**电子科技大学 信息与软件工程 学院**

**标 准 实 验 报 告**

**（实验）课程名称 编译技术**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：袁佳琪 学 号：2021090909029**

**指导教师：周尔强**

**实验地点：信软楼304 实验时间：2022.11.18**

**一、实验室名称：信软楼304**

**二、实验项目名称：LR语法分析**

**三、实验学时：4学时**

**四、实验目的、原理、内容及步骤：**

**目的：**通过本实验加深对编译技术中重点算法和编译技术的理解，提高学生的编程能力培养好的程序设计风格。了解和掌握LR语法分析的基本原理，根据给出的文法能够设计相应的语义子程序，生成AST节点。学习Bison语法，熟悉YACC语法分析工具的使用，为今后的代码生成打下基础。

**原理：**LR语法分析是建立在实验一词法分析的基础上，利用YACC工具、LR语法分析方法自动生成语法树，与实验二递归下降手动生成语法树形成了鲜明对比。从识别符号开始，再逐个对输入串以句柄进行规约，在相应语义动作的指导下生成AST。在识别过程中，可以将所有标志符及其属性保存至符号表。

**内容：**

1. 学习所提供的“表达式文法”的LR分析处理

理解 calc1.l, calc1.y, calc2.l, calc2.y的内容

在eclipse中建立工程,对calc3.l, calc3.y调试运行

2. 学习lrgram.txt所提供的文法

与递归下降分析所提供的文法作比较

3. 学习、理解 Makefile 文件的格式及写法（选做）

4. 编写SysY语言所提供文法的LR语法分析程序

(1)编写生成“语法树”的相关程序，包括

bison源程序 lrparser.y

flex源程序 lrlex.l

语法树相关程序 ast.h 和 ast.c

(2)其它相关函数(如main函数)等 main.c,

(3)使编译得到的rdparser最终从命令行读取要分析的程序test.c,分析后调用showAst打印该程序的结构。

5. 将分析中的所有标识符及其属性保存至符号表

分析并解决不同作用域下相同变量名的保存 与 查找问题

**实验步骤：**

1、理解Bison语法分析器的结构，理解程序的编译链接过程，编写MakeFile文件，最终生成可执行文件rdparser。

|  |
| --- |
| CC = gcc  CFLAGS = -O0 -g  NAME = rdparser  $(NAME):y.tab.c lex.yy.c ast.c main.c      $(CC) y.tab.c lex.yy.c ast.c main.c -o $(NAME) $(CFLAGS)  y.tab.c:lrparser.y      bison -d lrparser.y  lex.yy.c:lrlex.l      flex lrlex.l  .PHONY: clean  clean:      $(RM) \*.o \*.exe y.tab.c y.tab.h lex.yy.c y.output $(NAME) |

2.编写ast.c，在以便lrparser.y中执行语义动作：

为了统一不同结点类型的创建，减少代码量，我将结点的类型作为参数传入；同时建立链表，代码如下：

past newNode(char\* nodeType, char\* id, int ivalue, past list, past left, past right){

   past var = newAstNode();

   var->nodeType = strdup(nodeType);

   var->id = id;

   var->ivalue = ivalue;

   var->list = list;

   var->left = left;

   var->right = right;

   return var;

}

past ListNode(past HeadNode, past node){

    if(HeadNode->list == NULL){

        HeadNode->list = node;

    }

    else{

        past cur = HeadNode->list;

        while(cur->list != NULL){

            cur = cur->list;

        }

        cur->list = node;

    }

    return HeadNode;

}

3.编写lrparser.y程序，用bison实现语法树的自动生成。

（1）先在Bison文件的定义区，定义yyval的类型，并指定文法的类型：

%union{

    int     number;

    char    str[50];

    past    pAst;

};

%token      <str>       ID

%token      <number>    INTEGER

%token                  INT

%token      <str>       VOID

%token      <str>       CONST

%token      <str>       IF

%token      <str>       ELSE

%token      <str>       WHILE

%token      <str>       BREAK

%token      <str>       CONTINUE

%token      <str>       RETURN

%token      <str>       LESSEQ GREATEQ NOTEQ EQ AND OR

%type       <pAst>      \_CompUnit CompUnit Decl ConstDecl \_ConstDecl ConstDef \_ConstDef ConstInitVal \_ConstInitVal

%type       <pAst>      VarDecl \_VarDecl VarDef \_VarDef InitVal \_InitVal

%type       <pAst>      FuncDef FuncFParams \_FuncFParams FuncFParam \_FuncFParam

%type       <pAst>      Block \_Block BlockItem Stmt ElseStmt Exp Cond LVal \_LVal PrimaryExp

%type       <pAst>      Number UnaryExp FuncRParams \_FuncRParams

%type       <pAst>      MulExp AddExp RelExp EqExp LAndExp LOrExp ConstExp

再编写语法规则和相应的语义动作：

|  |
| --- |
| %%  \_CompUnit: CompUnit             {root = $$; $$ = $1;}           ;  CompUnit: Decl CompUnit         {$$ = ListNode($1,$2);}          | FuncDef CompUnit      {$$ = ListNode($1,$2);}          | Decl                  {$$ = $1;}          | FuncDef               {$$ = $1;}          ;  Decl: ConstDecl           {$$ = newNode("Decl",strdup("const int"),NOINT,NULL,$1,NULL);}      | VarDecl                   {$$ = newNode("Decl",strdup("int"),NOINT,NULL,$1,NULL);}      ;  ConstDecl: CONST INT \_ConstDecl ';'     {$$ = $3;}           ;  \_ConstDecl: ConstDef ',' \_ConstDecl     {$$ = ListNode($1,$3);}            | ConstDef                    {$$ = $1;}            ;  ConstDef: ID \_ConstDef '=' ConstInitVal         {$$ = newNode("ConstDef",strdup($1),NOINT,NULL,$2,$4);}          | ID '=' ConstInitVal                   {$$ = newNode("ConstDef",strdup($1),NOINT,NULL,$3,NULL);}          ;  \_ConstDef: '[' ConstExp ']'  \_ConstDef          {$$ = ListNode($2,$4);}           | '[' ConstExp ']'                     {$$ = $2;}           ;  ConstInitVal: ConstExp                          {$$ = newNode("ConstInitVal",NULL,NOINT,NULL,$1,NULL);}              | '{' \_ConstInitVal '}'             {$$ = newNode("ConstInitVal",NULL,NOINT,NULL,$2,NULL);}              | '{' '}'                           {$$ = newNode("noConstInitVal",NULL,NOINT,NULL,NULL,NULL);}              ;  \_ConstInitVal: ConstInitVal ',' \_ConstInitVal   {$$ = ListNode($1,$3);}               | ConstInitVal                     {$$ = $1;}               ;  VarDecl: INT \_VarDecl ';'               {$$ = $2;}         ;    \_VarDecl: VarDef ',' \_VarDecl            {$$ = ListNode($1,$3);}          | VarDef                        {$$ = $1;}          ;  VarDef: ID                              {$$ = newNode("VarDef",strdup($1),NOINT,NULL,NULL,NULL);}        | ID \_VarDef                      {$$ = newNode("VarDef",strdup($1),NOINT,NULL,$2,NULL);}        | ID '=' InitVal                  {$$ = newNode("VarDef",strdup($1),NOINT,NULL,$3,NULL);}        | ID \_VarDef '=' InitVal          {$$ = newNode("VarDef",strdup($1),NOINT,NULL,$2,$4);}        ;  \_VarDef: '[' ConstExp ']' \_VarDef               {$$ = ListNode($2,$4);}         | '[' ConstExp ']'                       {$$ = $2;}         ;  InitVal: Exp                            {$$ = newNode("InitVal",NULL,NOINT,NULL,$1,NULL);}         | '{' \_InitVal '}'               {$$ = newNode("InitVal",NULL,NOINT,NULL,$2,NULL);}         | '{' '}'                        {$$ = newNode("noInitVal",NULL,NOINT,NULL,NULL,NULL);}         ;  \_InitVal: InitVal ',' \_InitVal           {$$ = ListNode($1,$3);}          | InitVal                        {$$ = $1;}          ;  FuncDef: VOID ID '(' FuncFParams ')' Block      {$$ = newNode("FuncDef  void",strdup($2),NOINT,NULL,$4,$6);}         | INT ID '(' FuncFParams ')' Block       {$$ = newNode("FuncDef  int",strdup($2),NOINT,NULL,$4,$6);}         | VOID ID '(' ')' Block                  {$$ = newNode("FuncDef  void",strdup($2),NOINT,NULL,$5,NULL);}         | INT ID '(' ')' Block                   {$$ = newNode("FuncDef  int",strdup($2),NOINT,NULL,$5,NULL);}         ;  FuncFParams: \_FuncFParams            {$$ = $1;}             ;  \_FuncFParams: FuncFParam ',' \_FuncFParams       {$$ = ListNode($1,$3);}              | FuncFParam                        {$$ = $1;}              ;  FuncFParam: INT ID '[' ']'                      {$$ = newNode("FuncFParam  int",strdup($2),NOINT,NULL,NULL,NULL);}            | INT ID '[' ']' \_FuncFParam          {$$ = newNode("FuncFParam  int",strdup($2),NOINT,NULL,$5,NULL);}            | INT ID                              {$$ = newNode("FuncFParam  int",strdup($2),NOINT,NULL,NULL,NULL);}            ;  \_FuncFParam: '[' Exp ']' \_FuncFParam            {$$ = ListNode($2,$4);}             | '[' Exp ']'                        {$$ = $2;}             ;  Block: '{' \_Block '}'           {$$ = newNode("Block",NULL,NOINT,NULL,$2,NULL);}       | '{' '}'                  {$$ = newNode("noBlock",NULL,NOINT,NULL,NULL,NULL);}       ;  \_Block: BlockItem \_Block        {$$ = ListNode($1,$2);}        | BlockItem               {$$ = $1;}        ;  BlockItem: Decl                 {$$ = $1;}           | Stmt                 {$$ = $1;}           ;  Stmt: LVal '=' Exp ';'                  {$$ = newNode("AssignStmt",NULL,NOINT,NULL,$1,$3);}      | Exp ';'                           {$$ = newNode("ExpStmt",NULL,NOINT,NULL,$1,NULL);}      | ';'                               {$$ = newNode("noStmt",NULL,NOINT,NULL,NULL,NULL);}      | Block                             {$$ = $1;}      | IF '(' Cond ')' Stmt ElseStmt     {$$ = newNode("IfStmt",NULL,NOINT,$6,$3,$5);}      | IF '(' Cond ')' Stmt              {$$ = newNode("IfStmt",NULL,NOINT,NULL,$3,$5);}      | WHILE '(' Cond ')' Stmt           {$$ = newNode("WhileStmt",NULL,NOINT,NULL,$3,$5);}      | BREAK ';'                         {$$ = newNode("BreakStmt",NULL,NOINT,NULL,NULL,NULL);}      | CONTINUE ';'                      {$$ = newNode("ContinueStmt",NULL,NOINT,NULL,NULL,NULL);}      | RETURN Exp ';'                    {$$ = newNode("ReturnStmt",NULL,NOINT,NULL,$2,NULL);}      | RETURN ';'                        {$$ = newNode("ReturnStmt",NULL,NOINT,NULL,NULL,NULL);}      ;  ElseStmt: ELSE Stmt                     {$$ = newNode("ElseStmt",NULL,NOINT,NULL,$2,NULL);}          ;  Exp: AddExp                     {$$ = $1;}     ;  Cond: LOrExp                    {$$ = $1;}      ;  LVal: ID                        {$$ = newNode("LVal",strdup($1),NOINT,NULL,NULL,NULL);}      | ID \_LVal                  {$$ = newNode("LVal",strdup($1),NOINT,NULL,$2,NULL);}      ;  \_LVal: '[' Exp ']' \_LVal        {$$ = ListNode($2,$4);}       | '[' Exp ']'              {$$ = $2;}       ;  PrimaryExp: '(' Exp ')'         {$$ = $2;}            | LVal                {$$ = $1;}            | Number              {$$ = $1;}            ;  Number: INTEGER                 {$$ = newNode("Integer",NULL,$1,NULL,NULL,NULL);}        ;    UnaryExp: PrimaryExp                    {$$ = newNode("UnaryExp",NULL,NOINT,NULL,$1,NULL);}          | ID '(' FuncRParams ')'        {$$ = newNode("UnaryExp",strdup($1),NOINT,NULL,$3,NULL);}          | ID '(' ')'                    {$$ = newNode("UnaryExp",strdup($1),NOINT,NULL,NULL,NULL);}          | '+' UnaryExp                  {$$ = newNode("UnaryExp",strdup("+"),NOINT,NULL,$2,NULL);}          | '-' UnaryExp                  {$$ = newNode("UnaryExp",strdup("-"),NOINT,NULL,$2,NULL);}          | '!' UnaryExp                  {$$ = newNode("UnaryExp",strdup("!"),NOINT,NULL,$2,NULL);}          ;  FuncRParams: \_FuncRParams               {$$ = newNode("FuncRParams",NULL,NOINT,NULL,$1,NULL);}             ;  \_FuncRParams: \_FuncRParams ',' Exp      {$$ = ListNode($1,$3);}              | Exp                       {$$ = $1;}              ;  MulExp: UnaryExp                        {$$ = $1;}        | UnaryExp '\*' MulExp             {$3->id = strdup("\*");$$ = ListNode($1,$3);}        | UnaryExp '/' MulExp             {$3->id = strdup("/");$$ = ListNode($1,$3);}        | UnaryExp '%' MulExp             {$3->id = strdup("%");$$ = ListNode($1,$3);}        ;  AddExp: MulExp                          {$$ = $1;}        | MulExp '+' AddExp               {$3->id = strdup("+");$$ = ListNode($1,$3);}        | MulExp '-' AddExp               {$3->id = strdup("-");$$ = ListNode($1,$3);}        ;  RelExp: AddExp                          {$$ = $1;}        | AddExp '<' RelExp               {$3->id = strdup("<");$$ = ListNode($1,$3);}        | AddExp '>' RelExp               {$3->id = strdup(">");$$ = ListNode($1,$3);}        | AddExp LESSEQ RelExp            {$3->id = strdup($2);$$ = ListNode($1,$3);}        | AddExp GREATEQ RelExp           {$3->id = strdup($2);$$ = ListNode($1,$3);}        ;  EqExp: RelExp                           {$$ = $1;}       | RelExp EQ EqExp                  {$3->id = strdup($2);$$ = ListNode($1,$3);}       | RelExp NOTEQ EqExp               {$3->id = strdup($2);$$ = ListNode($1,$3);}       ;  LAndExp: EqExp                          {$$ = $1;}         | EqExp AND LAndExp              {$3->id = strdup($2);$$ = ListNode($1,$3);}         ;  LOrExp: LAndExp                         {$$ = $1;}        | LAndExp OR LOrExp               {$3->id = strdup($2);$$ = ListNode($1,$3);}        ;  ConstExp: AddExp                        {$$ = $1;}          ;  %% |

4.使用test\_cases中文件进行测试，并使用gdb工具debug以及改进代码。

5. 编写syboltable.c程序，用链表将分析中的所有标识符及其属性保存至符号表，为代码生成打下基础。代码如下：

pidNode newIdNode(){

    pidNode node = malloc(sizeof(idNode));

    if(node == NULL){

        printf("Error: Run out of memory.\n");

        exit(0);

    }

    memset(node, 0, sizeof(idNode));

    return node;

}

//头插法

pidNode newIdList(pidNode head, int CurLevel){

    pidNode new = newIdNode();

    new->count = CurLevel;

    if(head != NULL){

        new->list = head;

        head = new;

    }

    else{

        head = new;

    }

    return head;

}

pidNode newIdNext(pidNode head, char\* id, int count){

    pidNode new = newIdNode();

    new->id = id;

    new->count = count;

    if(head->next != NULL){

        new->next = head->next;

        head->next = new;

    }

    else{

        head->next = new;

    }

    return head;

}

void popList(pidNode head){

    if(head != NULL){

        pidNode pre = head;

        pidNode cur = pre->next;

        head = head->list;

        free(pre);

        while(cur != NULL){

            pre = cur;

            cur = cur->next;

            free(pre);

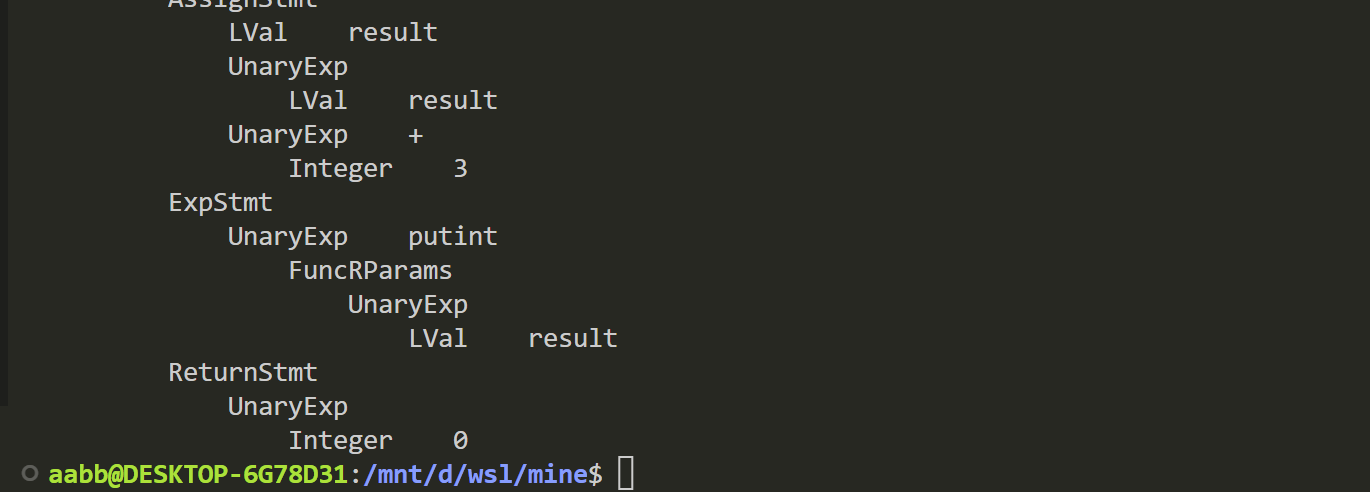
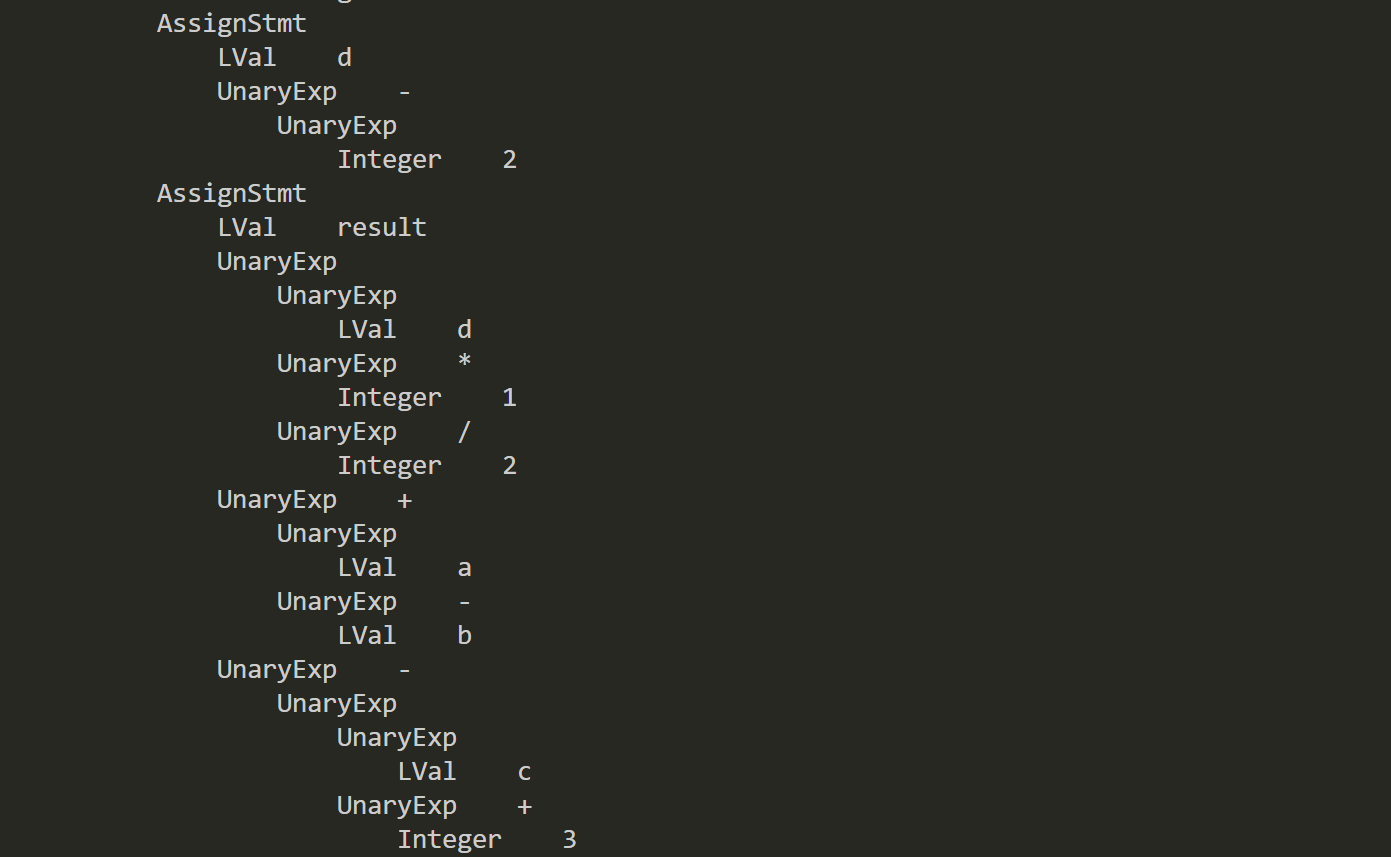
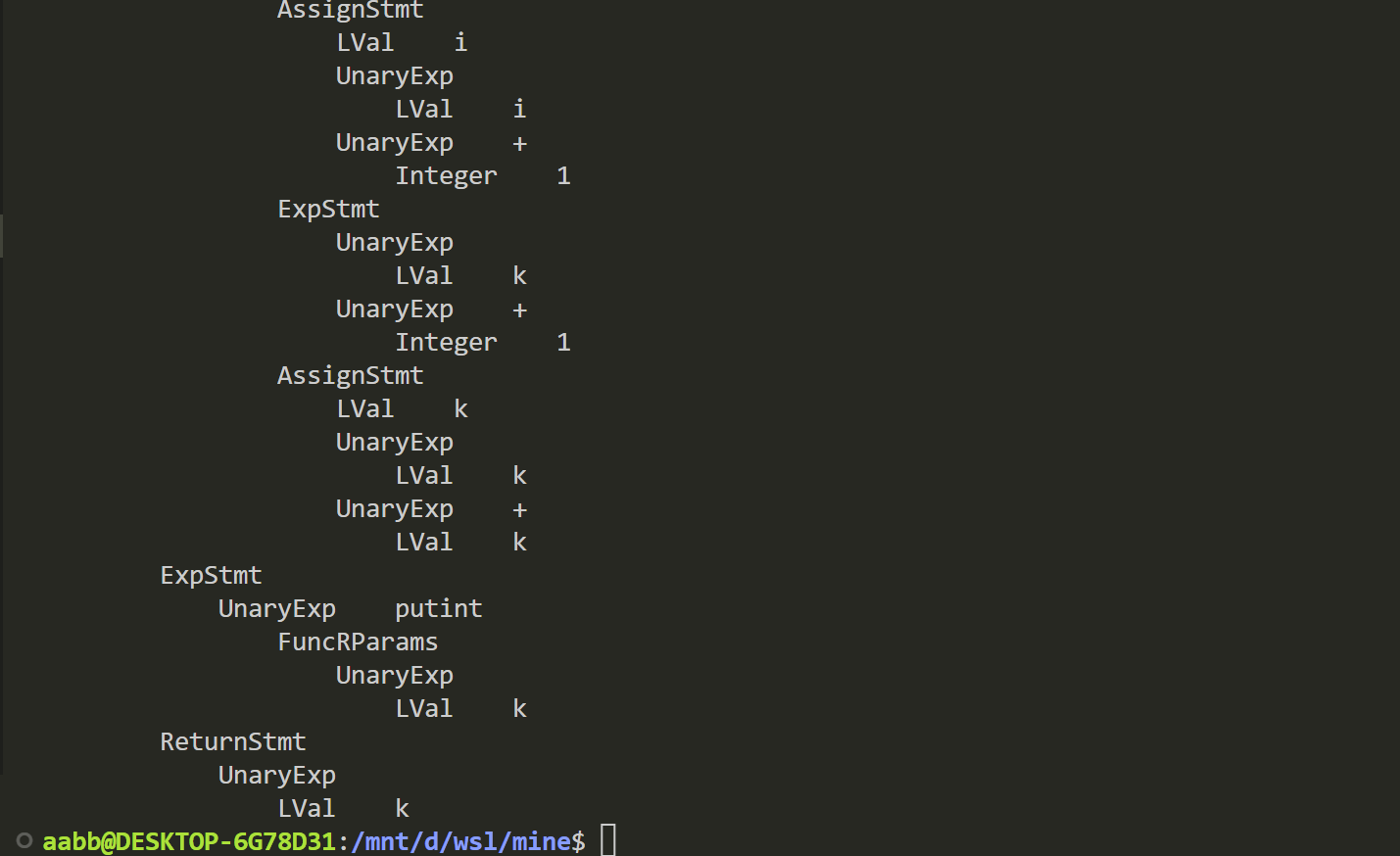
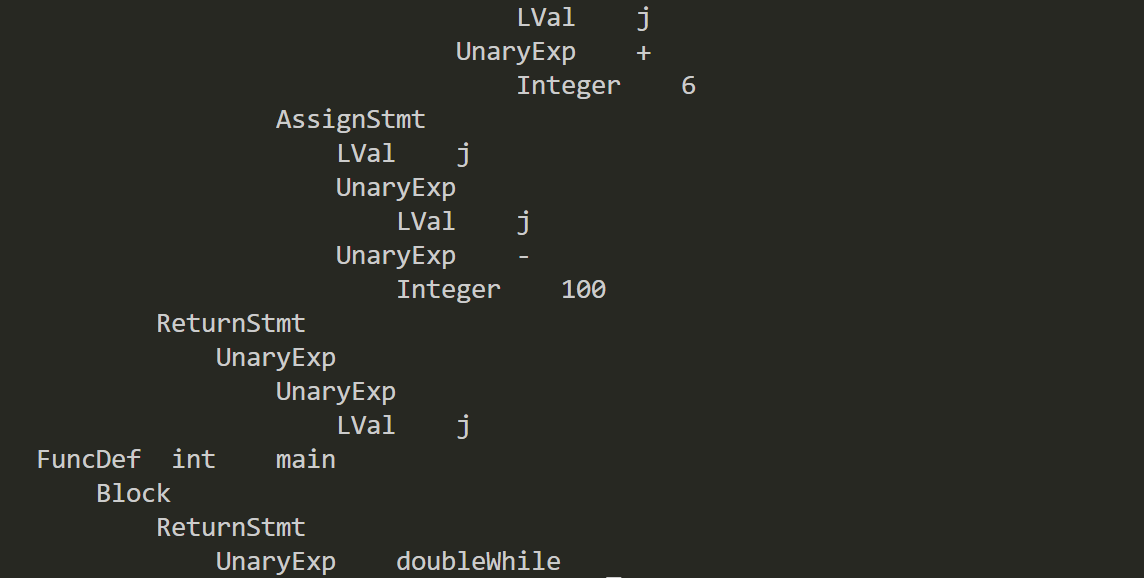
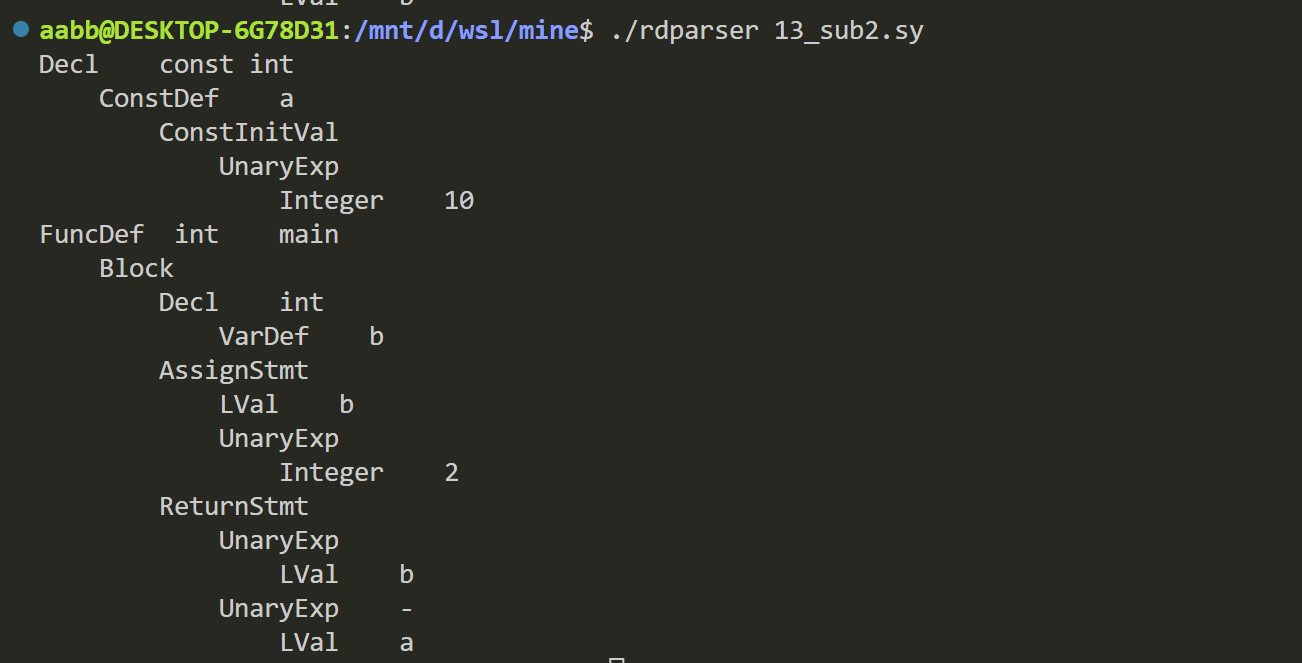
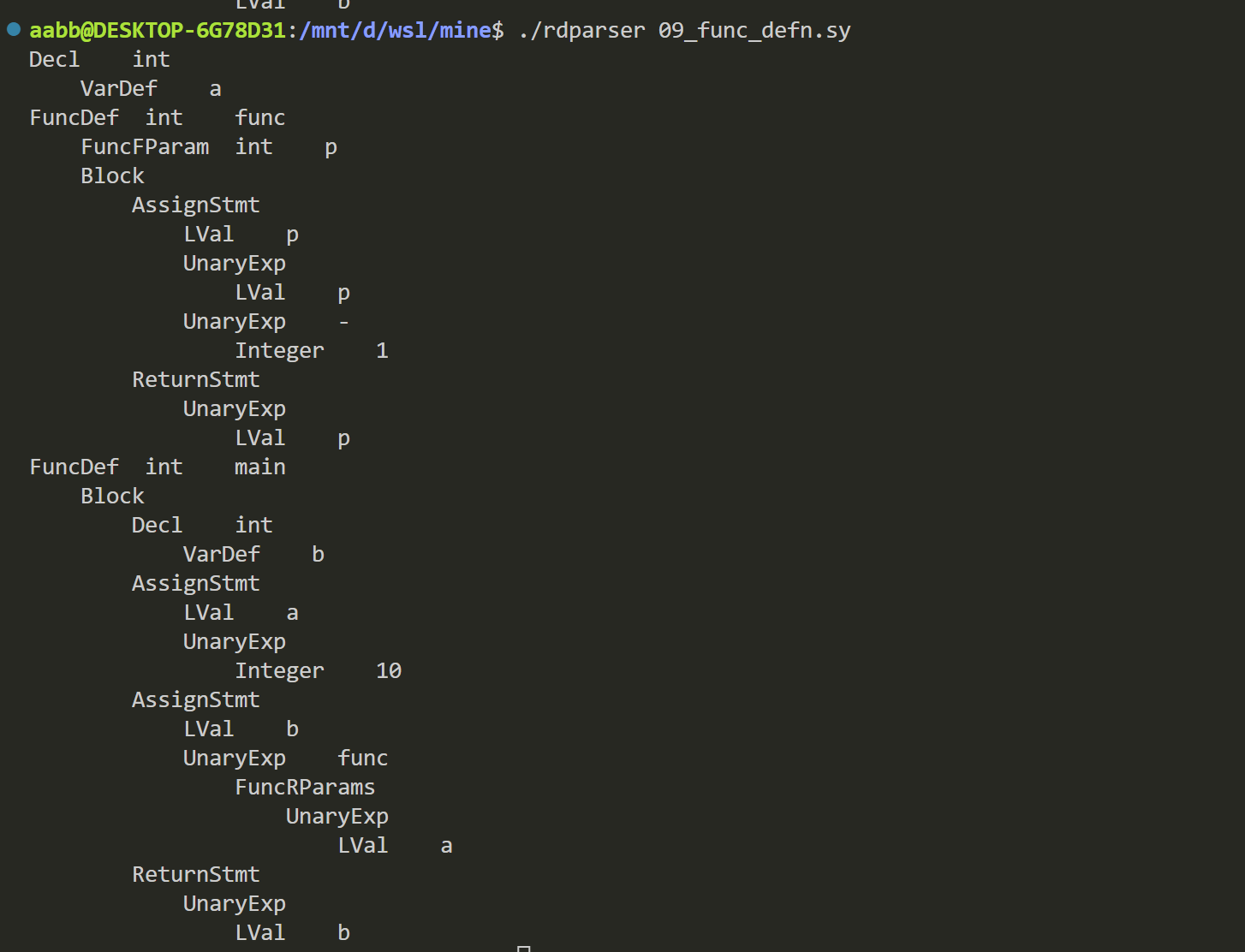
        }

    }

}

**五、实验运行结果：**

以下为test\_cases中部分文件的测试结果。



由于test.c中含有Bison中未枚举的类型str，所以没有测试成功。

**六、实验结论与总结：**

本次实验是第三次实验：基于LR的语法分析方法，利用Yacc工具执行语义动作并生成AST，以此作为代码生成的基础。对比实验二中工作量巨大，需手动编写的递归下降语法分析器，本次实验要轻松许多，得益于yacc工具的使用，实现了语法分析器的自动生成。经过测试，两种方法生成的AST几乎一样。在这次实验中，我还学习了Makefile的编写，极大地简化了每次文件的编译链接过程。

通过本次LR语法分析器的编写，让我对递归下降语法分析和自下而上的LR语法分析有了更深的领悟。同时也熟悉了bison和lex工具的使用和联动方法，使得我对编译这门课的理解不仅仅停留在课本中，而是将所学真正用于编译器的实现。

**报告评分：**

**指导教师签字：**