3. 语义分析

本章实验为**实验二**，任务是在词法分析和语法分析程序的基础上编写一个程序，对C--源代码进行语义分析和类型检查，并打印分析结果。与实验一不同的是，实验二不再借助已有的工具，所有的任务都必须手写代码来完成。另外，虽然语义分析在整个编译器的实现中并不是难度最大的任务，但却是最细致、琐碎的任务。因此需要用心地设计诸如符号表、变量类型等数据结构的实现细节，从而正确、高效地实现语义分析的各种功能。

需要注意的是，由于在后面的实验中还会用到本次实验已经写好的代码，因此保持一个良好的代码风格、系统地设计代码结构和各模块之间的接口对于整个实验来讲相当重要。

3.1 实验内容

3.1.1 实验要求

在本次实验中，我们对C--语言做如下假设，你可以认为这些就是C--语言的特性（注意，假设3、4、5可能因后面的不同选做要求而有所改变）：

1) **假设1**：整型（int）变量不能与浮点型（float）变量相互赋值或者相互运算。

2) **假设2**：仅有int型变量才能进行逻辑运算或者作为if和while语句的条件；仅有int型和float型变量才能参与算术运算。

3) **假设3**：任何函数只进行一次定义，无法进行函数声明。

4) **假设4**：所有变量（包括函数的形参）的作用域都是全局的，即程序中所有变量均不能重名。

5) **假设5**：结构体间的类型等价机制采用**名等价（Name Equivalence）**的方式。

6) **假设6**：函数无法进行嵌套定义。

7) **假设7**：结构体中的域不与变量重名，并且不同结构体中的域互不重名。

以上假设1至7也可视为要求，违反即会导致各种语义错误，不过我们只对后面讨论的17种错误类型进行考察。此外，你可以安全地假设输入文件中不包含注释、八进制数、十六进制数、以及指数形式的浮点数，也不包含任何词法或语法错误（除了特别说明的针对选做要求的测试）。

你的程序需要对输入文件进行语义分析（输入文件中可能包含函数、结构体、一维和高维数组）并检查如下类型的错误：

1) **错误类型1**：变量在使用时未经定义。

2) **错误类型2**：函数在调用时未经定义。

3) **错误类型3**：变量出现重复定义，或变量与前面定义过的结构体名字重复。

4) **错误类型4**：函数出现重复定义（即同样的函数名出现了不止一次定义）。

5) **错误类型5**：赋值号两边的表达式类型不匹配。

6) **错误类型6**：赋值号左边出现一个只有右值的表达式。

7) **错误类型7**：操作数类型不匹配或操作数类型与操作符不匹配（例如整型变量与数组变量相加减，或数组（或结构体）变量与数组（或结构体）变量相加减）。

8) **错误类型8**：return语句的返回类型与函数定义的返回类型不匹配。

9) **错误类型9**：函数调用时实参与形参的数目或类型不匹配。

10) **错误类型10**：对非数组型变量使用“[…]”（数组访问）操作符。

11) **错误类型11**：对普通变量使用“(…)”或“()”（函数调用）操作符。

12) **错误类型12**：数组访问操作符“[…]”中出现非整数（例如a[1.5]）。

13) **错误类型13**：对非结构体型变量使用“.”操作符。

14) **错误类型14**：访问结构体中未定义过的域。

15) **错误类型15**：结构体中域名重复定义（指同一结构体中），或在定义时对域进行初始化（例如struct A { int a = 0; }）。

16) **错误类型16**：结构体的名字与前面定义过的结构体或变量的名字重复。

17) **错误类型17**：直接使用未定义过的结构体来定义变量。

其中，要注意三点：一是关于数组类型的等价机制，同C语言一样，只要数组的基类型和维数相同我们即认为类型是匹配的，例如int a[10][2]和int b[5][3]即属于同一类型；二是我们允许类型等价的结构体变量之间的直接赋值（见后面的测试样例），这时的语义是，对应的域相应赋值（数组域也如此，按相对地址赋值直至所有数组元素赋值完毕或目标数组域已经填满）；三是对于结构体类型等价的判定，每个匿名的结构体类型我们认为均具有一个独有的隐藏名字，以此进行名等价判定。

除此之外，你的程序可以选择完成以下部分或全部的要求：

1) **要求2.1**：修改前面的C--语言假设3，使其变为“函数除了在定义之外还可以进行声明”。函数的定义仍然不可以重复出现，但函数的声明在相互一致的情况下可以重复出现。任一函数无论声明与否，其定义必须在源文件中出现。在新的假设3下，你的程序还需要检查两类新的错误和增加新的产生式：

a) **错误类型18**：函数进行了声明，但没有被定义。

b) **错误类型19**：函数的多次声明互相冲突（即函数名一致，但返回类型、形参数量或者形参类型不一致），或者声明与定义之间互相冲突。

c) 由于C--语言文法中并没有与函数声明相关的产生式，因此你需要先对该文法进行适当修改。对于函数声明来说，我们并不要求支持像“int foo(int, float)”这样省略参数名的函数声明。在修改的时候要留意，你的改动应该以不影响其它错误类型的检查为原则。

2) **要求2.2**：修改前面的C--语言假设4，使其变为“变量的定义受可嵌套作用域的影响，外层语句块中定义的变量可在内层语句块中重复定义（但此时在内层语句块中就无法访问到外层语句块的同名变量），内层语句块中定义的变量到了外层语句块中就会消亡，不同函数体内定义的局部变量可以相互重名”。在新的假设4下，完成错误类型1至17的检查。

3) **要求2.3**：修改前面的C--语言假设5，将结构体间的类型等价机制由名等价改为**结构等价（Structural Equivalence）**。例如，虽然名称不同，但两个结构体类型struct a { int x; float y; }和struct b { int y; float z; }仍然是等价的类型。注意，在结构等价时不要将数组展开来判断，例如struct A { int a; struct { float f; int i; } b[10]; }和struct B { struct { int i; float f; } b[10]; int b;}是不等价的。在新的假设5下，完成错误类型1至17的检查。

3.1.2 输入格式

你的程序的输入是一个包含C--源代码的文本文件，该源代码中可能会有语义错误。你的程序需要能够接收一个输入文件名作为参数。例如，假设你的程序名为cc、输入文件名为test1、程序和输入文件都位于当前目录下，那么在Linux命令行下运行./cc test1即可获得以test1作为输入文件的输出结果。

3.1.3 输出格式

实验二要求通过标准输出打印程序的运行结果。对于那些没有语义错误的输入文件，你的程序不需要输出任何内容。对于那些存在语义错误的输入文件，你的程序应当输出相应的错误信息，这些信息包括错误类型、出错的行号以及说明文字，其格式为：

```
Error type [错误类型] at Line [行号]: [说明文字].
```

说明文字的内容没有具体要求，但是错误类型和出错的行号一定要正确，因为这是判断输出的错误提示信息是否正确的唯一标准。请严格遵守实验要求中给定的错误分类，否则将影响

你的实验评分。

输入文件中可能包含一个或者多个错误（但每行最多只有一个错误），你的程序需要将它们全部检查出来。当然，有些时候输入文件中的一个错误会产生连锁反应，导致别的地方出现多个错误（例如，一个未定义的变量在使用时由于无法确定其类型，会使所有包含该变量的表达式产生类型错误），我们只会去考察你的程序是否报告了较本质那个的错误（如果难以确定哪个错误更本质一些，建议你报告所有发现的错误）。但是，如果源程序里有错而你的程序没有报错或报告的错误类型不对，又或者源程序里没有错但你的程序却报错，都会影响你的实验评分。

3.1.4 测试环境

你的程序将在如下环境中被编译并运行（同实验一）：

1) GNU Linux Release: Ubuntu 20.04, kernel version 5.13.0-44-generic;
2) GCC version 7.5.0;
3) GNU Flex version 2.6.4;
4) GNU Bison version 3.5.1。

一般而言，只要避免使用过于冷门的特性，使用其它版本的Linux或者GCC等，也基本上不会出现兼容性方面的问题。注意，实验二的检查过程中不会去安装或尝试引用各类方便编程的函数库（如glib等），因此请不要在你的程序中使用它们。

3.1.5 提交要求

实验二要求提交如下内容（同实验一）：

1) Flex、Bison以及C语言的可被正确编译运行的源程序。
2) 一份PDF格式的实验报告，内容包括：
   a) 你的程序实现了哪些功能？简要说明如何实现这些功能。清晰的说明有助于助教对你的程序所实现的功能进行合理的测试。
   b) 你的程序应该如何被编译？可以使用脚本、makefile或逐条输入命令进行编译，请详细说明应该如何编译你的程序。无法顺利编译将导致助教无法对你的程序所实现的功能进行任何测试，从而丢失相应的分数。
   c) 实验报告的长度不得超过三页！所以实验报告中需要重点描述的是你的程序中的亮点，是你认为最个性化、最具独创性的内容，而相对简单的、任何人都可以做

的内容则可不提或简单地提一下，尤其要避免大段地向报告里贴代码。实验报告中所出现的最小字号不得小于五号字（或英文11号字）。

3.1.6 样例（必做内容）

实验二的样例包括**必做内容样例**与**选做要求样例**两部分，分别对应于实验要求中的必做内容和选做要求。请仔细阅读样例，以加深对实验要求以及输出格式要求的理解。这节列举必做内容样例。

样例1:

输入:

```
1  int main()
2  {
3    int i = 0;
4    j = i + 1;
5  }
```

输出:

样例输入中变量“j”未定义，因此你的程序可以输出如下的错误提示信息:

```
Error type 1 at Line 4: Undefined variable "j".
```

样例2:

输入:

```
1  int main()
2  {
3    int i = 0;
4    inc(i);
5  }
```

输出:

样例输入中函数“inc”未定义，因此你的程序可以输出如下的错误提示信息:

```
Error type 2 at Line 4: Undefined function "inc".
```

样例3:

输入:

```
1  int main()
2  {
3    int i, j;
4    int i;
5  }
```

输出:

样例输入中变量“i”被重复定义，因此你的程序可以输出如下的错误提示信息:

```
Error type 3 at Line 4: Redefined variable "i".
```

样例4:

输入:

```
 1  int func(int i)
 2  {
 3    return i;
 4  }
 5
 6  int func()
 7  {
 8    return 0;
 9  }
10
11  int main()
12  {
13  }
```

输出:

样例输入中函数“func”被重复定义，因此你的程序可以输出如下的错误提示信息:

```
Error type 4 at Line 6: Redefined function "func".
```

样例5:

输入:

```
1  int main()
2  {
3    int i;
4    i = 3.7;
5  }
```

输出:

样例输入中错将一个浮点常数赋值给一个整型变量，因此你的程序可以输出如下的错误提示信息:

```
Error type 5 at Line 4: Type mismatched for assignment.
```

样例6:

输入:

```
1  int main()
2  {
3    int i;
4    10 = i;
5  }
```

输出:

样例输入中整数“10”出现在了赋值号的左边，因此你的程序可以输出如下的错误提示信息:

```
Error type 6 at Line 4: The left-hand side of an assignment must be a varia-
   ble.
```

样例7:

输入:

```
1  int main()
2  {
3    float j;
4    10 + j;
5  }
```

输出:

样例输入中表达式“10 + j”的两个操作数的类型不匹配，因此你的程序可以输出如下的错误提示信息:

```
Error type 7 at Line 4: Type mismatched for operands.
```

样例8:

输入:

```
1  int main()
2  {
3    float j = 1.7;
4    return j;
5  }
```

输出:

样例输入中“main”函数返回值的类型不正确，因此你的程序可以输出如下的错误提示信息:

```
Error type 8 at Line 4: Type mismatched for return.
```

样例9:

输入:

```
1  int func(int i)
2  {
3    return i;
4  }
5
6  int main()
7  {
8    func(1, 2);
9  }
```

输出:

样例输入中调用函数“func”时实参数目不正确，因此你的程序可以输出如下的错误提示信息:

```
Error type 9 at Line 8: Function "func(int)" is not applicable for arguments
   "(int, int)".
```

样例10:

输入：

```
1  int main()
2  {
3    int i;
4    i[0];
5  }
```

输出：

样例输入中变量“i”非数组型变量，因此你的程序可以输出如下的错误提示信息：

```
Error type 10 at Line 4: "i" is not an array.
```

样例11：

输入：

```
1  int main()
2  {
3    int i;
4    i(10);
5  }
```

输出：

样例输入中变量“i”不是函数，因此你的程序可以输出如下的错误提示信息：

```
Error type 11 at Line 4: "i" is not a function.
```

样例12：

输入：

```
1  int main()
2  {
3    int i[10];
4    i[1.5] = 10;
5  }
```

输出：

样例输入中数组访问符中出现了非整型常数“1.5”，因此你的程序可以输出如下的错误提示信息：

```
Error type 12 at Line 4: "1.5" is not an integer.
```

样例13：

输入：

```
 1  struct Position
 2  {
 3    float x, y;
 4  };
 5
 6  int main()
 7  {
 8    int i;
 9    i.x;
10  }
```

输出：

样例输入中变量“i”非结构体类型变量，因此你的程序可以输出如下的错误提示信息：

```
Error type 13 at Line 9: Illegal use of ".".
```

样例14：

输入：

```
1  struct Position
2  {
3    float x, y;
4  };
5
6  int main()
7  {
8    struct Position p;
9    if (p.n == 3.7)
10     return 0;
11 }
```

输出：

样例输入中结构体变量“p”访问了未定义的域“n”，因此你的程序可以输出如下的错误提示信息：

```
Error type 14 at Line 9: Non-existent field "n".
```

样例15：

输入：

```
1  struct Position
2  {
3    float x, y;
4    int x;
5  };
6
7  int main()
8  {
9  }
```

输出：

样例输入中结构体的域“x”被重复定义，因此你的程序可以输出如下的错误信息：

```
Error type 15 at Line 4: Redefined field "x".
```

样例16：

输入：

```
1  struct Position
2  {
3    float x;
4  };
5
6  struct Position
7  {
8    int y;
9  };
```

```
10
11  int main()
12  {
13  }
```

输出：

样例输入中两个结构体的名字重复，因此你的程序可以输出如下的错误信息：

```
Error type 16 at Line 6: Duplicated name "Position".
```

样例17：

输入：

```
1  int main()
2  {
3    struct Position pos;
4  }
```

输出：

样例输入中结构体“Position”未经定义，因此你的程序可以输出如下的错误信息：

```
Error type 17 at Line 3: Undefined structure "Position".
```

3.1.7 样例（选做要求）

这节列举选做要求样例。

样例1：

输入：

```
 1  int func(int a);
 2
 3  int func(int a)
 4  {
 5    return 1;
 6  }
 7
 8  int main()
 9  {
10  }
```

输出：

如果你的程序需要完成要求2.1，这个样例输入不存在任何词法、语法或语义错误，因此不需要输出。

如果你的程序不需要完成要求2.1，这个样例输入存在语法错误，因此你的程序可以输出如下的错误提示信息：

```
Error type B at Line 1: Incomplete definition of function "func".
```

样例2：

输入：

```
 1  struct Position
 2  {
 3    float x,y;
 4  };
 5
 6  int func(int a);
 7
 8  int func(struct Position p);
 9
10  int main()
11  {
12  }
```

输出：

如果你的程序需要完成要求2.1，这个样例输入存在两处语义错误：一是函数“func”的两次声明不一致；二是函数“func”未定义，因此你的程序可以输出如下的错误提示信息：

```
Error type 19 at Line 8: Inconsistent declaration of function "func".
Error type 18 at Line 6: Undefined function "func".
```

注意，我们对错误提示信息的顺序不做要求。

如果你的程序不需要完成要求2.1，这个样例输入存在两处语法错误，因此你的程序可以输出如下的错误提示信息：

```
Error type B at Line 6: Incomplete definition of function "func".
Error type B at Line 8: Incomplete definition of function "func".
```

样例3：

输入：

```
 1  int func()
 2  {
 3    int i = 10;
 4    return i;
 5  }
 6
 7  int main()
 8  {
 9    int i;
10    i = func();
11  }
```

输出：

如果你的程序需要完成要求2.2，这个样例输入不存在任何词法、语法或语义错误，因此不需要输出。

如果你的程序不需要完成要求2.2，样例输入中的变量“i”被重复定义，因此你的程序可以输出如下的错误提示信息：

```
Error type 3 at Line 9: Redefined variable "i".
```

样例4：

输入：

```
 1  int func()
 2  {
 3    int i = 10;
 4    return i;
 5  }
 6
 7  int main()
 8  {
 9    int i;
10    int i, j;
11    i = func();
12  }
```

输出：

如果你的程序需要完成要求2.2，样例输入中的变量“i”被重复定义，因此你的程序可以输出如下的错误提示信息：

```
Error type 3 at Line 10: Redefined variable "i".
```

如果你的程序不需要完成要求2.2，样例输入中的变量“i”被重复定义了两次，因此你的程序可以输出如下的错误提示信息：

```
Error type 3 at Line 9: Redefined variable "i".
Error type 3 at Line 10: Redefined variable "i".
```

样例5：

输入：

```
 1  struct Temp1
 2  {
 3    int i;
 4    float j;
 5  };
 6
 7  struct Temp2
 8  {
 9    int x;
10    float y;
11  };
12
13  int main()
14  {
15    struct Temp1 t1;
16    struct Temp2 t2;
17    t1 = t2;
18  }
```

输出：

如果你的程序需要完成要求2.3，这个样例输入不存在任何词法、语法或语义错误，因此不需要输出。

如果你的程序不需要完成要求2.3，样例输入中的语句“t1 = t2;”其赋值号两边变量的类型不匹配，因此你的程序可以输出如下的错误提示信息：

```
Error type 5 at Line 17: Type mismatched for assignment.
```

样例6：

输入：

```
1  struct Temp1
2  {
3    int i;
4    float j;
5  };
6
7  struct Temp2
8  {
9    int x;
10 };
11
12 int main()
13 {
14   struct Temp1 t1;
15   struct Temp2 t2;
16   t1 = t2;
17 }
```

输出：

如果你的程序需要完成要求2.3，样例输入中的语句“t1 = t2;”其赋值号两边变量的类型不匹配，因此你的程序可以输出如下的错误提示信息：

```
Error type 5 at Line 16: Type mismatched for assignment.
```

如果你的程序不需要完成要求2.3，应该输出与上述一样的错误提示信息：

```
Error type 5 at Line 16: Type mismatched for assignment.
```