

姓名：刘权祥

学号：2019300414

测试说明文档

测试说明文档

语义检查原理简介

测试说明

命名冲突

- 测试用例1——不冲突的简单情况（所有ID都不一样）
- 测试用例2——不冲突的复杂情况（同名ID在不同局部作用域）
- 测试用例3——同一个struct空间下不能有同名变量
- 测试用例4——同一个module空间下不能有同名struct
- 测试用例5——和上级作用域名字冲突
- 测试用例6——与上级作用域名字冲突的复杂情况

未定义即使用

- 测试用例1——正常情况1
- 测试用例2——正常情况2
- 测试用例3——未定义即使用的情况1
- 测试用例4——未定义即使用情况2

字面量类型检查

- 测试用例1——表达式类型判定1-相同类型相加
- 测试用例2——表达式类型判定2-类型兼容
- 测试用例3——表达式类型判定3-对不支持类型报错
- 测试用例4——正常的声明情况
- 测试用例5——类型冲突情况1
- 测试用例6——类型冲突情况2
- 测试用例7——类型冲突情况3
- 测试用例8——数组1-只有声明没有赋值
- 测试用例9——数组2-有赋值的正常情况
- 测试用例10——数组3-有赋值但是数量不够
- 测试用例11——数组4-有赋值但是数量过多
- 测试用例12——数组5-有赋值但类型不统一
- 测试用例13——对最大值的检查

语义检查原理简介

- 对于命名冲突和未定义即使用：

我设计了 `SymbolTable` 类来保存符号表。在编译器遍历抽象语法树的同时，维护一个**根符号表**和**当前符号表**。

每当进入新的作用域时：为新作用域构建新的符号表，并且设置当前符号表为新的符号表

每当从作用域退出时：设置当前符号表为退出的作用域的符号表。

每当遇到新的声明时：

- 当前符号表检查是否存在未定义即使用的情况；
- 并且检查新的ID是否存在冲突的情况；

另外：我额外实现了**检查ID是否与上级作用域冲突的情况**，比如局部作用域使用了和全局作用域相同ID的情况，**这种情况应该有警告**，防止编程人员错误使用变量。

- 对于字面量类型检查：

我继承了实验一设计的抽象语法树类，为Literal、Unary_expr等构建了专门的抽象语法树，在抽象语法树中写入了**属性文法**，用于做类型检查。

另外：对于数组类型，我实现了检查索引是否正确，检查给数组赋值的时候是否超出了数组的最大长度。

测试说明

命名冲突

测试用例1——不冲突的简单情况（所有ID都不一样）

源代码：

```
module space{
    struct A{
        short i1=10;
    };
};
```

测试结果：（正常就输出符号表，否则报错）



```
main.py × MIDLVisitor.py × SymbolTable.py × ASTree.py × test.idl × module_struct.idl ×
1  module space{
2      struct A{
3          short i1=10;
4      };
5  };
6
运行: main ×
/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
=====显示符号表=====
root
    space
module space
    A
struct A
    i1
=====原始输入=====
module space{
    struct A{
        short i1=10;
    };
};
```

测试是否通过：通过

测试用例2——不冲突的复杂情况（同名ID在不同局部作用域）

源代码：

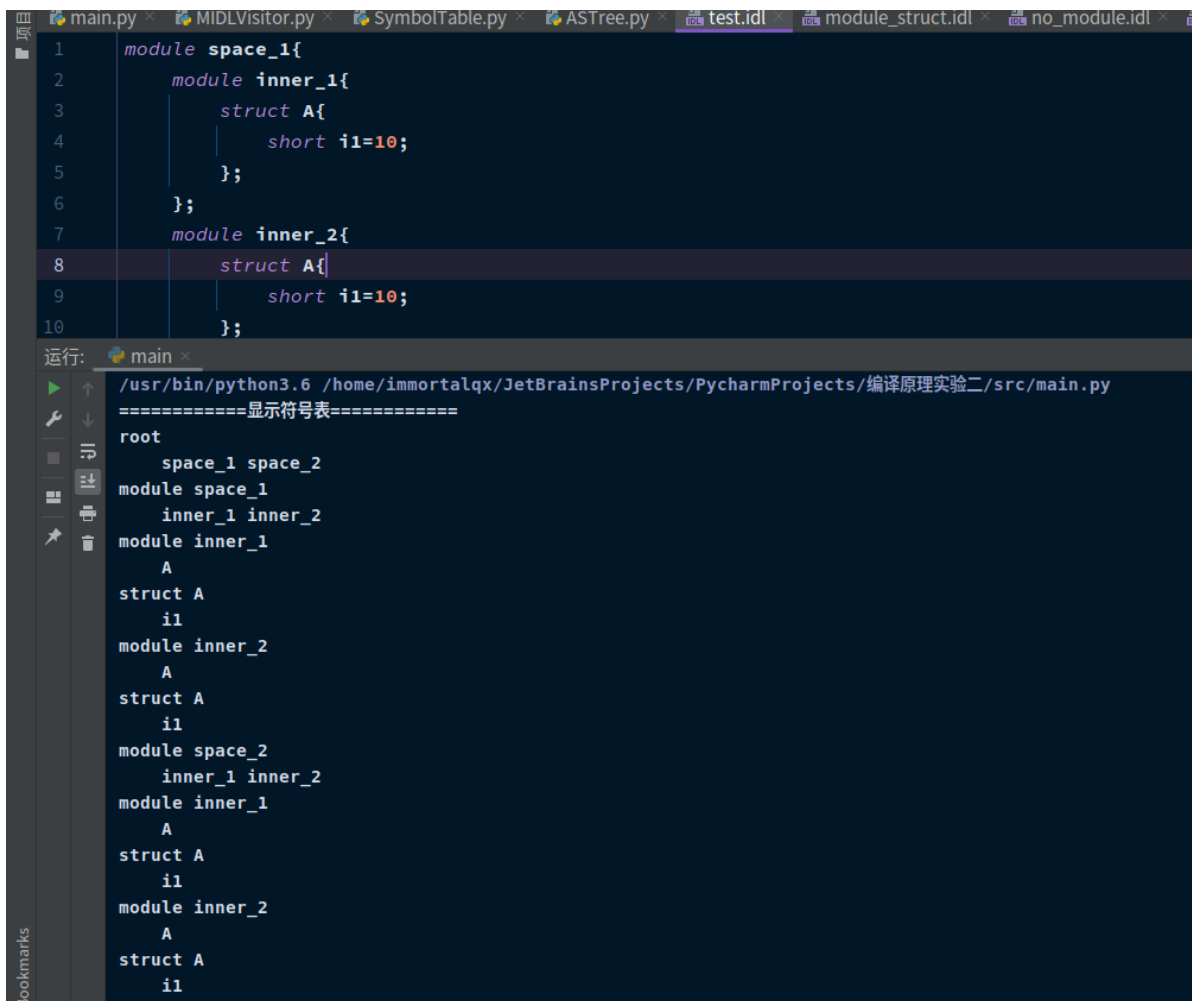
```
module space_1{
    module inner_1{
        module A{
            struct B;
        };
    };
    module inner_2{
        struct A{
            short i1=10;
        };
    };
};
```

```

};
};
module space_2{
    module inner_1{
        struct A{
            short i1=10;
        };
    };
    module inner_2{
        struct A{
            short i1=10;
        };
    };
};
};

```

测试结果：（正常就输出符号表，否则报错）



The screenshot shows an IDE with a code editor and a console window. The code editor displays the following C code:

```

1  module space_1{
2      module inner_1{
3          struct A{
4              short i1=10;
5          };
6      };
7      module inner_2{
8          struct A{
9              short i1=10;
10         };

```

The console window shows the output of a Python script that generates a symbol table. The output is as follows:

```

运行: /usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
=====显示符号表=====
root
    space_1 space_2
module space_1
    inner_1 inner_2
module inner_1
    A
    struct A
        i1
module inner_2
    A
    struct A
        i1
module space_2
    inner_1 inner_2
module inner_1
    A
    struct A
        i1
module inner_2
    A
    struct A
        i1

```

测试是否通过：通过

测试用例3——同一个struct空间下不能有同名变量

源代码：

```

module space{
    struct A{
        short i1=10;
        int8 i1;
    };
};

```

测试结果：（正常就输出符号表，否则报错）



The screenshot shows an IDE with several tabs: main.py, MIDLVisitor.py, SymbolTable.py, ASTree.py, test.idl, and module_struct.idl. The active file is test.idl, which contains the following code:

```
1 module space{
2     struct A{
3         short i1=10;
4         int8 i1;
5     };
6 };
```

Below the code editor, the '运行' (Run) tab is active, showing the command: `/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py`. The output shows an error: `错误: ID "i1" 在当前作用域多次定义!` (Error: ID "i1" is defined multiple times in the current scope!). Below the error, it says `进程已结束,退出代码255` (Process ended, exit code 255).

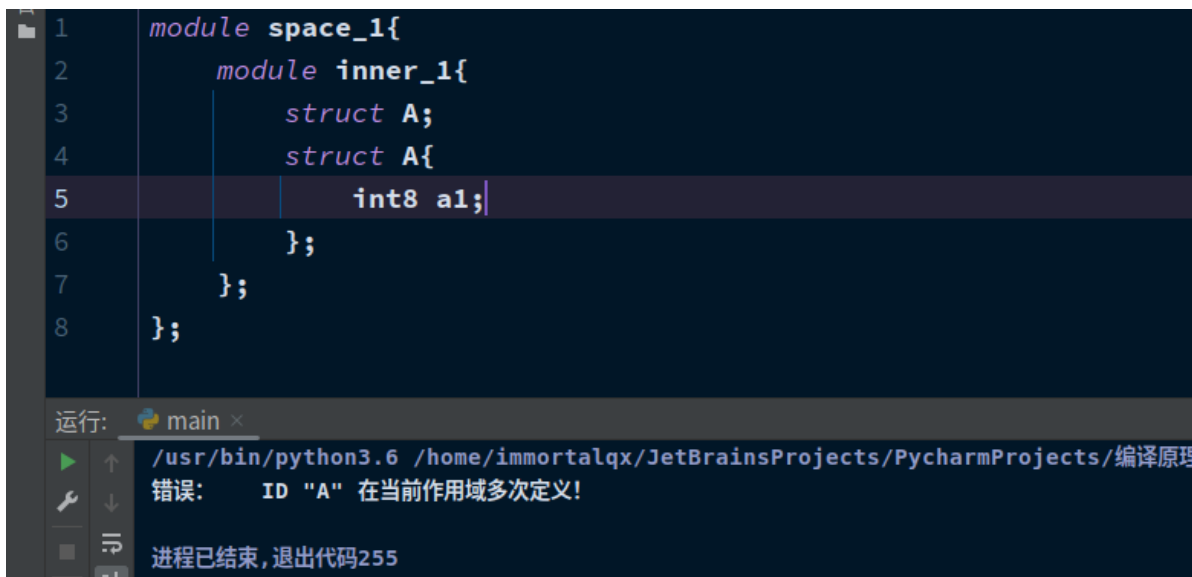
测试是否通过：通过

测试用例4——同一个module空间下不能有同名struct

源代码：

```
module space_1{
    module inner_1{
        struct A;
        struct A{
            int8 a1;
        };
    };
};
```

测试结果：（正常就输出符号表，否则报错）



The screenshot shows an IDE with the same tabs as before. The active file is test.idl, which contains the following code:

```
1 module space_1{
2     module inner_1{
3         struct A;
4         struct A{
5             int8 a1;
6         };
7     };
8 };
```

Below the code editor, the '运行' (Run) tab is active, showing the command: `/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py`. The output shows an error: `错误: ID "A" 在当前作用域多次定义!` (Error: ID "A" is defined multiple times in the current scope!). Below the error, it says `进程已结束,退出代码255` (Process ended, exit code 255).

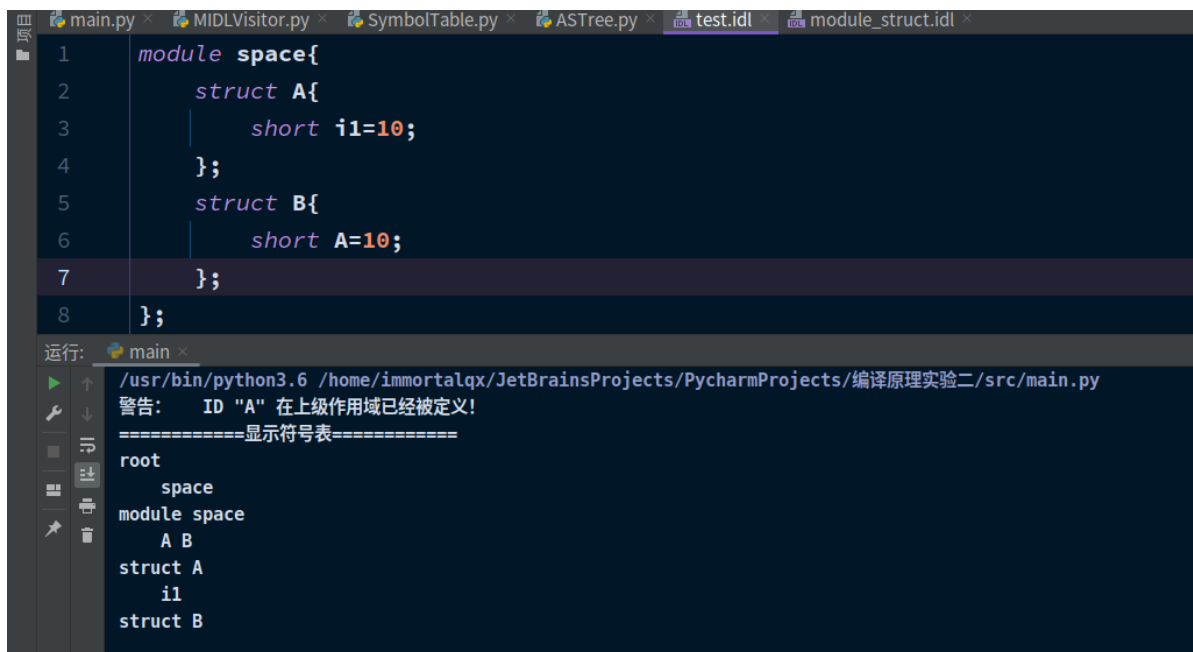
测试是否通过：通过

测试用例5——和上级作用域名字冲突

源代码：

```
module space{
    struct A{
        short i1=10;
    };
    struct B{
        short A=10;
    };
};
```

测试结果：（正常就输出符号表，否则报错）



```
1  module space{
2      struct A{
3          short i1=10;
4      };
5      struct B{
6          short A=10;
7      };
8  };

运行: main x
/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
警告: ID "A" 在上级作用域已经被定义!
=====显示符号表=====
root
  space
    module space
      A B
      struct A
        i1
      struct B
```

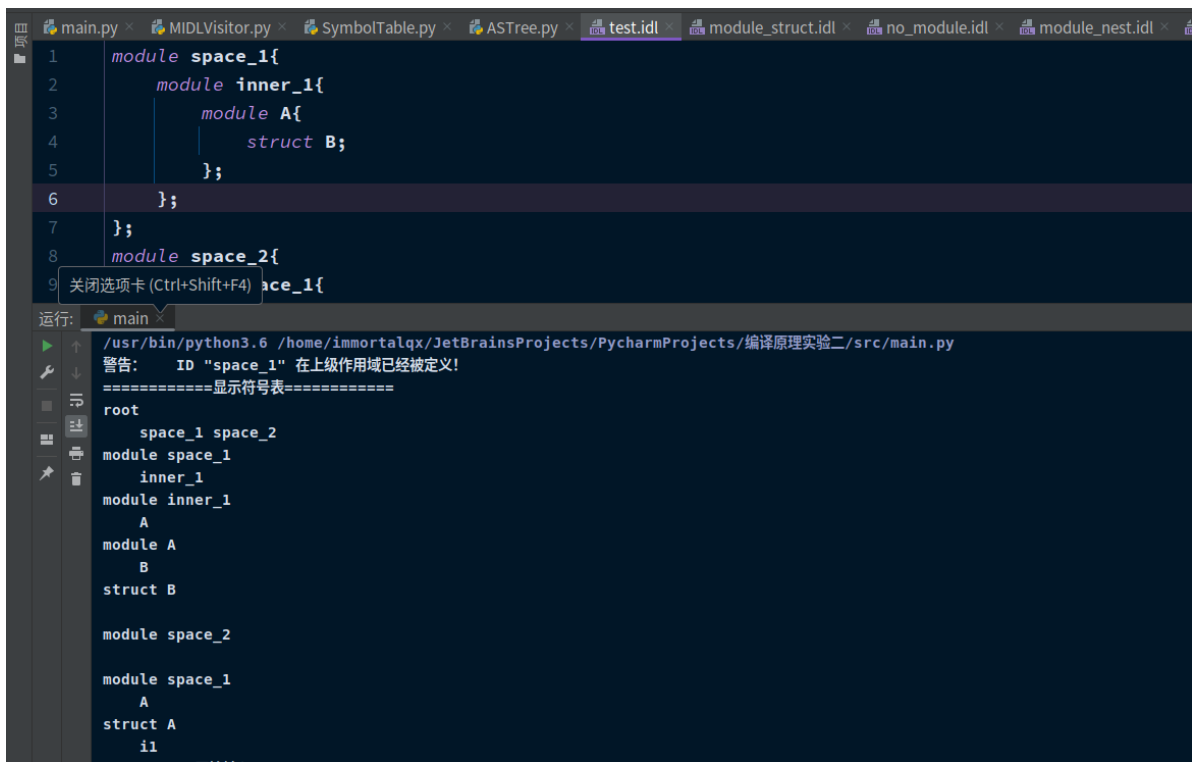
测试是否通过：通过

测试用例6——与上级作用域名字冲突的复杂情况

源代码：

```
module space_1{
    module inner_1{
        module A{
            struct B;
        };
    };
};
module space_2{
    module space_1{
        struct A{
            short i1=10;
        };
    };
};
```

测试结果：（正常就输出符号表，否则报错）



测试是否通过：通过

未定义即使用

测试用例1——正常情况1

源代码：

```

module space_1{
  module inner_1{
    module A{
      struct B;
    };
  };
};
module space_2{
  module inner{
    struct A{
      space_1::inner_1::A::B il=10;
    };
  };
};
```

测试结果：（正常就输出符号表，否则报错）

```
运行: main x
/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
=====显示符号表=====
root
    space_1 space_2
module space_1
    inner_1
module inner_1
    A
module A
    B
struct B

module space_2
    inner
module inner
    A
struct A
    i1
```

测试是否通过：通过

测试用例2——正常情况2

源代码：

```
module space_1{
    module inner_1{
        struct A;
    };
};
struct B{
    space_1::inner_1::A i1;
};
```

测试结果：（正常就输出符号表，否则报错）

```
运行: main x
/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
=====显示符号表=====
root
    space_1 B
module space_1
    inner_1
module inner_1
    A
struct A

struct B
    i1
```

测试是否通过：通过

测试用例3——未定义即使用的情况1

源代码：

```
module space_1{
    module inner_1{
        module A{
            struct B;
        };
    };
};
module space_2{
```

```

module inner{
    struct A{
        A::B i1=10;
    };
};
};

```

测试结果：（正常就输出符号表，否则报错）

The screenshot shows a PyCharm IDE with a C++ code file. The code defines a module `space_2` containing an inner module `inner` with a struct `A` that has a member `i1` of type `A::B` initialized to 10. The terminal window shows the command `/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py` and the output `A::B 未定义即使用!` (A::B undefined before use!), followed by `进程已结束, 退出代码255` (Process ended, exit code 255).

测试是否通过：通过

测试用例4——未定义即使用情况2

源代码：

```

module space_1{
    module inner_1{
        struct A;
    };
};
struct B{
    A i1;
};

```

测试结果：（正常就输出符号表，否则报错）

The screenshot shows a PyCharm IDE with a C++ code file. The code defines a module `space_1` containing an inner module `inner_1` with a struct `A`. Below `inner_1`, there is a struct `B` with a member `i1` of type `A`. The terminal window shows the command `/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py` and the output `A 未定义即使用!` (A undefined before use!), followed by `进程已结束, 退出代码255` (Process ended, exit code 255).

测试是否通过：通过

字面量类型检查

前三个是表达式类型判定，后面才是字面量类型检查

测试用例1——表达式类型判定1-相同类型相加

源代码：

```
struct A{
    int8 a1=1+2+3+4;
};
```

测试结果：

下面是抽象语法树，可以看到每一个Literal都是uint64类型（设置的整数默认类型），而最后推断出的Or_expr也是uint64类型。

```
Terminator ID
Terminator =
  Or_expr uint64
    Xor_expr uint64
      And_expr uint64
        Shift_expr uint64
          Add_expr uint64
            Mult_expr uint64
              Unary_expr uint64
                Terminator Literal uint64
              Terminator +
                Mult_expr uint64
                  Unary_expr uint64
                    Terminator Literal uint64
                Terminator +
                  Mult_expr uint64
                    Unary_expr uint64
                      Terminator Literal uint64
                Terminator +
                  Mult_expr uint64
                    Unary_expr uint64
                      Terminator Literal uint64
```

测试是否通过：通过

测试用例2——表达式类型判定2-类型兼容

源代码：

```
struct A{
    float a1=1+2+0.111;
};
```

测试结果：

从下面的抽象语法树可以看到，最后一个literal是float类型，而另外两个literal是uint64类型，而1+2+0.111 整个作为Add_expr就转化为了float类型

```

Terminator ID
  Terminator =
    Or_expr float
      Xor_expr float
        And_expr float
          Shift_expr float
            Add_expr float
              Mult_expr uint64
                Unary_expr uint64
                  Terminator Literal uint64
                Terminator +
                  Mult_expr uint64
                    Unary_expr uint64
                      Terminator Literal uint64
                  Terminator +
                    Mult_expr float
                      Unary_expr float
                        Terminator Literal float

```

测试是否通过：通过

测试用例3——表达式类型判定3-对不支持类型报错

源代码：

```

struct A{
    float a1=1+2+"0.111";
};

```

测试结果：

报错，并且说明为什么出了问题。

The screenshot shows a code editor with the following C++ code:

```

1 struct A{
2     float a1=1+2+"0.111";
3 };

```

Below the code editor, the console output shows the following error message:

```

运行: main
/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
错误: STRING类型不支持+、-运算!
进程已结束,退出代码255

```

测试是否通过：通过

测试用例4——正常的声明情况

源代码：

```
struct A{  
    float a1=2.1*1+2;  
};
```

测试结果：

对表达式类型进行了判断，判断为float类型，并且与a1类型兼容。

```
Terminator ID  
Terminator =  
    Or_expr float  
        Xor_expr float  
            And_expr float  
                Shift_expr float  
                    Add_expr float  
                        Mult_expr float  
                            Unary_expr float  
                                Terminator Literal float  
                                    Terminator *  
                                        Unary_expr uint64  
                                            Terminator Literal uint64  
Terminator +  
    Mult_expr uint64  
        Unary_expr uint64  
            Terminator Literal uint64
```

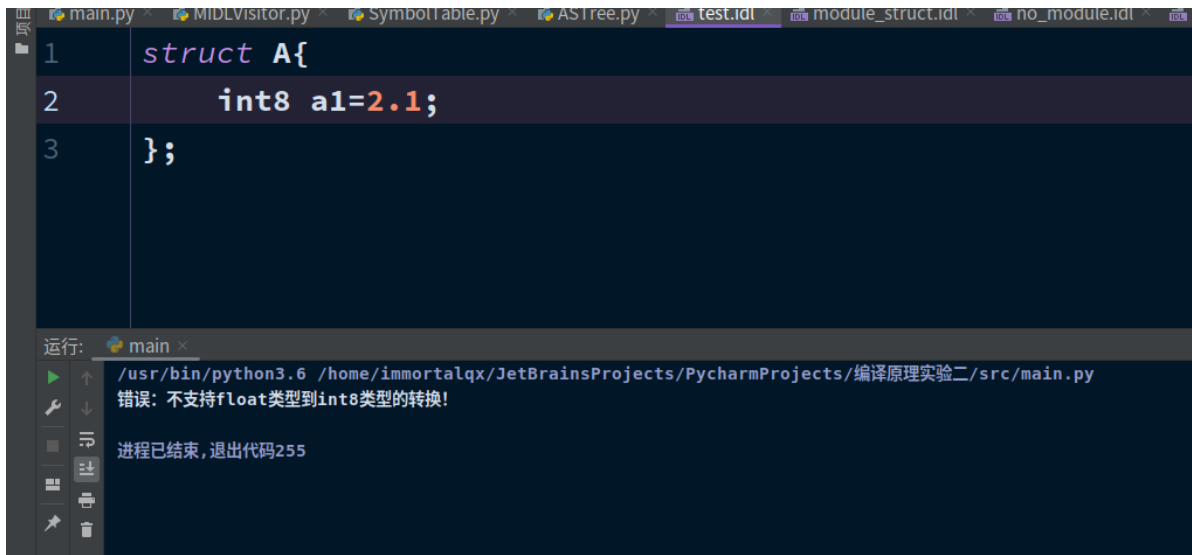
测试是否通过：通过

测试用例5——类型冲突情况1

源代码：

```
struct A{  
    int8 a1=2.1;  
};
```

测试结果：



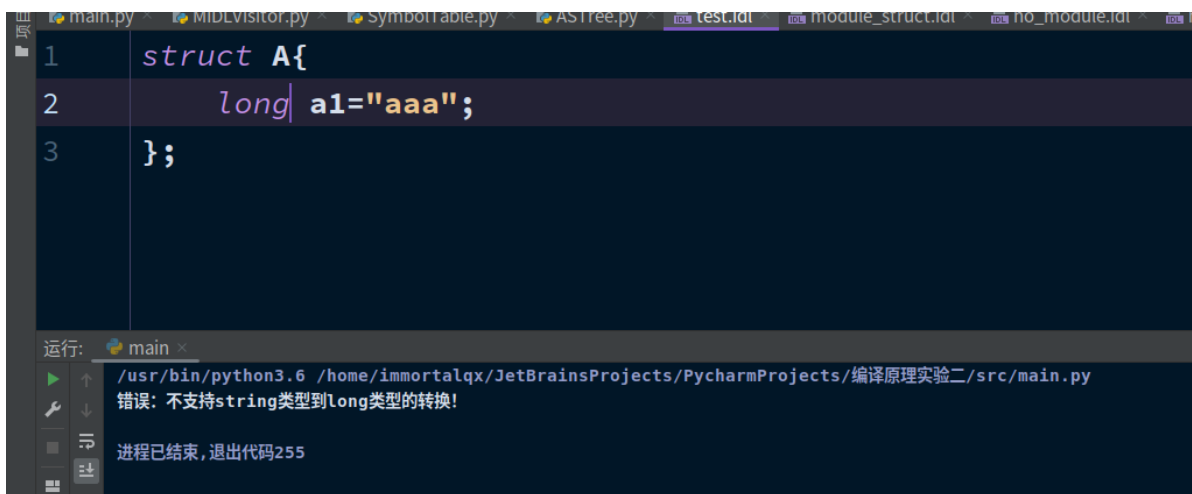
测试是否通过：通过

测试用例6——类型冲突情况2

源代码：

```
struct A{
    long a1="aaa";
};
```

测试结果：



测试是否通过：通过

测试用例7——类型冲突情况3

源代码：

```
struct A{
    long a1=false;
};
```

测试结果：

The screenshot shows an IDE with several tabs: main.py, MIDLVisitor.py, SymbolTable.py, ASTree.py, test.idl, module_struct.idl, no_module.idl, and mod. The active file is test.idl, which contains the following C code:

```
1 struct A{
2     long a1=false;
3 };
```

Below the code editor, a terminal window titled '运行: main' shows the command and error:

```
/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
错误: 不支持boolean类型到long类型的转换!
进程已结束,退出代码255
```

测试是否通过：通过

测试用例8——数组1-只有声明没有赋值

源代码：

```
struct A{
    long a1[5];
};
```

测试结果：

The screenshot shows the AST output for the code 'struct A{ long a1[5]; };'. The output is as follows:

```
Type_spec long
  Base_type_spec
    Integer_type
      Signed_int
        Terminator long
  Declarators uint64
    Declarator
      Array_declarator
        Terminator ID
        Or_expr uint64
          Xor_expr uint64
            And_expr uint64
              Shift_expr uint64
                Add_expr uint64
                  Mult_expr uint64
                    Unary_expr uint64
                      Terminator Literal uint64
```

测试是否通过：通过

测试用例9——数组2-有赋值的正常情况

源代码：

```
struct A{
    long a1[5]=[1,2,3,4,5];
};
```

测试结果：

```

Array_declarator
  Terminator ID
    Or_expr uint64
      Xor_expr uint64
        And_expr uint64
          Shift_expr uint64
            Add_expr uint64
              Mult_expr uint64
                Unary_expr uint64
                  Terminator Literal uint64
      Terminator =
        Exp_list uint64
          Or_expr uint64
            Xor_expr uint64
              And_expr uint64
                Shift_expr uint64
                  Add_expr uint64
                    Mult_expr uint64
                      Unary_expr uint64
                        Terminator Literal uint64
          Or_expr uint64
            Xor_expr uint64
              And_expr uint64
                Shift_expr uint64
                  Add_expr uint64
                    Mult_expr uint64
                      Unary_expr uint64
                        Terminator Literal uint64

```

测试是否通过：通过

测试用例10——数组3-有赋值但是数量不够

源代码：

```

struct A{
    long a1[5]=[1,2];
};

```

测试结果：

这种情况我设置的是只有警告的。

```

运行: main x
/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
警告：数组长度小于定义的长度！
=====显示符号表=====
root
  A
  struct A
    a1
=====原始输入=====
struct A{
    long a1[5]=[1,2];
};
=====抽象语法树结构=====
Specification
  Definition
    Type_decl

```

测试是否通过：通过

测试用例11——数组4-有赋值但是数量过多

源代码：

```
struct A{  
    long a1[5]=[1,2,3,4,5,6];  
};
```

测试结果：



```
1 struct A{  
2     long a1[5]=[1,2,3,4,5,6];  
3 };
```

运行: main ×
/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
错误：数组长度超过定义的长度！
进程已结束, 退出代码255

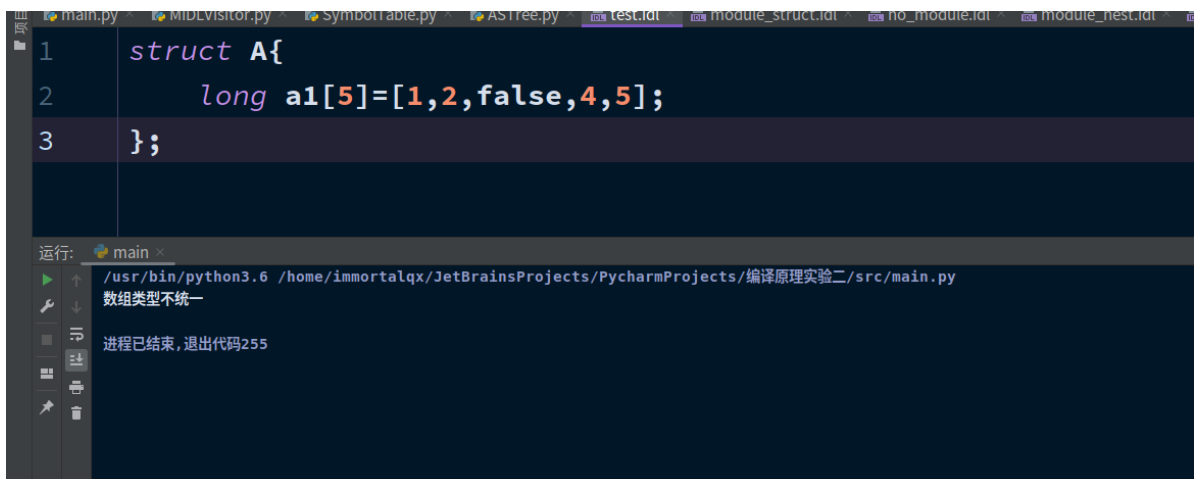
测试是否通过：通过

测试用例12——数组5-有赋值但类型不统一

源代码：

```
struct A{  
    long a1[5]=[1,2,false,4,5];  
};
```

测试结果：



```
1 struct A{  
2     long a1[5]=[1,2,false,4,5];  
3 };
```

运行: main ×
/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py
数组类型不统一
进程已结束, 退出代码255

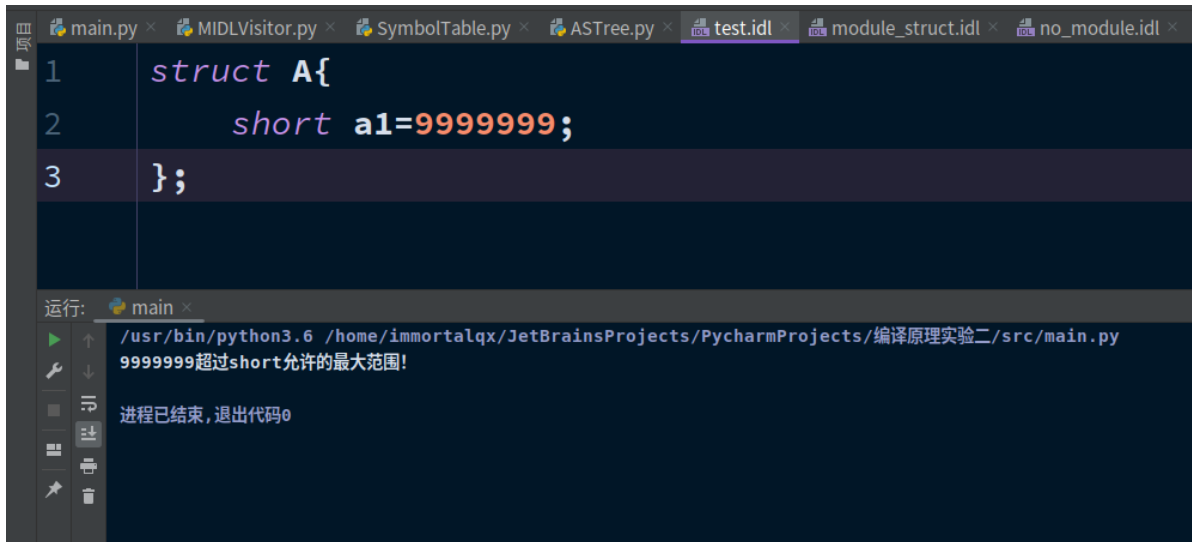
测试是否通过：通过

测试用例13——对最大值的检查

源代码：

```
struct A{  
    short a1=9999999;  
};
```

测试结果：



The screenshot shows an IDE window with several tabs: main.py, MIDLVisitor.py, SymbolTable.py, ASTree.py, test.idl (active), module_struct.idl, and no_module.idl. The active tab displays the following C code:

```
1 struct A{  
2     short a1=9999999;  
3 };
```

Below the code editor, the '运行' (Run) tab is active, showing the command: `/usr/bin/python3.6 /home/immortalqx/JetBrainsProjects/PycharmProjects/编译原理实验二/src/main.py`. The output indicates a runtime error: `9999999超过short允许的最大范围!` (9999999 exceeds the maximum range allowed for short!). The process ended with a return code of 0.

测试是否通过：通过