类型检查规则

1. 基本类型体系

WhileDT 语言支持如下类型：

- `TBASIC: TShort | TTInt | TLong | TLongLong`
- `TPTR: TPtr(T)`

类型优先级：`TShort < TInt < TLong < TLongLong`

指针类型层级式结构：

```
TPtr(TShort)
TPtr(TInt)
TPtr(TPtr(TInt))...
```

---

2. 显式类型转换

WhileDT 中只允许：

- 基础类型向基础类型的转换
- 指针类型向指针类型的转换（不关心指针指向的类型，例如 `TInt* -> long*` 、 `TInt* -> TInt**` ）

不允许基础类型与指针类型之间的转换

3. 隐式类型转换

WhileDT 模仿 C 语言的整数与指针转换机制。

3.1. 不允许涉及指针类型的隐式转换（必须显式化指针相关的类型转换）

3.2. 常规算术转换

用于算术运算与比较运算，使两个操作数类型一致：

- 两个操作数提升到类型等级更高者 `TShort -> TInt -> TLong -> TLongLong`
- 运算返回结果取提升后的类型

3.3. 布尔上下文转换

用于 `&&` 、 `||` 、 `!` 以及 `CIF` 、 `CWHILE` 中的条件表达式：

- **逻辑运算不会进行整数提升或算术转换。**任意 `TBASIC` 类型非零视为真，零视为假，直接参与运算，无需隐式转换。
- 运算结果为 `TInt` 类型的右值表达式。即“Bool”类型的存储模型是 `TInt` 。但如上一条中所述，逻辑运算过程中，任意的基础类型都可以直接判断是否等于0，因而无需转换。

```
TInt x; long y;
if (x && y) then { skip } // 无需隐式转换
```

---

4. 表达式与运算符的类型规则

4.1. 算术运算符：加减乘除模 `+ - * / %`

1. 两操作数必须是**相同的基础整数类型** `TBASIC`
2. 可以应用常规算术转换，使得不满足第一条的操作数类型一致
3. 结果类型为转换后的类型

4.2. 比较运算符： `<` 、 `<=` 、 `>` 、 `>=`

1. 两操作数必须为**相同的基础整数类型** `TBASIC`
2. 可以应用常规算术转换，使得不满足第一条的操作数类型一致
3. 返回结果类型为 `TInt`

4.3. 比较运算符特例： `==` 、 `!=`

1. 两操作数必须为**相同的基础整数类型** `TBASIC` **或完全相同（递归嵌套相同）的指针类型**
2. 基础整数类型可以应用常规算术转换，使得不满足第一条的操作数类型一致；指针类型必须要求完全一致
3. 返回结果类型为 `TInt`

4.4. 二元逻辑运算符：与或 `&&` 、 `||`

1. 操作数是**基础整数类型** `TBASIC` （不要求类型相同）
2. 无需隐式转换，任意基础类型可直接参与运算
3. 返回类型为 `TInt`

4.5. 一元运算符

`-x` （算术负号）

- 操作数为**基础整数类型** `TBASIC`
- 结果类型与操作数类型一致

`!x`（逻辑取反）

- 操作数为**基础整数类型** `TBASIC`
- 结果类型为 `TInt`

4.6. `&`（取地址运算符）

- 操作数必须为**左值表达式**（即有且仅有的两类 `EVAR` 、 `EDEREF` ）
- 结果类型为表达式类型外套一层指针： `TPtr(expr_type)`

4.7. `*`（解引用）

- 操作时必须为**指针类型** `TPTR`
- 结果类型为表达式类型去掉最外层一层指针： `TPtr(T) -> T`

---

5. 赋值语句类型规则

两种赋值形式：

```
1   int x; int *p;
2   x = e;             // ASGN
3   *p = e             // ASGNDEREF
```

5.1. 变量赋值

- 要求左右类型一致，或通过合法的显式/隐式类型转换后一致。
- **基础类型**：隐式转换支持，可以进行常规算术转换
- **指针类型**：无隐式转换，必须类型完全一致或显式转换

5.2. 解引用赋值

```
1   *e1 = e2
```

- `e1` 必须为**指针类型** `TPTR`
- `*e1` 类型必须与 `e2` 类型一致，或通过合法的显式/隐式转换后类型一致

---

6. 变量与作用域规则

- 每个作用域维持自己的类型环境 `struct VarTypeEnv` ，有两个字段 `map <string, VarTypes> vartype_tables; VarTypeEnv* parent`
- 变量声明时加入当前作用域，记录对应变量名及其类型
- 查找变量时先查局部当前作用域，未找到再查父级作用域，直到全局作用域；都未找到则该变量未声明或已结束生命周期
- `CIF` 的两个分支和 `CWHILE` 的循环体会各自创建新的子作用域

7. 类型分析方法

7.1. 递归分析语法树上的每个节点：

`checkexpr` 检查表达式是否合法，合法则返回表达式的类型

`checkcmd` 检查命令是否合法（满足所有类型检查规则）

```
1   VarType checkexpr(struct Expr *e, struct VarTypeEnv *env);
2   void checkcmdlicit(struct Cmd *c, struct VarTypeEnv *env);
```

从 `cmd` 进入 `expr` 的类型计算，每一个表达式最终都会还原到 `ECONST` 或 `EVAR`

- `ECONST` ：根据数值范围进行类型判定：首先，若该数值位于 `TInt` 类型的取值范围内，则默认将其设定为 `TInt` 类型；其次，若不满足上述条件，则进一步检查该数值是否落在 `TLongLong` 类型的取值区间内，若是，则将其设为 `TLongLong` 类型；最后，如果该数值超出以上两种类型的取值范围，则视为发生了**常数溢出**，程序将报错并终止执行。
- `EVAR` ：先查询当前可见的作用域中对应变量名的类型，再参与类型计算。

7.2. 隐式类型转换分析的体现：

类型分析的过程中，根据隐式转换规则，同步动态修改AST。即将隐式类型转换显式化体现在修改后的AST上