

Phase 2 report

Basic implement

在阶段二中为了实现语义检查，我们设计了四张符号表（`varTable`, `structTable`, `funcTable`, `arrayTable`），表的具体实现方式为链表。`varTable` 用来记载所有非数组变量，包括私有变量（`char`, `int`, `float`）和自定义结构体变量。`arrayTable` 用来记录数组，`structTable` 用来记录定义的结构体信息，`funcTable` 用来记录定义的函数信息。

Bonus implement

Bonus 1

我们实现了变量的作用域判断。我们在 `varTable` 和 `arrayTable` 的节点中加入了一个 `level` 值，这个值代表了该变量被定义时所处的程序深度。

```
typedef struct var_  
{  
    // int a = 1; a is the name  
    char *name;  
    // int is the typr  
    char *type;  
    // combine instruct and strucNum to judge if two instruct vars that have the  
    same name are different  
    int level;  
    struct var_ *next;  
    struct var_ *before;  
}var;
```

程序深度随 `parser` 的运行实时改变，每经过一个 `LC({})`，程序深度会加一，每经过一个 `RC({})`，程序深度会减一。当变量或数组被定义时，会将节点 `level` 值设置为当前程序深度，当在程序某个时刻我们需要查变量表时，我们只能在表中小于等于当前深度的节点中寻找对象。

```
int main(){  
    int a;  
    if (true){  
        int b;  
    }  
    b = 1;  
    return 0;  
}
```

比如上面这个简单的例子中，`a` 的 `level` 为1，而 `b` 的 `level` 为2，当程序离开 `if` 语句时，程序深度为1，所以找不到 `varTable` 中的 `b`，此时 `parser` 报错。

Bonus 2

我们还实现了通过 `structural equivalence` 来判断两个结构体是否一致，具体做法为在将一个结构体加入 `structTable` 时，我们会按顺序记录结构体内部的参数类型，即 `struc_` 中的 `type` 字符指针数组。由于我们在向 `type` 中加值时是按照 `parser` 运行顺序进行的，所以 `type` 中同样蕴含了结构体中参数的顺序信息。

```
typedef struct struc_  
{  
    char *name;  
    int typeNum;  
    char *type[10];  
    struct struc_ *next;  
    struct struc_ *before;  
}Struct;
```

当我们定义一个结构体时，我们需要判断两个点：

1. 在 structTable 中是否已经有同名结构体
2. 在 structTable 中是否有结构体与新定义结构体相同的结构，即两者的 type 数组长度和内容完全一致，如果有我们同样认为此时发生了 `redefine`