编译原理实验1实验报告

201840009田永上

1.任务说明

编号49，选做3，要求如下：

要求1.3：识别“//”和“/.../”形式的注释。若输入文件中包含符合定义的“//”和“/.../”形式的注释，你的程序需要能够滤除这样的注释；若输入文件中包含不符合定义的注释（如“/.../”注释中缺少“/*”），你的程序需要给出由不符合定义的注释所引发的错误的提示信息。

2.树和操作的设计

节点设计

```
struct TN
{
    int node_line;//行数
    BOOL if_token;//真表示是终结符，假表示非终结符
    char* node_name;//节点名称(如INT,FLOAT之类的)
    char* node_val;//节点的值(可以没有)
    //采用多叉树结构
    struct TN* node_child;//第一个子节点
    struct TN* node_next;//下一个子节点
};
```

新增节点操作设计

```
struct TN* newNode(char* name,char* value)
{
    yylval.node=(struct TN*)malloc(sizeof(struct TN));
    yylval.node->node_line=yylineno;
    yylval.node->if_token=true;//标记为终结符
    yylval.node->node_name = malloc((strlen(name) + 1) * sizeof(char));//分配略大的内存
    yylval.node->node_val = malloc((strlen(value) + 1) * sizeof(char));
    strcpy(yylval.node->node_name,name);//复制内容
    strcpy(yylval.node->node_val,value);
    yylval.node->node_child=NULL;
    yylval.node->node_next=NULL;
}
```

添加节点操作设计

采用动态数量的参数的设计，基于stdarg.h库，主要原因是产生式体中文法符号的数量并不固定，通过动态参数数量，可以简单地把这些文法符号的属性全部传入add_node函数中。

add_node函数主要用于添加非终结符关联的节点

```
struct TN* add_node(char* name, int num,...)//num为当前节点的子节点个数，等于一个node_child和num-1个node_next
{
    struct TN* cur=(struct TN*)malloc(sizeof(struct TN));
    struct TN* tmp=(struct TN*)malloc(sizeof(struct TN));
    va_list list;
```

```
    va_start(list,num);
    tmp=va_arg(list,struct TN*);
    cur->node_line=tmp->node_line;
    cur->if_token=false;
    cur->node_name=malloc((strlen(name) + 1) * sizeof(char));
    cur->node_val=malloc(10 * sizeof(char));
    strcpy(cur->node_name,name);
    strcpy(cur->node_val,"");
    cur->node_child=tmp;
    int cnt=num-1;
    while(cnt--)
    {
        tmp->node_next=va_arg(list,struct TN*);
        if(tmp->node_next!=NULL)
        {
            tmp=tmp->node_next;
        }
    }
    return cur;
}
```

打印树的操作

```
void print_tree(struct TN* root,int depth)
{
    if(root==NULL) return;
    for(int i=0;i<depth;i++) printf("  ");//注意空格数量
    printf("%s",root->node_name);
    if(root->if_token==false) printf(" (%d)",root->node_line);
    else if(strcmp(root->node_name,"ID")==0||strcmp(root-
>node_name,"TYPE")==0||strcmp(root->node_name,"INT")==0) printf(": %s",root-
>node_val);//通过strcmp来比较字符名称，switch适用于类型多的场景
    else if(strcmp(root->node_name,"FLOAT")==0)  printf(": %lf",atof(root->node_val));
    printf("\n");
    print_tree(root->node_child,depth+1);//向下一层移动
    print_tree(root->node_next,depth);//移动到下一个子节点
}
```

3.选做内容设计

识别注释

对于"//"设置一下

```
COMMENT "//"
```

通过用input，消耗掉所在行的所有内容，从而模拟注释的效果

```
{COMMENT} {char getin=input(); while(getin!='\n') getin=input();}
```

对于" /* ..... */ "，采用如下处理

```
"/*" {
    char c;
    c=input();
    while (c!= '\0') { // '\0'是字符数组结束的标志
        if (c == '*') {
            while (c == '*') c=input();
```

```
            if (c == '/') { // 找到 */
                break;
            }
        }
        c=input(); //持续消耗字符，模拟注释的效果
    }
    if (c == '\0') //若读到*/那么c的值为'/',否则是'\0',所以在该情况下报错
    {
        printf("Error type B at Line %d: Syntax error.\n",yylineno);
        right=1;
    }
}
```

4.错误发现和恢复

定义一个变量right来表示是否出错

```
typedef int BOOL;
#define true 1
#define false 0
int right=true;
```

在另外的代码中要使用right则需要如下声明：

```
extern int right;
```

修改了yyerror，使其可以按照需求报错

```
int yyerror(char* msg)
{
    fprintf(stderr, "Error type B at line %d:%s\n",yylineno,msg);
    return 0;
}
```

5.执行

```
make
编译项目代码
make clean
清楚内容
```