完成了 C--词法分析的十七种基本错误的分析，也完成了选做二，变量的定义受嵌套作用域的影响，采用的数据结构为 hash 表存储变量符号加十字链表确定作用域的方法.

参考了讲义上的数据结构实现：

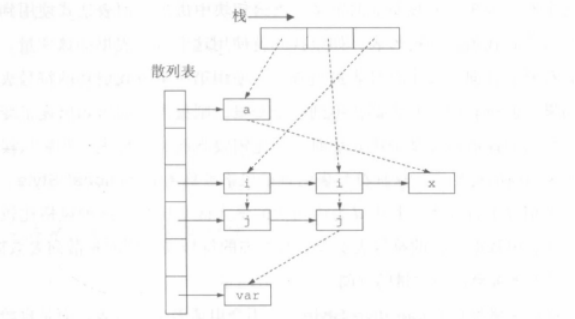

图 2-1 基于十字链表和 open hashing 散列表的符号表

```c
typedef struct fieldList {
    char* name;       // 域的名字
    pType type;       // 域的类型
    pFieldList tail;  // 下一个域
} FieldList;

typedef struct tableItem {
    int symbolDepth;    //深度
    pFieldList field;   //字段列表
    pItem nextSymbol;  // 同一深度的下一个符号
    pItem nextHash;    // 相同hash的下一个符号
} TableItem;

typedef struct hashTable {
    pItem* hashArray;   //指针数组，指向tableitem
} HashTable;

typedef struct stack {
    pItem* stackArray;  //指针数组
    int curStackDepth;  //快速确定当前的符号深度
} Stack;

typedef struct table {
    pHash hash;
    pStack stack;
    int unNamedStructNum;
} Table;
```

为了更清晰地区分函数符号和结构体的定义与声明，在教材中对类型进行了调整。我们现在存储了结构体和函数的相关信息。对于结构体的定义，结构体的名称会被保存在上一级的 FieldList 中，而在类型中的 structname 会保持为 null。而对于声明，上一级的 FieldList 会存储变量的名称，而类型中会存储变量结构体类型的名称。对于匿名结构体，我们使用数字编号作为其名称，以避免与其他符号的冲突。这些编号是通过在表格结构中记录匿名结构体的数量来获得的。

```c
typedef struct type {
    Kind kind;
    union {
        // 基本类型
        BasicType basic;
        // 数组类型信息包括元素类型与数组大小构成
        struct {
            pType elem;
            int size;
        } array;
        // 结构体类型信息是一个链表
        struct {
            char* structName;
            pFieldList field;
        } structure;

        struct {
            int argc;          // argument counter
            pFieldList argv;   // argument vector
            pType returnType;  // returnType
        } function;
    } u;
} Type;
```

语义分析中，当程序碰到符号需要处理时必定是进入了 ExtDef 后，所以我们的程序遍

历语法树，遇到 ExtDef 节点便从该节点开始进行语义分析，然后对 ExtDef 展开会得到的所有非终结符节点进行处理对符号表进行操作以及查找语义错误。

运行：

编写了 makefile 文件，在 Code 目录下 make 即可：

```
parser: syntax $(filter-out $(LFO),$(OBJS))
    $(CC) -o parser $(filter-out $(LFO),$(OBJS)) -lfl -ly

syntax: lexical syntax-c
    $(CC) -c $(YFC) -o $(YFO)

lexical: $(LFILE)
    $(FLEX) -o $(LFC) $(LFILE)

syntax-c: $(YFILE)
    $(BISON) -o $(YFC) -d -v $(YFILE)

-include $(patsubst %.o, %.d, $(OBJS))
```

编写了两个脚本，更方便的测试：

$ parse.sh

```
1  mkdir -p out2
2  file_name=$(basename $1)
3  ./Code/parser $1 > ./out2/${file_name}_out.txt 2>&1
```

$ auto-test.sh

```
1  for file in ./lab2_test/*
2      do
3          echo Testing with $file
4          ./parse.sh $file
5      done
```

根目录下运行 auto-test.sh 即可将生成的结果保存到 out2 目录下