



# 静态类型检查

李诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心  
计算机科学与技术学院

2025年12月08日



## □ 变量的类型

- 限定了变量在程序执行期间的取值范围和存储空间消耗

## □ 类型化的语言(typed language)

- 变量都被给定类型的语言
- 表达式、语句等程序构造的类型都可以静态确定，运行时不需要额外的操作

## □ 未类型化的语言(untyped language)

- 不限制变量值范围的语言,如JavaScript、Perl

# 类型的两个典型应用



## □ 静态的语义分析

### ■ 类型检查

利用逻辑规则分析运算分量的类型与运算符预期是否匹配？

## □ 中间代码生成

- 声明语句的翻译
- 数组寻址的翻译
- 类型转换

- 通过声明语句收集变量或函数的类型
- 计算所占存储空间
- 分配相对地址
- 类型转换适配指令选择

# 主要内容



## □ 类型表达式

- 类型的结构

层次一：形式化描述类型结构

## □ 类型等价

- 结构等价和名字等价

层次二：判定两个类型相同的依据

## □ 类型检查

- 语法制导翻译方案实现
- 函数和算符的重载

层次三：定义一组逻辑规则检查语句或者表达式中是否存在类型错误

## □ 其他知识点

# 类型表达式 (Type expression)



- 类型可以是语法的一部分，因此也是结构的

考慮以下文法，D代表声明语句，S代表一般语句

$$P \rightarrow D ; S$$

$$D \rightarrow D ; D \mid \text{id} : T$$

$$T \rightarrow \text{boolean} \mid \text{integer} \mid \text{array [num] of } T \mid \uparrow T \mid T \xrightarrow{\cdot} T$$

# 类型表达式 (Type expression)



- 类型可以是语法的一部分，因此也是结构的

考虑以下文法，D代表声明语句，S代表一般语句

$$P \rightarrow D ; S$$

$$D \rightarrow D ; D \mid id : T$$

$$T \rightarrow \text{boolean} \mid \text{integer} \mid \text{array } [\text{num }] \text{ of } T \mid \uparrow T \mid T \text{ '}' \rightarrow ' T$$

数组

指针

函数

基本类型

复杂且可组合的类型

# 类型表达式 (Type expression)



## □ 基本类型是类型表达式

- *integer*
- *real*
- *char*
- *boolean*
- *type\_error* // 出错类型
- *void* // 无类型

在类型检查中  
传递错误

语句的类型

# 类型表达式 (Type expression)



- 基本类型是类型表达式
- 可为类型表达式命名, 类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子 *array*
  - 指针类型构造算子 *pointer*
  - 笛卡尔乘积类型构造算子  $\times$
  - 函数类型构造算子  $\rightarrow$
  - 记录类型构造算子 *record*
    - 若有标识符  $N_1, N_2, \dots, N_n$  以及对应的类型表达式  $T_1, T_2, \dots, T_n$ , 则  $record((N_1 \times T_1) \times (N_2 \times T_2) \times \dots \times (N_n \times T_n))$  也是类型表达式

# 主要内容



## □ 类型表达式

- 类型的结构

层次一：形式化描述类  
型结构

## □ 类型等价

- 结构等价和名字等价

层次二：判定两个类型  
相同的依据

## □ 类型检查

- 语法制导翻译方案实现
- 函数和算符的重载

层次三：定义一组逻辑  
规则检查语句或者表达  
式中是否存在类型错误

## □ 其他知识点

# 结构等价 (Structural equivalence)



## □ 两个类型表达式完全相同 (当无类型名时)

- 类型表达式树一样
- 相同的类型构造符作用于相同的子表达式

**type link = ↑cell;**

**var next : link;**

**last : link;**

**p : ↑cell;**

**q, r : ↑cell;**

# 结构等价 (Structural equivalence)



- 两个类型表达式完全相同 (当无类型名时)
- 有类型名时, 用它们所定义的类型表达式代换它们, 所得表达式完全相同 (类型定义无环时)

```
type link = ↑cell;
```

```
var next : link;
```

```
last : link;
```

```
p : ↑cell;
```

```
q, r : ↑cell;
```

这里隐藏了递归检查, 因此暂时不考虑有环的情况

next, last, p, q和r结构等价

# 结构等价 (Structural equivalence)



```
function sequiv(s, t) : boolean
```

```
{if s 和 t 是相同的基本类型 then
```

```
    return true
```

```
else if s == array(s1, s2) and t == array(t1, t2) then
```

```
    return sequiv(s1, t1) and sequiv(s2, t2)
```

```
else if s == s1 × s2 and t == t1 × t2 then
```

```
    return sequiv(s1, t1) and sequiv(s2, t2)
```

```
else if s == pointer(s1) and t == pointer(t1) then
```

```
    return sequiv(s1, t1)
```

```
else if s == s1 → s2 and t == t1 → t2 then
```

```
    return sequiv(s1, t1) and sequiv(s2, t2)
```

```
else return false
```

```
}
```

# 名等价 (name equivalence)



- 把每个类型名看成是一个可区别的类型
- 两个类型表达式名字等价当且仅当
  - 它们是相同的基本类型
  - 不进行名字代换就能结构等价

# 名等价 (name equivalence)



- 把每个类型名看成是一个可区别的类型
- 两个类型表达式名字等价当且仅当

- 它们是相同的基本类型
- 不进行名字代换就能结构等价

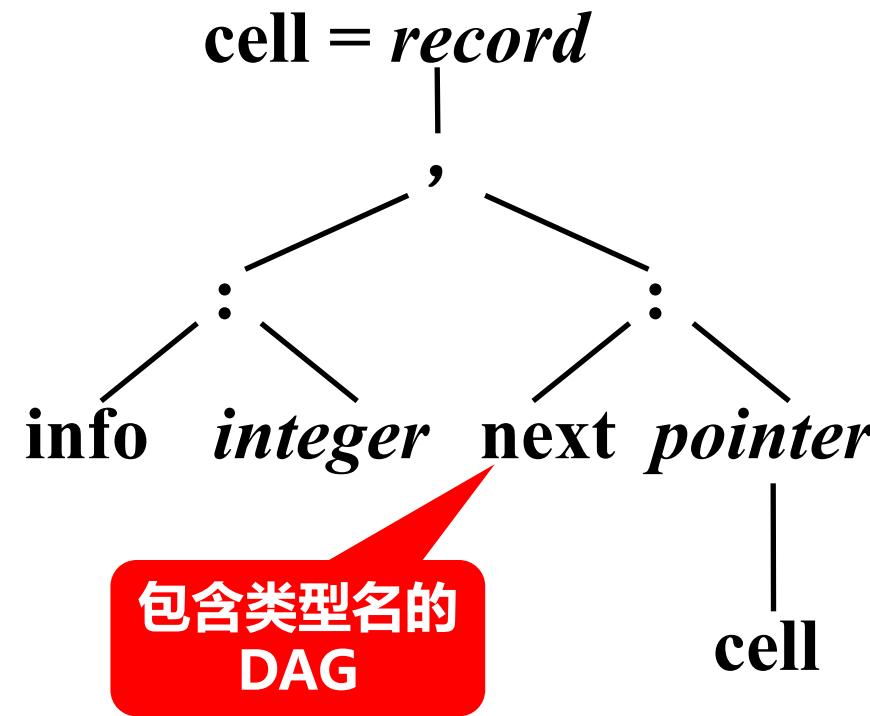
|                              |                |               |
|------------------------------|----------------|---------------|
| type link = $\uparrow$ cell; | 类型表达式          |               |
| var next : link;             | link           | next和last名字等价 |
| last : link;                 | link           | p, q和r名字等价    |
| p : $\uparrow$ cell;         | pointer (cell) |               |
| q, r : $\uparrow$ cell;      | pointer (cell) |               |

# 递归定义的类型



- ❑ Where: Linked Lists, Trees, etc
- ❑ How: records containing pointers to similar records

```
type link = ↑ cell ;  
cell = record  
    info : integer ;  
    next : link  
end;
```

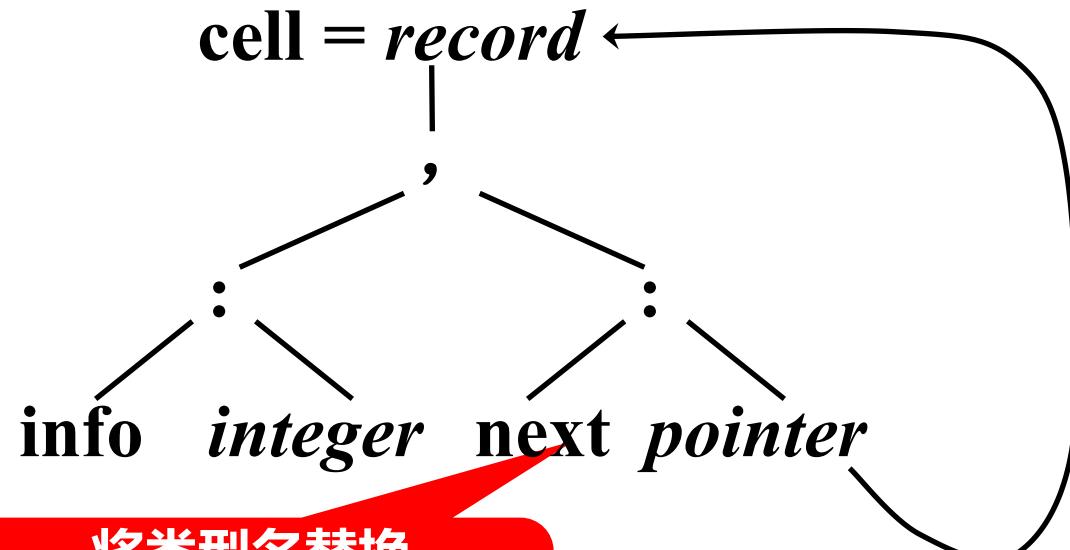


# 递归定义的类型



- ❑ Where: Linked Lists, Trees, etc
- ❑ How: records containing pointers to similar records

```
type link = ↑ cell ;  
cell = record  
    info : integer ;  
    next : link  
end;
```

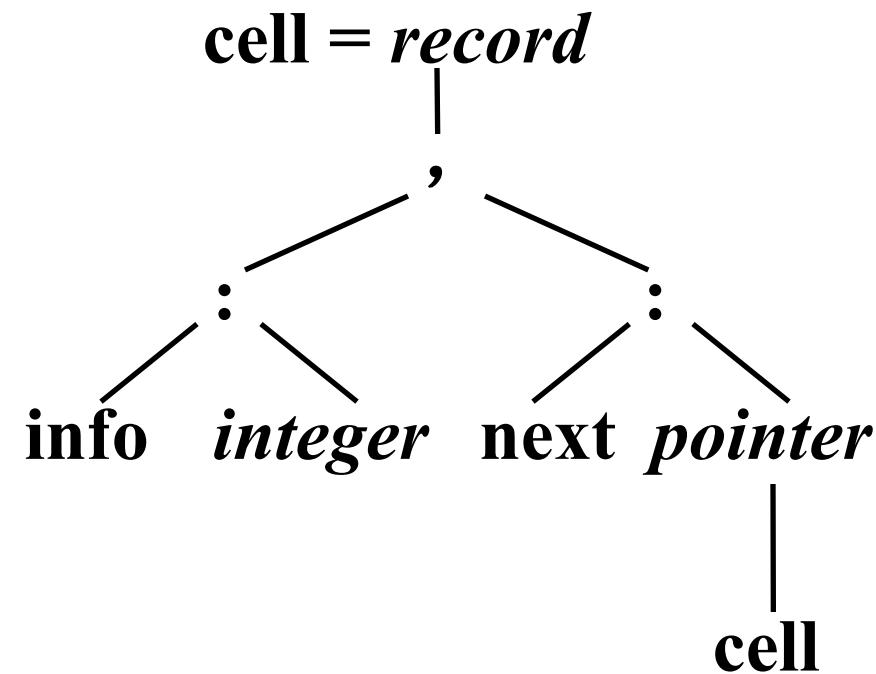


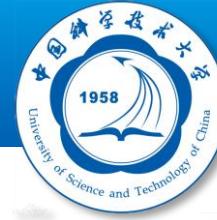
将类型名替换  
引入环，结构等价判定  
有可能不终止

# C语言中的递归定义的类型



- C语言对除记录（结构体）以外的所有类型使用结构等价，而记录类型用的是名字等价，以避免类型图中的环





在X86/Linux机器上，编译器报告最后一行有错误：

**incompatible types in return**

```
typedef int A1[10];           | A2 *fun1( ) {  
typedef int A2[10];           |     return(&a);  
A1 a;                         | }  
typedef struct {int i;}S1;      | S2 fun2( ) {  
typedef struct {int i;}S2;      |     return(s);  
S1 s;                          | }
```

在C语言中，数组和结构体都是构造类型，为什么上面第2个函数有类型错误，而第1个函数却没有？

# 主要内容



## □类型表达式

- 类型的结构

层次一：形式化描述类  
型结构

## □类型等价

- 结构等价和名字等价

层次二：判定两个类型  
相同的依据

## □类型检查

- 语法制导翻译方案实现
- 函数和算符的重载

层次三：定义一组逻辑  
规则检查语句或者表达  
式中是否存在类型错误

## □其他知识点

# 一个简单的语言



$P \rightarrow D; S$

$D \rightarrow D; D \mid id : T$

$T \rightarrow \text{boolean} \mid \text{integer} \mid \text{array [num ] of } T \mid$

$\uparrow T \mid T \xrightarrow{\quad} T$

$S \rightarrow id := E \mid \text{if } E \text{ then } S \mid \text{while } E \text{ do } S \mid S; S$

$E \rightarrow \text{truth} \mid \text{num} \mid id \mid E \text{ mod } E \mid E[E] \mid$

$E^\uparrow \mid E(E)$

例

$i : \text{integer};$

$j : \text{integer};$

$j := i \bmod 2000$

# 类型检查——声明语句



$D \rightarrow D; D$

$D \rightarrow \text{id} : T \quad \{ \text{addtype}(\text{id.entry}, T.type) \}$

*addtype*: 把类型信息填入符号表

# 类型检查——声明语句



$D \rightarrow D; D$

$D \rightarrow \text{id} : T \quad \{ \text{addtype}(\text{id.entry}, T.type) \}$

$T \rightarrow \text{boolean} \quad \{ T.type = \text{boolean} \}$

$T \rightarrow \text{integer} \quad \{ T.type = \text{integer} \}$

$T \rightarrow \uparrow T_1 \quad \{ T.type = \text{pointer}(T_1.type) \}$

# 类型检查——声明语句



$D \rightarrow D; D$

$D \rightarrow \text{id} : T \quad \{ \text{addtype}(\text{id.entry}, T.type) \}$

$T \rightarrow \text{boolean} \quad \{ T.type = \text{boolean} \}$

$T \rightarrow \text{integer} \quad \{ T.type = \text{integer} \}$

$T \rightarrow \uparrow T_1 \quad \{ T.type = \text{pointer}(T_1.type) \}$

$T \rightarrow \text{array} [\text{num}] \text{ of } T_1$

$\{ T.type = \text{array}(\text{num.val}, T_1.type) \}$

# 类型检查——声明语句



$D \rightarrow D; D$

$D \rightarrow \text{id} : T \quad \{ \text{addtype}(\text{id.entry}, T.type) \}$

$T \rightarrow \text{boolean} \quad \{ T.type = \text{boolean} \}$

$T \rightarrow \text{integer} \quad \{ T.type = \text{integer} \}$

$T \rightarrow \uparrow T_1 \quad \{ T.type = \text{pointer}(T_1.type) \}$

$T \rightarrow \text{array} [\text{num}] \text{ of } T_1$

$\{ T.type = \text{array}(\text{num.val}, T_1.type) \}$

$T \rightarrow T_1 \rightarrow T_2 \quad \{ T.type = T_1.type \rightarrow T_2.type \}$

# 类型检查——表达式



- |                              |   |
|------------------------------|---|
| $E \rightarrow \text{truth}$ | $\{E.\text{type} = \text{boolean}\}$          |
| $E \rightarrow \text{num}$   | $\{E.\text{type} = \text{integer}\}$          |
| $E \rightarrow \text{id}$    | $\{E.\text{type} = \text{lookup(id.entry)}\}$ |

# 类型检查——表达式



$E \rightarrow \text{truth}$        $\{E.\text{type} = \text{boolean}\}$

$E \rightarrow \text{num}$        $\{E.\text{type} = \text{integer}\}$

$E \rightarrow \text{id}$        $\{E.\text{type} = \text{lookup(id.entry)}\}$

$E \rightarrow E_1 \bmod E_2$

$\{E.\text{type} = \text{if } E_1.\text{type} == \text{integer} \text{ and}$   
 $E_2.\text{type} == \text{integer} \text{ then } \text{integer}$   
 $\text{else } \text{type\_error}\}$

# 类型检查——表达式


$$E \rightarrow E_1 [E_2] \{ E.type = \text{if } E_2.type == \text{integer} \text{ and } \\ E_1.type == \text{array}(s, t) \text{ then } t \\ \text{else } type\_error \}$$

# 类型检查——表达式


$$E \rightarrow E_1 [E_2] \{ E.type = \text{if } E_2.type == \text{integer} \text{ and } \\ E_1.type == \text{array}(s, t) \text{ then } t \\ \text{else type\_error} \}$$
$$E \rightarrow E_1 \uparrow \{ E.type = \text{if } E_1.type == \text{pointer}(t) \text{ then } t \\ \text{else type\_error} \}$$

# 类型检查——表达式


$$E \rightarrow E_1 [E_2] \{E.type = \text{if } E_2.type == \text{integer} \text{ and } \\ E_1.type == \text{array}(s, t) \text{ then } t \\ \text{else type\_error}\}$$
$$E \rightarrow E_1 \uparrow \{E.type = \text{if } E_1.type == \text{pointer}(t) \text{ then } t \\ \text{else type\_error}\}$$
$$E \rightarrow E_1 (E_2) \{E.type = \text{if } E_2.type == s \text{ and } \\ E_1.type == s \rightarrow t \text{ then } t \\ \text{else type\_error}\}$$

# 类型检查——语句



$S \rightarrow \text{id} := E \{ \text{if } (\text{id.type} == E.type \text{ && } E.type \in \{\text{boolean, integer}\}) \text{ } S.type = \text{void},$   
 $\text{else } S.type = \text{type\_error}; \}$

# 类型检查——语句



$S \rightarrow \text{id} := E \{ \text{ if } (\text{id.type} == E.type \&\& E.type \in \{\text{boolean, integer}\}) \text{ } S.type = \text{void};$   
 $\text{ else } S.type = \text{type\_error}; \}$

$S \rightarrow \text{if } E \text{ then } S_1 \{ S.type = \text{if } E.type == \text{boolean}$   
 $\text{ then } S_1.type$   
 $\text{ else } \text{type\_error} \}$

# 类型检查——语句



$S \rightarrow \text{while } E \text{ do } S_1$

$\{S.\text{type} = \text{if } E.\text{type} == \text{boolean} \text{ then } S_1.\text{type}$   
 $\text{else } \text{type\_error}\}$

# 类型检查——语句



$S \rightarrow \text{while } E \text{ do } S_1$

$\{S.\text{type} = \text{if } E.\text{type} == \text{boolean} \text{ then } S_1.\text{type}$   
 $\quad \quad \quad \text{else } \text{type\_error}\}$

$S \rightarrow S_1; S_2$

$\{S.\text{type} = \text{if } S_1.\text{type} == \text{void} \text{ and}$   
 $\quad \quad \quad S_2.\text{type} == \text{void} \text{ then } \text{void}$   
 $\quad \quad \quad \text{else } \text{type\_error}\}$

# 类型检查——程序



$P \rightarrow D; S$

$\{P. type = \text{if } S.type == \text{void} \text{ then void}$   
 $\text{else type\_error}\}$

# 主要内容



## □类型表达式

- 类型的结构

层次一：形式化描述类  
型结构

## □类型等价

- 结构等价和名字等价

层次二：判定两个类型  
相同的依据

## □类型检查

- 语法制导翻译方案实现
- 函数和算符的重载

层次三：定义一组逻辑  
规则检查语句或者表达  
式中是否存在类型错误

## □其他知识点



## □ 重载符号

- 有多个含义，但在每个引用点的含义都是唯一的

## □ 例如：

- 加法算符+可用于不同类型， "+" 是多个函数的名字，而不是一个多态函数的名字

## □ 重载的消除

- 在重载符号的引用点，其含义能确定到唯一



## □ 例 Ada语言的声明：

```
function "*" (i, j : integer) return complex;
```

```
function "*" (x, y : complex) return complex;
```

使得算符\*重载，可能的类型包括：

integer × integer → integer --这是预定义的类型

integer × integer → complex

2 \* (3 \* 5)

complex × complex → complex

(3 \* 5) \* z z是复型



## □ 以函数应用为例，考虑类型检查

- 在每个表达式都有唯一的类型时，函数应用的类型检查是：

$E \rightarrow E_1(E_2) \{ E.type = \text{if } E_2.type == s \text{ and } E_1.type == s \rightarrow t \text{ then } t \text{ else type\_error } \}$

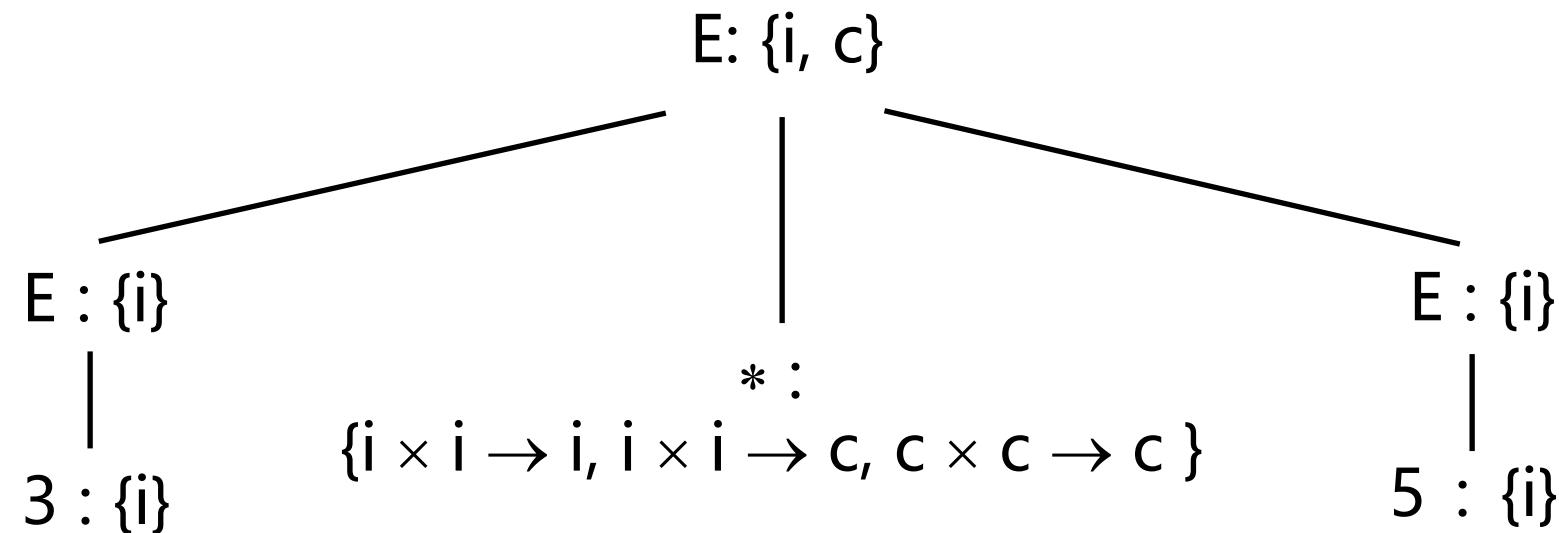
- 确定表达式可能类型的集合（类型可能不唯一）

| 产生式                      | 语义规则   |
|--------------------------|--|
| $E' \rightarrow E$       | $E'.types = E.types$   |
| $E \rightarrow id$       | $E.types = \text{lookup}(id.entry)$  |
| $E \rightarrow E_1(E_2)$ | $E.types = \{t \mid E_2.types \text{ 中存在一个 } s, \text{ 使得 } s \rightarrow t \text{ 属于 } E_1.types\}$ |

# 子表达式的可能类型集合



## 例：表达式 $3 * 5$ 可能的类型集合



# 主要内容



## □ 类型表达式

- 类型的结构

层次一：形式化描述类型结构

## □ 类型等价

- 结构等价和名字等价

层次二：判定两个类型相同的依据

## □ 类型检查

- 语法制导翻译方案实现
- 函数和算符的重载

层次三：定义一组逻辑规则检查语句或者表达式中是否存在类型错误

## □ 其他知识点



## □ 控制流检查

- 控制流语句必须使控制转移到合法的地方。
- 例如，在C语言中**break**语句使控制跳离包括该语句的最小**while**、**for**或者**switch**语句；否则就报错。

```
main() {  
    printf( "\n%d\n" ,gcd(4,12));  
    continue;  
}
```

编译时的报错如下：

example.c: In function ‘main’ :

example.c:3: continue statement not within a loop



## □ 上下文相关检查

- 标识符没有声明
- 标识符重复声明

## □ 唯一性检查

- **Switch**语句的分支常量表达式不能有重复
- 枚举类型的元素不能重复

## □ 编译时的唯一性检查的例子

```
main() {  
    int i;  
    switch(i){  
        case 10: printf( "%d\n" , 10); break;  
        case 20: printf( "%d\n" , 20); break;  
        case 10: printf( "%d\n" , 10); break;  
    }  
}
```

### ■ 编译时的报错如下：

```
switch.c: In function 'main' :  
switch.c:6: duplicate case value  
switch.c:4: this is the first entry for that value
```



# 一起努力 打造国产基础系统软件体系！

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心  
计算机科学与技术学院

2025年12月08日