Санкт-Петербургский государственный университет

Кафедра системного программирования

Восстановление синтаксического анализа RuC после обнаружения ошибок

Автор: Илья Алексеевич Андреев, 444 группа

Научный руководитель: д. ф.-м. н. проф. А. Н. Терехов

Санкт-Петербургский государственный университет
Кафедра системного программирования

4 декабря 2020г.

Введение

- Синтаксический анализ – это первый этап компиляции, результатом которого является синтаксическое дерево разбора. В процессе его работы происходит сопоставление токенов языка с его формальной грамматикой
- В предыдущей реализации синтаксического анализатора RuC, если встречалась ошибка в исходном коде, то анализатор заканчивал работу

Постановка задачи

Целью работы является реализация синтаксического анализатора RuC, который сможет продолжать анализ после нахождения ошибки, обнаружив при этом как можно больше действительных ошибок и как можно меньше наведенных.

Задачи:

- Для каждой ошибки, распознаваемой компилятором RuC, написать алгоритм восстановления
- Подготовить тесты и протестировать анализатор
- Сравнить с аналогами

Обзор

- Синтаксический анализатор в RuC устроен так, что в каждый момент времени читаются два подряд идущих токена: currtoken и nexttoken.
- Можно видеть дальше текущего токена, но при этом не нужно было делать возвраты
- Этот механизм важен для анализатора, умеющего восстанавливаться после ошибок.

Реализация

Функция mustbe(Token, errorType): используется, когда из-за несовпадения ожидаемого и читаемого токенов не нужно предпринимать особых действий

- Примеры использования: скобки при разборе операторов, ';' в конце операторов или описаний

Реализация

Тип Undefined: используется для сокращения числа наведенных ошибок

- Например, при задании размера массива необходим целочисленный или символьный тип:

```
int func(intt a){
 int t[a];
}
```

Будет выдана одна ошибка о том, что тип intt не был описан

Реализация

Функция skipUntil(Token, unsigned flags): используется для пропуска токенов до конца текущего блока. Используется при обнаружении:

- Ошибок в выражениях
- Ошибок в списках объявлений
- Ошибок в операторах
- Ошибок в списках инициализации
- Ошибок в списках аргументов
- Ошибок в объявлении массива

Пример 1

- Код:

```
int func(typespec1 a, typespec2 b) {
 int t1[5];
 int t2 = t1[b] + a
 int t3[b];
}
```

- Вывод анализатора RuC:

```
Использование неописанного идентификатора 'typespec1'
Использование неописанного идентификатора 'typespec2'
Ожидалась ';' после списка описаний
Функция, имеющая непустой тип, не возвращает значения
```

- Вывод анализатора clang:

```
test.c:1:10: error: unknown type name 'typespec1'
test.c:1:23: error: unknown type name 'typespec2'
```

Пример 2

- Код:

```
int main() {
 if a == 0)
  b += 5(a);
 return 0;
}
```

- Вывод анализатора RuC:

```
Ожидалась левая скобка перед условием
Использование неописанного идентификатора 'a'
Использование неописанного идентификатора 'b'
Постфиксное выражение должно применяться к идентификатору
```

- Вывод анализатора clang:

```
test.c:3:5: error: expected '(' after 'if'
```

Результаты

- Для каждой ошибки, распознаваемой компилятором RuC, написан алгоритм восстановления
- Подготовлены тесты и анализатор протестирован
- Произведено сравнение с аналогами