```
%{

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


typedef struct
{
 char q[41624];
 char thestr[25];
 int ival;
 int ttype;
}tstruct ;

#define YYSTYPE  tstruct



int yylex();
void yyerror( char *s );
int erc = 0;

#include "symtab.c"



%}

%token tstart
%token tfinish
%token tmaxx
%token tmaxy
%token tminx
%token tminy
%token tint
%token tfloat
%token tput
%token tget
%token tgraph
%token tsollin
%token tsolquad
%token tsolcube
%token tnum
%token tfnum
%token tid
%token tstrlit
%token tfor
%token twhile
%token tbool
```

```
%token tsemi
%token telse
%token tif
%token tob
%token tcb


%define locations

%%

p : prog {
        FILE *fp;
        fp = fopen(".tom.c", "w+");
        fprintf(fp, "%s", $1.q);
        fclose(fp);

        printf("There are %d errors\n", erc);
        };

prog : tstart tsemi  tfinish tsemi {
                sprintf($$.q, "int main() {\nreturn 0;\n}\n");
        }
 |  tstart tsemi AL tfinish tsemi  {
                sprintf($$.q, "#include <stdio.h>\n"
                        "#include <math.h>\n"
                        "#include <stdlib.h>\n\n"
                        "int main() {\n"
                        "int minx = 0;\n"
                        "int maxx = 10;\n"
                        "int miny = 0;\n"
                        "int maxy = 10;\n"
                        "float scaley;\n"
                        "float scalex;\n"
                        "float x1, x2, x3, x4, x0, root1, root2, mid,
i1, root3, i2, i3, f1, g1, h1, R1, S1, T1, U1, j1, k1, L1, M1, N1,
P1;;\n"
                        "int least;\n"
                        "float x;\n"
                        "\n%s\n%s\n"
                        "return 0;\n}\n", $3.q, $4.q);
        }
 ;

AL : SL AL      {sprintf($$.q, "%s%s", $1.q, $2.q);}
   | DL AL      {sprintf($$.q, "%s%s", $1.q, $2.q);}
   | SL         {sprintf($$.q, "%s", $1.q);}
   | DL         {sprintf($$.q, "%s", $1.q);}
   ;
```

```
DL :  DL D   {sprintf($$.q, "%s%s", $1.q, $2.q);}
  | D        {sprintf($$.q, "%s", $1.q);}
  ;

D : Dtail tsemi           { $$.ttype = $1.ttype;
                          sprintf($$.q, "%s;\n", $1.q);
                  }

Dtail : Dtail ',' tid     { addtab($3.thestr);
                           addtype($3.thestr, $1.ttype);
                           $$.ttype = $1.ttype;
                            sprintf($$.q, "%s, %s", $1.q, $3.thestr);
                          }
      | type tid    {     addtab($2.thestr);
                          addtype($2.thestr, $1.ttype);
                          $$.ttype = $1.ttype;
                          sprintf($$.q, "%s %s", $1.q, $2.thestr);
                          }
      ;

type: tint       {$$.ttype = 10;
                  sprintf($$.q, "int");
                 }
    | tfloat     {$$.ttype = 20;
                  sprintf($$.q, "float");
                 }
    ;

SL : SL S   {sprintf($$.q, "%s%s", $1.q, $2.q);}
   | S        {sprintf($$.q, "%s", $1.q);}
   ;

S : tput tstrlit tsemi   {sprintf($$.q, "printf(%s);\n", $2.thestr);}
  | tput tid tsemi       {if($2.ttype == 10)
                                sprintf($$.q, "printf(\"%%d\", %s);
\n", $2.thestr);
                          else
                                sprintf($$.q, "printf(\"%%f\", %s);
\n", $2.thestr);}
  | tget tid tsemi       {if($2.ttype == 10)
                                sprintf($$.q, "scanf(\"%%d\", &%s);
\n", $2.thestr);
                          else
                                sprintf($$.q, "scanf(\"%%f\", &%s);
\n", $2.thestr);}
  | tid '=' expr tsemi
              {
                $1.ttype = gettype($1.thestr);
                if ($1.ttype > 0 )
                {
```

```
                  if ($1.ttype == $3.ttype || ($1.ttype == 20 &&
$3.ttype == 10) )
                        sprintf($$.q, "%s = %s;\n", $1.thestr, $3.q);
                  else
                     {
                    yyerror("Type Mismatch Error :::");
                         erc++;//error message
                     }
                }
                else {
                    yyerror("Type Error :::");
                     erc++;
                 }

              }
  | tmaxx expr tsemi  { $2.ttype == 10;
                        sprintf($$.q, "maxx = %s;\n", $2.q);}
  | tminx expr tsemi  { $2.ttype = 10;
                        sprintf($$.q, "minx = %s;\n", $2.q);}
  | tmaxy expr tsemi  { $2.ttype = 10;
                        sprintf($$.q, "maxy = %s;\n", $2.q);}
  | tminy expr tsemi  { $2.ttype = 10;
                        sprintf($$.q, "miny = %s;\n", $2.q);}
  | tgraph expr tsemi { sprintf($$.q, "scaley = (maxy - miny * 1.0) /
20;\n"
        "scalex = (maxx - minx * 1.0) / 40;\n"
        "for(float i = maxy; i >= miny; i -= scaley) {\n"
                "printf(\"%%10.1f|\", i);\n"
                "for(x = minx; x <= maxx; x += scalex) {\n"
                        "float yValue;\n"
                        "yValue = %s;\n"
                        "if(yValue >= i && yValue < (i + scaley) )\n"
                                "printf(\"#\");\n"
                        "else\n"
                                "printf(\" \");\n"
                        "}\n"
                "printf(\"\\n\");\n"
                "}\n", $2.q);


                sprintf($$.q, "%s\nwhile(maxx%%5 != 0)"
                        "maxx++;\n"
                "while(minx%%5 != 0)\n"
                        "minx--;\n"
                "scalex = (maxx-minx*1.0)/20;\n"
                "least = ( (int)(log10(abs(minx))+1) > (int)
(log10(abs(maxx))+1) ) ? (int)(log10(abs(minx))+1) : (int)
(log10(abs(maxx))+1);\n"
                "for(int i = least; i > 0; i--) {\n"
                       "printf(\"          \");\n"
```

```
                        "for(float j = minx; j <= maxx; j += scalex)
{\n"
                       "char toprint;\n"
                        "if(j == (int)j)\n"
                                "toprint = \'0\' + ((int)(abs(j) /
pow(10, i-1))) %% 10;\n"
                                 "else\n"
                                "toprint = \' \';\n"
                        "printf(\" %%c\", toprint);\n}\n"
        "printf(\"\\n\");\n}\n", $$.q);
        }
  | tfor tid '=' expr ',' expr tbool expr ',' tid '=' expr tob tsemi
SL tcb tsemi {
        sprintf($$.q, "for( %s = %s; %s %s %s; %s = %s) {\n%s}\n",
$2.thestr, $4.q, $6.q, $7.thestr, $8.q, $10.thestr, $12.q, $15.q);
  }
  | twhile expr tbool expr tob tsemi SL tcb tsemi {
       sprintf($$.q, "while( %s %s %s) {\n%s}\n", $2.q, $3.thestr,
$4.q, $7.q);
  }

  | tsollin expr tid '+' expr '=' expr tid '+' expr tsemi
        {
                sprintf($$.q, ""
                "printf(\"[%sx+%s=%sx+%s]\\n\");"
                ,$2.q, $5.q, $7.q, $10.q);
                sprintf($$.q,"%s"
                "x1=%s - %s;\n"
                "x0=%s - %s;\n"
                "root1 = -x0 / x1;\n"
                "printf(\"Linear Solve = :%%f\\n\", root1);\n",$$.q,
$2.q, $7.q, $5.q, $10.q);

        }
  | tsollin expr tid '+' expr '=' expr tid tsemi
        {
                sprintf($$.q, ""
                "printf(\"[%sx+%s=%sx]\\n\");"
                ,$2.q, $5.q, $7.q);
                sprintf($$.q,"%s"
                "x1=%s - %s;\n"
                "x0=%s ;\n"
                "root1 = -x0 / x1;\n"
                "printf(\"Linear Solve = :%%f\\n\", root1);\n",$$.q,
$2.q, $7.q, $5.q);
        }
  | tsollin expr tid '+' expr '=' expr tsemi
        {
                sprintf($$.q, ""
                "printf(\"[%sx+%s=%s]\\n\");"
```

```
                ,$2.q, $5.q, $7.q);

                sprintf($$.q,"%s"
                "x1=%s ;\n"
                "x0=%s - %s ;\n"
                "root1 = -x0 / x1;\n"
                "printf(\"Linear Solve = :%%f\\n\", root1);\n",$$.q,
$2.q, $5.q, $7.q);
        }
  | tsolquad expr tid '^' tnum '+' expr tid '+' expr '=' expr tsemi
        {
                sprintf($$.q, ""
                "printf(\"[%sx^2+%sx+%s=%s]\\n\");"
                ,$2.q, $7.q, $10.q, $12.q);

                sprintf($$.q, "%s"
                "x2 = %s ;\n"
                "x1 = %s ;\n"
                "x0 = %s - %s ;\n"
                " mid = pow(x1, 2) - 4 * x2 * x0;\n"
                "if (mid < 0) {\n"
                "root1 = -x1 / 2 / x2;\n"
                "i1 = sqrt(-mid) / 2 / x2;\n"
                "printf(\"Quadratic Solve: %%f+%%fi, %%f-%%fi\\n\",
root1, i1, root1, i1);\n"
                "}\n"
                "else {\n"
                "root1 = (-x1 + sqrt(mid)) / 2 / x2;\n"
                "root2 = (-x1 - sqrt(mid)) / 2 / x2;\n"
                "printf(\"Quadratic Solve: %%f, %%f\\n\", root1,
root2);\n"
                "}\n", $$.q, $2.q, $7.q, $10.q, $12.q);
        }
  | tsolcube expr tid '^' tnum '+' expr tid '^' tnum '+' expr tid '+'
expr '=' expr tsemi
    {
        sprintf($$.q, ""
        "x3 = %s;\n"
        "x2 = %s;\n"
        "x1 = %s;\n"
        "x0 = %s - %s ;\n"
        "f1 = ((3 * x1 / x3) - (pow(x2, 2) / pow(x3, 2))) / 3;\n"
        "g1 =((2*pow(x2,3)/pow(x3,3))-(9*x2*x1/pow(x3,2))+(27*x0/x3))/
27;\n"
        "h1 = (pow(g1,2) / 4) + (pow(f1,3)/27);\n"
        "if (f1 == 0 && g1 == 0 && h1 == 0) {\n"
        "root1 = -1 * cbrt(x0/x3);\n"
        "root2 = root1;\n"
        "root3 = root1;\n"
        "printf(\"Cubic Solve: %%f, %%f, %%f \\n\", root1, root2,
```

```
root3);\n"
        "}\n"
        "else if (h1 > 0) {\n"
        "R1 = -(g1 / 2)+sqrt(h1);\n"
        "S1 = cbrt(R1);\n"
        "T1 = -(g1 / 2)-sqrt(h1);\n"
        "U1 = cbrt(T1);\n"
        "root1 = (S1 + U1) - (x2 / 3 / x3);\n"
        "root2 = -(S1 + U1) / 2 - (x2 / 3 / x3);\n"
        "root3 = root2;\n"
        "i2 = (S1 - U1) * sqrt(3) / 2;\n"
        "i3 = i2;\n"
        "printf(\"Cubic Solve: %%f, %%f+%%fi, %%f-%%fi \\n\", root1,
root2, i2, root3, i3);\n"
        "}\n"
        "else {\n"
        "i1 = sqrt((pow(g1, 2) / 4) - h1);\n"
        "j1 = cbrt(i1);\n"
        "k1 = acos(-(g1 / 2 / i1));\n"
        "L1 = j1 * -1;\n"
        "M1 = cos(k1 / 3);\n"
        "N1 = sqrt(3) * sin(k1 / 3);\n"
        "P1 = -1 * (x2 / 3 / x3);\n"
        "root1 = 2 * j1 * cos(k1 / 3) - (x2 / 3 / x3);\n"
        "root2 = L1 * (M1 + N1) + P1;\n"
        "root3 = L1 * (M1 - N1) + P1;\n"
        "printf(\"Cubic Solve: %%f, %%f, %%f\\n\", root1, root2,
root3);\n"
        "}\n", $2.q, $7.q, $12.q, $15.q, $17.q);
  }

        | ie { sprintf($$.q, "%s", $1.q);  }
  ;

ie : ie elselist { sprintf($$.q, "%s%s", $1.q, $2.q);  }
   | iflist { sprintf($$.q, "%s", $1.q);  }
;
elselist : telse tif expr tbool expr tob tsemi SL tcb tsemi {
        sprintf($$.q, "else if( %s %s %s) {\n%s}\n", $3.q, $4.thestr,
$5.q, $8.q);
        }
        | telse tob tsemi SL tcb tsemi {
        sprintf($$.q, "else {\n%s}\n", $4.q);
        }


;
iflist :tif expr tbool expr tob tsemi SL tcb tsemi{
        sprintf($$.q, "if( %s %s %s) {\n%s}\n", $2.q, $3.thestr, $4.q,
$7.q);
```

```
        }
 ;


expr : expr '+' term
             {
               if ($1.ttype == 10 && $3.ttype == 10)  $$.ttype = 10;
               else if ($1.ttype == 20 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 10 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 20 && $3.ttype == 10)  $$.ttype =
20;
               else $$.ttype = -1;
                sprintf($$.q, "%s + %s", $1.q, $3.q);
             }
  |  term      { $$.ttype = $1.ttype;
                 sprintf($$.q, "%s", $1.q);}
  ;

expr : expr '-' term
             {
               if ($1.ttype == 10 && $3.ttype == 10)  $$.ttype = 10;
               else if ($1.ttype == 20 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 10 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 20 && $3.ttype == 10)  $$.ttype =
20;
               else $$.ttype = -1;
                sprintf($$.q, "%s + %s", $1.q, $3.q);
             }
  ;

term : term '*' carot
             {
               if ($1.ttype == 10 && $3.ttype == 10)  $$.ttype = 10;
               else if ($1.ttype == 20 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 10 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 20 && $3.ttype == 10)  $$.ttype =
20;
               else $$.ttype = -1;
                sprintf($$.q, "%s * %s", $1.q, $3.q);
             }
  | carot   { $$.ttype = $1.ttype;
              sprintf($$.q, "%s", $1.q);}
  ;
```

```
term : term '/' carot
             {
               if ($1.ttype == 10 && $3.ttype == 10)  $$.ttype = 20;
               else if ($1.ttype == 20 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 10 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 20 && $3.ttype == 10)  $$.ttype =
20;
               else $$.ttype = -1;
                sprintf($$.q, "%s / %s", $1.q, $3.q);
             }
  ;

term : term '%' carot
             {
               if ($1.ttype == 10 && $3.ttype == 10)  $$.ttype = 10;
               else $$.ttype = -1;
                sprintf($$.q, "%s % %s", $1.q, $3.q);
             }
  ;

carot : factor '^' carot {
               if ($1.ttype == 10 && $3.ttype == 10)  $$.ttype = 10;
               else if ($1.ttype == 20 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 10 && $3.ttype == 20)  $$.ttype =
20;
               else if ($1.ttype == 20 && $3.ttype == 10)  $$.ttype =
20;
               else $$.ttype = -1;
               sprintf($$.q, "pow(%s, %s)", $1.q, $3.q);
        }
  | factor {$$.ttype = $1.ttype;
                sprintf($$.q, "%s", $1.q);}

factor : '(' expr ')'
        {
                $$.ttype = $2.ttype;
                sprintf($$.q, "( %s )", $2.q);
        }
 | tid
              {
                if(strcmp($1.thestr, "x") ) {
                $$.ttype = gettype($1.thestr);
                if ($$.ttype > 0 )
                      sprintf($$.q, "%s", $1.thestr);
                else
                    erc++;//yyerror("Type Error :::");
                } else {
```

```
                        sprintf($$.q, "x");
                        $$.ttype = 10;
                }
                }

 | tnum   {$$.ttype = 10;
            sprintf($$.q, "%s", $1.thestr);}
 | tfnum  {$$.ttype = 20;
            sprintf($$.q, "%s", $1.thestr);}
 ;

%%


int main()
{

//  printf("int main() {\n");
//  printf("int maxx = 10;\nint minx = 0;\nint maxy = 10;\nint miny =
0;\nfloat x;");

  yyparse ();

//  printf("return 0;\n");
  return 0;
}


//void yyerror(char *s)  /* Called by yyparse on error */
//{
// printf ("\terror: %s\n", s);
// printf ("ERROR: %s at line %d\n", s, yylineno);
//}
```