실험 1 : 文法解读作业

숙제 파일공간 문법

서문

컴파일러 기술 과목의 실험 과제에서는 주어진 문법에 따라 작은 규모의 컴파일러를 구현해야 합니다. 이 컴파일러의 구현은 여러 단계로 나뉘어 각 단계별 과제 요구에 해당합니다. 각 과제의 힌트에 따라 컴파일러 모듈을 점진적으로 추가하여 특정 목표 코드를 생성하는 완전한 컴파일러를 완성해야 합니다.

2023년 컴파일러 과목의 설계문법은 "2023 컴파일러 기술 실험 문법 정의 및 관련 설명" 문서에서 자세히 확인하실 수 있습니다. 이번 학기 실험의 모든 과제는 이 문법을 대상으로 하며, 후속 과제는 이전 과제를 기반으로 점진적으로 개발되어야 합니다.

또한, 이번 학기의 모든 과제는 독립적으로 수행되어야 합니다. 유사한 코드 발견 시 부정행위로 간주될 수 있습니다.

（问题描述）문서

코드 컴파일 작업을 시작하기 전에 주어진 문법을 주의 깊게 읽고 분석해야 합니다. 또한, 컴파일러의 전체 구조와 각 모듈 간의 인터페이스, 각 단계에서 발생할 수 있는 오류 유형을 고려해야 합니다. 이를 위해 첫 번째 과제는 문법 해석으로 시작합니다.

아래의 문법을 주의 깊게 읽고 각 규칙이 정의하는 구문 구성 요소의 역할, 제한 조건, 조합 가능성, 가능한 문장을 이해해야 합니다. 이를 기반으로 4-6개의 테스트 프로그램을 작성해야 합니다. 모든 테스트 프로그램은 문법 규칙과 규칙 내의 일반적인 조합을 커버해야 하며, 각 테스트 프로그램은 정확성을 확인하기 위해 10개의 출력 문장을 가져야 합니다.

각 시험프로그램의 첫 번째 글쓰기 문은 printf"('<문자열>')로 자신의 학번을 출력하고, 기타 글쓰기 문장은 프로그램 정의의 데이터와 연산결과를 최대한 반영하여 후속 단계에서 다양한 문법성분의 번역의 정확성을 테스트할 수 있도록 해야 합니다.

문서

입력파일과 출력파일을 문서에 넣고, 이름은 testfile1.txt이면, 각 input1.txt, output1.txt라고 한다.

- 입력파일 : <读语句>가 있으면 제공, 없으면 빈 파일 제공
- 출력파일 : 틀리면 알려줌

시작하는법

문법을 주의 깊게 읽어보고, 각 규칙이 정의하는 구문 구성 요소의 역할, 제한 조건, 조합 가능성 및 생성될 수 있는 문장에 대해 분석해보시기 바랍니다.

다음은 4-6개의 테스트 프로그램을 작성하는 것을 요청드리며, 각 테스트 프로그램이 모든 문법 규칙과 일반적인 조합을 커버할 수 있도록 해야합니다.

각 테스트 프로그램은 10개의 출력 줄을 가져야 합니다.

첫 번째 문장에서는 학번을 출력하도록 요청하였습니다.

다른 문장들은 프로그램에서 정의한 데이터와 해당 데이터의 연산 결과를 최대한 반영하도록 해주시기 바랍니다.

점수

1. 평가 프로그램은 각 테스트 프로그램을 컴파일하고 실행하며 입출력이 주어진 파일과 일치하는지 확인하고 각 테스트 프로그램의 적용 범위와 모든 테스트 프로그램의 누적 적용 범위를 계산하고 점수 = 누적 적용 범위 항목 수/검사의 총 항목 수를 계산합니다.90% 미만의 점수에서 힌트를 주지 않는 경우 문법 보충을 주의 깊게 읽거나 테스트 절차를 개선하십시오. 90% 이상의 점수에서 덮어쓰기되지 않은 규칙 또는 조합에 대한 힌트를 얻을 수 있습니다(평가 세부 정보에서 다운로드 오류를 클릭하면 다운로드 가능).
2. 테스트 프로그램의 난이도는 A, B, C의 3단계로 나뉘며 난이도는 점차 감소합니다.그 중 B급 난이도에는 문법의 노란색 하이라이트 내용이 포함되어야 하며, A급 난이도에는 녹색 하이라이트 내용이 포함되어야 하며, 그 중 녹색 하이라이트 부분, 즉 프로그램의 조건 판단에 [단락 평가] 규칙이 필요합니다.파일을 업로드할 때 A급 난이도 testfile은 1번, B급 난이도 testfile은 2번과 3번, 나머지는 C급 난이도 testfile은 2번과 2번만 있는지 확인하십시오.즉, testfile 번호가 작을수록 난이도가 높아집니다.
3. 평가기 출력 설명

   각 testfile에 대한 덮어쓰기 정보는 세 가지입니다

   testfile runtime error: testfile을 실행하는 중 오류가 발생했습니다. input, output 파일이 제공되지 않았거나 testfile 실행 오류일 수 있습니다.
   testfile syntax error: 제공된 testfile의 구문 성분이 잘못되었습니다.
   백분율: 개별 테스트 지점의 적용 범위를 나타냅니다.
   전체 적용 범위는 모든 문서의 전체 적용 범위를 나타냅니다.
4. 제출하기 전에 파일을 utf-8(bom 없음) 또는 ANSI 형식으로 저장하십시오. 그렇지 않으면 평가 결과에 영향을 미칠 수 있습니다.
5. 입력이 없더라도 빈 입력 파일을 주셔야 합니다.
6. 일부 학생들은 입출력 불일치 오류가 발생할 수 있으며 출력 파일의 줄 끝에 출력해야 할 공간이 부족합니다.이 경우 컴퓨터에 내장된 텍스트 편집기를 사용하여 파일을 저장하고 업로드하십시오. 일부 IDE는 jetbrain과 같은 경우 저장 후 자동으로 행 끝 공백이 지워져 출력이 일치하지 않습니다.
7. 학생들은 현지에서 clang10.0을 사용할 수 있습니다.0 이상의 버전으로 조정되며, 이때 입력한 getint() 함수를 scanf로 대체해야 합니다(2023 컴파일 기술 실험 문법 정의 및 관련 지침 1절 참조).
8. 테스트 결과에 이의가 있을 경우 각 반 조교에게 연락주시면 수업팀에서 오류 조사를 진행하겠습니다.

문법

0 개요

SysY 언어는 실험실에서 개발한 컴파일러의 소스 언어로, C 언어의 하위 집합입니다. 이 파일에는 main이라는 이름의 주 함수 정의가 하나만 있어야 하며, 전역 변수 선언, 상수 선언 및 다른 함수 정의를 포함할 수도 있습니다. SysY 언어는 int 형과 행 우선 저장 방식으로 구성된 다차원 배열 형식을 지원합니다. int 형 정수는 32비트 부호 있는 정수이며, const 한정자는 상수를 선언하는 데 사용됩니다.

1. 함수: 함수는 매개변수를 가질 수도 있고 가지지 않을 수도 있으며, 매개변수의 유형은 int 또는 배열 유형일 수 있습니다. 함수는 int 유형의 값을 반환하거나 반환하지 않을 수 있습니다(void 유형으로 선언됨). 매개변수가 int인 경우 값으로 전달되며, 배열 유형인 경우 실제로 전달되는 것은 배열의 시작 주소이며, 형식 매개변수는 첫 번째 차원의 길이만 비워둘 수 있습니다. 함수 본문은 여러 변수 선언과 문으로 구성됩니다.
2. 변수/상수 선언: 한 번에 여러 변수 또는 상수를 선언하는 변수/상수 선언 문에서 초기화 식을 포함하여 선언할 수 있습니다. 모든 변수/상수는 선언하기 전에 정의되어야 합니다. 함수 외부에서 선언된 것은 전역 변수/상수이며, 함수 내부에서 선언된 것은 지역 변수/상수입니다.
3. 문（语句）: 문은 할당문, 표현식 문(표현식은 비워둘 수 있음), 문 블록, if 문, for 문, break 문, continue 문, return 문으로 구성됩니다. 문 블록에는 여러 변수 선언과 문을 포함할 수 있습니다.
4. 표현식: 기본적인 산술 연산(+, -, *, /, %), 관계 연산(==, !=, <, >, <=, >=) 및 논리 연산(!, &&, ||)을 지원합니다. 0이 아닌 값은 참을 나타내고 0은 거짓을 나타냅니다. 관계나 논리 연산의 결과는 1을 참으로 나타내고 0을 거짓으로 나타냅니다. 연산자의 우선순위, 결합성 및 계산 규칙(논리 연산의 "단락 평가" 포함)은 C 언어와 동일합니다.

一、输入输出函数定义

```
<InStmt> → <LVal> = 'getint''('')'
<OutStmt> → 'printf''('<FormatString>{,<Exp>}')'
```

FormatString : 형식 문자열 터미널로서,다음과 같은 규칙을 따릅니다:

1. clang 평가 결과와 일치하기 위해, <NormalChar>의 이스케이프 문자는 '\n'만 허용되며,
2. 다른 경우에는 단독으로 사용되는 " 는 유효하지 않습니다.”

"Format 문자열에서는 \n 만이 줄바꿈을 나타내는 이스케이프 문자로 사용됩니다. 다른 이스케이프 상황은 고려할 필요가 없습니다. 따라서 <Stmt> 문의 오른쪽에는 위의 입력 및 출력 문장을 추가해야 합니다.”

二、文法及测试程序覆盖要求

1. 커버리지 요구사항

   테스트 프로그램은 컴파일러를 테스트하기 위해 작성된 SysY 언어 프로그램으로, "문법 해석 과제"(과제 1)에서 학생들이 테스트 프로그램을 작성해야 합니다. 테스트 프로그램은 문법의 모든 문법 규칙(즉, 모든 유도 규칙의 모든 후보 항목)을 커버하고, 문법의 의미 제약조건을 충족해야 합니다(즉, 테스트 프로그램은 올바른 프로그램이어야 합니다). 다음 섹션에서는 주석 형식으로 커버해야 할 경우를 제시합니다.

2. 문법

   SysY 언어의 문법은 확장된 Backus-Naur Form(EBNF)을 사용하여 표현되며 다음과 같습니다:

   - [...] ： 대괄호 안에 포함된 것이 선택 사항임을 나타냅니다.
   - {...} ： 중괄호 안에 포함된 항목이 0회 이상 반복될 수 있음을 나타냅니다.
   - 작은 따옴표로 묶인 문자열이나 Ident, InstConst는 **터미널** 기호다.

SysY 언어의 문법은 다음과 같이 표현됩니다. 여기서 CompUnit은 시작 기호입니다.

중요한 점: 문법의 의미 제약조건과 함께 섹션 3의 문법 본문을 함께 참조하는 것이 좋습니다.

三、단어 약속

```
 // 0-> 1.是否存在Decl 2.是否存在FuncDef
编译单元 CompUnit → {Decl} {FuncDef} MainFuncDef

// 1<-0 覆盖两种声明
声明 Decl → ConstDecl | VarDecl
基本类型 BType → 'int' // 存在即可

//ConstDecl부분
// 2<-1 1.花括号内重复0 次 2.花括号内重复多次
常量声明 ConstDecl → 'const' BType ConstDef { ',' ConstDef } ';'
// 3<-2 변수, 1차원, 2차원 배열을 포함
常数定义 ConstDef → Ident { '[' ConstExp ']' } '=' ConstInitVal
// 4<-3 위에서 선언한 변수, 1차원, 2차원 배열을 초기화
常量初值 ConstInitVal → ConstExp | '{' [ ConstInitVal { ',' ConstInitVal } ] '}'

// 5<-1 1花括号内重复0次 2.花括号内重复多次
变量声明 VarDecl → BType VarDef { ',' VarDef } ';'
// 包含普通变量、一维数组、二维数组定义
变量定义 VarDef → Ident { '[' ConstExp ']' } | Ident { '[' ConstExp ']' } '=' InitVal
// 1.表达式初值 2.一维数 组初值 3.二维数组初值
变量初值 InitVal → Exp | '{' [ InitVal { ',' InitVal } ] '}'
 /*
 -> a. Exp : 1
 -> b. { Exp } : {1}
 -> 0 { Exp } : {}
 -> c. { Exp, Exp ... } : { 1, 2, 3 }
 -> d. { initVal } -> {{ InitVal }} -> {{Exp}} : {{1,2}}
 -> e.                {{ InitVal }} -> {{Exp,Exp...}} : {{1,2},{3,4}}
*/

// 0. 메인 함수
主函数定义 MainFuncDef → 'int' 'main' '(' ')' Block

// 일반 함수
函数定义 FuncDef → FuncType Ident '(' [FuncFParams] ')' Block // 1.无形参 2.有形参
// 覆盖两种类型的函数
函数类型 FuncType → 'void' | 'int'
// 1. 1.花括号内重复0次 2.花括号内重复多次
函数形参表 FuncFParams → FuncFParam { ',' FuncFParam }
// 1.普通变量 2.一维数组变量 3.二维数组变量<- 이거안함
函数形参 FuncFParam → BType Ident ['[' ']' { '[' ConstExp ']' }]



// 1.花括号内重复0次 2.花括号内重复多次
语句块 Block → '{' { BlockItem } '}'
// 覆盖两种语句块项
语句块项 BlockItem → Decl | Stmt

// 每种类型的语句都要覆盖
语句 Stmt →
LVal '=' Exp ';'
| [Exp] ';' //有无Exp两种情况
| Block
| 'if' '(' Cond ')' Stmt [ 'else' Stmt ] // 1.有else 2.无else
//1. 无缺省 2. 缺省第一个 ForStmt 3. 缺省Cond 4. 缺省第二个ForStmt
| 'for' '(' [ForStmt] ';' [Cond] ';' [ForStmt] ')' Stmt
| 'break' ';' | 'continue' ';'
| 'return' [Exp] ';' // 1.有Exp 2.无Exp
```

```
| LVal '=' 'getint''('')'';'
| 'printf''('FormatString{','Exp}')'';' // 1.有Exp 2.无Exp

语句 ForStmt → LVal '=' Exp // 存在即可
表达式 Exp → AddExp 注:SysY 表达式是int 型表达式 // 存在即可
条件表达式 Cond → LOrExp // 存在即可
// 1.普通变量 2.一维数组 3.二维数组
左值表达式 LVal → Ident {'[' Exp ']'}

// 3
// 三种情况均需覆盖
基本表达式 PrimaryExp → '(' Exp ')' | LVal | Number
数值 Number → IntConst // 存在即可

// 3种情况均需覆盖,函数调用也需要覆盖FuncRParams的不同情况
一元表达式 UnaryExp → PrimaryExp | Ident '(' [FuncRParams] ')' | UnaryOp UnaryExp // 存在即可
单目运算符 UnaryOp → '+' | '−' | '!' 注:'!'仅出现在条件表达式中 // 三种均需覆盖
// 1.花括号内重复0次 2.花括号内重复多次 3.Exp需要覆盖数组传参和部分数组传参
函数实参表 FuncRParams → Exp { ',' Exp }


// 1// 1.UnaryExp 2.* 3./ 4.% 均需覆盖
乘除模表达式 MulExp → UnaryExp | MulExp ('*' | '/' | '%') UnaryExp
加减表达式 AddExp → MulExp | AddExp ('+' | '−') MulExp // 1.MulExp 2.+ 需覆盖 3.- 需 覆盖

关系表达式 RelExp → AddExp | RelExp ('<' | '>' | '<=' | '>=') AddExp // 1.AddExp 2.< 3.> 4.<= 5.>= 均需覆盖
相等性表达式 EqExp → RelExp | EqExp ('==' | '!=') RelExp // 1.RelExp 2.== 3.!= 均需 覆盖

逻辑与表达式 LAndExp → EqExp | LAndExp '&&' EqExp // 1.EqExp 2.&& 均需覆盖
逻辑或表达式 LOrExp → LAndExp | LOrExp '||' LAndExp // 1.LAndExp 2.|| 均需覆盖
常量表达式 ConstExp → AddExp 注:使用的Ident 必须是常量 // 存在即可
```

- FuncDef : 함수정의
- ConstDef : 상수
- ident : 변수(상수)의 이름
- 표현식 :
  - 조건식 : 참/거짓의 값을 요구한다. if, while
- 구문 : 어떤 표현이 구문인지 표현식인지 살펴보려면 x=…의 우변에 들어가서 작동하는지를 체크해보면 된다.

  for문,
- 声明 ： 선언
- 变量定义： 변수 선언 : a，a[], a[][]
- 初值：초기화
- constExp : 숫자
- stmt :
  1. Stmt의 if 유형 문은 근접 일치를 따릅니다.
  2. 개별 Exp는 Stmt로 사용할 수 있습니다.Exp는 값이 매겨지고 값이 매겨집니다.
- LVal : 변수 이름f

1. LVal은 변수 또는 배열 요소가 될 수 있는 왼쪽 값의 식을 나타냅니다.
2. LVal이 배열을 나타낼 때 대괄호 개수는 배열 변수의 차원과 같아야 합니다(즉, 요소에 위치).
3. 3. LVal이 개별 변수를 나타낼 때 다음 대괄호가 나타날 수 없습니다.

- EXP, Cond :
  1. Exp는 SysY에서 int형 식을 나타내므로 AddExp로 정의되고 Cond는 조건식을 나타내므로 LOrExp로 정의됩니다.전자의 단목 연산자에 ! 가 나타나지 않습니다!후자가 나타날 수 있습니다.
  2. LVal은 Exp 문 앞에 정의된 현재 작용 영역 내 변수 또는 상수여야 하며 할당 번호 왼쪽에 있는 LVal의 경우 변수여야 합니다.
  3. 함수 호출 형식은 Ident'("FuncRParams")이며 여기서 FuncRParams는 실제 매개변수를 나타냅니다.실제 매개변수의 유형과 개수는 Ident에 해당하는 함수로 정의된 참조와 완전히 일치해야 합니다.
  4. SysY에서 연산자의 우선 순위와 결합성은 C언어와 일치하며, 우선 순위와 결합성의 정의는 위의 SysY 문법에서 구현되었습니다.
- Exp (expression) : 표현식. 수학식처럼 우리가 사용하는 변수이다. 피연산자 : operand(상수 / 변수)와 연산자 : operator (+,-,=,*,/)등이 있다.
  - Primary Expression : 피연산자로만 이루어진 가장 간단한 표현식. 식별자 혹은 상수로 표현 가능하다.
    - 식별자 : int a=x; //x가 식별자이다.
    - 상수 : int a=10; //10이 상수이다.
  - UnaryExpression : 단항 연산자

    "Unary Expression"은 프로그래밍 언어에서 사용되는 용어로, 피연산자가 하나인 연산자 표현식을 의미합니다. 다음은 일반적으로 사용되는 "Unary Expression"의 종류입니다:

    1. 증가/감소 연산자 (Increment/Decrement Operators): 피연산자의 값을 1씩 증가시키거나 감소시킵니다. 주로 변수의 값을 증가시키거나 감소시키는 데 사용됩니다.
       - ++ (전위 증가 연산자, Pre-increment Operator): 피연산자의 값을 1 증가시킨 후, 결과 값을 반환합니다.
       - - (전위 감소 연산자, Pre-decrement Operator): 피연산자의 값을 1 감소시킨 후, 결과 값을 반환합니다.
       - (예시) `++x`, `-y`
    2. 부호 연산자 (Sign Operator): 피연산자의 부호를 변경합니다. 양수를 음수로, 음수를 양수로 바꿉니다.
       - (양수 표시 연산자, Plus Operator): 피연산자의 부호를 유지합니다.
       - (음수 표시 연산자, Minus Operator): 피연산자의 부호를 반전시킵니다.
       - (예시) `+x`, `y`
    3. 논리 부정 연산자 (Logical Negation Operator): 피연산자의 논리 값을 반전시킵니다. 참을 거짓으로, 거짓을 참으로 바꿉니다.

- ! (논리 부정 연산자, Logical NOT Operator)
- (예시) `!flag`, `!(x > 5)`

4. 비트 반전 연산자 (Bitwise Complement Operator): 피연산자의 비트를 반전시킵니다. 0은 1로, 1은 0으로 바꿉니다.
   - ~ (비트 반전 연산자, Bitwise NOT Operator)
   - (예시) `~num`