#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "type.h"

/\*

typedef enum op {OP\_NULL, LOD,LDX,LDXB,LDA,LITI, STO,STOB,STX,STXB,SUBI,SUBF,DIVI,DIVF,ADDI,ADDF,OFFSET,MULI,MULF,MOD,LSSI,LSSF,GTRI,GTRF,LEQI,LEQF,GEQI,GEQF,NEQI,NEQF,EQLI,EQLF,NOT,OR,AND,CVTI,CVTF,JPC,JPCR,JMP,JPT,JPTR,INT,INCI,INCF,DECI,DECF,SUP,CAL,ADDR,RET,MINUSI,MINUSF,CHK,LDI,LDIB,POP,POPB} OPCODE;

\*/

char \*opcode\_name[]={ "OP\_NULL", "LOD","LDX","LDXB", "LDA", "LITI","STO","STOB","STX","STXB","SUBI","SUBF","DIVI","DIVF","ADDI","ADDF","OFFSET","MULI","MULF", "MOD", "LSSI","LSSF","GTRI","GTRF", "LEQI","LEQF","GEQI", "GEQF","NEQI","NEQF","EQLI","EQLF", "NOT", "OR", "AND","CVTI","CVTF", "JPC","JPCR","JMP","JPT","JPTR","INT","INCI","INCF","DECI","DECF","SUP","CAL","ADDR","RET","MINUSI","MINUSF","CHK","LDI","LDIB","SWITCH","SWVALUE","SWDEFAULT", "SWLABEL", "SWEND","POP","POPB","CHK"} ;

typedef enum {SW\_VALUE,SW\_DEFAULT} SW\_KIND;

typedef struct sw {SW\_KIND kind; int val; int label;} A\_SWITCH;

void code\_generation(A\_NODE \*);

void gen\_literal\_table();

void gen\_program(A\_NODE \*);

void gen\_expression(A\_NODE \*);

void gen\_expression\_left(A\_NODE \*);

void gen\_arg\_expression(A\_NODE \*);

void gen\_statement(A\_NODE \*,int, int, A\_SWITCH [], int \*);

void gen\_statement\_list(A\_NODE \*,int, int, A\_SWITCH [], int \*);

void gen\_initializer\_global(A\_NODE \*, A\_TYPE \*, int);

void gen\_initializer\_local(A\_NODE \*, A\_TYPE \*, int);

void gen\_declaration\_list(A\_ID \*); void gen\_declaration(A\_ID \*);

void gen\_code\_i(OPCODE,int,int); void gen\_code\_f(OPCODE,int,float);

void gen\_code\_s(OPCODE,int,char \*); void gen\_code\_l(OPCODE,int,int);

void gen\_label\_number(int);

void gen\_label\_name(char \*);

void gen\_error();

int get\_label();

int label\_no=0;

int gen\_err=0;

extern FILE \*fout;

extern A\_TYPE \*int\_type, \*float\_type, \*char\_type, \*void\_type, \*string\_type;

extern A\_LITERAL literal\_table[];

extern int literal\_no;

void code\_generation(A\_NODE \*node)

{

gen\_program(node);

gen\_literal\_table();

}

void gen\_literal\_table() {

int i;

for (i=1;i<=literal\_no; i++) {

fprintf(fout,".literal %5d ",literal\_table[i].addr);

if (literal\_table[i].type==int\_type)

fprintf(fout,"%d\n",literal\_table[i].value.i);

else if (literal\_table[i].type==float\_type)

fprintf(fout,"%f\n",literal\_table[i].value.f);

else if (literal\_table[i].type==char\_type)

fprintf(fout,"%d\n",literal\_table[i].value.c);

else if (literal\_table[i].type==string\_type)

fprintf(fout,"%s\n",literal\_table[i].value.s);

}

}

void gen\_program(A\_NODE \*node) {

switch(node->name) {

case N\_PROGRAM :

gen\_code\_i(INT,0,node->value);

gen\_code\_s(SUP,0,"main");

gen\_code\_i(RET,0,0);

gen\_declaration\_list(node->clink);

break;

default :

gen\_error(100,node->line);

break;

}

}

void gen\_expression(A\_NODE \*node)

{

A\_ID \*id;

A\_TYPE \*t;

int i,ll;

switch(node->name) {

case N\_EXP\_IDENT :

id=node->clink;

t=id->type;

switch (id->kind) {

case ID\_VAR:

case ID\_PARM:

switch (t->kind) {

case T\_ENUM:

case T\_POINTER:

gen\_code\_i(LOD, id->level, id->address);

break;

case T\_ARRAY:

if (id->kind==ID\_VAR)

gen\_code\_i(LDA,id->level, id->address);

else

gen\_code\_i(LOD, id->level, id->address);

break;

case T\_STRUCT:

case T\_UNION:

gen\_code\_i(LDA, id->level, id->address);

i=id->type->size;

gen\_code\_i(LDI,0,i%4?i/4+1:i/4);

break;

default: gen\_error(11,id->line);

break;

}

break;

case ID\_ENUM\_LITERAL:

gen\_code\_i(LITI,0,id->init);

break;

default: gen\_error(11,node->line);

break;

}

break;

case N\_EXP\_INT\_CONST :

gen\_code\_i(LITI, 0, node->clink);

break;

case N\_EXP\_FLOAT\_CONST :

i=node->clink;

gen\_code\_i(LOD, 0, literal\_table[i].addr);

break;

case N\_EXP\_CHAR\_CONST :

gen\_code\_i(LITI, 0, node->clink);

break;

case N\_EXP\_STRING\_LITERAL :

i=node->clink;

gen\_code\_i(LDA,0, literal\_table[i].addr);

break;

case N\_EXP\_ARRAY :

gen\_expression(node->llink);

gen\_expression(node->rlink);

// gen\_code\_i(CHK,0,node->llink->type->expr);

if (node->type->size>1) {

gen\_code\_i(LITI,0,node->type->size);

gen\_code\_i(MULI,0,0);

}

gen\_code\_i(OFFSET,0,0);

if (!isArrayType(node->type)) {

i=node->type->size;

if (i==1)

gen\_code\_i(LDIB,0,0);

else

gen\_code\_i(LDI,0, i%4?i/4+1:i/4);

}

break;

case N\_EXP\_FUNCTION\_CALL :

t=node->llink->type;

i=t->element\_type->element\_type->size;

if (i%4) i=i/4\*4+4;

if(node->rlink){

gen\_code\_i(INT,0,12+i);

gen\_arg\_expression(node->rlink);

gen\_code\_i(POP,0,node->rlink->value/4+3);

}

else

gen\_code\_i(INT,0,i);

gen\_expression(node->llink);

gen\_code\_i(CAL,0,0);

break;

case N\_EXP\_STRUCT :

gen\_expression\_left(node->llink);

id=node->rlink;

if(id->address>0) {

gen\_code\_i(LITI,0,id->address);

gen\_code\_i(OFFSET,0,0);

}

if (!isArrayType(node->type)) {

i=node->type->size;

if (i==1)

gen\_code\_i(LDIB,0,0);

else

gen\_code\_i(LDI,0, i%4?i/4+1:i/4);

}

break;

case N\_EXP\_ARROW:

gen\_expression(node->llink);

id=node->rlink;

if(id->address>0) {

gen\_code\_i(LITI,0,id->address);

gen\_code\_i(OFFSET,0,0);

}

if (!isArrayType(node->type)) {

i=node->type->size;

if (i==1)

gen\_code\_i(LDIB,0,0);

else

gen\_code\_i(LDI,0, i%4?i/4+1:i/4);

}

break;

case N\_EXP\_POST\_INC :

gen\_expression(node->clink);

gen\_expression\_left(node->clink);

t=node->type;

if (node->type->size==1)

gen\_code\_i(LDXB,0,0);

else

gen\_code\_i(LDX,0,1);

if (isPointerOrArrayType(node->type)) {

gen\_code\_i(LITI,0,node->type->element\_type->size);

gen\_code\_i(ADDI,0,0);

}

else if (isFloatType(node->type))

gen\_code\_i(INCF,0,0);

else

gen\_code\_i(INCI,0,0);

if (node->type->size==1)

gen\_code\_i(STOB,0,0);

else

gen\_code\_i(STO,0,1);

break;

case N\_EXP\_POST\_DEC :

gen\_expression(node->clink);

gen\_expression\_left(node->clink);

t=node->type;

if (node->type->size==1)

gen\_code\_i(LDXB,0,0);

else

gen\_code\_i(LDX,0,1);

if (isPointerOrArrayType(node->type)) {

gen\_code\_i(LITI,0,node->type->element\_type->size);

gen\_code\_i(SUBI,0,0);}

else if (isFloatType(node->type))

gen\_code\_i(DECF,0,0);

else

gen\_code\_i(DECI,0,0);

if (node->type->size==1)

gen\_code\_i(STOB,0,0);

else

gen\_code\_i(STO,0,1);

break;

case N\_EXP\_PRE\_INC :

gen\_expression\_left(node->clink);

t=node->type;

if (node->type->size==1)

gen\_code\_i(LDXB,0,0);

else

gen\_code\_i(LDX,0,1);

if (isPointerOrArrayType(node->type)) {

gen\_code\_i(LITI,0,node->type->element\_type->size);

gen\_code\_i(ADDI,0,0);

}

else if (isFloatType(node->type))

gen\_code\_i(INCF,0,0);

else

gen\_code\_i(INCI,0,0);

if (node->type->size==1)

gen\_code\_i(STXB,0,0);

else

gen\_code\_i(STX,0,1);

break;

case N\_EXP\_PRE\_DEC :

gen\_expression\_left(node->clink);

t=node->type;

if (node->type->size==1)

gen\_code\_i(LDXB,0,0);

else

gen\_code\_i(LDX,0,1);

if (isPointerOrArrayType(node->type)) {

gen\_code\_i(LITI,0,node->type->element\_type->size);

gen\_code\_i(SUBI,0,0);

}

else if (isFloatType(node->type))

gen\_code\_i(DECF,0,0);

else

gen\_code\_i(DECI,0,0);

if (node->type->size==1)

gen\_code\_i(STXB,0,0);

else

gen\_code\_i(STX,0,1);

break;

case N\_EXP\_NOT :

gen\_expression(node->clink);

gen\_code\_i(NOT,0,0);

break;

case N\_EXP\_PLUS :

gen\_expression(node->clink);

break;

case N\_EXP\_MINUS :

gen\_expression(node->clink);

if (isFloatType(node->type))

gen\_code\_i(MINUSF,0,0);

else

gen\_code\_i(MINUSI,0,0);

break;

case N\_EXP\_AMP :

gen\_expression\_left(node->clink);

break;

case N\_EXP\_STAR :

gen\_expression(node->clink);

i=node->type->size;

if (i==1)

gen\_code\_i(LDIB,0,0);

else

gen\_code\_i(LDI,0, i%4?i/4+1:i/4);

break;

case N\_EXP\_SIZE\_EXP :

gen\_code\_i(LITI, 0,node->clink);

break;

case N\_EXP\_SIZE\_TYPE :

gen\_code\_i(LITI, 0,node->clink);

break;

case N\_EXP\_CAST :

gen\_expression(node->rlink);

if (node->type!=node->rlink->type)

if (isFloatType(node->type))

gen\_code\_i(CVTF,0,0);

else if (isFloatType(node->rlink->type))

gen\_code\_i(CVTI,0,0);

break;

case N\_EXP\_MUL :

gen\_expression(node->llink);

gen\_expression(node->rlink);

if (isFloatType(node->type))

gen\_code\_i(MULF,0,0);

else

gen\_code\_i(MULI,0,0);

break;

case N\_EXP\_DIV :

gen\_expression(node->llink);

gen\_expression(node->rlink);

if (isFloatType(node->type))

gen\_code\_i(DIVF,0,0);

else

gen\_code\_i(DIVI,0,0);

break;

case N\_EXP\_MOD :

gen\_expression(node->llink);

gen\_expression(node->rlink);

gen\_code\_i(MOD, 0,0);

break;

case N\_EXP\_ADD :

gen\_expression(node->llink);

if (isPointerOrArrayType(node->rlink->type)){

gen\_code\_i(LITI,0,node->rlink->type->element\_type->size);

gen\_code\_i(MULI,0,0);

}

gen\_expression(node->rlink);

if (isPointerOrArrayType(node->llink->type)){

gen\_code\_i(LITI,0,node->llink->type->element\_type->size);

gen\_code\_i(MULI,0,0);

}

if (isFloatType(node->type))

gen\_code\_i(ADDF,0,0);

else

gen\_code\_i(ADDI,0,0);

break;

case N\_EXP\_SUB :

gen\_expression(node->llink);

gen\_expression(node->rlink);

if (isPointerOrArrayType(node->llink->type) && !isPointerOrArrayType(node->rlink->type)) {

gen\_code\_i(LITI,0,node->llink->type->element\_type->size);

gen\_code\_i(MULI,0,0);

}

if (isFloatType(node->type))

gen\_code\_i(SUBF,0,0);

else

gen\_code\_i(SUBI,0,0);

break;

case N\_EXP\_LSS :

gen\_expression(node->llink);

gen\_expression(node->rlink);

if (isFloatType(node->llink->type))

gen\_code\_i(LSSF,0,0);

else

gen\_code\_i(LSSI,0,0);

break;

case N\_EXP\_GTR :

gen\_expression(node->llink);

gen\_expression(node->rlink);

if (isFloatType(node->llink->type))

gen\_code\_i(GTRF,0,0);

else

gen\_code\_i(GTRI,0,0);

break;

case N\_EXP\_LEQ :

gen\_expression(node->llink);

gen\_expression(node->rlink);

if (isFloatType(node->llink->type))

gen\_code\_i(LEQF,0,0);

else

gen\_code\_i(LEQI,0,0);

break;

case N\_EXP\_GEQ :

gen\_expression(node->llink);

gen\_expression(node->rlink);

if (isFloatType(node->llink->type))

gen\_code\_i(GEQF,0,0);

else

gen\_code\_i(GEQI,0,0);

break;

case N\_EXP\_NEQ :

gen\_expression(node->llink);

gen\_expression(node->rlink);

if (isFloatType(node->llink->type))

gen\_code\_i(NEQF,0,0);

else

gen\_code\_i(NEQI,0,0);

break;

case N\_EXP\_EQL :

gen\_expression(node->llink);

gen\_expression(node->rlink);

if (isFloatType(node->llink->type))

gen\_code\_i(EQLF,0,0);

else

gen\_code\_i(EQLI,0,0);

break;

case N\_EXP\_AND :

gen\_expression(node->llink);

gen\_code\_l(JPCR,0,i=get\_label());

gen\_expression(node->rlink);

gen\_label\_number(i);

break;

case N\_EXP\_OR :

gen\_expression(node->llink);

gen\_code\_l(JPTR,0,i=get\_label());

gen\_expression(node->rlink);

gen\_label\_number(i);

break;

case N\_EXP\_ASSIGN :

gen\_expression\_left(node->llink);

gen\_expression(node->rlink);

i=node->type->size;

if (i==1)

gen\_code\_i(STXB,0,0);

else

gen\_code\_i(STX,0,i%4?i/4+1:i/4);

break;

default :

gen\_error(100,node->line);

break;

}

}

void gen\_expression\_left(A\_NODE \*node)

{

A\_ID \*id;

A\_TYPE \*t;

int result;

switch(node->name) {

case N\_EXP\_IDENT :

id=node->clink;

t=id->type;

switch (id->kind) {

case ID\_VAR:

case ID\_PARM:

switch (t->kind) {

case T\_ENUM:

case T\_POINTER:

case T\_STRUCT:

case T\_UNION:

gen\_code\_i(LDA, id->level, id->address);

break;

case T\_ARRAY:

if (id->kind==ID\_VAR)

gen\_code\_i(LDA, id->level, id->address);

else

gen\_code\_i(LOD, id->level, id->address);

break;

default:

gen\_error(13,node->line,id->name);

break;

}

break;

case ID\_FUNC:

gen\_code\_s(ADDR, 0, id->name);

break;

default:

gen\_error(13,node->line,id->name);

break;

}

break;

case N\_EXP\_ARRAY :

gen\_expression(node->llink);

gen\_expression(node->rlink);

// gen\_code\_i(CHK,0,node->llink->type->expr);

if (node->type->size>1) {

gen\_code\_i(LITI,0,node->type->size);

gen\_code\_i(MULI,0,0);

}

gen\_code\_i(OFFSET,0,0);

break;

case N\_EXP\_STRUCT :

gen\_expression\_left(node->llink);

id=node->rlink;

if(id->address>0) {

gen\_code\_i(LITI,0,id->address);

gen\_code\_i(OFFSET,0,0);

}

break;

case N\_EXP\_ARROW :

gen\_expression(node->llink);

id=node->rlink;

if(id->address>0) {

gen\_code\_i(LITI,0,id->address);

gen\_code\_i(OFFSET,0,0);

}

break;

case N\_EXP\_STAR :

gen\_expression(node->clink);

break;

case N\_EXP\_INT\_CONST :

case N\_EXP\_FLOAT\_CONST :

case N\_EXP\_CHAR\_CONST :

case N\_EXP\_STRING\_LITERAL :

case N\_EXP\_FUNCTION\_CALL :

case N\_EXP\_POST\_INC :

case N\_EXP\_POST\_DEC :

case N\_EXP\_PRE\_INC :

case N\_EXP\_PRE\_DEC :

case N\_EXP\_NOT :

case N\_EXP\_MINUS :

case N\_EXP\_SIZE\_EXP :

case N\_EXP\_SIZE\_TYPE :

case N\_EXP\_CAST :

case N\_EXP\_MUL :

case N\_EXP\_DIV :

case N\_EXP\_MOD :

case N\_EXP\_ADD :

case N\_EXP\_SUB :

case N\_EXP\_LSS :

case N\_EXP\_GTR :

case N\_EXP\_LEQ :

case N\_EXP\_GEQ :

case N\_EXP\_NEQ :

case N\_EXP\_EQL :

case N\_EXP\_AMP :

case N\_EXP\_AND :

case N\_EXP\_OR :

case N\_EXP\_ASSIGN :

gen\_error(12,node->line);

break;

default :

gen\_error(100,node->line);

break;

}

}

void gen\_arg\_expression(A\_NODE \*node)

{

A\_NODE \*n;

switch(node->name) {

case N\_ARG\_LIST :

gen\_expression(node->llink);

gen\_arg\_expression(node->rlink);

break;

case N\_ARG\_LIST\_NIL :

break;

default :

gen\_error(100,node->line);

break;

}

}

int get\_label()

{

label\_no++;

return(label\_no);

}

void gen\_statement(A\_NODE \*node, int cont\_label, int break\_label, A\_SWITCH sw[], int \*sn)

{

A\_SWITCH switch\_table[100];

int switch\_no=0;

A\_NODE \*n;

int i,l1,l2,l3;

switch(node->name) {

case N\_STMT\_LABEL\_CASE :

/\*

\* if (sw) {

\*sn=\*sn+1;

sw[\*sn].kind=SW\_VALUE;

sw[\*sn].val=node->llink;

sw[\*sn].label=l1=get\_label();

gen\_label\_number(l1);

}

else

gen\_error(21,node->line);

gen\_statement(node->rlink,cont\_label,break\_label,sw,sn);

\*/

break;

case N\_STMT\_LABEL\_DEFAULT :

/\*

\* if (sw) {

\*sn=\*sn+1;

sw[\*sn].kind=SW\_DEFAULT;

sw[\*sn].label=l1=get\_label();

gen\_label\_number(l1);

}

else

gen\_error(20,node->line);

gen\_statement(node->clink,cont\_label,break\_label,sw,sn);

\*/

break;

case N\_STMT\_COMPOUND:

if(node->llink)

gen\_declaration\_list(node->llink);

gen\_statement\_list(node->rlink,cont\_label,break\_label,sw,sn);

break;

case N\_STMT\_EMPTY:

break;

case N\_STMT\_EXPRESSION:

n=node->clink;

gen\_expression(n);

i=n->type->size;

if (i)

gen\_code\_i(POP,0,i%4?i/4+1:i/4);

break;

case N\_STMT\_IF:

gen\_expression(node->llink);

gen\_code\_l(JPC, 0, l1=get\_label());

gen\_statement(node->rlink,cont\_label,break\_label,0,0);

gen\_label\_number(l1);

break;

case N\_STMT\_IF\_ELSE:

gen\_expression(node->llink);

gen\_code\_l(JPC, 0, l1=get\_label());

gen\_statement(node->clink,cont\_label,break\_label,0,0);

gen\_code\_l(JMP, 0, l2=get\_label());

gen\_label\_number(l1);

gen\_statement(node->rlink,cont\_label,break\_label,0,0);

gen\_label\_number(l2);

break;

case N\_STMT\_SWITCH:

//gen\_expression(node->llink);

//gen\_code\_l(SWITCH, 0,l1=get\_label());

//gen\_code\_l(JMP,0,l2=get\_label());

//gen\_statement(node->rlink,cont\_label,l2,switch\_table,&switch\_no);

//gen\_label\_number(l1);

//for (i=1;i<=switch\_no;i++) {

// if (switch\_table[i].kind==SW\_VALUE)

// gen\_code\_i(SWVALUE,0,switch\_table[i].val);

// else

// gen\_code\_i(SWDEFAULT,0,0);

// gen\_code\_l(SWLABEL,0,switch\_table[i].label);

//}

//gen\_code\_i(SWEND,0,0);

//gen\_label\_number(l2);

break;

case N\_STMT\_WHILE:

l3=get\_label();

gen\_label\_number(l1=get\_label());

gen\_expression(node->llink);

gen\_code\_l(JPC, 0, l2=get\_label());

gen\_statement(node->rlink,l3,l2,0,0);

gen\_label\_number(l3);

gen\_code\_l(JMP, 0, l1);

gen\_label\_number(l2);

break;

case N\_STMT\_DO:

l3=get\_label();

l2=get\_label();

gen\_label\_number(l1=get\_label());

gen\_statement(node->llink,l2,l3,0,0);

gen\_label\_number(l2);

gen\_expression(node->rlink);

gen\_code\_l(JPT, 0, l1);

gen\_label\_number(l3);

break;

case N\_STMT\_FOR:

n=node->llink;

l3=get\_label();

if (n->llink) {

gen\_expression(n->llink);

i=n->llink->type->size;

if (i)

gen\_code\_i(POP,0,i%4?i/4+1:i/4);

}

gen\_label\_number(l1=get\_label());

l2=get\_label();

if (n->clink) {

gen\_expression(n->clink);

gen\_code\_l(JPC, 0, l2);

}

gen\_statement(node->rlink,l3,l2,0,0);

gen\_label\_number(l3);

if (n->rlink) {

gen\_expression(n->rlink);

i=n->rlink->type->size;

if (i)

gen\_code\_i(POP,0,i%4?i/4+1:i/4);

}

gen\_code\_l(JMP, 0, l1);

gen\_label\_number(l2);

break;

case N\_STMT\_CONTINUE:

if (cont\_label)

gen\_code\_l(JMP,0,cont\_label);

else

gen\_error(22,node->line);

break;

case N\_STMT\_BREAK:

if (break\_label)

gen\_code\_l(JMP,0,break\_label);

else

gen\_error(23,node->line);

break;

case N\_STMT\_RETURN:

n=node->clink;

if(n) {

i=n->type->size;

if (i%4) i=i/4\*4+4;

gen\_code\_i(LDA, 1, -i);

gen\_expression(n);

gen\_code\_i(STO, 0, i/4);

}

gen\_code\_i(RET,0,0);

break;

default :

gen\_error(100,node->line);

break;

}

}

void gen\_statement\_list(A\_NODE \*node, int cont\_label, int break\_label, A\_SWITCH sw[], int \*sn)

{

switch(node->name) {

case N\_STMT\_LIST:

gen\_statement(node->llink,cont\_label, break\_label,sw,sn);

gen\_statement\_list(node->rlink,cont\_label, break\_label,sw,sn);

break;

case N\_STMT\_LIST\_NIL:

break;

default :

gen\_error(100,node->line);

break;

}

}

void gen\_initializer\_global(A\_NODE \*node, A\_TYPE \*t, int addr) {

}

void gen\_initializer\_local(A\_NODE \*node, A\_TYPE \*t, int addr) {

}

void gen\_declaration\_list(A\_ID \*id)

{

while (id) {

gen\_declaration(id);

id=id->link;

}

}

void gen\_declaration(A\_ID \*id)

{

int i;

A\_NODE \*node;

switch (id->kind) {

case ID\_VAR:

if (id->init)

if (id->level==0)

gen\_initializer\_global(id->init,id->type,id->address);

else

gen\_initializer\_local(id->init,id->type,id->address);

break;

case ID\_FUNC:

if (id->type->expr) {

gen\_label\_name(id->name);

gen\_code\_i(INT,0,id->type->local\_var\_size);

gen\_statement(id->type->expr,0,0,0,0);

gen\_code\_i(RET,0,0);

}

break;

case ID\_PARM:

case ID\_TYPE:

case ID\_ENUM:

case ID\_STRUCT:

case ID\_FIELD:

case ID\_ENUM\_LITERAL:

case ID\_NULL:

break;

default:

gen\_error(100,id->line);

break;

}

}

void gen\_error(int i, int ll, char \*s )

{

gen\_err++;

printf("\*\*\* error at line %d: ",ll);

switch (i) {

case 11:

printf("illegal identifier in expression \n");

break;

case 12:

printf("illegal l-value expression \n");

break;

case 13:

printf("identifier %s not l-value expression \n",s);

break;

case 20:

printf("illegal default label in switch statement \n");

break;

case 21:

printf("illegal case label in switch statement \n");

break;

case 22:

printf("no destination for continue statement \n");

break;

case 23:

printf("no destination for break statement \n");

break;

case 100:

printf("fatal compiler error during code generation\n");

break;

default:

printf("unknown \n");

break;

}

}

void gen\_code\_i(OPCODE op, int l, int a)

{

fprintf(fout,"\t%9s %d, %d\n",opcode\_name[op],l,a);

}

void gen\_code\_f(OPCODE op, int l, float a)

{

fprintf(fout,"\t%9s %d, %f\n",opcode\_name[op],l,a);

}

void gen\_code\_s(OPCODE op, int l, char \*a)

{

fprintf(fout,"\t%9s %d, %s\n",opcode\_name[op],l,a);

}

void gen\_code\_l(OPCODE op, int l, int a)

{

fprintf(fout,"\t%9s %d, L%d\n",opcode\_name[op],l,a);

}

void gen\_label\_number(int i)

{

fprintf(fout,"L%d:\n",i);

}

void gen\_label\_name(char \*s)

{

fprintf(fout,"%s:\n",s);

}