lab1 C语言声明分析器

&修改说明

本次的修改变动在对函数参数的处理部分，即Part2部分。改动后可以处理函数的嵌套问题。

&实验要求

实现一个递归下降的分析处理程序，实现C语言变量声明分析，其功能如下：

> 1.能够将输入中各个声明变量的类型表达式等相关信息加以输出;
>
> 2.可进行简单的类型检查，如函数返回值类型不能是数组，而数组的元素类型也 不能是函数类型等。
>
> 3.可输出相关类型的大小。假设 int 与指针类型大小均为 1，那么 int a[20]的大小 为 20。

例如输入

```
int *pa[20][30]; //表达式1
int (*p(int * s,int (*t)(int *m, int n, int (*l())[20]),int k[10]))[10][20];//表达式2
```

得到输出

```
int *pa[20][30]; //原声明
pa is type of : array(20,array(30,pointer(int)))//pa类型
Type Chencking ...  OK! //pa类型检查
type size : 600 //pa类型大小
ValueType is : array(30,pointer(int)) //数组pa内存的值的类型

function(pointer(int) X pointer(function(pointer(int) X int X function(void =>
pointer(array(20,int))) => int)) X array(10,int) => pointer(array(10,array(20,int))))
```

&调研报告

Part1 C语言声明的组成

一个声明的组成

名字	出现形式	数量
类型说明符[type-specifier]	VOID, INT	>=1
init声明器[init_declarator]	声明器	仅一个
更多声明器[declarator]	声明器	>=0
分号	;	仅一个

一个声明器[declarator]的组成

标识符以及与它组合在一起的任何指针、 函数括号、 数组下标等

Part2 声明非法情况

发现非法操作直接报错，不继续分析

- 函数的返回值不能是一个函数， 所以像 lyt() () 这样是非法的
- 函数的返回值不能是一个数组， 所以像 lyt() [] 这样是非法的
- 数组里面不能有函数， 所以像 lyt[] ()这样是非法的

Part3 C语言声明优先级规则

1.开始读取

> 声明从名字开始读取，然后按照优先级顺序依次读取

2.优先级从高到低依次为

> (1)声明中被括号括起来的那部分
> (2)后缀操作符
> 括号（）表示这是一个函数，而方括号[]表示这是一个数组
> (3)前缀操作符
> *表示“指向...的指针”

3.函数的层级嵌套示例

4.数据结构图解

(1) `Deal_With_Declarator()`

(2) `Deal_With_Arrays()`

> step1: 处理n维数组下标，如3维数组下标"[][][]";
> step2: 数组里面不能有函数， 所以像 lyt[] () 这样是非法的，处理这样的非法表达式并报错

(3) `Deal_With_Function_Args()`

```
进入一层函数，则函数嵌套层次加一，同时开辟一个新栈供新层次的函数使用。
新函数层级中的表达式同样调用 Deal_With_Declarator()分析
```

(4) `Deal_With_Pointer()`

```
step1: 判断指针类型是函数指针，数组指针，还是普通指针
step2: 如果是函数指针，调用Deal_With_Function_Args()处理;
       如果是数组指针，调用 Deal_With_Arrays()处理
       如果是普通指针，直接弹出打印
```

&实验过程

Part1 一个基本的Cdecl+增加功能:变量类型/类型大小/存值类型

Step 1 一个基本的Cdecl+变量类型打印

设计方案

```
数据结构: 一个堆栈
具体操作:
从左向右读取，把各个标记依次压入堆栈，直到读到标识符为止。然后我们继续向右读入一个标记，也就是标识符右边的那个标记。接着，观察标识符左边的那个标记（需要从堆栈中弹出）
```

主要数据结构

```
struct token
{
    char type;
    char string[MAXTOKENLEN];
}

struct token stack[MAXTOKENS]; //保存第一个identifier前的所有token

struct token latest;//刚读入的token
```

词法分析

```
//F1:字符串分类
ClassifyString()
{
    查看当前标记;
```

```
    通过latest.type返回一个值，内容为"type","qualifier",或"identifier";
}

//F2:取标记
GetToken()
{
    把下一个标记读入latest.string;
    if 是字母数字组合
    {
        ClassifyString();
    }
    else //是一个单字符标记
    {
        latest.type=该标记;
    }
    结束this.string;
}

//F3:读至第一个标识符
Read_To_First_Identifier()
{
    GetToken();
    while latest.type != identifier
    {
        push(latest.string);
        GetToken();
    }
    Print "latest.string is type of:"
    GetToken();
}
```

声明解析

```
//F4:处理函数参数
Deal_With_Function_Args()
{
    读取越过右括号")"后，打印"函数返回"
}

//F5:处理函数数组
Deal_With_Arrays()
{
    读取"[size]"后将其打印并继续读取
}

//F6:处理指针
Deal_With_Pointer()
{
    从堆栈中读取"*"时，打印"指向...的指针";
    将该"*"弹出堆栈;
}

//F7:处理声明器
```

```
Deal_With_Declarator()
{
    if latest.type == '['
        Deal_With_Arrays();
    if latest.type == '('
        Deal_With_Function_Args();
    Deal_With_Pointer();
    while 堆栈非空
        if stack.top == '('
            弹出堆栈;
            GetToken();
            Deal_With_Declarator();
        else
            将其弹出堆栈并打印;
}
```

主程序

```
void main()
{
    while(读取每一行)
    {
        Read_To_First_Identifier();
        Deal_With_Declarator();
    }
}
```

Step 2 增加功能:类型大小/存值类型

存值类型设计方案

使用一个全局的token数组，在变量类型输出时，按输出顺序将各个token存入数组中，遍历数组计算类型大小，输出存值类型
按变量类型分类如下:
(1)array: 如array(20,T)，则存值类型为T
(2)pointer: 如pointer(T)，则存值类型为T
(3)int: 存值类型为int
(4)其他:不输出

类型大小设计方案

TypeSize初始大小为1，遇到数字则与其相乘，遇到pointer则停止计算

Part2 增加功能:int i,j;型声明的分析+函数参数分析

int i,j;型声明的分析以及处理

主要思想：先分析前一个变量的定义声明，利用while循环，当读到‘,’时，继续下一个定义声明的分析；

```
init_declarator_list //初始声
    : init_declarator
    | init_declarator_list ',' init_declarator
    ;
```

我们的前缀分析，可以共用最前面的int/void，所有在存前缀的栈中，将栈顶指针指向最开始的int，也就是top=0，即可； 在while循环中的处理和之前的处理一致：

```
    while(latest.type==',')
        {
            printf("\n");
            S_top =0;                       //栈顶指针指向共同的部分，最开始的int/void的位置
            err = 0;
            Read_To_First_Identifier();  //前缀的分析
            Deal_With_Declarator();      //读到变量后的分析
            type_chaking();                  //类型检查函数
        }
```

函数参数分析

数据结构：

```
int preO_top;//输出参数的输出栈的指针
prestack[MAXLAYER];//备份栈
pretop[MAXLAYER];//备份栈顶指针
preO_top; //备份栈顶指针
para_num;//参数统计量
func_layer;//当前函数所在层次
para_P[MAXPARA]; //各参数在该行的位置指针
```

实现思路：

1.修改函数Deal_With_Function_Args()，保存该层次的栈状态和栈顶指针后清空当前工作栈，再将层次统计量func_layer递增，表示当前所在位置是第几层函数

2.输出提示信息“function(”,取下一个记号，若是’)’说明该函数无参数类型，输出”void”,否则依次分析各个参数并递增para_num，记录StrLine_P到para_P数组，直到碰到’)’为止

3.此时该层次已分析完，将func_layer递减，恢复上一层次的工作栈，输出“=>”等待进一步分析给出函数的返回值类型

实现细节

•实际上，main函数分析某一行时首先清零para_num，再多次调用Read_To_First_Identifier和Deal_With_Declarator，直到遇到分号，给出所有的标识符信息；

•此时的para_num=k即是该行的参数个数，且它们出现的位置已经依次记录在偏移量数组para_P中；循环k次，从para_P[i]处对参数i分析（调用Read_To_First_Parameter和Deal_With_Declarator）给出它的表达式。

•函数Read_To_First_Parameter()和Skip_First_Parameter():

辅助函数，类似Read_To_First_Identifier(),只是输出不同

Part3 增加功能: 类型检查

输入有错类型

```
        case 1:The length of Array must be digits:                      a[int/void
init_declarator_list] 或 a[int/void f(parameter_list)]
        case 2:The return of Functions can''t be Array!:                int/void
pointer/kong f(parameter_list)[declaration]
        case 3:The return of Functions can''t be Function!:             int/void
pointer/kong f(parameter_list)(parameter_list)
        case 4:Array of Function is not allowed!:
 a[constant_int](parameter_list)
```

数据利用

```
增加int err为全局变量，记录错误类型;
```

具体的类型检查函数

```
case1:在数组分析里面，读完'[',之后读到非数字，就记录错误类型等于1;
case2：在检查完函数之后，也就是在Deal_With_Declarator中分析完之后，要是读取的下一个是'['，就记录错误类型等于2;
case3：在检查完函数之后，也就是在Deal_With_Declarator中分析完之后，要是读取的下一个是' ('，就记录错误类型等于3;
case4：在数组分析里面，要是分析完']',读取到的下一个是‘（’，记录错误类型4;
```

&测试结果

测试文件

```
int *p;
int *pa[20][30];
int (**pp),*tip;
int (**ptr[20])[30];
int *f(int a[2],int b);
int (*f(int i,int *j))[20],i,j;
int a[b];
int *fa(int i)[20];
int af[20](int k);
```

运行结果

```
声明式:int *p;
p is type of: pointer(int)
TypeChecking...OK!
ValueType is:int
type size: 1


声明式:int *pa[20][30];
pa is type of: array(20,array(30,pointer(int)))
TypeChecking...OK!
ValueType is:array(30,pointer(int))
type size: 600


声明式:int (**pp);
pp is type of: pointer(pointer(int))
TypeChecking...OK!
ValueType is:pointer(int)
type size: 1


声明式:int (**ap[20])[30];
ap is type of: array(20,pointer(pointer(array(30,int))))
TypeChecking...OK!
ValueType is:pointer(pointer(array(30,int)))
type size: 20


声明式:int (*fpa(int i,int *j))[20];
fpa is type of: function(int X pointer(int)=>pointer(array(20,int)))
TypeChecking...OK!


parameter i is type of: int
parameter j is type of: pointer(int)

声明式:int * fa(int i)[20][30][40]; //type-error!
fa is type of: function(int=>array(20,array(30,array(40,pointer(int)))))
TypeChecking...
Error 2:The return of Functions can't be Array!
```

```
声明式:int af[20](int k); // type-error!
af is type of: array(20,function(int=>int))
TypeChecking...
Error 4:Array of Function is not allowed!
error!


parameter k is type of: int

声明式:void (*(*paa)[10])(int a);
paa is type of: pointer(array(10,pointer(function(int=>void))))
TypeChecking...OK!
ValueType is:array(10,pointer(function(int=>void)))
type size: 1

parameter a is type of: int

声明式:void (*afp[10]) (int b);
afp is type of: array(10,pointer(function(int=>void)))
TypeChecking...OK!
ValueType is:pointer(function(int=>void))
type size: 10

parameter b is type of: int

声明式:void (*afp[10]) (int b)[20];
afp is type of: array(10,pointer(function(int=>array(20,void))))
TypeChecking...
Error 2:The return of Functions can't be Array!
error!


parameter b is type of: int
```

```
声明式:int (*(*(*pg())(int x))[20])(int *y);
pg is type of: function(void=>pointer(function(int=>pointer(array(20,pointer(
function(pointer(int)=>int)))))))
TypeChecking...OK!


parameter x is type of: int
parameter y is type of: pointer(int)

声明式:int (*p(int * s,int (*t)(int *m, int n, int (*l())[20]),int k[10]))[10
][20];p is type of: function(pointer(int) X pointer(function(pointer(int) X i
nt X function(void=>pointer(array(20,int)))=>int)) X array(10,int)=>pointer(a
rray(10,array(20,int))))
TypeChecking...OK!


parameter s is type of: pointer(int)
parameter t is type of: pointer(function(pointer(int) X int X function(void=>
pointer(array(20,int)))=>int))
parameter m is type of: pointer(int)
parameter n is type of: int
parameter l is type of: function(void=>pointer(array(20,int)))
parameter k is type of: array(10,int)


--------------------------------
Process exited after 0.4232 seconds with return value 0
请按任意键继续. . .
```

&源代码

CDecl.cpp文件