );
```

```
	if ( YY_CURRENT_BUFFER_LVALUE->yy_fill_buffer == 0 )
		{ /* Don't try to fill the buffer, so this is an EOF. */
		if ( (yy_c_buf_p) - (yytext_ptr) - YY_MORE_ADJ == 1 )
			{
			/* We matched a single character, the EOB, so
			 * treat this as a final EOF.
			 */
			return EOB_ACT_END_OF_FILE;
			}

		else
			{
			/* We matched some text prior to the EOB, first
			 * process it.
			 */
			return EOB_ACT_LAST_MATCH;
			}
		}

	/* Try to read more data. */

	/* First move last chars to start of buffer. */
	number_to_move = (int) ((yy_c_buf_p) - (yytext_ptr) - 1);

	for ( i = 0; i < number_to_move; ++i )
		*(dest++) = *(source++);

	if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status ==
YY_BUFFER_EOF_PENDING )
		/* don't do the read, it's not guaranteed to return an EOF,
		 * just force an EOF
		 */
		YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars) = 0;

	else
		{
			int num_to_read =
			YY_CURRENT_BUFFER_LVALUE->yy_buf_size - number_to_move - 1;

		while ( num_to_read <= 0 )
			{ /* Not enough room in the buffer - grow it. */

			/* just a shorter name for the current buffer */
			YY_BUFFER_STATE b = YY_CURRENT_BUFFER_LVALUE;

			int yy_c_buf_p_offset =
				(int) ((yy_c_buf_p) - b->yy_ch_buf);

			if ( b->yy_is_our_buffer )
				{
				int new_size = b->yy_buf_size * 2;
```

```
                        if ( new_size <= 0 )
                                b->yy_buf_size += b->yy_buf_size / 8;
                        else
                                b->yy_buf_size *= 2;

                        b->yy_ch_buf = (char *)
                                /* Include room in for 2 EOB chars. */
                                yyrealloc( (void *) b->yy_ch_buf,
                                                (yy_size_t) (b->yy_buf_size + 2)  );
                        }
                else
                        /* Can't grow it, we don't own it. */
                        b->yy_ch_buf = NULL;

                if ( ! b->yy_ch_buf )
                        YY_FATAL_ERROR(
                        "fatal error - scanner input buffer overflow" );

                (yy_c_buf_p) = &b->yy_ch_buf[yy_c_buf_p_offset];

                num_to_read = YY_CURRENT_BUFFER_LVALUE->yy_buf_size -
                                        number_to_move - 1;

                }

        if ( num_to_read > YY_READ_BUF_SIZE )
                num_to_read = YY_READ_BUF_SIZE;

        /* Read in more data. */
        YY_INPUT( (&YY_CURRENT_BUFFER_LVALUE-
>yy_ch_buf[number_to_move]),
                (yy_n_chars), num_to_read );

        YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
        }

    if ( (yy_n_chars) == 0 )
        {
        if ( number_to_move == YY_MORE_ADJ )
                {
                ret_val = EOB_ACT_END_OF_FILE;
                yyrestart( yyin  );
                }

        else
                {
                ret_val = EOB_ACT_LAST_MATCH;
                YY_CURRENT_BUFFER_LVALUE->yy_buffer_status =
                        YY_BUFFER_EOF_PENDING;
                }
        }
```

```
	else
		ret_val = EOB_ACT_CONTINUE_SCAN;

	if (((yy_n_chars) + number_to_move) > YY_CURRENT_BUFFER_LVALUE-
>yy_buf_size) {
		/* Extend the array by 50%, plus the number we really need. */
		int new_size = (yy_n_chars) + number_to_move + ((yy_n_chars) >> 1);
		YY_CURRENT_BUFFER_LVALUE->yy_ch_buf = (char *) yyrealloc(
			(void *) YY_CURRENT_BUFFER_LVALUE->yy_ch_buf, (yy_size_t)
new_size  );
		if ( ! YY_CURRENT_BUFFER_LVALUE->yy_ch_buf )
			YY_FATAL_ERROR( "out of dynamic memory in yy_get_next_buffer()" );
		/* "- 2" to take care of EOB's */
		YY_CURRENT_BUFFER_LVALUE->yy_buf_size = (int) (new_size - 2);
	}

	(yy_n_chars) += number_to_move;
	YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] =
YY_END_OF_BUFFER_CHAR;
	YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars) + 1] =
YY_END_OF_BUFFER_CHAR;

	(yytext_ptr) = &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[0];

	return ret_val;
}

/* yy_get_previous_state - get the state just before the EOB char was reached */

   static yy_state_type yy_get_previous_state (void)
{
	yy_state_type yy_current_state;
	char *yy_cp;

	yy_current_state = (yy_start);

	for ( yy_cp = (yytext_ptr) + YY_MORE_ADJ; yy_cp < (yy_c_buf_p); ++yy_cp )
		{
		YY_CHAR yy_c = (*yy_cp ? yy_ec[YY_SC_TO_UI(*yy_cp)] : 1);
		if ( yy_accept[yy_current_state] )
			{
			(yy_last_accepting_state) = yy_current_state;
			(yy_last_accepting_cpos) = yy_cp;
			}
		while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
			{
			yy_current_state = (int) yy_def[yy_current_state];
			if ( yy_current_state >= 30 )
				yy_c = yy_meta[yy_c];
			}
		yy_current_state = yy_nxt[yy_base[yy_current_state] + yy_c];
```

```
                }

        return yy_current_state;
}

/* yy_try_NUL_trans - try to make a transition on the NUL character
 *
 * synopsis
 *      next_state = yy_try_NUL_trans( current_state );
 */
   static yy_state_type yy_try_NUL_trans  (yy_state_type yy_current_state )
{
        int yy_is_jam;
        char *yy_cp = (yy_c_buf_p);

        YY_CHAR yy_c = 1;
        if ( yy_accept[yy_current_state] )
                {
                (yy_last_accepting_state) = yy_current_state;
                (yy_last_accepting_cpos) = yy_cp;
                }
        while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
                {
                yy_current_state = (int) yy_def[yy_current_state];
                if ( yy_current_state >= 30 )
                        yy_c = yy_meta[yy_c];
                }
        yy_current_state = yy_nxt[yy_base[yy_current_state] + yy_c];
        yy_is_jam = (yy_current_state == 29);

                return yy_is_jam ? 0 : yy_current_state;
}

#ifndef YY_NO_UNPUT

   static void yyunput (int c, char * yy_bp )
{
        char *yy_cp;

   yy_cp = (yy_c_buf_p);

        /* undo effects of setting up yytext */
        *yy_cp = (yy_hold_char);

        if ( yy_cp < YY_CURRENT_BUFFER_LVALUE->yy_ch_buf + 2 )
                { /* need to shift things up to make room */
                /* +2 for EOB chars. */
                int number_to_move = (yy_n_chars) + 2;
                char *dest = &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[
                                        YY_CURRENT_BUFFER_LVALUE->yy_buf_size + 2];
                char *source =
```

```
                                &YY_CURRENT_BUFFER_LVALUE-
>yy_ch_buf[number_to_move];

                while ( source > YY_CURRENT_BUFFER_LVALUE->yy_ch_buf )
                        *--dest = *--source;

                yy_cp += (int) (dest - source);
                yy_bp += (int) (dest - source);
                YY_CURRENT_BUFFER_LVALUE->yy_n_chars =
                        (yy_n_chars) = (int) YY_CURRENT_BUFFER_LVALUE->yy_buf_size;

                if ( yy_cp < YY_CURRENT_BUFFER_LVALUE->yy_ch_buf + 2 )
                        YY_FATAL_ERROR( "flex scanner push-back overflow" );
                }

        *--yy_cp = (char) c;

        (yytext_ptr) = yy_bp;
        (yy_hold_char) = *yy_cp;
        (yy_c_buf_p) = yy_cp;
}

#endif

#ifndef YY_NO_INPUT
#ifdef __cplusplus
   static int yyinput (void)
#else
   static int input  (void)
#endif

{
        int c;

        *(yy_c_buf_p) = (yy_hold_char);

        if ( *(yy_c_buf_p) == YY_END_OF_BUFFER_CHAR )
                {
                /* yy_c_buf_p now points to the character we want to return.
                 * If this occurs *before* the EOB characters, then it's a
                 * valid NUL; if not, then we've hit the end of the buffer.
                 */
                if ( (yy_c_buf_p) < &YY_CURRENT_BUFFER_LVALUE-
>yy_ch_buf[(yy_n_chars)] )
                        /* This was really a NUL. */
                        *(yy_c_buf_p) = '\0';

                else
                        { /* need more input */
                        int offset = (int) ((yy_c_buf_p) - (yytext_ptr));
                        ++(yy_c_buf_p);
```

```
                        switch ( yy_get_next_buffer(  ) )
                                {
                                case EOB_ACT_LAST_MATCH:
                                        /* This happens because yy_g_n_b()
                                         * sees that we've accumulated a
                                         * token and flags that we need to
                                         * try matching the token before
                                         * proceeding.  But for input(),
                                         * there's no matching to consider.
                                         * So convert the EOB_ACT_LAST_MATCH
                                         * to EOB_ACT_END_OF_FILE.
                                         */

                                        /* Reset buffer status. */
                                        yyrestart( yyin );

                                        /*FALLTHROUGH*/

                                case EOB_ACT_END_OF_FILE:
                                        {
                                        if ( yywrap(  ) )
                                                return 0;

                                        if ( ! (yy_did_buffer_switch_on_eof) )
                                                YY_NEW_FILE;
#ifdef __cplusplus
                                        return yyinput();
#else
                                        return input();
#endif
                                        }

                                case EOB_ACT_CONTINUE_SCAN:
                                        (yy_c_buf_p) = (yytext_ptr) + offset;
                                        break;
                                }
                        }
                }

        c = *(unsigned char *) (yy_c_buf_p);            /* cast for 8-bit char's */
        *(yy_c_buf_p) = '\0';  /* preserve yytext */
        (yy_hold_char) = *++(yy_c_buf_p);

        return c;
}
#endif  /* ifndef YY_NO_INPUT */

/** Immediately switch to a different input stream.
 * @param input_file A readable stream.
 *
 * @note This function does not reset the start condition to @c INITIAL .
 */
```

```
    void yyrestart  (FILE * input_file )
{

        if ( ! YY_CURRENT_BUFFER ){
      yyensure_buffer_stack ();
                YY_CURRENT_BUFFER_LVALUE =
        yy_create_buffer( yyin, YY_BUF_SIZE );
        }

        yy_init_buffer( YY_CURRENT_BUFFER, input_file );
        yy_load_buffer_state(  );
}

/** Switch to a different input buffer.
 * @param new_buffer The new input buffer.
 *
 */
    void yy_switch_to_buffer  (YY_BUFFER_STATE  new_buffer )
{

        /* TODO. We should be able to replace this entire function body
         * with
         *              yypop_buffer_state();
         *              yypush_buffer_state(new_buffer);
     */
        yyensure_buffer_stack ();
        if ( YY_CURRENT_BUFFER == new_buffer )
                return;

        if ( YY_CURRENT_BUFFER )
                {
                /* Flush out information for old buffer. */
                *(yy_c_buf_p) = (yy_hold_char);
                YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
                YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
                }

        YY_CURRENT_BUFFER_LVALUE = new_buffer;
        yy_load_buffer_state(  );

        /* We don't actually know whether we did this switch during
         * EOF (yywrap()) processing, but the only time this flag
         * is looked at is after yywrap() is called, so it's safe
         * to go ahead and always set it.
         */
        (yy_did_buffer_switch_on_eof) = 1;
}

static void yy_load_buffer_state  (void)
{
        (yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
        (yytext_ptr) = (yy_c_buf_p) = YY_CURRENT_BUFFER_LVALUE->yy_buf_pos;
```

```
	yyin = YY_CURRENT_BUFFER_LVALUE->yy_input_file;
	(yy_hold_char) = *(yy_c_buf_p);
}

/** Allocate and initialize an input buffer state.
 * @param file A readable stream.
 * @param size The character buffer size in bytes. When in doubt, use @c YY_BUF_SIZE.
 *
 * @return the allocated buffer state.
 */
    YY_BUFFER_STATE yy_create_buffer  (FILE * file, int  size )
{
	YY_BUFFER_STATE b;

	b = (YY_BUFFER_STATE) yyalloc( sizeof( struct yy_buffer_state )  );
	if ( ! b )
		YY_FATAL_ERROR( "out of dynamic memory in yy_create_buffer()" );

	b->yy_buf_size = size;

	/* yy_ch_buf has to be 2 characters longer than the size given because
	 * we need to put in 2 end-of-buffer characters.
	 */
	b->yy_ch_buf = (char *) yyalloc( (yy_size_t) (b->yy_buf_size + 2)  );
	if ( ! b->yy_ch_buf )
		YY_FATAL_ERROR( "out of dynamic memory in yy_create_buffer()" );

	b->yy_is_our_buffer = 1;

	yy_init_buffer( b, file );

	return b;
}

/** Destroy the buffer.
 * @param b a buffer created with yy_create_buffer()
 *
 */
    void yy_delete_buffer (YY_BUFFER_STATE  b )
{

	if ( ! b )
		return;

	if ( b == YY_CURRENT_BUFFER ) /* Not sure if we should pop here. */
		YY_CURRENT_BUFFER_LVALUE = (YY_BUFFER_STATE) 0;

	if ( b->yy_is_our_buffer )
		yyfree( (void *) b->yy_ch_buf  );

	yyfree( (void *) b  );
}
```

```
/* Initializes or reinitializes a buffer.
 * This function is sometimes called more than once on the same buffer,
 * such as during a yyrestart() or at EOF.
 */
    static void yy_init_buffer  (YY_BUFFER_STATE  b, FILE * file )

{
        int oerrno = errno;

        yy_flush_buffer( b );

        b->yy_input_file = file;
        b->yy_fill_buffer = 1;

    /* If b is the current buffer, then yy_init_buffer was _probably_
     * called from yyrestart() or through yy_get_next_buffer.
     * In that case, we don't want to reset the lineno or column.
     */
    if (b != YY_CURRENT_BUFFER){
        b->yy_bs_lineno = 1;
        b->yy_bs_column = 0;
    }

        b->yy_is_interactive = file ? (isatty( fileno(file) ) > 0) : 0;

        errno = oerrno;
}

/** Discard all buffered characters. On the next scan, YY_INPUT will be called.
 * @param b the buffer state to be flushed, usually @c YY_CURRENT_BUFFER.
 *
 */
    void yy_flush_buffer (YY_BUFFER_STATE  b )
{
        if ( ! b )
                return;

        b->yy_n_chars = 0;

        /* We always need two end-of-buffer characters.  The first causes
         * a transition to the end-of-buffer state.  The second causes
         * a jam in that state.
         */
        b->yy_ch_buf[0] = YY_END_OF_BUFFER_CHAR;
        b->yy_ch_buf[1] = YY_END_OF_BUFFER_CHAR;

        b->yy_buf_pos = &b->yy_ch_buf[0];

        b->yy_at_bol = 1;
        b->yy_buffer_status = YY_BUFFER_NEW;
```

```
	if ( b == YY_CURRENT_BUFFER )
		yy_load_buffer_state(  );
}

/** Pushes the new state onto the stack. The new state becomes
 *  the current state. This function will allocate the stack
 *  if necessary.
 *  @param new_buffer The new state.
 *
 */
void yypush_buffer_state (YY_BUFFER_STATE new_buffer )
{
	if (new_buffer == NULL)
		return;

	yyensure_buffer_stack();

	/* This block is copied from yy_switch_to_buffer. */
	if ( YY_CURRENT_BUFFER )
		{
		/* Flush out information for old buffer. */
		*(yy_c_buf_p) = (yy_hold_char);
		YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
		YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
		}

	/* Only push if top exists. Otherwise, replace top. */
	if (YY_CURRENT_BUFFER)
		(yy_buffer_stack_top)++;
	YY_CURRENT_BUFFER_LVALUE = new_buffer;

	/* copied from yy_switch_to_buffer. */
	yy_load_buffer_state(  );
	(yy_did_buffer_switch_on_eof) = 1;
}

/** Removes and deletes the top of the stack, if present.
 *  The next element becomes the new top.
 *
 */
void yypop_buffer_state (void)
{
	if (!YY_CURRENT_BUFFER)
		return;

	yy_delete_buffer(YY_CURRENT_BUFFER );
	YY_CURRENT_BUFFER_LVALUE = NULL;
	if ((yy_buffer_stack_top) > 0)
		--(yy_buffer_stack_top);

	if (YY_CURRENT_BUFFER) {
		yy_load_buffer_state(  );
```

```
                (yy_did_buffer_switch_on_eof) = 1;
        }
}

/* Allocates the stack if it does not exist.
 *  Guarantees space for at least one push.
 */
static void yyensure_buffer_stack (void)
{
        yy_size_t num_to_alloc;

        if (!(yy_buffer_stack)) {

                /* First allocation is just for 2 elements, since we don't know if this
                 * scanner will even need a stack. We use 2 instead of 1 to avoid an
                 * immediate realloc on the next call.
       */
     num_to_alloc = 1; /* After all that talk, this was set to 1 anyways... */
                (yy_buffer_stack) = (struct yy_buffer_state**)yyalloc
                                                        (num_to_alloc * sizeof(struct
yy_buffer_state*)
                                                        );
                if ( ! (yy_buffer_stack) )
                        YY_FATAL_ERROR( "out of dynamic memory in
yyensure_buffer_stack()" );

                memset((yy_buffer_stack), 0, num_to_alloc * sizeof(struct yy_buffer_state*));

                (yy_buffer_stack_max) = num_to_alloc;
                (yy_buffer_stack_top) = 0;
                return;
        }

        if ((yy_buffer_stack_top) >= ((yy_buffer_stack_max)) - 1){

                /* Increase the buffer to prepare for a possible push. */
                yy_size_t grow_size = 8 /* arbitrary grow size */;

                num_to_alloc = (yy_buffer_stack_max) + grow_size;
                (yy_buffer_stack) = (struct yy_buffer_state**)yyrealloc
                                                        ((yy_buffer_stack),
                                                        num_to_alloc * sizeof(struct
yy_buffer_state*)
                                                        );
                if ( ! (yy_buffer_stack) )
                        YY_FATAL_ERROR( "out of dynamic memory in
yyensure_buffer_stack()" );

                /* zero only the new slots.*/
                memset((yy_buffer_stack) + (yy_buffer_stack_max), 0, grow_size * sizeof(struct
yy_buffer_state*));
                (yy_buffer_stack_max) = num_to_alloc;
```

```
	}
}

/** Setup the input buffer state to scan directly from a user-specified character buffer.
 * @param base the character buffer
 * @param size the size in bytes of the character buffer
 *
 * @return the newly allocated buffer state object.
 */
YY_BUFFER_STATE yy_scan_buffer  (char * base, yy_size_t  size )
{
	YY_BUFFER_STATE b;

	if ( size < 2 ||
	    base[size-2] != YY_END_OF_BUFFER_CHAR ||
	    base[size-1] != YY_END_OF_BUFFER_CHAR )
		/* They forgot to leave room for the EOB's. */
		return NULL;

	b = (YY_BUFFER_STATE) yyalloc( sizeof( struct yy_buffer_state )  );
	if ( ! b )
		YY_FATAL_ERROR( "out of dynamic memory in yy_scan_buffer()" );

	b->yy_buf_size = (int) (size - 2);	/* "- 2" to take care of EOB's */
	b->yy_buf_pos = b->yy_ch_buf = base;
	b->yy_is_our_buffer = 0;
	b->yy_input_file = NULL;
	b->yy_n_chars = b->yy_buf_size;
	b->yy_is_interactive = 0;
	b->yy_at_bol = 1;
	b->yy_fill_buffer = 0;
	b->yy_buffer_status = YY_BUFFER_NEW;

	yy_switch_to_buffer( b  );

	return b;
}

/** Setup the input buffer state to scan a string. The next call to yylex() will
 * scan from a @e copy of @a str.
 * @param yystr a NUL-terminated string to scan
 *
 * @return the newly allocated buffer state object.
 * @note If you want to scan bytes that may contain NUL values, then use
 *       yy_scan_bytes() instead.
 */
YY_BUFFER_STATE yy_scan_string (const char * yystr )
{

	return yy_scan_bytes( yystr, (int) strlen(yystr) );
}
```

```
/** Setup the input buffer state to scan the given bytes. The next call to yylex() will
 * scan from a @e copy of @a bytes.
 * @param yybytes the byte buffer to scan
 * @param _yybytes_len the number of bytes in the buffer pointed to by @a bytes.
 *
 * @return the newly allocated buffer state object.
 */
YY_BUFFER_STATE yy_scan_bytes  (const char * yybytes, int  _yybytes_len )
{
	YY_BUFFER_STATE b;
	char *buf;
	yy_size_t n;
	int i;

	/* Get memory for full buffer, including space for trailing EOB's. */
	n = (yy_size_t) (_yybytes_len + 2);
	buf = (char *) yyalloc( n  );
	if ( ! buf )
		YY_FATAL_ERROR( "out of dynamic memory in yy_scan_bytes()" );

	for ( i = 0; i < _yybytes_len; ++i )
		buf[i] = yybytes[i];

	buf[_yybytes_len] = buf[_yybytes_len+1] = YY_END_OF_BUFFER_CHAR;

	b = yy_scan_buffer( buf, n );
	if ( ! b )
		YY_FATAL_ERROR( "bad buffer in yy_scan_bytes()" );

	/* It's okay to grow etc. this buffer, and we should throw it
	 * away when we're done.
	 */
	b->yy_is_our_buffer = 1;

	return b;
}

#ifndef YY_EXIT_FAILURE
#define YY_EXIT_FAILURE 2
#endif

static void yynoreturn yy_fatal_error (const char* msg )
{
			fprintf( stderr, "%s\n", msg );
	exit( YY_EXIT_FAILURE );
}

/* Redefine yyless() so it works in section 3 code. */

#undef yyless
#define yyless(n) \
	do \
```

```
                { \
                /* Undo effects of setting up yytext. */ \
        int yyless_macro_arg = (n); \
        YY_LESS_LINENO(yyless_macro_arg);\
                yytext[yyleng] = (yy_hold_char); \
                (yy_c_buf_p) = yytext + yyless_macro_arg; \
                (yy_hold_char) = *(yy_c_buf_p); \
                *(yy_c_buf_p) = '\0'; \
                yyleng = yyless_macro_arg; \
                } \
           while ( 0 )

/* Accessor  methods (get/set functions) to struct members. */

/** Get the current line number.
 *
 */
int yyget_lineno  (void)
{

    return yylineno;
}

/** Get the input stream.
 *
 */
FILE *yyget_in  (void)
{
        return yyin;
}

/** Get the output stream.
 *
 */
FILE *yyget_out  (void)
{
        return yyout;
}

/** Get the length of the current token.
 *
 */
int yyget_leng  (void)
{
        return yyleng;
}

/** Get the current token.
 *
 */

char *yyget_text  (void)
```

```
{
	return yytext;
}

/** Set the current line number.
 * @param _line_number line number
 *
 */
void yyset_lineno (int  _line_number )
{

    yylineno = _line_number;
}

/** Set the input stream. This does not discard the current
 * input buffer.
 * @param _in_str A readable stream.
 *
 * @see yy_switch_to_buffer
 */
void yyset_in (FILE *  _in_str )
{
	yyin = _in_str ;
}

void yyset_out (FILE *  _out_str )
{
	yyout = _out_str ;
}

int yyget_debug  (void)
{
	return yy_flex_debug;
}

void yyset_debug (int  _bdebug )
{
	yy_flex_debug = _bdebug ;
}

static int yy_init_globals (void)
{
	/* Initialization is the same as for the non-reentrant scanner.
     * This function is called from yylex_destroy(), so don't allocate here.
     */

    (yy_buffer_stack) = NULL;
    (yy_buffer_stack_top) = 0;
    (yy_buffer_stack_max) = 0;
    (yy_c_buf_p) = NULL;
    (yy_init) = 0;
    (yy_start) = 0;
```

```
/* Defined in main.c */
#ifdef YY_STDINIT
    yyin = stdin;
    yyout = stdout;
#else
    yyin = NULL;
    yyout = NULL;
#endif

    /* For future reference: Set errno on error, since we are called by
     * yylex_init()
     */
    return 0;
}

/* yylex_destroy is for both reentrant and non-reentrant scanners. */
int yylex_destroy  (void)
{

    /* Pop the buffer stack, destroying each element. */
        while(YY_CURRENT_BUFFER){
                yy_delete_buffer( YY_CURRENT_BUFFER  );
                YY_CURRENT_BUFFER_LVALUE = NULL;
                yypop_buffer_state();
        }

        /* Destroy the stack itself. */
        yyfree((yy_buffer_stack) );
        (yy_buffer_stack) = NULL;

    /* Reset the globals. This is important in a non-reentrant scanner so the next time
     * yylex() is called, initialization will occur. */
    yy_init_globals( );

    return 0;
}

/*
 * Internal utility routines.
 */

#ifndef yytext_ptr
static void yy_flex_strncpy (char* s1, const char * s2, int n )
{

        int i;
        for ( i = 0; i < n; ++i )
                s1[i] = s2[i];
}
#endif
```

```
#ifdef YY_NEED_STRLEN
static int yy_flex_strlen (const char * s )
{
	int n;
	for ( n = 0; s[n]; ++n )
		;

	return n;
}
#endif

void *yyalloc (yy_size_t  size )
{
				return malloc(size);
}

void *yyrealloc  (void * ptr, yy_size_t  size )
{

	/* The cast to (char *) in the following accommodates both
	 * implementations that use char* generic pointers, and those
	 * that use void* generic pointers.  It works with the latter
	 * because both ANSI C and C++ allow castless assignment from
	 * any pointer type to void*, and deal with argument conversions
	 * as though doing an assignment.
	 */
	return realloc(ptr, size);
}

void yyfree (void * ptr )
{
				free( (char *) ptr );	/* see yyrealloc() for (char *) cast */
}

#define YYTABLES_NAME "yytables"

#line 42 "pgm.l"

int main()
{
	yyin=fopen("input.txt","r");
	printf("Lexeme\tLine\tToken\n");
	yylex();
}
```

input.txt

```
void main(){

	int a,b,c=0;
	char ch;
```

```
	a=b+c;
	if(a<=b)
		a=b;
	printf("Hello World");
}
```

output

Lexeme	Line	Token
void	1	keyword
main	1	Identifier
int	3	keyword
a	3	Identifier
b	3	Identifier
c	3	Identifier
=	3	Assignment operator, EQ
0	3	Literal
char	4	keyword
ch	4	Identifier
a	5	Identifier
=	5	Assignment operator, EQ
b	5	Identifier
+	5	Arithmetic opeator, ADD
c	5	Identifier
if	6	keyword
a	6	Identifier
<=	6	Relational Operator, LE
b	6	Identifier
a	7	Identifier
=	7	Assignment operator, EQ
b	7	Identifier
printf	8	Identifier
Hello	8	Identifier
World	8	Identifier

PGM 12

lex.yy.c

```
#line 3 "lex.yy.c"

#define  YY_INT_ALIGNED short int

/* A lexical scanner generated by flex */

#define FLEX_SCANNER
#define YY_FLEX_MAJOR_VERSION 2
#define YY_FLEX_MINOR_VERSION 6
```

```
#define YY_FLEX_SUBMINOR_VERSION 4
#if YY_FLEX_SUBMINOR_VERSION > 0
#define FLEX_BETA
#endif

/* First, we deal with  platform-specific or compiler-specific issues. */

/* begin standard C headers. */
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>

/* end standard C headers. */

/* flex integer type definitions */

#ifndef FLEXINT_H
#define FLEXINT_H

/* C99 systems have <inttypes.h>. Non-C99 systems may or may not. */

#if defined (__STDC_VERSION__) && __STDC_VERSION__ >= 199901L

/* C99 says to define __STDC_LIMIT_MACROS before including stdint.h,
 * if you want the limit (max/min) macros for int types.
 */
#ifndef __STDC_LIMIT_MACROS
#define __STDC_LIMIT_MACROS 1
#endif

#include <inttypes.h>
typedef int8_t flex_int8_t;
typedef uint8_t flex_uint8_t;
typedef int16_t flex_int16_t;
typedef uint16_t flex_uint16_t;
typedef int32_t flex_int32_t;
typedef uint32_t flex_uint32_t;
#else
typedef signed char flex_int8_t;
typedef short int flex_int16_t;
typedef int flex_int32_t;
typedef unsigned char flex_uint8_t;
typedef unsigned short int flex_uint16_t;
typedef unsigned int flex_uint32_t;

/* Limits of integral types. */
#ifndef INT8_MIN
#define INT8_MIN               (-128)
#endif
#ifndef INT16_MIN
#define INT16_MIN              (-32767-1)
```

```
#endif
#ifndef INT32_MIN
#define INT32_MIN	(-2147483647-1)
#endif
#ifndef INT8_MAX
#define INT8_MAX	(127)
#endif
#ifndef INT16_MAX
#define INT16_MAX	(32767)
#endif
#ifndef INT32_MAX
#define INT32_MAX	(2147483647)
#endif
#ifndef UINT8_MAX
#define UINT8_MAX	(255U)
#endif
#ifndef UINT16_MAX
#define UINT16_MAX	(65535U)
#endif
#ifndef UINT32_MAX
#define UINT32_MAX	(4294967295U)
#endif

#ifndef SIZE_MAX
#define SIZE_MAX	(~(size_t)0)
#endif

#endif /* ! C99 */

#endif /* ! FLEXINT_H */

/* begin standard C++ headers. */

/* TODO: this is always defined, so inline it */
#define yyconst const

#if defined(__GNUC__) && __GNUC__ >= 3
#define yynoreturn __attribute__((__noreturn__))
#else
#define yynoreturn
#endif

/* Returned upon end-of-file. */
#define YY_NULL 0

/* Promotes a possibly negative, possibly signed char to an
 *   integer in range [0..255] for use as an array index.
 */
#define YY_SC_TO_UI(c) ((YY_CHAR) (c))

/* Enter a start condition.  This macro really ought to take a parameter,
 * but we do it the disgusting crufty way forced on us by the ()-less
```

```
 * definition of BEGIN.
 */
#define BEGIN (yy_start) = 1 + 2 *
/* Translate the current start state into a value that can be later handed
 * to BEGIN to return to the state.  The YYSTATE alias is for lex
 * compatibility.
 */
#define YY_START (((yy_start) - 1) / 2)
#define YYSTATE YY_START
/* Action number for EOF rule of a given start state. */
#define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
/* Special action meaning "start processing a new file". */
#define YY_NEW_FILE yyrestart( yyin  )
#define YY_END_OF_BUFFER_CHAR 0

/* Size of default input buffer. */
#ifndef YY_BUF_SIZE
#ifdef __ia64__
/* On IA-64, the buffer size is 16k, not 8k.
 * Moreover, YY_BUF_SIZE is 2*YY_READ_BUF_SIZE in the general case.
 * Ditto for the __ia64__ case accordingly.
 */
#define YY_BUF_SIZE 32768
#else
#define YY_BUF_SIZE 16384
#endif /* __ia64__ */
#endif

/* The state buf must be large enough to hold one state per character in the main buffer.
 */
#define YY_STATE_BUF_SIZE   ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))

#ifndef YY_TYPEDEF_YY_BUFFER_STATE
#define YY_TYPEDEF_YY_BUFFER_STATE
typedef struct yy_buffer_state *YY_BUFFER_STATE;
#endif

#ifndef YY_TYPEDEF_YY_SIZE_T
#define YY_TYPEDEF_YY_SIZE_T
typedef size_t yy_size_t;
#endif

extern int yyleng;

extern FILE *yyin, *yyout;

#define EOB_ACT_CONTINUE_SCAN 0
#define EOB_ACT_END_OF_FILE 1
#define EOB_ACT_LAST_MATCH 2

    #define YY_LESS_LINENO(n)
    #define YY_LINENO_REWIND_TO(ptr)
```

```
/* Return all but the first "n" matched characters back to the input stream. */
#define yyless(n) \
	do \
		{ \
		/* Undo effects of setting up yytext. */ \
	int yyless_macro_arg = (n); \
	YY_LESS_LINENO(yyless_macro_arg);\
		*yy_cp = (yy_hold_char); \
		YY_RESTORE_YY_MORE_OFFSET \
		(yy_c_buf_p) = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ; \
		YY_DO_BEFORE_ACTION; /* set up yytext again */ \
		} \
	while ( 0 )
#define unput(c) yyunput( c, (yytext_ptr)  )

#ifndef YY_STRUCT_YY_BUFFER_STATE
#define YY_STRUCT_YY_BUFFER_STATE
struct yy_buffer_state
	{
	FILE *yy_input_file;

	char *yy_ch_buf;		/* input buffer */
	char *yy_buf_pos;		/* current position in input buffer */

	/* Size of input buffer in bytes, not including room for EOB
	 * characters.
	 */
	int yy_buf_size;

	/* Number of characters read into yy_ch_buf, not including EOB
	 * characters.
	 */
	int yy_n_chars;

	/* Whether we "own" the buffer - i.e., we know we created it,
	 * and can realloc() it to grow it, and should free() it to
	 * delete it.
	 */
	int yy_is_our_buffer;

	/* Whether this is an "interactive" input source; if so, and
	 * if we're using stdio for input, then we want to use getc()
	 * instead of fread(), to make sure we stop fetching input after
	 * each newline.
	 */
	int yy_is_interactive;

	/* Whether we're considered to be at the beginning of a line.
	 * If so, '^' rules will be active on the next match, otherwise
	 * not.
	 */
```

```
        int yy_at_bol;

    int yy_bs_lineno; /**< The line count. */
    int yy_bs_column; /**< The column count. */

        /* Whether to try to fill the input buffer when we reach the
         * end of it.
         */
        int yy_fill_buffer;

        int yy_buffer_status;

#define YY_BUFFER_NEW 0
#define YY_BUFFER_NORMAL 1
        /* When an EOF's been seen but there's still some text to process
         * then we mark the buffer as YY_EOF_PENDING, to indicate that we
         * shouldn't try reading from the input source any more.  We might
         * still have a bunch of tokens to match, though, because of
         * possible backing-up.
         *
         * When we actually see the EOF, we change the status to "new"
         * (via yyrestart()), so that the user can continue scanning by
         * just pointing yyin at a new input file.
         */
#define YY_BUFFER_EOF_PENDING 2

        };
#endif /* !YY_STRUCT_YY_BUFFER_STATE */

/* Stack of input buffers. */
static size_t yy_buffer_stack_top = 0; /**< index of top of stack. */
static size_t yy_buffer_stack_max = 0; /**< capacity of stack. */
static YY_BUFFER_STATE * yy_buffer_stack = NULL; /**< Stack as an array. */

/* We provide macros for accessing buffer states in case in the
 * future we want to put the buffer states in a more general
 * "scanner state".
 *
 * Returns the top of the stack, or NULL.
 */
#define YY_CURRENT_BUFFER ( (yy_buffer_stack) \
                          ? (yy_buffer_stack)[(yy_buffer_stack_top)] \
                          : NULL)
/* Same as previous macro, but useful when we know that the buffer stack is not
 * NULL or when we need an lvalue. For internal use only.
 */
#define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]

/* yy_hold_char holds the character lost when yytext is formed. */
static char yy_hold_char;
static int yy_n_chars;          /* number of characters read into yy_ch_buf */
int yyleng;
```

```
/* Points to current character in buffer. */
static char *yy_c_buf_p = NULL;
static int yy_init = 0;             /* whether we need to initialize */
static int yy_start = 0; /* start state number */

/* Flag which is used to allow yywrap()'s to do buffer switches
 * instead of setting up a fresh yyin.  A bit of a hack ...
 */
static int yy_did_buffer_switch_on_eof;

void yyrestart ( FILE *input_file  );
void yy_switch_to_buffer ( YY_BUFFER_STATE new_buffer  );
YY_BUFFER_STATE yy_create_buffer ( FILE *file, int size  );
void yy_delete_buffer ( YY_BUFFER_STATE b  );
void yy_flush_buffer ( YY_BUFFER_STATE b  );
void yypush_buffer_state ( YY_BUFFER_STATE new_buffer  );
void yypop_buffer_state ( void );

static void yyensure_buffer_stack ( void );
static void yy_load_buffer_state ( void );
static void yy_init_buffer ( YY_BUFFER_STATE b, FILE *file  );
#define YY_FLUSH_BUFFER yy_flush_buffer( YY_CURRENT_BUFFER )

YY_BUFFER_STATE yy_scan_buffer ( char *base, yy_size_t size  );
YY_BUFFER_STATE yy_scan_string ( const char *yy_str  );
YY_BUFFER_STATE yy_scan_bytes ( const char *bytes, int len  );

void *yyalloc ( yy_size_t  );
void *yyrealloc ( void *, yy_size_t  );
void yyfree ( void *  );

#define yy_new_buffer yy_create_buffer
#define yy_set_interactive(is_interactive) \
        { \
        if ( ! YY_CURRENT_BUFFER ){ \
      yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
        yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
        }
#define yy_set_bol(at_bol) \
        { \
        if ( ! YY_CURRENT_BUFFER ){\
      yyensure_buffer_stack (); \
                YY_CURRENT_BUFFER_LVALUE =    \
        yy_create_buffer( yyin, YY_BUF_SIZE ); \
        } \
        YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
        }
#define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
```

```
/* Begin user sect3 */
typedef flex_uint8_t YY_CHAR;

FILE *yyin = NULL, *yyout = NULL;

typedef int yy_state_type;

extern int yylineno;
int yylineno = 1;

extern char *yytext;
#ifdef yytext_ptr
#undef yytext_ptr
#endif
#define yytext_ptr yytext

static yy_state_type yy_get_previous_state ( void );
static yy_state_type yy_try_NUL_trans ( yy_state_type current_state  );
static int yy_get_next_buffer ( void );
static void yynoreturn yy_fatal_error ( const char* msg  );

/* Done after the current pattern has been matched and before the
 * corresponding action - sets up yytext.
 */
#define YY_DO_BEFORE_ACTION \
        (yytext_ptr) = yy_bp; \
        yyleng = (int) (yy_cp - yy_bp); \
        (yy_hold_char) = *yy_cp; \
        *yy_cp = '\0'; \
        (yy_c_buf_p) = yy_cp;
#define YY_NUM_RULES 5
#define YY_END_OF_BUFFER 6
/* This struct is not used in this scanner,
   but its presence is necessary. */
struct yy_trans_info
        {
        flex_int32_t yy_verify;
        flex_int32_t yy_nxt;
        };
static const flex_int16_t yy_accept[10] =
    {   0,
        0,    0,    6,    4,    2,    3,    1,    1,    0
    } ;

static const YY_CHAR yy_ec[256] =
    {   0,
        1,    1,    1,    1,    1,    1,    1,    1,    2,    3,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
        1,    1,    1,    1,    1,    1,    1,    4,    4,    4,
```

```
    4,   4,   4,   4,   4,   4,   4,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,

    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,

    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
    1,   1,   1,   1,   1
  } ;

static const YY_CHAR yy_meta[5] =
  {   0,
      1,   1,   1,   2
  } ;

static const flex_int16_t yy_base[11] =
  {   0,
      0,   0,   6,   7,   7,   7,   0,   0,   7,   3
  } ;

static const flex_int16_t yy_def[11] =
  {   0,
      9,   1,   9,   9,   9,   9,  10,  10,   0,   9
  } ;

static const flex_int16_t yy_nxt[12] =
  {   0,
      4,   5,   6,   7,   8,   9,   3,   9,   9,   9,
      9
  } ;

static const flex_int16_t yy_chk[12] =
  {   0,
      1,   1,   1,   1,  10,   3,   9,   9,   9,   9,
      9
  } ;
```

```
static yy_state_type yy_last_accepting_state;
static char *yy_last_accepting_cpos;

extern int yy_flex_debug;
int yy_flex_debug = 0;

/* The intent behind this definition is that it'll catch
 * any uses of REJECT which flex missed.
 */
#define REJECT reject_used_but_not_detected
#define yymore() yymore_used_but_not_detected
#define YY_MORE_ADJ 0
#define YY_RESTORE_YY_MORE_OFFSET
char *yytext;
#line 1 "pgm.l"
#line 2 "pgm.l"
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
#line 447 "lex.yy.c"
#line 448 "lex.yy.c"

#define INITIAL 0

#ifndef YY_NO_UNISTD_H
/* Special case for "unistd.h", since it is non-ANSI. We include it way
 * down here because we want the user's section 1 to have been scanned first.
 * The user has a chance to override it with an option.
 */
#include <unistd.h>
#endif

#ifndef YY_EXTRA_TYPE
#define YY_EXTRA_TYPE void *
#endif

static int yy_init_globals ( void );

/* Accessor methods to globals.
   These are made visible to non-reentrant scanners for convenience. */

int yylex_destroy ( void );

int yyget_debug ( void );

void yyset_debug ( int debug_flag  );

YY_EXTRA_TYPE yyget_extra ( void );

void yyset_extra ( YY_EXTRA_TYPE user_defined  );

FILE *yyget_in ( void );
```

```
void yyset_in  ( FILE * _in_str  );

FILE *yyget_out ( void );

void yyset_out  ( FILE * _out_str  );

                        int yyget_leng ( void );

char *yyget_text ( void );

int yyget_lineno ( void );

void yyset_lineno ( int _line_number  );

/* Macros after this point can all be overridden by user definitions in
 * section 1.
 */

#ifndef YY_SKIP_YYWRAP
#ifdef __cplusplus
extern "C" int yywrap ( void );
#else
extern int yywrap ( void );
#endif
#endif

#ifndef YY_NO_UNPUT

    static void yyunput ( int c, char *buf_ptr  );

#endif

#ifndef yytext_ptr
static void yy_flex_strncpy ( char *, const char *, int );
#endif

#ifdef YY_NEED_STRLEN
static int yy_flex_strlen ( const char * );
#endif

#ifndef YY_NO_INPUT
#ifdef __cplusplus
static int yyinput ( void );
#else
static int input ( void );
#endif

#endif

/* Amount of stuff to slurp up with each read. */
#ifndef YY_READ_BUF_SIZE
```

```
#ifdef __ia64__
/* On IA-64, the buffer size is 16k, not 8k */
#define YY_READ_BUF_SIZE 16384
#else
#define YY_READ_BUF_SIZE 8192
#endif /* __ia64__ */
#endif

/* Copy whatever the last rule matched to the standard output. */
#ifndef ECHO
/* This used to be an fputs(), but since the string might contain NUL's,
 * we now use fwrite().
 */
#define ECHO do { if (fwrite( yytext, (size_t) yyleng, 1, yyout )) {} } while (0)
#endif

/* Gets input and stuffs it into "buf".  number of characters read, or YY_NULL,
 * is returned in "result".
 */
#ifndef YY_INPUT
#define YY_INPUT(buf,result,max_size) \
        if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
                { \
                int c = '*'; \
                int n; \
                for ( n = 0; n < max_size && \
                             (c = getc( yyin )) != EOF && c != '\n'; ++n ) \
                        buf[n] = (char) c; \
                if ( c == '\n' ) \
                        buf[n++] = (char) c; \
                if ( c == EOF && ferror( yyin ) ) \
                        YY_FATAL_ERROR( "input in flex scanner failed" ); \
                result = n; \
                } \
        else \
                { \
                errno=0; \
                while ( (result = (int) fread(buf, 1, (yy_size_t) max_size, yyin)) == 0 &&
ferror(yyin)) \
                        { \
                        if( errno != EINTR) \
                                { \
                                YY_FATAL_ERROR( "input in flex scanner failed" ); \
                                break; \
                                } \
                        errno=0; \
                        clearerr(yyin); \
                        } \
                }\
\

#endif
```

```
/* No semi-colon after return; correct usage is to write "yyterminate();" -
 * we don't want an extra ';' after the "return" because that will cause
 * some compilers to complain about unreachable statements.
 */
#ifndef yyterminate
#define yyterminate() return YY_NULL
#endif

/* Number of entries by which start-condition stack grows. */
#ifndef YY_START_STACK_INCR
#define YY_START_STACK_INCR 25
#endif

/* Report a fatal error. */
#ifndef YY_FATAL_ERROR
#define YY_FATAL_ERROR(msg) yy_fatal_error( msg )
#endif

/* end tables serialization structures and prototypes */

/* Default declaration of generated scanner - a define so the user can
 * easily add parameters.
 */
#ifndef YY_DECL
#define YY_DECL_IS_OURS 1

extern int yylex (void);

#define YY_DECL int yylex (void)
#endif /* !YY_DECL */

/* Code executed at the beginning of each rule, after yytext and yyleng
 * have been set up.
 */
#ifndef YY_USER_ACTION
#define YY_USER_ACTION
#endif

/* Code executed at the end of each rule. */
#ifndef YY_BREAK
#define YY_BREAK /*LINTED*/break;
#endif

#define YY_RULE_SETUP \
	YY_USER_ACTION

/** The main scanner function which does all the work.
 */
YY_DECL
{
	yy_state_type yy_current_state;
```

```
	char *yy_cp, *yy_bp;
	int yy_act;

	if ( !(yy_init) )
		{
		(yy_init) = 1;

#ifdef YY_USER_INIT
		YY_USER_INIT;
#endif

		if ( ! (yy_start) )
			(yy_start) = 1; /* first start state */

		if ( ! yyin )
			yyin = stdin;

		if ( ! yyout )
			yyout = stdout;

		if ( ! YY_CURRENT_BUFFER ) {
			yyensure_buffer_stack ();
			YY_CURRENT_BUFFER_LVALUE =
				yy_create_buffer( yyin, YY_BUF_SIZE );
		}

		yy_load_buffer_state(  );
		}

	{
#line 6 "pgm.l"

#line 667 "lex.yy.c"

	while ( /*CONSTCOND*/1 )		/* loops until end-of-file is reached */
		{
		yy_cp = (yy_c_buf_p);

		/* Support of yytext. */
		*yy_cp = (yy_hold_char);

		/* yy_bp points to the position in yy_ch_buf of the start of
		 * the current run.
		 */
		yy_bp = yy_cp;

		yy_current_state = (yy_start);
yy_match:
		do
			{
			YY_CHAR yy_c = yy_ec[YY_SC_TO_UI(*yy_cp)] ;
			if ( yy_accept[yy_current_state] )
```

```
                        {
                        (yy_last_accepting_state) = yy_current_state;
                        (yy_last_accepting_cpos) = yy_cp;
                        }
                    while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
                        {
                        yy_current_state = (int) yy_def[yy_current_state];
                        if ( yy_current_state >= 10 )
                            yy_c = yy_meta[yy_c];
                        }
                    yy_current_state = yy_nxt[yy_base[yy_current_state] + yy_c];
                    ++yy_cp;
                    }
                while ( yy_base[yy_current_state] != 7 );

yy_find_action:
                yy_act = yy_accept[yy_current_state];
                if ( yy_act == 0 )
                    { /* have to back up */
                    yy_cp = (yy_last_accepting_cpos);
                    yy_current_state = (yy_last_accepting_state);
                    yy_act = yy_accept[yy_current_state];
                    }

                YY_DO_BEFORE_ACTION;

do_action:      /* This label is used only to access EOF actions. */

                switch ( yy_act )
        { /* beginning of action switch */
                    case 0: /* must back up */
                    /* undo the effects of YY_DO_BEFORE_ACTION */
                    *yy_cp = (yy_hold_char);
                    yy_cp = (yy_last_accepting_cpos);
                    yy_current_state = (yy_last_accepting_state);
                    goto yy_find_action;

case 1:
YY_RULE_SETUP
#line 7 "pgm.l"
{ yylval=atoi(yytext); return NUMBER;}
        YY_BREAK
case 2:
YY_RULE_SETUP
#line 8 "pgm.l"
;
        YY_BREAK
case 3:
/* rule 3 can match eol */
YY_RULE_SETUP
#line 9 "pgm.l"
return 0;
```

```
	YY_BREAK
case 4:
YY_RULE_SETUP
#line 10 "pgm.l"
return yytext[0];
	YY_BREAK
case 5:
YY_RULE_SETUP
#line 11 "pgm.l"
ECHO;
	YY_BREAK
#line 750 "lex.yy.c"
case YY_STATE_EOF(INITIAL):
	yyterminate();

	case YY_END_OF_BUFFER:
		{
		/* Amount of text matched not including the EOB char. */
		int yy_amount_of_matched_text = (int) (yy_cp - (yytext_ptr)) - 1;

		/* Undo the effects of YY_DO_BEFORE_ACTION. */
		*yy_cp = (yy_hold_char);
		YY_RESTORE_YY_MORE_OFFSET

		if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status == YY_BUFFER_NEW
)
			{
			/* We're scanning a new file or input source.  It's
			 * possible that this happened because the user
			 * just pointed yyin at a new source and called
			 * yylex().  If so, then we have to assure
			 * consistency between YY_CURRENT_BUFFER and our
			 * globals.  Here is the right place to do so, because
			 * this is the first action (other than possibly a
			 * back-up) that will match for the new input source.
			 */
			(yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
			YY_CURRENT_BUFFER_LVALUE->yy_input_file = yyin;
			YY_CURRENT_BUFFER_LVALUE->yy_buffer_status =
YY_BUFFER_NORMAL;
			}

		/* Note that here we test for yy_c_buf_p "<=" to the position
		 * of the first EOB in the buffer, since yy_c_buf_p will
		 * already have been incremented past the NUL character
		 * (since all states make transitions on EOB to the
		 * end-of-buffer state).  Contrast this with the test
		 * in input().
		 */
		if ( (yy_c_buf_p) <= &YY_CURRENT_BUFFER_LVALUE-
>yy_ch_buf[(yy_n_chars)] )
			{ /* This was really a NUL. */
```

```
		yy_state_type yy_next_state;

		(yy_c_buf_p) = (yytext_ptr) + yy_amount_of_matched_text;

		yy_current_state = yy_get_previous_state(  );

		/* Okay, we're now positioned to make the NUL
		 * transition.  We couldn't have
		 * yy_get_previous_state() go ahead and do it
		 * for us because it doesn't know how to deal
		 * with the possibility of jamming (and we don't
		 * want to build jamming into it because then it
		 * will run more slowly).
		 */

		yy_next_state = yy_try_NUL_trans( yy_current_state );

		yy_bp = (yytext_ptr) + YY_MORE_ADJ;

		if ( yy_next_state )
			{
			/* Consume the NUL. */
			yy_cp = ++(yy_c_buf_p);
			yy_current_state = yy_next_state;
			goto yy_match;
			}

		else
			{
			yy_cp = (yy_c_buf_p);
			goto yy_find_action;
			}
		}

	else switch ( yy_get_next_buffer(  ) )
		{
		case EOB_ACT_END_OF_FILE:
			{
			(yy_did_buffer_switch_on_eof) = 0;

			if ( yywrap(  ) )
				{
				/* Note: because we've taken care in
				 * yy_get_next_buffer() to have set up
				 * yytext, we can now set up
				 * yy_c_buf_p so that if some total
				 * hoser (like flex itself) wants to
				 * call the scanner after we return the
				 * YY_NULL, it'll still work - another
				 * YY_NULL will get returned.
				 */
				(yy_c_buf_p) = (yytext_ptr) + YY_MORE_ADJ;
```

```
                        yy_act = YY_STATE_EOF(YY_START);
                        goto do_action;
                        }

                    else
                        {
                        if ( ! (yy_did_buffer_switch_on_eof) )
                            YY_NEW_FILE;
                        }
                    break;
                    }

                case EOB_ACT_CONTINUE_SCAN:
                    (yy_c_buf_p) =
                        (yytext_ptr) + yy_amount_of_matched_text;

                    yy_current_state = yy_get_previous_state(  );

                    yy_cp = (yy_c_buf_p);
                    yy_bp = (yytext_ptr) + YY_MORE_ADJ;
                    goto yy_match;

                case EOB_ACT_LAST_MATCH:
                    (yy_c_buf_p) =
                    &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)];

                    yy_current_state = yy_get_previous_state(  );

                    yy_cp = (yy_c_buf_p);
                    yy_bp = (yytext_ptr) + YY_MORE_ADJ;
                    goto yy_find_action;
                }
            break;
            }

        default:
            YY_FATAL_ERROR(
                "fatal flex scanner internal error--no action found" );
        } /* end of action switch */
            } /* end of scanning one token */
        } /* end of user's declarations */
} /* end of yylex */

/* yy_get_next_buffer - try to read in a new buffer
 *
 * Returns a code representing an action:
 *      EOB_ACT_LAST_MATCH -
 *      EOB_ACT_CONTINUE_SCAN - continue scanning from current position
 *      EOB_ACT_END_OF_FILE - end of file
 */
static int yy_get_next_buffer (void)
```

```
{
	char *dest = YY_CURRENT_BUFFER_LVALUE->yy_ch_buf;
	char *source = (yytext_ptr);
	int number_to_move, i;
	int ret_val;

	if ( (yy_c_buf_p) > &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars) + 1] )
		YY_FATAL_ERROR(
		"fatal flex scanner internal error--end of buffer missed" );

	if ( YY_CURRENT_BUFFER_LVALUE->yy_fill_buffer == 0 )
		{ /* Don't try to fill the buffer, so this is an EOF. */
		if ( (yy_c_buf_p) - (yytext_ptr) - YY_MORE_ADJ == 1 )
			{
			/* We matched a single character, the EOB, so
			 * treat this as a final EOF.
			 */
			return EOB_ACT_END_OF_FILE;
			}

		else
			{
			/* We matched some text prior to the EOB, first
			 * process it.
			 */
			return EOB_ACT_LAST_MATCH;
			}
		}

	/* Try to read more data. */

	/* First move last chars to start of buffer. */
	number_to_move = (int) ((yy_c_buf_p) - (yytext_ptr) - 1);

	for ( i = 0; i < number_to_move; ++i )
		*(dest++) = *(source++);

	if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status ==
YY_BUFFER_EOF_PENDING )
		/* don't do the read, it's not guaranteed to return an EOF,
		 * just force an EOF
		 */
		YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars) = 0;

	else
		{
			int num_to_read =
			YY_CURRENT_BUFFER_LVALUE->yy_buf_size - number_to_move - 1;

		while ( num_to_read <= 0 )
			{ /* Not enough room in the buffer - grow it. */
```

```
                        /* just a shorter name for the current buffer */
                        YY_BUFFER_STATE b = YY_CURRENT_BUFFER_LVALUE;

                        int yy_c_buf_p_offset =
                                (int) ((yy_c_buf_p) - b->yy_ch_buf);

                        if ( b->yy_is_our_buffer )
                                {
                                int new_size = b->yy_buf_size * 2;

                                if ( new_size <= 0 )
                                        b->yy_buf_size += b->yy_buf_size / 8;
                                else
                                        b->yy_buf_size *= 2;

                                b->yy_ch_buf = (char *)
                                        /* Include room in for 2 EOB chars. */
                                        yyrealloc( (void *) b->yy_ch_buf,
                                                        (yy_size_t) (b->yy_buf_size + 2)  );
                                }
                        else
                                /* Can't grow it, we don't own it. */
                                b->yy_ch_buf = NULL;

                        if ( ! b->yy_ch_buf )
                                YY_FATAL_ERROR(
                                "fatal error - scanner input buffer overflow" );

                        (yy_c_buf_p) = &b->yy_ch_buf[yy_c_buf_p_offset];

                        num_to_read = YY_CURRENT_BUFFER_LVALUE->yy_buf_size -
                                                number_to_move - 1;

                        }

                if ( num_to_read > YY_READ_BUF_SIZE )
                        num_to_read = YY_READ_BUF_SIZE;

                /* Read in more data. */
                YY_INPUT( (&YY_CURRENT_BUFFER_LVALUE-
>yy_ch_buf[number_to_move]),
                        (yy_n_chars), num_to_read );

                YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
                }

        if ( (yy_n_chars) == 0 )
                {
                if ( number_to_move == YY_MORE_ADJ )
                        {
                        ret_val = EOB_ACT_END_OF_FILE;
                        yyrestart( yyin  );
```

```
                        }

                else
                        {
                        ret_val = EOB_ACT_LAST_MATCH;
                        YY_CURRENT_BUFFER_LVALUE->yy_buffer_status =
                                YY_BUFFER_EOF_PENDING;
                        }
                }

        else
                ret_val = EOB_ACT_CONTINUE_SCAN;

        if (((yy_n_chars) + number_to_move) > YY_CURRENT_BUFFER_LVALUE-
>yy_buf_size) {
                /* Extend the array by 50%, plus the number we really need. */
                int new_size = (yy_n_chars) + number_to_move + ((yy_n_chars) >> 1);
                YY_CURRENT_BUFFER_LVALUE->yy_ch_buf = (char *) yyrealloc(
                        (void *) YY_CURRENT_BUFFER_LVALUE->yy_ch_buf, (yy_size_t)
new_size  );
                if ( ! YY_CURRENT_BUFFER_LVALUE->yy_ch_buf )
                        YY_FATAL_ERROR( "out of dynamic memory in yy_get_next_buffer()" );
                /* "- 2" to take care of EOB's */
                YY_CURRENT_BUFFER_LVALUE->yy_buf_size = (int) (new_size - 2);
        }

        (yy_n_chars) += number_to_move;
        YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] =
YY_END_OF_BUFFER_CHAR;
        YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars) + 1] =
YY_END_OF_BUFFER_CHAR;

        (yytext_ptr) = &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[0];

        return ret_val;
}

/* yy_get_previous_state - get the state just before the EOB char was reached */

   static yy_state_type yy_get_previous_state (void)
{
        yy_state_type yy_current_state;
        char *yy_cp;

        yy_current_state = (yy_start);

        for ( yy_cp = (yytext_ptr) + YY_MORE_ADJ; yy_cp < (yy_c_buf_p); ++yy_cp )
                {
                YY_CHAR yy_c = (*yy_cp ? yy_ec[YY_SC_TO_UI(*yy_cp)] : 1);
                if ( yy_accept[yy_current_state] )
                        {
                        (yy_last_accepting_state) = yy_current_state;
```

```
                        (yy_last_accepting_cpos) = yy_cp;
                        }
                while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
                        {
                        yy_current_state = (int) yy_def[yy_current_state];
                        if ( yy_current_state >= 10 )
                                yy_c = yy_meta[yy_c];
                        }
                yy_current_state = yy_nxt[yy_base[yy_current_state] + yy_c];
                }

        return yy_current_state;
}

/* yy_try_NUL_trans - try to make a transition on the NUL character
 *
 * synopsis
 *      next_state = yy_try_NUL_trans( current_state );
 */
   static yy_state_type yy_try_NUL_trans  (yy_state_type yy_current_state )
{
        int yy_is_jam;
        char *yy_cp = (yy_c_buf_p);

        YY_CHAR yy_c = 1;
        if ( yy_accept[yy_current_state] )
                {
                (yy_last_accepting_state) = yy_current_state;
                (yy_last_accepting_cpos) = yy_cp;
                }
        while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
                {
                yy_current_state = (int) yy_def[yy_current_state];
                if ( yy_current_state >= 10 )
                        yy_c = yy_meta[yy_c];
                }
        yy_current_state = yy_nxt[yy_base[yy_current_state] + yy_c];
        yy_is_jam = (yy_current_state == 9);

                return yy_is_jam ? 0 : yy_current_state;
}

#ifndef YY_NO_UNPUT

   static void yyunput (int c, char * yy_bp )
{
        char *yy_cp;

   yy_cp = (yy_c_buf_p);

        /* undo effects of setting up yytext */
        *yy_cp = (yy_hold_char);
```

```
	if ( yy_cp < YY_CURRENT_BUFFER_LVALUE->yy_ch_buf + 2 )
		{ /* need to shift things up to make room */
		/* +2 for EOB chars. */
		int number_to_move = (yy_n_chars) + 2;
		char *dest = &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[
					YY_CURRENT_BUFFER_LVALUE->yy_buf_size + 2];
		char *source =
				&YY_CURRENT_BUFFER_LVALUE-
>yy_ch_buf[number_to_move];

		while ( source > YY_CURRENT_BUFFER_LVALUE->yy_ch_buf )
			*--dest = *--source;

		yy_cp += (int) (dest - source);
		yy_bp += (int) (dest - source);
		YY_CURRENT_BUFFER_LVALUE->yy_n_chars =
			(yy_n_chars) = (int) YY_CURRENT_BUFFER_LVALUE->yy_buf_size;

		if ( yy_cp < YY_CURRENT_BUFFER_LVALUE->yy_ch_buf + 2 )
			YY_FATAL_ERROR( "flex scanner push-back overflow" );
		}

	*--yy_cp = (char) c;

	(yytext_ptr) = yy_bp;
	(yy_hold_char) = *yy_cp;
	(yy_c_buf_p) = yy_cp;
}

#endif

#ifndef YY_NO_INPUT
#ifdef __cplusplus
   static int yyinput (void)
#else
   static int input  (void)
#endif

{
	int c;

	*(yy_c_buf_p) = (yy_hold_char);

	if ( *(yy_c_buf_p) == YY_END_OF_BUFFER_CHAR )
		{
		/* yy_c_buf_p now points to the character we want to return.
		 * If this occurs *before* the EOB characters, then it's a
		 * valid NUL; if not, then we've hit the end of the buffer.
		 */
		if ( (yy_c_buf_p) < &YY_CURRENT_BUFFER_LVALUE-
>yy_ch_buf[(yy_n_chars)] )
```

```
                        /* This was really a NUL. */
                        *(yy_c_buf_p) = '\0';

                    else
                        { /* need more input */
                        int offset = (int) ((yy_c_buf_p) - (yytext_ptr));
                        ++(yy_c_buf_p);

                        switch ( yy_get_next_buffer(  ) )
                            {
                            case EOB_ACT_LAST_MATCH:
                                /* This happens because yy_g_n_b()
                                 * sees that we've accumulated a
                                 * token and flags that we need to
                                 * try matching the token before
                                 * proceeding.  But for input(),
                                 * there's no matching to consider.
                                 * So convert the EOB_ACT_LAST_MATCH
                                 * to EOB_ACT_END_OF_FILE.
                                 */

                                /* Reset buffer status. */
                                yyrestart( yyin );

                                /*FALLTHROUGH*/

                            case EOB_ACT_END_OF_FILE:
                                {
                                if ( yywrap(  ) )
                                    return 0;

                                if ( ! (yy_did_buffer_switch_on_eof) )
                                    YY_NEW_FILE;
#ifdef __cplusplus
                                return yyinput();
#else
                                return input();
#endif
                                }

                            case EOB_ACT_CONTINUE_SCAN:
                                (yy_c_buf_p) = (yytext_ptr) + offset;
                                break;
                            }
                        }
                    }

                c = *(unsigned char *) (yy_c_buf_p);        /* cast for 8-bit char's */
                *(yy_c_buf_p) = '\0';  /* preserve yytext */
                (yy_hold_char) = *++(yy_c_buf_p);

                return c;
```

```
}
#endif  /* ifndef YY_NO_INPUT */

/** Immediately switch to a different input stream.
 * @param input_file A readable stream.
 *
 * @note This function does not reset the start condition to @c INITIAL .
 */
    void yyrestart  (FILE * input_file )
{

          if ( ! YY_CURRENT_BUFFER ){
        yyensure_buffer_stack ();
                    YY_CURRENT_BUFFER_LVALUE =
          yy_create_buffer( yyin, YY_BUF_SIZE );
          }

          yy_init_buffer( YY_CURRENT_BUFFER, input_file );
          yy_load_buffer_state(  );
}

/** Switch to a different input buffer.
 * @param new_buffer The new input buffer.
 *
 */
    void yy_switch_to_buffer  (YY_BUFFER_STATE  new_buffer )
{

          /* TODO. We should be able to replace this entire function body
           * with
           *              yypop_buffer_state();
           *              yypush_buffer_state(new_buffer);
     */
          yyensure_buffer_stack ();
          if ( YY_CURRENT_BUFFER == new_buffer )
                    return;

          if ( YY_CURRENT_BUFFER )
                    {
                    /* Flush out information for old buffer. */
                    *(yy_c_buf_p) = (yy_hold_char);
                    YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
                    YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
                    }

          YY_CURRENT_BUFFER_LVALUE = new_buffer;
          yy_load_buffer_state(  );

          /* We don't actually know whether we did this switch during
           * EOF (yywrap()) processing, but the only time this flag
           * is looked at is after yywrap() is called, so it's safe
           * to go ahead and always set it.
```

```
	 */
	(yy_did_buffer_switch_on_eof) = 1;
}

static void yy_load_buffer_state  (void)
{
	(yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
	(yytext_ptr) = (yy_c_buf_p) = YY_CURRENT_BUFFER_LVALUE->yy_buf_pos;
	yyin = YY_CURRENT_BUFFER_LVALUE->yy_input_file;
	(yy_hold_char) = *(yy_c_buf_p);
}

/** Allocate and initialize an input buffer state.
 * @param file A readable stream.
 * @param size The character buffer size in bytes. When in doubt, use @c YY_BUF_SIZE.
 *
 * @return the allocated buffer state.
 */
   YY_BUFFER_STATE yy_create_buffer  (FILE * file, int  size )
{
	YY_BUFFER_STATE b;

	b = (YY_BUFFER_STATE) yyalloc( sizeof( struct yy_buffer_state )  );
	if ( ! b )
		YY_FATAL_ERROR( "out of dynamic memory in yy_create_buffer()" );

	b->yy_buf_size = size;

	/* yy_ch_buf has to be 2 characters longer than the size given because
	 * we need to put in 2 end-of-buffer characters.
	 */
	b->yy_ch_buf = (char *) yyalloc( (yy_size_t) (b->yy_buf_size + 2)  );
	if ( ! b->yy_ch_buf )
		YY_FATAL_ERROR( "out of dynamic memory in yy_create_buffer()" );

	b->yy_is_our_buffer = 1;

	yy_init_buffer( b, file );

	return b;
}

/** Destroy the buffer.
 * @param b a buffer created with yy_create_buffer()
 *
 */
   void yy_delete_buffer (YY_BUFFER_STATE  b )
{

	if ( ! b )
		return;
```

```
	if ( b == YY_CURRENT_BUFFER ) /* Not sure if we should pop here. */
		YY_CURRENT_BUFFER_LVALUE = (YY_BUFFER_STATE) 0;

	if ( b->yy_is_our_buffer )
		yyfree( (void *) b->yy_ch_buf  );

	yyfree( (void *) b  );
}

/* Initializes or reinitializes a buffer.
 * This function is sometimes called more than once on the same buffer,
 * such as during a yyrestart() or at EOF.
 */
   static void yy_init_buffer  (YY_BUFFER_STATE  b, FILE * file )

{
	int oerrno = errno;

	yy_flush_buffer( b );

	b->yy_input_file = file;
	b->yy_fill_buffer = 1;

   /* If b is the current buffer, then yy_init_buffer was _probably_
    * called from yyrestart() or through yy_get_next_buffer.
    * In that case, we don't want to reset the lineno or column.
    */
   if (b != YY_CURRENT_BUFFER){
      b->yy_bs_lineno = 1;
      b->yy_bs_column = 0;
   }

      b->yy_is_interactive = file ? (isatty( fileno(file) ) > 0) : 0;

	errno = oerrno;
}

/** Discard all buffered characters. On the next scan, YY_INPUT will be called.
 * @param b the buffer state to be flushed, usually @c YY_CURRENT_BUFFER.
 *
 */
   void yy_flush_buffer (YY_BUFFER_STATE  b )
{
	if ( ! b )
		return;

	b->yy_n_chars = 0;

	/* We always need two end-of-buffer characters.  The first causes
	 * a transition to the end-of-buffer state.  The second causes
	 * a jam in that state.
	 */
```

```
	b->yy_ch_buf[0] = YY_END_OF_BUFFER_CHAR;
	b->yy_ch_buf[1] = YY_END_OF_BUFFER_CHAR;

	b->yy_buf_pos = &b->yy_ch_buf[0];

	b->yy_at_bol = 1;
	b->yy_buffer_status = YY_BUFFER_NEW;

	if ( b == YY_CURRENT_BUFFER )
		yy_load_buffer_state(  );
}

/** Pushes the new state onto the stack. The new state becomes
 *  the current state. This function will allocate the stack
 *  if necessary.
 *  @param new_buffer The new state.
 *
 */
void yypush_buffer_state (YY_BUFFER_STATE new_buffer )
{
	if (new_buffer == NULL)
		return;

	yyensure_buffer_stack();

	/* This block is copied from yy_switch_to_buffer. */
	if ( YY_CURRENT_BUFFER )
		{
		/* Flush out information for old buffer. */
		*(yy_c_buf_p) = (yy_hold_char);
		YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
		YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
		}

	/* Only push if top exists. Otherwise, replace top. */
	if (YY_CURRENT_BUFFER)
		(yy_buffer_stack_top)++;
	YY_CURRENT_BUFFER_LVALUE = new_buffer;

	/* copied from yy_switch_to_buffer. */
	yy_load_buffer_state(  );
	(yy_did_buffer_switch_on_eof) = 1;
}

/** Removes and deletes the top of the stack, if present.
 *  The next element becomes the new top.
 *
 */
void yypop_buffer_state (void)
{
	if (!YY_CURRENT_BUFFER)
		return;
```

```
	yy_delete_buffer(YY_CURRENT_BUFFER );
	YY_CURRENT_BUFFER_LVALUE = NULL;
	if ((yy_buffer_stack_top) > 0)
		--(yy_buffer_stack_top);

	if (YY_CURRENT_BUFFER) {
		yy_load_buffer_state(  );
		(yy_did_buffer_switch_on_eof) = 1;
	}
}

/* Allocates the stack if it does not exist.
 *  Guarantees space for at least one push.
 */
static void yyensure_buffer_stack (void)
{
	yy_size_t num_to_alloc;

	if (!(yy_buffer_stack)) {

		/* First allocation is just for 2 elements, since we don't know if this
		 * scanner will even need a stack. We use 2 instead of 1 to avoid an
		 * immediate realloc on the next call.
       */
    num_to_alloc = 1; /* After all that talk, this was set to 1 anyways... */
		(yy_buffer_stack) = (struct yy_buffer_state**)yyalloc
							(num_to_alloc * sizeof(struct
yy_buffer_state*)
							);
		if ( ! (yy_buffer_stack) )
			YY_FATAL_ERROR( "out of dynamic memory in
yyensure_buffer_stack()" );

		memset((yy_buffer_stack), 0, num_to_alloc * sizeof(struct yy_buffer_state*));

		(yy_buffer_stack_max) = num_to_alloc;
		(yy_buffer_stack_top) = 0;
		return;
	}

	if ((yy_buffer_stack_top) >= ((yy_buffer_stack_max)) - 1){

		/* Increase the buffer to prepare for a possible push. */
		yy_size_t grow_size = 8 /* arbitrary grow size */;

		num_to_alloc = (yy_buffer_stack_max) + grow_size;
		(yy_buffer_stack) = (struct yy_buffer_state**)yyrealloc
							((yy_buffer_stack),
							num_to_alloc * sizeof(struct
yy_buffer_state*)
							);
```

```
		if ( ! (yy_buffer_stack) )
			YY_FATAL_ERROR( "out of dynamic memory in
yyensure_buffer_stack()" );

		/* zero only the new slots.*/
		memset((yy_buffer_stack) + (yy_buffer_stack_max), 0, grow_size * sizeof(struct
yy_buffer_state*));
		(yy_buffer_stack_max) = num_to_alloc;
	}
}

/** Setup the input buffer state to scan directly from a user-specified character buffer.
 * @param base the character buffer
 * @param size the size in bytes of the character buffer
 *
 * @return the newly allocated buffer state object.
 */
YY_BUFFER_STATE yy_scan_buffer  (char * base, yy_size_t  size )
{
	YY_BUFFER_STATE b;

	if ( size < 2 ||
	    base[size-2] != YY_END_OF_BUFFER_CHAR ||
	    base[size-1] != YY_END_OF_BUFFER_CHAR )
		/* They forgot to leave room for the EOB's. */
		return NULL;

	b = (YY_BUFFER_STATE) yyalloc( sizeof( struct yy_buffer_state )  );
	if ( ! b )
		YY_FATAL_ERROR( "out of dynamic memory in yy_scan_buffer()" );

	b->yy_buf_size = (int) (size - 2);	/* "- 2" to take care of EOB's */
	b->yy_buf_pos = b->yy_ch_buf = base;
	b->yy_is_our_buffer = 0;
	b->yy_input_file = NULL;
	b->yy_n_chars = b->yy_buf_size;
	b->yy_is_interactive = 0;
	b->yy_at_bol = 1;
	b->yy_fill_buffer = 0;
	b->yy_buffer_status = YY_BUFFER_NEW;

	yy_switch_to_buffer( b  );

	return b;
}

/** Setup the input buffer state to scan a string. The next call to yylex() will
 * scan from a @e copy of @a str.
 * @param yystr a NUL-terminated string to scan
 *
 * @return the newly allocated buffer state object.
 * @note If you want to scan bytes that may contain NUL values, then use
```

```
 *      yy_scan_bytes() instead.
 */
YY_BUFFER_STATE yy_scan_string (const char * yystr )
{

        return yy_scan_bytes( yystr, (int) strlen(yystr) );
}

/** Setup the input buffer state to scan the given bytes. The next call to yylex() will
 * scan from a @e copy of @a bytes.
 * @param yybytes the byte buffer to scan
 * @param _yybytes_len the number of bytes in the buffer pointed to by @a bytes.
 *
 * @return the newly allocated buffer state object.
 */
YY_BUFFER_STATE yy_scan_bytes  (const char * yybytes, int  _yybytes_len )
{
        YY_BUFFER_STATE b;
        char *buf;
        yy_size_t n;
        int i;

        /* Get memory for full buffer, including space for trailing EOB's. */
        n = (yy_size_t) (_yybytes_len + 2);
        buf = (char *) yyalloc( n  );
        if ( ! buf )
                YY_FATAL_ERROR( "out of dynamic memory in yy_scan_bytes()" );

        for ( i = 0; i < _yybytes_len; ++i )
                buf[i] = yybytes[i];

        buf[_yybytes_len] = buf[_yybytes_len+1] = YY_END_OF_BUFFER_CHAR;

        b = yy_scan_buffer( buf, n );
        if ( ! b )
                YY_FATAL_ERROR( "bad buffer in yy_scan_bytes()" );

        /* It's okay to grow etc. this buffer, and we should throw it
         * away when we're done.
         */
        b->yy_is_our_buffer = 1;

        return b;
}

#ifndef YY_EXIT_FAILURE
#define YY_EXIT_FAILURE 2
#endif

static void yynoreturn yy_fatal_error (const char* msg )
{
                        fprintf( stderr, "%s\n", msg );
```

```
		exit( YY_EXIT_FAILURE );
}

/* Redefine yyless() so it works in section 3 code. */

#undef yyless
#define yyless(n) \
	do \
		{ \
		/* Undo effects of setting up yytext. */ \
	int yyless_macro_arg = (n); \
	YY_LESS_LINENO(yyless_macro_arg);\
		yytext[yyleng] = (yy_hold_char); \
		(yy_c_buf_p) = yytext + yyless_macro_arg; \
		(yy_hold_char) = *(yy_c_buf_p); \
		*(yy_c_buf_p) = '\0'; \
		yyleng = yyless_macro_arg; \
		} \
	while ( 0 )

/* Accessor  methods (get/set functions) to struct members. */

/** Get the current line number.
 *
 */
int yyget_lineno  (void)
{

    return yylineno;
}

/** Get the input stream.
 *
 */
FILE *yyget_in  (void)
{
	return yyin;
}

/** Get the output stream.
 *
 */
FILE *yyget_out  (void)
{
	return yyout;
}

/** Get the length of the current token.
 *
 */
int yyget_leng  (void)
{
```

```
	return yyleng;
}

/** Get the current token.
 *
 */

char *yyget_text  (void)
{
	return yytext;
}

/** Set the current line number.
 * @param _line_number line number
 *
 */
void yyset_lineno (int  _line_number )
{

    yylineno = _line_number;
}

/** Set the input stream. This does not discard the current
 * input buffer.
 * @param _in_str A readable stream.
 *
 * @see yy_switch_to_buffer
 */
void yyset_in (FILE *  _in_str )
{
	yyin = _in_str ;
}

void yyset_out (FILE *  _out_str )
{
	yyout = _out_str ;
}

int yyget_debug  (void)
{
	return yy_flex_debug;
}

void yyset_debug (int  _bdebug )
{
	yy_flex_debug = _bdebug ;
}

static int yy_init_globals (void)
{
	/* Initialization is the same as for the non-reentrant scanner.
     * This function is called from yylex_destroy(), so don't allocate here.
```

```
     */

    (yy_buffer_stack) = NULL;
    (yy_buffer_stack_top) = 0;
    (yy_buffer_stack_max) = 0;
    (yy_c_buf_p) = NULL;
    (yy_init) = 0;
    (yy_start) = 0;

/* Defined in main.c */
#ifdef YY_STDINIT
    yyin = stdin;
    yyout = stdout;
#else
    yyin = NULL;
    yyout = NULL;
#endif

    /* For future reference: Set errno on error, since we are called by
     * yylex_init()
     */
    return 0;
}

/* yylex_destroy is for both reentrant and non-reentrant scanners. */
int yylex_destroy  (void)
{

    /* Pop the buffer stack, destroying each element. */
        while(YY_CURRENT_BUFFER){
                yy_delete_buffer( YY_CURRENT_BUFFER  );
                YY_CURRENT_BUFFER_LVALUE = NULL;
                yypop_buffer_state();
        }

        /* Destroy the stack itself. */
        yyfree((yy_buffer_stack) );
        (yy_buffer_stack) = NULL;

    /* Reset the globals. This is important in a non-reentrant scanner so the next time
     * yylex() is called, initialization will occur. */
    yy_init_globals( );

    return 0;
}

/*
 * Internal utility routines.
 */

#ifndef yytext_ptr
static void yy_flex_strncpy (char* s1, const char * s2, int n )
```

```
{

        int i;
        for ( i = 0; i < n; ++i )
                s1[i] = s2[i];
}
#endif

#ifdef YY_NEED_STRLEN
static int yy_flex_strlen (const char * s )
{
        int n;
        for ( n = 0; s[n]; ++n )
                ;

        return n;
}
#endif

void *yyalloc (yy_size_t  size )
{
                        return malloc(size);
}

void *yyrealloc  (void * ptr, yy_size_t  size )
{

        /* The cast to (char *) in the following accommodates both
         * implementations that use char* generic pointers, and those
         * that use void* generic pointers.  It works with the latter
         * because both ANSI C and C++ allow castless assignment from
         * any pointer type to void*, and deal with argument conversions
         * as though doing an assignment.
         */
        return realloc(ptr, size);
}

void yyfree (void * ptr )
{
                        free( (char *) ptr );     /* see yyrealloc() for (char *) cast */
}

#define YYTABLES_NAME "yytables"

#line 11 "pgm.l"

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

pgm.l

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ { yylval=atoi(yytext); return NUMBER;}
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

pgm.y

```
%{
#include<stdio.h>
int flag=0;
%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
ArithmeticExpression: E{printf("output:%d\n",$$);return 0;};
E:E'+'E {$$=$1+$3;}
|E'-'E {$$=$1-$3;}
|E'*'E {$$=$1*$3;}
|E'/'E {$$=$1/$3;}
|E'%'E {$$=$1%$3;}
|'('E')' {$$=$2;}
| NUMBER {$$=$1;}
;
%%
void main()
{
printf("\nEnter the input:\n");
yyparse();
if(flag==0)
printf("Expression is Valid\n");
}
void yyerror()
{
printf("Expression is invalid\n");
flag=1;
}
```

```
//yacc -d Pg13.y
//lex Pg13.l
//gcc lex.yy.c y.tab.c -w
//./a.out
```

y.tab.c

```
/* A Bison parser, made by GNU Bison 3.8.2.  */

/* Bison implementation for Yacc-like parsers in C

   Copyright (C) 1984, 1989-1990, 2000-2015, 2018-2021 Free Software Foundation,
   Inc.

   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 3 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, see <https://www.gnu.org/licenses/>.  */

/* As a special exception, you may create a larger work that contains
   part or all of the Bison parser skeleton and distribute that work
   under terms of your choice, so long as that work isn't itself a
   parser generator using the skeleton or a modified version thereof
   as a parser skeleton.  Alternatively, if you modify or redistribute
   the parser skeleton itself, you may (at your option) remove this
   special exception, which will cause the skeleton and the resulting
   Bison output files to be licensed under the GNU General Public
   License without this special exception.

   This special exception was added by the Free Software Foundation in
   version 2.2 of Bison.  */

/* C LALR(1) parser skeleton written by Richard Stallman, by
   simplifying the original so-called "semantic" parser.  */

/* DO NOT RELY ON FEATURES THAT ARE NOT DOCUMENTED in the manual,
```

```
   especially those whose name start with YY_ or yy_.  They are
   private implementation details that can be changed or removed.  */

/* All symbols defined below should begin with yy or YY, to avoid
   infringing on user name space.  This should be done even for local
   variables, as they might otherwise be expanded by user macros.
   There are some unavoidable exceptions within include files to
   define necessary library symbols; they are noted "INFRINGES ON
   USER NAME SPACE" below.  */

/* Identify Bison output, and Bison version.  */
#define YYBISON 30802

/* Bison version string.  */
#define YYBISON_VERSION "3.8.2"

/* Skeleton name.  */
#define YYSKELETON_NAME "yacc.c"

/* Pure parsers.  */
#define YYPURE 0

/* Push parsers.  */
#define YYPUSH 0

/* Pull parsers.  */
#define YYPULL 1




/* First part of user prologue.  */
#line 1 "pgm.y"

#include<stdio.h>
int flag=0;

#line 76 "y.tab.c"

# ifndef YY_CAST
#  ifdef __cplusplus
#   define YY_CAST(Type, Val) static_cast<Type> (Val)
#   define YY_REINTERPRET_CAST(Type, Val) reinterpret_cast<Type> (Val)
#  else
#   define YY_CAST(Type, Val) ((Type) (Val))
#   define YY_REINTERPRET_CAST(Type, Val) ((Type) (Val))
#  endif
# endif
# ifndef YY_NULLPTR
#  if defined __cplusplus
#   if 201103L <= __cplusplus
#    define YY_NULLPTR nullptr
```

```
#   else
#    define YY_NULLPTR 0
#   endif
#  else
#   define YY_NULLPTR ((void*)0)
#  endif
# endif

/* Use api.header.include to #include this header
   instead of duplicating it here.  */
#ifndef YY_YY_Y_TAB_H_INCLUDED
# define YY_YY_Y_TAB_H_INCLUDED
/* Debug traces.  */
#ifndef YYDEBUG
# define YYDEBUG 0
#endif
#if YYDEBUG
extern int yydebug;
#endif

/* Token kinds.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
  enum yytokentype
  {
    YYEMPTY = -2,
    YYEOF = 0,                     /* "end of file"  */
    YYerror = 256,                 /* error  */
    YYUNDEF = 257,                   /* "invalid token"  */
    NUMBER = 258                   /* NUMBER  */
  };
  typedef enum yytokentype yytoken_kind_t;
#endif
/* Token kinds.  */
#define YYEMPTY -2
#define YYEOF 0
#define YYerror 256
#define YYUNDEF 257
#define NUMBER 258

/* Value type.  */
#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
typedef int YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define YYSTYPE_IS_DECLARED 1
#endif


extern YYSTYPE yylval;


int yyparse (void);
```

```
#endif /* !YY_YY_Y_TAB_H_INCLUDED  */
/* Symbol kind.  */
enum yysymbol_kind_t
{
  YYSYMBOL_YYEMPTY = -2,
  YYSYMBOL_YYEOF = 0,                      /* "end of file"  */
  YYSYMBOL_YYerror = 1,                    /* error  */
  YYSYMBOL_YYUNDEF = 2,                    /* "invalid token"  */
  YYSYMBOL_NUMBER = 3,                     /* NUMBER  */
  YYSYMBOL_4_ = 4,                         /* '+'  */
  YYSYMBOL_5_ = 5,                         /* '-'  */
  YYSYMBOL_6_ = 6,                         /* '*'  */
  YYSYMBOL_7_ = 7,                         /* '/'  */
  YYSYMBOL_8_ = 8,                         /* '%'  */
  YYSYMBOL_9_ = 9,                         /* '('  */
  YYSYMBOL_10_ = 10,                       /* ')'  */
  YYSYMBOL_YYACCEPT = 11,                  /* $accept  */
  YYSYMBOL_ArithmeticExpression = 12,      /* ArithmeticExpression  */
  YYSYMBOL_E = 13                          /* E  */
};
typedef enum yysymbol_kind_t yysymbol_kind_t;




#ifdef short
# undef short
#endif

/* On compilers that do not define __PTRDIFF_MAX__ etc., make sure
   <limits.h> and (if available) <stdint.h> are included
   so that the code can choose integer types of a good width.  */

#ifndef __PTRDIFF_MAX__
# include <limits.h> /* INFRINGES ON USER NAME SPACE */
# if defined __STDC_VERSION__ && 199901 <= __STDC_VERSION__
#  include <stdint.h> /* INFRINGES ON USER NAME SPACE */
#  define YY_STDINT_H
# endif
#endif

/* Narrow types that promote to a signed type and that can represent a
   signed or unsigned integer of at least N bits.  In tables they can
   save space and decrease cache pressure.  Promoting to a signed type
   helps avoid bugs in integer arithmetic.  */

#ifdef __INT_LEAST8_MAX__
typedef __INT_LEAST8_TYPE__ yytype_int8;
#elif defined YY_STDINT_H
typedef int_least8_t yytype_int8;
```

```
#else
typedef signed char yytype_int8;
#endif

#ifdef __INT_LEAST16_MAX__
typedef __INT_LEAST16_TYPE__ yytype_int16;
#elif defined YY_STDINT_H
typedef int_least16_t yytype_int16;
#else
typedef short yytype_int16;
#endif

/* Work around bug in HP-UX 11.23, which defines these macros
   incorrectly for preprocessor constants.  This workaround can likely
   be removed in 2023, as HPE has promised support for HP-UX 11.23
   (aka HP-UX 11i v2) only through the end of 2022; see Table 2 of
   <https://h20195.www2.hpe.com/V2/getpdf.aspx/4AA4-7673ENW.pdf>.  */
#ifdef __hpux
# undef UINT_LEAST8_MAX
# undef UINT_LEAST16_MAX
# define UINT_LEAST8_MAX 255
# define UINT_LEAST16_MAX 65535
#endif

#if defined __UINT_LEAST8_MAX__ && __UINT_LEAST8_MAX__ <= __INT_MAX__
typedef __UINT_LEAST8_TYPE__ yytype_uint8;
#elif (!defined __UINT_LEAST8_MAX__ && defined YY_STDINT_H \
       && UINT_LEAST8_MAX <= INT_MAX)
typedef uint_least8_t yytype_uint8;
#elif !defined __UINT_LEAST8_MAX__ && UCHAR_MAX <= INT_MAX
typedef unsigned char yytype_uint8;
#else
typedef short yytype_uint8;
#endif

#if defined __UINT_LEAST16_MAX__ && __UINT_LEAST16_MAX__ <= __INT_MAX__
typedef __UINT_LEAST16_TYPE__ yytype_uint16;
#elif (!defined __UINT_LEAST16_MAX__ && defined YY_STDINT_H \
       && UINT_LEAST16_MAX <= INT_MAX)
typedef uint_least16_t yytype_uint16;
#elif !defined __UINT_LEAST16_MAX__ && USHRT_MAX <= INT_MAX
typedef unsigned short yytype_uint16;
#else
typedef int yytype_uint16;
#endif

#ifndef YYPTRDIFF_T
# if defined __PTRDIFF_TYPE__ && defined __PTRDIFF_MAX__
#  define YYPTRDIFF_T __PTRDIFF_TYPE__
#  define YYPTRDIFF_MAXIMUM __PTRDIFF_MAX__
# elif defined PTRDIFF_MAX
#  ifndef ptrdiff_t
```

```
#   include <stddef.h> /* INFRINGES ON USER NAME SPACE */
#  endif
#  define YYPTRDIFF_T ptrdiff_t
#  define YYPTRDIFF_MAXIMUM PTRDIFF_MAX
# else
#  define YYPTRDIFF_T long
#  define YYPTRDIFF_MAXIMUM LONG_MAX
# endif
#endif

#ifndef YYSIZE_T
# ifdef __SIZE_TYPE__
#  define YYSIZE_T __SIZE_TYPE__
# elif defined size_t
#  define YYSIZE_T size_t
# elif defined __STDC_VERSION__ && 199901 <= __STDC_VERSION__
#  include <stddef.h> /* INFRINGES ON USER NAME SPACE */
#  define YYSIZE_T size_t
# else
#  define YYSIZE_T unsigned
# endif
#endif

#define YYSIZE_MAXIMUM                                  \
  YY_CAST (YYPTRDIFF_T,                                 \
           (YYPTRDIFF_MAXIMUM < YY_CAST (YYSIZE_T, -1)  \
            ? YYPTRDIFF_MAXIMUM                         \
            : YY_CAST (YYSIZE_T, -1)))

#define YYSIZEOF(X) YY_CAST (YYPTRDIFF_T, sizeof (X))


/* Stored state numbers (used for stacks). */
typedef yytype_int8 yy_state_t;

/* State numbers in computations.  */
typedef int yy_state_fast_t;

#ifndef YY_
# if defined YYENABLE_NLS && YYENABLE_NLS
#  if ENABLE_NLS
#   include <libintl.h> /* INFRINGES ON USER NAME SPACE */
#   define YY_(Msgid) dgettext ("bison-runtime", Msgid)
#  endif
# endif
# ifndef YY_
#  define YY_(Msgid) Msgid
# endif
#endif


#ifndef YY_ATTRIBUTE_PURE
```

```
# if defined __GNUC__ && 2 < __GNUC__ + (96 <= __GNUC_MINOR__)
#  define YY_ATTRIBUTE_PURE __attribute__ ((__pure__))
# else
#  define YY_ATTRIBUTE_PURE
# endif
#endif

#ifndef YY_ATTRIBUTE_UNUSED
# if defined __GNUC__ && 2 < __GNUC__ + (7 <= __GNUC_MINOR__)
#  define YY_ATTRIBUTE_UNUSED __attribute__ ((__unused__))
# else
#  define YY_ATTRIBUTE_UNUSED
# endif
#endif

/* Suppress unused-variable warnings by "using" E.  */
#if ! defined lint || defined __GNUC__
# define YY_USE(E) ((void) (E))
#else
# define YY_USE(E) /* empty */
#endif

/* Suppress an incorrect diagnostic about yylval being uninitialized.  */
#if defined __GNUC__ && ! defined __ICC && 406 <= __GNUC__ * 100 +
__GNUC_MINOR__
# if __GNUC__ * 100 + __GNUC_MINOR__ < 407
#  define YY_IGNORE_MAYBE_UNINITIALIZED_BEGIN                           \
    _Pragma ("GCC diagnostic push")                                     \
    _Pragma ("GCC diagnostic ignored \"-Wuninitialized\"")
# else
#  define YY_IGNORE_MAYBE_UNINITIALIZED_BEGIN                           \
    _Pragma ("GCC diagnostic push")                                     \
    _Pragma ("GCC diagnostic ignored \"-Wuninitialized\"")              \
    _Pragma ("GCC diagnostic ignored \"-Wmaybe-uninitialized\"")
# endif
# define YY_IGNORE_MAYBE_UNINITIALIZED_END      \
    _Pragma ("GCC diagnostic pop")
#else
# define YY_INITIAL_VALUE(Value) Value
#endif
#ifndef YY_IGNORE_MAYBE_UNINITIALIZED_BEGIN
# define YY_IGNORE_MAYBE_UNINITIALIZED_BEGIN
# define YY_IGNORE_MAYBE_UNINITIALIZED_END
#endif
#ifndef YY_INITIAL_VALUE
# define YY_INITIAL_VALUE(Value) /* Nothing. */
#endif

#if defined __cplusplus && defined __GNUC__ && ! defined __ICC && 6 <= __GNUC__
# define YY_IGNORE_USELESS_CAST_BEGIN                           \
    _Pragma ("GCC diagnostic push")                             \
    _Pragma ("GCC diagnostic ignored \"-Wuseless-cast\"")
```

```
# define YY_IGNORE_USELESS_CAST_END            \
    _Pragma ("GCC diagnostic pop")
#endif
#ifndef YY_IGNORE_USELESS_CAST_BEGIN
# define YY_IGNORE_USELESS_CAST_BEGIN
# define YY_IGNORE_USELESS_CAST_END
#endif


#define YY_ASSERT(E) ((void) (0 && (E)))

#if !defined yyoverflow

/* The parser invokes alloca or malloc; define the necessary symbols.  */

# ifdef YYSTACK_USE_ALLOCA
#  if YYSTACK_USE_ALLOCA
#   ifdef __GNUC__
#    define YYSTACK_ALLOC __builtin_alloca
#   elif defined __BUILTIN_VA_ARG_INCR
#    include <alloca.h> /* INFRINGES ON USER NAME SPACE */
#   elif defined _AIX
#    define YYSTACK_ALLOC __alloca
#   elif defined _MSC_VER
#    include <malloc.h> /* INFRINGES ON USER NAME SPACE */
#    define alloca _alloca
#   else
#    define YYSTACK_ALLOC alloca
#    if ! defined _ALLOCA_H && ! defined EXIT_SUCCESS
#     include <stdlib.h> /* INFRINGES ON USER NAME SPACE */
     /* Use EXIT_SUCCESS as a witness for stdlib.h.  */
#     ifndef EXIT_SUCCESS
#      define EXIT_SUCCESS 0
#     endif
#    endif
#   endif
#  endif
# endif

# ifdef YYSTACK_ALLOC
   /* Pacify GCC's 'empty if-body' warning.  */
#  define YYSTACK_FREE(Ptr) do { /* empty */; } while (0)
#  ifndef YYSTACK_ALLOC_MAXIMUM
    /* The OS might guarantee only one guard page at the bottom of the stack,
       and a page size can be as small as 4096 bytes.  So we cannot safely
       invoke alloca (N) if N exceeds 4096.  Use a slightly smaller number
       to allow for a few compiler-allocated temporary stack slots.  */
#   define YYSTACK_ALLOC_MAXIMUM 4032 /* reasonable circa 2006 */
#  endif
# else
#  define YYSTACK_ALLOC YYMALLOC
#  define YYSTACK_FREE YYFREE
```

```
#  ifndef YYSTACK_ALLOC_MAXIMUM
#   define YYSTACK_ALLOC_MAXIMUM YYSIZE_MAXIMUM
#  endif
#  if (defined __cplusplus && ! defined EXIT_SUCCESS \
      && ! ((defined YYMALLOC || defined malloc) \
          && (defined YYFREE || defined free)))
#   include <stdlib.h> /* INFRINGES ON USER NAME SPACE */
#   ifndef EXIT_SUCCESS
#    define EXIT_SUCCESS 0
#   endif
#  endif
#  ifndef YYMALLOC
#   define YYMALLOC malloc
#   if ! defined malloc && ! defined EXIT_SUCCESS
void *malloc (YYSIZE_T); /* INFRINGES ON USER NAME SPACE */
#   endif
#  endif
#  ifndef YYFREE
#   define YYFREE free
#   if ! defined free && ! defined EXIT_SUCCESS
void free (void *); /* INFRINGES ON USER NAME SPACE */
#   endif
#  endif
# endif
#endif /* !defined yyoverflow */

#if (! defined yyoverflow \
     && (! defined __cplusplus \
         || (defined YYSTYPE_IS_TRIVIAL && YYSTYPE_IS_TRIVIAL)))

/* A type that is properly aligned for any stack member.  */
union yyalloc
{
  yy_state_t yyss_alloc;
  YYSTYPE yyvs_alloc;
};

/* The size of the maximum gap between one aligned stack and the next.  */
# define YYSTACK_GAP_MAXIMUM (YYSIZEOF (union yyalloc) - 1)

/* The size of an array large to enough to hold all stacks, each with
   N elements.  */
# define YYSTACK_BYTES(N) \
     ((N) * (YYSIZEOF (yy_state_t) + YYSIZEOF (YYSTYPE)) \
      + YYSTACK_GAP_MAXIMUM)

# define YYCOPY_NEEDED 1

/* Relocate STACK from its old location to the new one.  The
   local variables YYSIZE and YYSTACKSIZE give the old and new number of
   elements in the stack, and YYPTR gives the new location of the
   stack.  Advance YYPTR to a properly aligned location for the next
```

```
   stack.  */
# define YYSTACK_RELOCATE(Stack_alloc, Stack)                           \
    do                                                                  \
      {                                                                 \
        YYPTRDIFF_T yynewbytes;                                         \
        YYCOPY (&yyptr->Stack_alloc, Stack, yysize);                    \
        Stack = &yyptr->Stack_alloc;                                    \
        yynewbytes = yystacksize * YYSIZEOF (*Stack) + YYSTACK_GAP_MAXIMUM; \
        yyptr += yynewbytes / YYSIZEOF (*yyptr);                        \
      }                                                                 \
    while (0)

#endif

#if defined YYCOPY_NEEDED && YYCOPY_NEEDED
/* Copy COUNT objects from SRC to DST.  The source and destination do
   not overlap.  */
# ifndef YYCOPY
#  if defined __GNUC__ && 1 < __GNUC__
#   define YYCOPY(Dst, Src, Count) \
      __builtin_memcpy (Dst, Src, YY_CAST (YYSIZE_T, (Count)) * sizeof (*(Src)))
#  else
#   define YYCOPY(Dst, Src, Count)              \
      do                                        \
        {                                       \
          YYPTRDIFF_T yyi;                      \
          for (yyi = 0; yyi < (Count); yyi++)   \
            (Dst)[yyi] = (Src)[yyi];            \
        }                                       \
      while (0)
#  endif
# endif
#endif /* !YYCOPY_NEEDED */

/* YYFINAL -- State number of the termination state.  */
#define YYFINAL  6
/* YYLAST -- Last index in YYTABLE.  */
#define YYLAST   26

/* YYNTOKENS -- Number of terminals.  */
#define YYNTOKENS  11
/* YYNNTS -- Number of nonterminals.  */
#define YYNNTS  3
/* YYNRULES -- Number of rules.  */
#define YYNRULES  9
/* YYNSTATES -- Number of states.  */
#define YYNSTATES  18

/* YYMAXUTOK -- Last valid token kind.  */
#define YYMAXUTOK   258
```

```
/* YYTRANSLATE(TOKEN-NUM) -- Symbol number corresponding to TOKEN-NUM
   as returned by yylex, with out-of-bounds checking.  */
#define YYTRANSLATE(YYX)                                \
  (0 <= (YYX) && (YYX) <= YYMAXUTOK                     \
   ? YY_CAST (yysymbol_kind_t, yytranslate[YYX])        \
   : YYSYMBOL_YYUNDEF)

/* YYTRANSLATE[TOKEN-NUM] -- Symbol number corresponding to TOKEN-NUM
   as returned by yylex.  */
static const yytype_int8 yytranslate[] =
{
       0,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     8,     2,     2,
       9,    10,     6,     4,     2,     5,     2,     7,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     2,     2,     2,     2,
       2,     2,     2,     2,     2,     2,     1,     2,     3
};

#if YYDEBUG
/* YYRLINE[YYN] -- Source line where rule number YYN was defined.  */
static const yytype_int8 yyrline[] =
{
       0,    10,    10,    11,    12,    13,    14,    15,    16,    17
};
#endif

/** Accessing symbol of state STATE.  */
#define YY_ACCESSING_SYMBOL(State) YY_CAST (yysymbol_kind_t, yystos[State])

#if YYDEBUG || 0
/* The user-facing name of the symbol whose (internal) number is
```

```
   YYSYMBOL.  No bounds checking.  */
static const char *yysymbol_name (yysymbol_kind_t yysymbol) YY_ATTRIBUTE_UNUSED;

/* YYTNAME[SYMBOL-NUM] -- String name of the symbol SYMBOL-NUM.
   First, the terminals, then, starting at YYNTOKENS, nonterminals.  */
static const char *const yytname[] =
{
  "\"end of file\"", "error", "\"invalid token\"", "NUMBER", "'+'", "'-'",
  "'*'", "'/'", "'%'", "'('", "')'", "$accept", "ArithmeticExpression",
  "E", YY_NULLPTR
};

static const char *
yysymbol_name (yysymbol_kind_t yysymbol)
{
  return yytname[yysymbol];
}
#endif

#define YYPACT_NINF (-6)

#define yypact_value_is_default(Yyn) \
  ((Yyn) == YYPACT_NINF)

#define YYTABLE_NINF (-1)

#define yytable_value_is_error(Yyn) \
  0

/* YYPACT[STATE-NUM] -- Index in YYTABLE of the portion describing
   STATE-NUM.  */
static const yytype_int8 yypact[] =
{
      12,    -6,    12,     4,    18,     6,    -6,    12,    12,    12,
      12,    12,    -6,    -5,    -5,    -6,    -6,    -6
};

/* YYDEFACT[STATE-NUM] -- Default reduction number in state STATE-NUM.
   Performed when YYTABLE does not specify something else to do.  Zero
   means the default is an error.  */
static const yytype_int8 yydefact[] =
{
       0,     9,     0,     0,     2,     0,     1,     0,     0,     0,
       0,     0,     8,     3,     4,     5,     6,     7
};

/* YYPGOTO[NTERM-NUM].  */
static const yytype_int8 yypgoto[] =
{
     -6,    -6,    -2
};
```

```
/* YYDEFGOTO[NTERM-NUM].  */
static const yytype_int8 yydefgoto[] =
{
      0,     3,     4
};

/* YYTABLE[YYPACT[STATE-NUM]] -- What to do in state STATE-NUM.  If
   positive, shift that token.  If negative, reduce the rule whose
   number is the opposite.  If YYTABLE_NINF, syntax error.  */
static const yytype_int8 yytable[] =
{
      5,     9,    10,    11,     6,    13,    14,    15,    16,    17,
      7,     8,     9,    10,    11,     1,    12,     0,     0,     0,
      0,     2,     7,     8,     9,    10,    11
};

static const yytype_int8 yycheck[] =
{
      2,     6,     7,     8,     0,     7,     8,     9,    10,    11,
      4,     5,     6,     7,     8,     3,    10,    -1,    -1,    -1,
     -1,     9,     4,     5,     6,     7,     8
};

/* YYSTOS[STATE-NUM] -- The symbol kind of the accessing symbol of
   state STATE-NUM.  */
static const yytype_int8 yystos[] =
{
      0,     3,     9,    12,    13,    13,     0,     4,     5,     6,
      7,     8,    10,    13,    13,    13,    13,    13
};

/* YYR1[RULE-NUM] -- Symbol kind of the left-hand side of rule RULE-NUM.  */
static const yytype_int8 yyr1[] =
{
      0,    11,    12,    13,    13,    13,    13,    13,    13,    13
};

/* YYR2[RULE-NUM] -- Number of symbols on the right-hand side of rule RULE-NUM.  */
static const yytype_int8 yyr2[] =
{
      0,     2,     1,     3,     3,     3,     3,     3,     3,     1
};


enum { YYENOMEM = -2 };

#define yyerrok         (yyerrstatus = 0)
#define yyclearin       (yychar = YYEMPTY)

#define YYACCEPT        goto yyacceptlab
#define YYABORT         goto yyabortlab
#define YYERROR         goto yyerrorlab
```

```
#define YYNOMEM         goto yyexhaustedlab


#define YYRECOVERING()  (!!yyerrstatus)

#define YYBACKUP(Token, Value)                                  \
  do                                                            \
    if (yychar == YYEMPTY)                                      \
      {                                                         \
        yychar = (Token);                                       \
        yylval = (Value);                                       \
        YYPOPSTACK (yylen);                                     \
        yystate = *yyssp;                                       \
        goto yybackup;                                          \
      }                                                         \
    else                                                        \
      {                                                         \
        yyerror (YY_("syntax error: cannot back up")); \
        YYERROR;                                                \
      }                                                         \
  while (0)

/* Backward compatibility with an undocumented macro.
   Use YYerror or YYUNDEF. */
#define YYERRCODE YYUNDEF


/* Enable debugging if requested.  */
#if YYDEBUG

# ifndef YYFPRINTF
#  include <stdio.h> /* INFRINGES ON USER NAME SPACE */
#  define YYFPRINTF fprintf
# endif

# define YYDPRINTF(Args)                        \
do {                                            \
  if (yydebug)                                  \
    YYFPRINTF Args;                             \
} while (0)




# define YY_SYMBOL_PRINT(Title, Kind, Value, Location)                    \
do {                                                                      \
  if (yydebug)                                                            \
    {                                                                     \
      YYFPRINTF (stderr, "%s ", Title);                                   \
      yy_symbol_print (stderr,                                            \
                  Kind, Value); \
      YYFPRINTF (stderr, "\n");                                           \
```

```
    }                                           \
} while (0)


/*--------------------------------.
| Print this symbol's value on YYO.  |
`--------------------------------*/

static void
yy_symbol_value_print (FILE *yyo,
                       yysymbol_kind_t yykind, YYSTYPE const * const yyvaluep)
{
  FILE *yyoutput = yyo;
  YY_USE (yyoutput);
  if (!yyvaluep)
    return;
  YY_IGNORE_MAYBE_UNINITIALIZED_BEGIN
  YY_USE (yykind);
  YY_IGNORE_MAYBE_UNINITIALIZED_END
}


/*---------------------------.
| Print this symbol on YYO.  |
`---------------------------*/

static void
yy_symbol_print (FILE *yyo,
                 yysymbol_kind_t yykind, YYSTYPE const * const yyvaluep)
{
  YYFPRINTF (yyo, "%s %s (",
             yykind < YYNTOKENS ? "token" : "nterm", yysymbol_name (yykind));

  yy_symbol_value_print (yyo, yykind, yyvaluep);
  YYFPRINTF (yyo, ")");
}

/*------------------------------------------------------------------.
| yy_stack_print -- Print the state stack from its BOTTOM up to its |
| TOP (included).                                                   |
`------------------------------------------------------------------*/

static void
yy_stack_print (yy_state_t *yybottom, yy_state_t *yytop)
{
  YYFPRINTF (stderr, "Stack now");
  for (; yybottom <= yytop; yybottom++)
    {
      int yybot = *yybottom;
      YYFPRINTF (stderr, " %d", yybot);
    }
  YYFPRINTF (stderr, "\n");
```

```
}

# define YY_STACK_PRINT(Bottom, Top)                            \
do {                                                            \
  if (yydebug)                                                  \
    yy_stack_print ((Bottom), (Top));                           \
} while (0)


/*------------------------------------------------.
| Report that the YYRULE is going to be reduced.  |
`------------------------------------------------*/

static void
yy_reduce_print (yy_state_t *yyssp, YYSTYPE *yyvsp,
                 int yyrule)
{
  int yylno = yyrline[yyrule];
  int yynrhs = yyr2[yyrule];
  int yyi;
  YYFPRINTF (stderr, "Reducing stack by rule %d (line %d):\n",
             yyrule - 1, yylno);
  /* The symbols being reduced.  */
  for (yyi = 0; yyi < yynrhs; yyi++)
    {
      YYFPRINTF (stderr, "   $%d = ", yyi + 1);
      yy_symbol_print (stderr,
                       YY_ACCESSING_SYMBOL (+yyssp[yyi + 1 - yynrhs]),
                       &yyvsp[(yyi + 1) - (yynrhs)]);
      YYFPRINTF (stderr, "\n");
    }
}

# define YY_REDUCE_PRINT(Rule)          \
do {                                    \
  if (yydebug)                          \
    yy_reduce_print (yyssp, yyvsp, Rule); \
} while (0)

/* Nonzero means print parse trace.  It is left uninitialized so that
   multiple parsers can coexist.  */
int yydebug;
#else /* !YYDEBUG */
# define YYDPRINTF(Args) ((void) 0)
# define YY_SYMBOL_PRINT(Title, Kind, Value, Location)
# define YY_STACK_PRINT(Bottom, Top)
# define YY_REDUCE_PRINT(Rule)
#endif /* !YYDEBUG */


/* YYINITDEPTH -- initial size of the parser's stacks.  */
#ifndef YYINITDEPTH
```

```
# define YYINITDEPTH 200
#endif

/* YYMAXDEPTH -- maximum size the stacks can grow to (effective only
   if the built-in stack extension method is used).

   Do not make this value too large; the results are undefined if
   YYSTACK_ALLOC_MAXIMUM < YYSTACK_BYTES (YYMAXDEPTH)
   evaluated with infinite-precision integer arithmetic.  */

#ifndef YYMAXDEPTH
# define YYMAXDEPTH 10000
#endif




/*-----------------------------------------------.
| Release the memory associated to this symbol.  |
`-----------------------------------------------*/

static void
yydestruct (const char *yymsg,
            yysymbol_kind_t yykind, YYSTYPE *yyvaluep)
{
  YY_USE (yyvaluep);
  if (!yymsg)
    yymsg = "Deleting";
  YY_SYMBOL_PRINT (yymsg, yykind, yyvaluep, yylocationp);

  YY_IGNORE_MAYBE_UNINITIALIZED_BEGIN
  YY_USE (yykind);
  YY_IGNORE_MAYBE_UNINITIALIZED_END
}


/* Lookahead token kind.  */
int yychar;

/* The semantic value of the lookahead symbol.  */
YYSTYPE yylval;
/* Number of syntax errors so far.  */
int yynerrs;




/*----------.
| yyparse.  |
`----------*/
```

```
int
yyparse (void)
{
    yy_state_fast_t yystate = 0;
    /* Number of tokens to shift before error messages enabled.  */
    int yyerrstatus = 0;

    /* Refer to the stacks through separate pointers, to allow yyoverflow
       to reallocate them elsewhere.  */

    /* Their size.  */
    YYPTRDIFF_T yystacksize = YYINITDEPTH;

    /* The state stack: array, bottom, top.  */
    yy_state_t yyssa[YYINITDEPTH];
    yy_state_t *yyss = yyssa;
    yy_state_t *yyssp = yyss;

    /* The semantic value stack: array, bottom, top.  */
    YYSTYPE yyvsa[YYINITDEPTH];
    YYSTYPE *yyvs = yyvsa;
    YYSTYPE *yyvsp = yyvs;

  int yyn;
  /* The return value of yyparse.  */
  int yyresult;
  /* Lookahead symbol kind.  */
  yysymbol_kind_t yytoken = YYSYMBOL_YYEMPTY;
  /* The variables used to return semantic value and location from the
     action routines.  */
  YYSTYPE yyval;



#define YYPOPSTACK(N)   (yyvsp -= (N), yyssp -= (N))

  /* The number of symbols on the RHS of the reduced rule.
     Keep to zero when no symbol should be popped.  */
  int yylen = 0;

  YYDPRINTF ((stderr, "Starting parse\n"));

  yychar = YYEMPTY; /* Cause a token to be read.  */

  goto yysetstate;


/*------------------------------------------------------------.
| yynewstate -- push a new state, which is found in yystate.  |
`------------------------------------------------------------*/
yynewstate:
```

```
  /* In all cases, when you get here, the value and location stacks
     have just been pushed.  So pushing a state here evens the stacks.  */
  yyssp++;


/*--------------------------------------------------------------------.
| yysetstate -- set current state (the top of the stack) to yystate.  |
`--------------------------------------------------------------------*/
yysetstate:
  YYDPRINTF ((stderr, "Entering state %d\n", yystate));
  YY_ASSERT (0 <= yystate && yystate < YYNSTATES);
  YY_IGNORE_USELESS_CAST_BEGIN
  *yyssp = YY_CAST (yy_state_t, yystate);
  YY_IGNORE_USELESS_CAST_END
  YY_STACK_PRINT (yyss, yyssp);

  if (yyss + yystacksize - 1 <= yyssp)
#if !defined yyoverflow && !defined YYSTACK_RELOCATE
    YYNOMEM;
#else
    {
      /* Get the current used size of the three stacks, in elements.  */
      YYPTRDIFF_T yysize = yyssp - yyss + 1;

# if defined yyoverflow
      {
        /* Give user a chance to reallocate the stack.  Use copies of
           these so that the &'s don't force the real ones into
           memory.  */
        yy_state_t *yyss1 = yyss;
        YYSTYPE *yyvs1 = yyvs;

        /* Each stack pointer address is followed by the size of the
           data in use in that stack, in bytes.  This used to be a
           conditional around just the two extra args, but that might
           be undefined if yyoverflow is a macro.  */
        yyoverflow (YY_("memory exhausted"),
                    &yyss1, yysize * YYSIZEOF (*yyssp),
                    &yyvs1, yysize * YYSIZEOF (*yyvsp),
                    &yystacksize);
        yyss = yyss1;
        yyvs = yyvs1;
      }
# else /* defined YYSTACK_RELOCATE */
      /* Extend the stack our own way.  */
      if (YYMAXDEPTH <= yystacksize)
        YYNOMEM;
      yystacksize *= 2;
      if (YYMAXDEPTH < yystacksize)
        yystacksize = YYMAXDEPTH;

      {
```

```
        yy_state_t *yyss1 = yyss;
        union yyalloc *yyptr =
          YY_CAST (union yyalloc *,
                   YYSTACK_ALLOC (YY_CAST (YYSIZE_T, YYSTACK_BYTES (yystacksize))));
        if (! yyptr)
          YYNOMEM;
        YYSTACK_RELOCATE (yyss_alloc, yyss);
        YYSTACK_RELOCATE (yyvs_alloc, yyvs);
#  undef YYSTACK_RELOCATE
        if (yyss1 != yyssa)
          YYSTACK_FREE (yyss1);
      }
# endif

      yyssp = yyss + yysize - 1;
      yyvsp = yyvs + yysize - 1;

      YY_IGNORE_USELESS_CAST_BEGIN
      YYDPRINTF ((stderr, "Stack size increased to %ld\n",
                  YY_CAST (long, yystacksize)));
      YY_IGNORE_USELESS_CAST_END

      if (yyss + yystacksize - 1 <= yyssp)
        YYABORT;
    }
#endif /* !defined yyoverflow && !defined YYSTACK_RELOCATE */


  if (yystate == YYFINAL)
    YYACCEPT;

  goto yybackup;


/*-----------.
| yybackup.  |
`-----------*/
yybackup:
  /* Do appropriate processing given the current state.  Read a
     lookahead token if we need one and don't already have one.  */

  /* First try to decide what to do without reference to lookahead token.  */
  yyn = yypact[yystate];
  if (yypact_value_is_default (yyn))
    goto yydefault;

  /* Not known => get a lookahead token if don't already have one.  */

  /* YYCHAR is either empty, or end-of-input, or a valid lookahead.  */
  if (yychar == YYEMPTY)
    {
      YYDPRINTF ((stderr, "Reading a token\n"));
```

```
    yychar = yylex ();
  }

if (yychar <= YYEOF)
  {
    yychar = YYEOF;
    yytoken = YYSYMBOL_YYEOF;
    YYDPRINTF ((stderr, "Now at end of input.\n"));
  }
else if (yychar == YYerror)
  {
    /* The scanner already issued an error message, process directly
       to error recovery.  But do not keep the error token as
       lookahead, it is too special and may lead us to an endless
       loop in error recovery. */
    yychar = YYUNDEF;
    yytoken = YYSYMBOL_YYerror;
    goto yyerrlab1;
  }
else
  {
    yytoken = YYTRANSLATE (yychar);
    YY_SYMBOL_PRINT ("Next token is", yytoken, &yylval, &yylloc);
  }

/* If the proper action on seeing token YYTOKEN is to reduce or to
   detect an error, take that action.  */
yyn += yytoken;
if (yyn < 0 || YYLAST < yyn || yycheck[yyn] != yytoken)
  goto yydefault;
yyn = yytable[yyn];
if (yyn <= 0)
  {
    if (yytable_value_is_error (yyn))
      goto yyerrlab;
    yyn = -yyn;
    goto yyreduce;
  }

/* Count tokens shifted since error; after three, turn off error
   status.  */
if (yyerrstatus)
  yyerrstatus--;

/* Shift the lookahead token.  */
YY_SYMBOL_PRINT ("Shifting", yytoken, &yylval, &yylloc);
yystate = yyn;
YY_IGNORE_MAYBE_UNINITIALIZED_BEGIN
*++yyvsp = yylval;
YY_IGNORE_MAYBE_UNINITIALIZED_END

/* Discard the shifted token.  */
```

```
  yychar = YYEMPTY;
  goto yynewstate;


/*-----------------------------------------------------------.
| yydefault -- do the default action for the current state.  |
`-----------------------------------------------------------*/
yydefault:
  yyn = yydefact[yystate];
  if (yyn == 0)
    goto yyerrlab;
  goto yyreduce;


/*-----------------------------.
| yyreduce -- do a reduction.  |
`-----------------------------*/
yyreduce:
  /* yyn is the number of a rule to reduce with.  */
  yylen = yyr2[yyn];

  /* If YYLEN is nonzero, implement the default value of the action:
     '$$ = $1'.

     Otherwise, the following line sets YYVAL to garbage.
     This behavior is undocumented and Bison
     users should not rely upon it.  Assigning to YYVAL
     unconditionally makes the parser a bit smaller, and it avoids a
     GCC warning that YYVAL may be used uninitialized.  */
  yyval = yyvsp[1-yylen];


  YY_REDUCE_PRINT (yyn);
  switch (yyn)
    {
  case 2: /* ArithmeticExpression: E  */
#line 10 "pgm.y"
                    {printf("output:%d\n",yyval);return 0;}
#line 1116 "y.tab.c"
    break;

  case 3: /* E: E '+' E  */
#line 11 "pgm.y"
         {yyval=yyvsp[-2]+yyvsp[0];}
#line 1122 "y.tab.c"
    break;

  case 4: /* E: E '-' E  */
#line 12 "pgm.y"
        {yyval=yyvsp[-2]-yyvsp[0];}
#line 1128 "y.tab.c"
    break;
```

```
  case 5: /* E: E '*' E  */
#line 13 "pgm.y"
      {yyval=yyvsp[-2]*yyvsp[0];}
#line 1134 "y.tab.c"
    break;

  case 6: /* E: E '/' E  */
#line 14 "pgm.y"
      {yyval=yyvsp[-2]/yyvsp[0];}
#line 1140 "y.tab.c"
    break;

  case 7: /* E: E '%' E  */
#line 15 "pgm.y"
      {yyval=yyvsp[-2]%yyvsp[0];}
#line 1146 "y.tab.c"
    break;

  case 8: /* E: '(' E ')'  */
#line 16 "pgm.y"
        {yyval=yyvsp[-1];}
#line 1152 "y.tab.c"
    break;

  case 9: /* E: NUMBER  */
#line 17 "pgm.y"
        {yyval=yyvsp[0];}
#line 1158 "y.tab.c"
    break;


#line 1162 "y.tab.c"

      default: break;
    }
  /* User semantic actions sometimes alter yychar, and that requires
     that yytoken be updated with the new translation.  We take the
     approach of translating immediately before every use of yytoken.
     One alternative is translating here after every semantic action,
     but that translation would be missed if the semantic action invokes
     YYABORT, YYACCEPT, or YYERROR immediately after altering yychar or
     if it invokes YYBACKUP.  In the case of YYABORT or YYACCEPT, an
     incorrect destructor might then be invoked immediately.  In the
     case of YYERROR or YYBACKUP, subsequent parser actions might lead
     to an incorrect destructor call or verbose syntax error message
     before the lookahead is translated.  */
  YY_SYMBOL_PRINT ("-> $$ =", YY_CAST (yysymbol_kind_t, yyr1[yyn]), &yyval, &yyloc);

  YYPOPSTACK (yylen);
  yylen = 0;
```

```
  *++yyvsp = yyval;

  /* Now 'shift' the result of the reduction.  Determine what state
     that goes to, based on the state we popped back to and the rule
     number reduced by.  */
  {
    const int yylhs = yyr1[yyn] - YYNTOKENS;
    const int yyi = yypgoto[yylhs] + *yyssp;
    yystate = (0 <= yyi && yyi <= YYLAST && yycheck[yyi] == *yyssp
               ? yytable[yyi]
               : yydefgoto[yylhs]);
  }

  goto yynewstate;


/*--------------------------------------.
| yyerrlab -- here on detecting error.  |
`--------------------------------------*/
yyerrlab:
  /* Make sure we have latest lookahead translation.  See comments at
     user semantic actions for why this is necessary.  */
  yytoken = yychar == YYEMPTY ? YYSYMBOL_YYEMPTY : YYTRANSLATE (yychar);
  /* If not already recovering from an error, report this error.  */
  if (!yyerrstatus)
    {
      ++yynerrs;
      yyerror (YY_("syntax error"));
    }

  if (yyerrstatus == 3)
    {
      /* If just tried and failed to reuse lookahead token after an
         error, discard it.  */

      if (yychar <= YYEOF)
        {
          /* Return failure if at end of input.  */
          if (yychar == YYEOF)
            YYABORT;
        }
      else
        {
          yydestruct ("Error: discarding",
                      yytoken, &yylval);
          yychar = YYEMPTY;
        }
    }

  /* Else will try to reuse lookahead token after shifting the error
     token.  */
  goto yyerrlab1;
```

```
/*---------------------------------------------------.
| yyerrorlab -- error raised explicitly by YYERROR.  |
`---------------------------------------------------*/
yyerrorlab:
  /* Pacify compilers when the user code never invokes YYERROR and the
     label yyerrorlab therefore never appears in user code.  */
  if (0)
    YYERROR;
  ++yynerrs;

  /* Do not reclaim the symbols of the rule whose action triggered
     this YYERROR.  */
  YYPOPSTACK (yylen);
  yylen = 0;
  YY_STACK_PRINT (yyss, yyssp);
  yystate = *yyssp;
  goto yyerrlab1;


/*-------------------------------------------------------------.
| yyerrlab1 -- common code for both syntax error and YYERROR.  |
`-------------------------------------------------------------*/
yyerrlab1:
  yyerrstatus = 3;      /* Each real token shifted decrements this.  */

  /* Pop stack until we find a state that shifts the error token.  */
  for (;;)
    {
      yyn = yypact[yystate];
      if (!yypact_value_is_default (yyn))
        {
          yyn += YYSYMBOL_YYerror;
          if (0 <= yyn && yyn <= YYLAST && yycheck[yyn] == YYSYMBOL_YYerror)
            {
              yyn = yytable[yyn];
              if (0 < yyn)
                break;
            }
        }

      /* Pop the current state because it cannot handle the error token.  */
      if (yyssp == yyss)
        YYABORT;


      yydestruct ("Error: popping",
                  YY_ACCESSING_SYMBOL (yystate), yyvsp);
      YYPOPSTACK (1);
      yystate = *yyssp;
      YY_STACK_PRINT (yyss, yyssp);
```

```
    }

  YY_IGNORE_MAYBE_UNINITIALIZED_BEGIN
  *++yyvsp = yylval;
  YY_IGNORE_MAYBE_UNINITIALIZED_END


  /* Shift the error token.  */
  YY_SYMBOL_PRINT ("Shifting", YY_ACCESSING_SYMBOL (yyn), yyvsp, yylsp);

  yystate = yyn;
  goto yynewstate;


/*-------------------------------------.
| yyacceptlab -- YYACCEPT comes here.  |
`-------------------------------------*/
yyacceptlab:
  yyresult = 0;
  goto yyreturnlab;


/*-----------------------------------.
| yyabortlab -- YYABORT comes here.  |
`-----------------------------------*/
yyabortlab:
  yyresult = 1;
  goto yyreturnlab;


/*-----------------------------------------------------------.
| yyexhaustedlab -- YYNOMEM (memory exhaustion) comes here.  |
`-----------------------------------------------------------*/
yyexhaustedlab:
  yyerror (YY_("memory exhausted"));
  yyresult = 2;
  goto yyreturnlab;


/*----------------------------------------------------------.
| yyreturnlab -- parsing is finished, clean up and return.  |
`----------------------------------------------------------*/
yyreturnlab:
  if (yychar != YYEMPTY)
    {
      /* Make sure we have latest lookahead translation.  See comments at
         user semantic actions for why this is necessary.  */
      yytoken = YYTRANSLATE (yychar);
      yydestruct ("Cleanup: discarding lookahead",
                  yytoken, &yylval);
    }
  /* Do not reclaim the symbols of the rule whose action triggered
```

```
    this YYABORT or YYACCEPT.  */
  YYPOPSTACK (yylen);
  YY_STACK_PRINT (yyss, yyssp);
  while (yyssp != yyss)
    {
      yydestruct ("Cleanup: popping",
                  YY_ACCESSING_SYMBOL (+*yyssp), yyvsp);
      YYPOPSTACK (1);
    }
#ifndef yyoverflow
  if (yyss != yyssa)
    YYSTACK_FREE (yyss);
#endif

  return yyresult;
}

#line 19 "pgm.y"

void main()
{
printf("\nEnter the input:\n");
yyparse();
if(flag==0)
printf("Expression is Valid\n");
}
void yyerror()
{
printf("Expression is invalid\n");
flag=1;
}
```

```
//yacc -d Pg13.y
//lex Pg13.l
//gcc lex.yy.c y.tab.c -w
//./a.out
```

y.tab.h

```
/* A Bison parser, made by GNU Bison 3.8.2.  */

/* Bison interface for Yacc-like parsers in C
```

/* DO NOT RELY ON FEATURES THAT ARE NOT DOCUMENTED in the manual,
especially those whose name start with YY_ or yy_. They are
private implementation details that can be changed or removed. */

```
#ifndef YY_YY_Y_TAB_H_INCLUDED
# define YY_YY_Y_TAB_H_INCLUDED
/* Debug traces.  */
#ifndef YYDEBUG
# define YYDEBUG 0
#endif
#if YYDEBUG
extern int yydebug;
#endif

/* Token kinds.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
  enum yytokentype
  {
    YYEMPTY = -2,
    YYEOF = 0,                     /* "end of file"  */
    YYerror = 256,                 /* error  */
    YYUNDEF = 257,                 /* "invalid token"  */
```

```
    NUMBER = 258                   /* NUMBER  */
  };
  typedef enum yytokentype yytoken_kind_t;
#endif
/* Token kinds.  */
#define YYEMPTY -2
#define YYEOF 0
#define YYerror 256
#define YYUNDEF 257
#define NUMBER 258

/* Value type.  */
#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
typedef int YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define YYSTYPE_IS_DECLARED 1
#endif


extern YYSTYPE yylval;


int yyparse (void);


#endif /* !YY_YY_Y_TAB_H_INCLUDED  */
```

OUTPUT

```
ubuntu@ubuntu:~/Downloads$ yacc -d pgm.y
ubuntu@ubuntu:~/Downloads$ lex pgm.l
ubuntu@ubuntu:~/Downloads$ gcc lex.yy.c y.tab.c -w
ubuntu@ubuntu:~/Downloads$ ./a.out

Enter the input:
5*2
output:10
Expression is Valid
```