LR 语法分析

请根据LR语法分析实验中给定的文法，利用bison设计一语法分析器，该分析器从标准输入读入源代码后，得到该源码对应的抽象语法树。

请在本页面上传本实验所有源代码及实验报告（源代码至少包括一个flex源文件，即 .l 文件，一个 bison 源文件，即 .y 文件，如有其它 .c 或 .h 文件，也请一并上传）

本实验中所构建的抽象语法树结点类型由 nodeType 标识，对该变量的定义包含在 node\_type.h 中，此头文件可直接使用，无需上传。

实验中所构建的抽象语法树的结构参照 clang 所生成语法树结构（clang生成语法树中的 ImplicitCastExpr 结点可忽略，其子结点提升到该 ImplicitCastExpr 所在位置），构建完成后，需调用 showAST 函数输出所有结点的类型及关键属性，showAST 的参考实现如下，其中node是抽象语法树根结点，nest 为嵌套的层次数，use\_blank为真时用缩进的形式展示，方便检查调试；use\_blank为假时则为检查所核对的输出样式。例如对于源代码

int main(){

return 3;

}

本实验所要求的输出为：

0FUNCTION\_DECL 'main'

1COMPOUND\_STMT

2RETURN\_STMT

3INTEGER\_LITERAL 3

void showAst(past node, int nest, bool use\_blank){

if(node == NULL)

return;

if(use\_blank) {

for (int i = 0; i < nest; i++)

printf(" ");

}else{

printf("%d", nest);

}

switch (node->nodeType){

case INTEGER\_LITERAL:

printf("%s %d\n", "INTEGER\_LITERAL", node->ivalue);

break;

case BINARY\_OPERATOR:

printf("%s ", "BINARY\_OPERATOR");

showToken(node->ivalue);

printf("\n");

break;

case UNARY\_OPERATOR:

printf("%s ", "UNARY\_OPERATOR");

showToken(node->ivalue);

printf("\n");

break;

case FUNCTION\_DECL:

printf("%s '%s'\n", "FUNCTION\_DECL", node->svalue);

break;

case CALL\_EXPR:

printf("%s\n", "CALL\_EXPR");

break;

case COMPOUND\_STMT:

printf("%s\n", "COMPOUND\_STMT");

break;

case DECL\_STMT:

printf("%s\n", "DECL\_STMT");

break;

case VARIABLE\_REF:

printf("%s\n", "VARIABLE\_REF");

break;

case PARM\_DECL:

printf("%s '%s'\n", "PARM\_DECL", node->svalue);

break;

case VAR\_DECL:

printf("%s: '%s'\n", "VAR\_DECL", node->svalue);

break;

case DECL\_REF\_EXPR:

printf("%s: '%s'\n", "DECL\_REF\_EXPR", node->svalue);

break;

case RETURN\_STMT:

printf("%s\n", "RETURN\_STMT");

break;

case WHILE\_STMT:

printf("%s\n", "WHILE\_STMT");

break;

case BREAK\_STMT:

printf("%s\n", "BREAK\_STMT");

break;

case CONTINUE\_STMT:

printf("%s\n", "CONTINUE\_STMT");

break;

case IF\_STMT:

printf("%s\n", "IF\_STMT");

showAst(node->if\_cond, nest+1, use\_blank);

break;

case INIT\_LIST\_EXPR:

printf("%s\n", "INIT\_LIST\_EXPR");

break;

case ARRAY\_SUBSCRIPT\_EXPR:

printf("%s\n", "ARRAY\_SUBSCRIPT\_EXPR");

break;

case PAREN\_EXPR:

printf("%s\n", "PAREN\_EXPR");

break;

default:

printf("%s\n", "NOT IMPlemented");

break;

}

showAst(node->left, nest+1, use\_blank);

showAst(node->right, nest+1, use\_blank);

showAst(node->next, nest, use\_blank);

}

void showToken(int Token){

switch (Token) {

case Y\_INT:

printf("%s", "INTEGER");

break;

case Y\_FLOAT:

printf("%s", "FLOAT");

break;

case Y\_CONST:

printf("%s", "CONST");

break;

case Y\_ADD:

case '+':

printf("'%c'", '+');

break;

case Y\_SUB:

case '-':

printf("'%c'", '-');

break;

case Y\_MUL:

case '\*':

printf("'%c'", '\*');

break;

case Y\_DIV:

printf("'%c'", '/');

break;

case Y\_MODULO:

case '%':

printf("'%c'", '%');

break;

case Y\_LESS:

case '<':

printf("'%c'", '<');

break;

case Y\_LESSEQ:

printf("'%s'", "<=");

break;

case Y\_GREAT:

case '>':

printf("'%c'", '>');

break;

case Y\_GREATEQ:

printf("'%s'", ">=");

break;

case Y\_NOTEQ:

printf("'%s'", "!=");

break;

case Y\_EQ:

printf("'%s'", "==");

break;

case Y\_NOT:

case '!':

printf("'%c'", '!');

break;

case Y\_AND:

printf("'%s'", "&&");

break;

case Y\_OR:

printf("'%s'", "||");

break;

case Y\_ASSIGN:

case '=':

printf("'%c'", '=');

break;

default:

break;

}

}

抽象语法树结点的参考设计如下：

typedef struct \_ast ast;

typedef struct \_ast \*past;

struct \_ast{

int ivalue;

float fvalue;

char\* svalue;

node\_type nodeType;

past left;

past right;

past if\_cond;

past next;

};