**数据结构及接口定义**

1. **词法分析**
2. **符号定义**
   1. **关键字token类型**

PROGRAM "program"

CONST "const"

VAR "var"

ARRAY "array"

OF "of"

INTEGER "integer"

REAL "real"

BOOLEAN "boolean"

CHAR "char"

PROCEDURE "procedure"

FUNCTION "function"

BEGIN "begin"

END "end"

IF "if"

THEN "then"

FOR "for"

TO "to"

DO "do"

READ "read"

WRITE "write"

ELSE "else"

NOT "not"

TRUE "true"

FALSE "false"

* 1. **运算符**

":=" { ASSIGN}

"<>" { NEQ }

"<=" { LEQ }

">=" { GEQ}

"=" { EQ; }

"<" { LT; }

">" { GT; }

"+" { PLUS; }

"-" { MINUS; }

"\*" { MULT; }

"/" { DIVISION; }

“div” { DIV; }

“mod” {MOD;}

"(" { LPAREN; }

")" { RPAREN; }

"[" { LBRACKET; }

"]" { RBRACKET; }

";" { SEMICOLON; }

"," { COMMA; }

":" { COLON; }

"." { DOT; }

* 1. **其他**

DIGIT [0-9]

LETTER [a-zA-Z]

ID {LETTER}({LETTER}|{DIGIT})\*

INTEGER\_NUM {DIGIT}+

REAL\_NUM {DIGIT}+\.{DIGIT}+(E[+-]?{DIGIT}+)?

CHARACTER \'([^\']|\'\')\'

STRING \'([^\']|\'\')\*\'

EPSILON

%token <num> INTEGER\_LIT

%token <fnum> REAL\_LIT

%token <ch> CHAR\_LIT

%token <str> STRING\_LIT IDENTIFIER

1. **接口**

// ------------------------- 接口定义 -------------------------

// 词法分析接口

int yylex();

输入：Pascal字符流

输出：Token流

1. **语法分析**
   1. **接口**

// ------------------------- 接口定义 -------------------------

// 语法分析接口

void yyparse(ASTNode \* root);

输入：Token流

参数：AST根节点指针

1. **语法树**
2. **数据结构**

// ------------------------- AST 节点类型枚举 -------------------------

typedef enum {

    TOKEN\_KEYWORD, TOKEN\_OPERATOR, TOKEN\_IDENTIFIER,

    TOKEN\_NUMBER, PROGRAMSTRUCT, PROGRAM\_HEAD, PROGRAM\_BODY,

    IDLIST, CONST\_DECLARATIONS, CONST\_DECLARATION,

    CONST\_VALUE, VAR\_DECLARATIONS, VAR\_DECLARATION, TYPE,

    BASIC\_TYPE, PERIOD, SUBPROGRAM\_DECLARATIONS,

    SUBPROGRAM, SUBPROGRAM\_HEAD, FORMAL\_PARAMETER,

    PARAMETER\_LIST, PARAMETER, VAR\_PARAMETER,

    VALUE\_PARAMETER, SUBPROGRAM\_BODY, COMPOUND\_STATEMENT,

    STATEMENT\_LIST, STATEMENT, VARIABLE\_LIST, VARIABLE,

    ID\_VARPART, PROCEDURE\_CALL, ELSE\_PART, EXPRESSION\_LIST,

    EXPRESSION, SIMPLE\_EXPRESSION, TERM, FACTOR

} NodeType;

// ------------------------- 关键字类型枚举 -------------------------

typedef enum {

    PROGRAM, CONST, VAR, ARRAY, OF, INTEGER, REAL, BOOLEAN,

    CHAR, PROCEDURE, FUNCTION, BEGIN, END, IF, THEN,

    FOR, TO, DO, READ, WRITE, ELSE, NOT, TRUE, FALSE

} KeywordType;

// ------------------------- 操作符类型枚举 -------------------------

typedef enum {

    OP\_EQ, OP\_NE, OP\_L, OP\_LE, OP\_G, OP\_GE,     //RELOP运算符

    OP\_ADD, OP\_SUB, OP\_OR,                      //ADDOP运算符

    OP\_MUL, OP\_D, OP\_DIV, OP\_MOD, OP\_AND,       //MULOP运算符

    OP\_ASSIGN                                   //ASSIGNOP运算符

} OperatorType;

// ------------------------- 字面量类型枚举 -------------------------

typedef enum {

    LIT\_INTEGER,    // 整数字面量

    LIT\_REAL,       // 实数字面量

    LIT\_CHAR,        // 字符字面量

    LIT\_BOOLEAN      // 布尔字面量

} LiteralType;

typedef enum{

    INTEGER\_TYPE,

    REAL\_TYPE,

    BOOLEAN\_TYPE,

    CHAR\_TYPE

}BasicType;

typedef enum{

    ARRAY\_TYPE

}AdvancedType;

// ------------------------- AST 节点结构定义 -------------------------

typedef struct ASTNode {

    NodeType node\_type;

    union{

        KeywordType key\_type;

        OperatorType op\_type;

        LiteralType lit\_type;

        BasicType bas\_type;

        AdvancedType adv\_type;

    }val\_type;

    union{

        char \* str;

        int val;

        float fval;

    }value;

    int lineno;

    struct ASTNode \* parent;

    struct ASTNode \* brother;

    struct ASTNode \* child;

} ASTNode;

1. **接口**

// ------------------------- AST 接口定义 -------------------------

ASTNode \*create\_astnode(NodeType node\_type);

void free\_astnode(ASTNode \*root);

bool check\_node\_type(ASTNode \*Node1, ASTNode \*Node2);

1. **符号表**
2. **数据结构**

// ------------------------- 符号类型枚举 -------------------------

typedef enum {

    SYM\_VARIABLE,       // 变量

    SYM\_FUNCTION,       // 函数（有返回值）

    SYM\_PROCEDURE,      // 过程（无返回值）

    SYM\_PARAMETER,      // 函数/过程的参数

    SYM\_TYPE,           // 类型定义（如数组、结构体）

    SYM\_CONSTANT,       // 常量

    SYM\_LABEL,          // 标签（goto 目标）

    SYM\_ENUM\_MEMBER     // 枚举成员

} SymbolType;

// ------------------------- 类型系统定义 -------------------------

typedef enum {

    TYPE\_BASIC,         // 基础类型（integer/real/boolean/char）

    TYPE\_ARRAY,         // 数组类型

    TYPE\_POINTER,       // 指针类型

    TYPE\_FUNCTION       // 函数类型

} TypeCategory;

// 基础类型枚举

typedef enum{

    INTEGER\_TYPE,       // 整形

    REAL\_TYPE,          // 浮点

    BOOLEAN\_TYPE,       // 布尔

    CHAR\_TYPE           // 字符

}BasicType;

// 数组范围定义

typedef struct {

    int lower\_bound;                // 数组下界

    int upper\_bound;                // 数组上界

    struct ArrayRange \*next\_range;  // 下一组维数指针

} ArrayRange;

// 数组类型详细信息

typedef struct {

    ArrayRange range;           // 数组范围（如 1..10）

    struct TypeInfo \*element\_type; // 元素类型

} ArrayInfo;

// 函数类型详细信息

typedef struct {

    struct TypeInfo \*return\_type;   // 返回值类型（过程为 NULL）

    struct ParamList \*params;       // 参数列表

} FunctionInfo;

// 类型信息结构体

typedef struct TypeInfo {

    TypeCategory category;      // 类型类别

    union {

        BasicType basic;        // 基础类型名

        ArrayInfo array;        // 数组类型信息

        FunctionInfo function;  // 函数类型信息

    } details;

} TypeInfo;

// ------------------------- 参数列表结构 -------------------------

typedef struct ParamList {

    char \*name;                 // 参数名

    struct TypeInfo \*type;      // 参数类型

    struct ParamList \*next;     // 下一个参数

} ParamList;

// ------------------------- 符号表条目 -------------------------

typedef struct SymbolEntry {

    char \*name;                 // 符号名称

    SymbolType sym\_type;        // 符号类型（变量/函数等）

    TypeInfo \*data\_type;        // 符号的类型信息

    int scope\_level;            // 作用域层级

    struct SymbolEntry \*next;   // 链表指针

} SymbolEntry;

// ------------------------- 符号表结构 -------------------------

typedef struct SymbolTable {

    SymbolEntry \*entries;       // 符号条目链表

    struct SymbolTable \*parent; // 父作用域

    int level;                  // 当前作用域层级

} SymbolTable;

1. **接口**

// ------------------------- 符号表操作接口 -------------------------

// 创建符号表

SymbolTable \*symtab\_create(int level, SymbolTable \*parent);

// 插入符号条目

void symtab\_insert(SymbolTable \*table, SymbolEntry \*entry);

// 查找符号条目（recursive 表示是否递归查找父作用域）

SymbolEntry \*symtab\_lookup(SymbolTable \*table, const char \*name, bool recursive);

// 销毁符号表（递归释放内存）

void symtab\_destroy(SymbolTable \*table);

// 创建符号条目

SymbolEntry \*symtab\_create\_entry(const char \*name, SymbolType sym\_type, TypeInfo \*data\_type);

// ------------------------- 类型系统操作接口 -------------------------

// 创建基础类型

TypeInfo \*type\_create\_basic(BasicType basic);

// 创建数组类型

TypeInfo \*type\_create\_array(ArrayRange range, TypeInfo \*element\_type);

// 创建函数类型

TypeInfo \*type\_create\_function(TypeInfo \*return\_type, ParamList \*params);

1. **代码生成**
   1. **接口**

// ------------------------- 接口定义 -------------------------

//代码生成接口

void generate(FILE \* outputfile, ASTNode \* root, SymbolTable \* symtab);

参数表：

|  |  |
| --- | --- |
| outputfile | 输出文件 |
| root | 语法树根节点指针 |
| SymbolTable | 符号表入口指针 |