Автономная некоммерческая общеобразовательная организация "Физтех-лицей" имени ПЛ. Капицы

Документация к языку программирования
Язык программирования ExQu

Выполнили ученицы класса 11«И»:
Бочкова Валерия Андреевна
Мулина Виктория Павловна

Долгопрудный, 2024

Содержание

Грамматика языка

<program> → < function > |

<function> → ((void|integer|bool|line|valid)id(< enumerationtype >)< operator >(| < function >)|(void|integer|bool|line|valid)(id|id[< prioritylevel1 >])

[= <priority level 1>] ;(id | id[<priority level 1>]) [= <priority level 1>] ! (| <function>)

<enumeration type> → (integer|bool|line|valid)id[, < enumerationtype >]

<enumeration of arithmetic> → < prioritylevel1 > ; < prioritylevel1 >

<multiple definition> → (integer|bool|line|valid)(id|id[< prioritylevel1 >])[=< prioritylevel1 >]; (id|id[< prioritylevel1 >])[=< prioritylevel1 >]!

<bool> → true|false

<assignment> → (id|id[< prioritylevel1 >]) =< operationobject >!

<operator> → < assignment > | < multipledefinition > |if(< prioritylevel1 >)< operator >

[else <operator>] ! | while (<priority level 1>) <internal operator> ! | toggle ((id | id[<priority level 1>])) <selection> ! | for ((<assignment>| <multiple definition>)!<priority level 1> ! <operator>) <internal operator> ! | output (<operation object>) ! | input ((id | id[<priority level 1>])) ! | id . add (< priority level1>) ! | id (<enumeration of arithmetic>) ! | id . pop !| !| return[<priority level1>] !

<internal operator> → < assignment > | < multipledefinition > |if(< prioritylevel1 >)< internaloperator > prioritylevel1 >)< internaloperator >!|

toggle ((id | id[<priority level 1>])) <selection> ! | for((id | id[<priority level 1>] | <multiple definition>) !<priority level 1> ! <internal operator>) <internal operator> ! | id (<enumeration of arithmetic>) ! | output (<operation object>) ! | input ((id | id[<priority level 1>])) ! | id . add (<priority level1 >) ! | id . pop ! | break! | continue! | | return [<priority level1>] !

<selection> → choice(< prioritylevel1 >)< internaloperator >(!| < selection >)

<priority level 1> → < prioritylevel2 > , < prioritylevel2 >

<priority level 2> → < prioritylevel3 > | < prioritylevel3 >

<priority level 3> → < prioritylevel4 > < prioritylevel4 >

<priority level 4> → < prioritylevel5 > (== | =) < prioritylevel5 >

<priority level 5> → < prioritylevel6 > (< | <= | > | >=) < prioritylevel6 >

<priority level 6> → < prioritylevel7 > (+|−) < prioritylevel7 >

<priority level 7> → < prioritylevel8 > (∗|/| < prioritylevel8 >

<priority level 8> → (| + |−) < prioritylevel8 > |(prioritylevel1)| < operationobject >

<operation object> → String| < bool > |Number|id|id.len|id[< prioritylevel1 >]

Описание основных элементов программы

В программе используются следующие классы и методы:

- **Класс Parser**

  Служит для реализации синтаксического анализатора методом рекурсивного спуска. Содержит методы, представляющие из себя рекурсивные функции в соответствии с грамматикой языка:

  - **void program()**
    Параметры функции: нет
    Результат работы: при обнаружении символа конца файла завершает работу программы, иначе вызывает следуующую, реализующую синтаксический анализ, функцию функцию
  - **void operator()**
    Параметры функции: нет
    Результат работы: производит синтаксический анализ тела блока программы интерпретируемого языка.
  - **void function()**
    Параметры функции: нет
    Результат работы: производит синтаксический анализ объявлений функций.
  - **void enumeration__type()**
    Параметры функции: нет
    множественная инициализация объектов
  - **void enumeration__arithmetic()**
    Параметры функции: нет
    перечисление объектов в аргуметах функций, при выводе
  - **void assignment()**
    Параметры функции: нет
    разбор присваивания
  - **void internal __operator()**
    Параметры функции: нет
    Результат работы: производит анализ тела цикла, оператора *toggle*. В отличие от функции *void operator* поддерживает анализ операторов *break*, *continue*
  - **void selection()**
    Параметры функции: нет
    Результат работы: производит разбор *choice* в теле оператора *toggle*
  - **void multiple __definition()**
    Параметры функции: нет
    Результат работы: производит разбор множественной инициализации и объявления объектов
  - **std::pair<type__of__object, bool> priority__level1(bool print)**
    Параметры функции: переменная типа *bool*, отвечающая за необходимость класть разобранное выражение в ПОЛИЗ
    Результат работы: первый, самый низкий уровень приоритета операций; соедержит операцию ,; возвращает пару значений: тип возвращаемого объекта и перемнную типа bool, обозначающую, является ли возвращаемый объект массивом
  - **std::pair<type__of__object, bool> priority__level2(bool print)**
    Параметры функции: переменная типа *bool*, отвечающая за необходимость класть разобранное выражение в ПОЛИЗ
    Результат работы: второй уровень приоритета операций; соедержит операцию *логическое или* '|'; возвращает пару значений: тип возвращаемого объекта и перемнную типа bool, обозначающую, является ли возвращаемый объект массивом

– std::pair<type__of__object, bool> priority__level3(bool print)

Параметры функции: переменная типа *bool*, отвечающая за необходимость класть разобранное выражение в ПОЛИЗ

Результат работы: третий уровень приоритета операций; соедержит операцию *логическое и ''*; возвращает пару значений: тип возвращаемого объекта и перемнную типа bool, обозначающую, является ли возвращаемый объект массивом

– std::pair<type__of__object, bool> priority__level4(bool print)

Параметры функции: переменная типа *bool*, отвечающая за необходимость класть разобранное выражение в ПОЛИЗ

Результат работы: четвертый уровень приоритета операций; соедержит операции *равенства '==' и неравенства '!='*; возвращает пару значений: тип возвращаемого объекта и перемнную типа bool, обозначающую, является ли возвращаемый объект массивом

– std::pair<type__of__object, bool> priority__level5(bool print)

Параметры функции: переменная типа *bool*, отвечающая за необходимость класть разобранное выражение в ПОЛИЗ

Результат работы: пятый уровень приоритета операций; соедержит операции сравнения *'>', '>=', '<=', '<'*; возвращает пару значений: тип возвращаемого объекта и перемнную типа bool, обозначающую, является ли возвращаемый объект массивом

– std::pair<type__of__object, bool> priority__level6(bool print)

Параметры функции: переменная типа *bool*, отвечающая за необходимость класть разобранное выражение в ПОЛИЗ

Результат работы: шестой уровень приоритета операций; соедержит бинарные операции сложения и вычетания *'+', '-'*; возвращает пару значений: тип возвращаемого объекта и перемнную типа bool, обозначающую, является ли возвращаемый объект массивом

– std::pair<type__of__object, bool> priority__level7(bool print)

Параметры функции: переменная типа *bool*, отвечающая за необходимость класть разобранное выражение в ПОЛИЗ

Результат работы: седьмой уровень приоритета операций; соедержит бинарные операции умножения, деления, взятия остатка *'*', '/', '%'*; возвращает пару значений: тип возвращаемого объекта и перемнную типа bool, обозначающую, является ли возвращаемый объект массивом

– std::pair<type__of__object, bool> priority__level8(bool print)

Параметры функции: переменная типа *bool*, отвечающая за необходимость класть разобранное выражение в ПОЛИЗ

Результат работы: восьмой уровень приоритета операций; соедержит унарные операции; возвращает пару значений: тип возвращаемого объекта и перемнную типа bool, обозначающую, является ли возвращаемый объект массивом

– std::pair<type__of__object, bool> operation__ object()

Параметры функции: переменная типа *bool*, отвечающая за необходимость класть объект выражения в ПОЛИЗ

Результат работы: неделимое выражение арифметики; может являться числом, переменной, ячейкой массива; возвращает пару значений; тип возвращаемого объекта и перемнную типа bool, обозначающую, является ли возвращаемый объект массивом

Все выше перечисленные методы имеют спецификатор доступа *public*

Основные поля класса:

– type __of __lexem c __type

тип рассматриваемой лексемы

- **std::string c __str**
  рассматриваемая лексема в виде строки
- **int c__number __str**
  номер строки кода, содержащей исходную лексему

Другие функции класса:

- **void get __lex()**
  Параметры функции: нет
  Результат работы: осуществляет заполненение основных полей класса для рассматриваемой лексемы
- **void analyze()**
  Параметры функции: нет
  Результат работы: является основополагающей функцией класса; служит для отлавливания ошибок при помощи блоков try-catch

- **Класс Poliz**

  Класс, служащий для построения ПОЛИЗА выражения, то есть его постфиксной записи. ПОЛИЗ выражения формируется в векторе, содержащем лексемы - элементы ПОЛИЗА. При генерации ПОЛИЗА используются дополнительные типы лексем, а именно

  - **GOTOFALSE**
  - **GOTO**
  - **LABEL**

  Генерация внутреннего представления программы происходит во время синтаксического анализа параллельно с контролем контекстных условий, поэтому для генерации используется информация, собранная синтаксическим и семантическим анализаторами.

  Основные поля класса:

  - **Lex* p**
  - **int size**
    размер ПОЛИЗА
  - **int free**
    свободное место для элемента ПОЛИЗА

  Основные методы класса:

  - **void put __lex()**
    Параметры функции: нет
    Результат работы: добавление лексемы в полиз
  - **void blank()**
    Параметры функции: нет
    Результат работы: освобождение ячейки в ПОЛИЗЕ
  - **void antiblank()**
    Параметры функции: нет
    Результат работы: удаление лишней ячейки ПОЛИЗА
  - **int get __free()**
    Параметры функции: нет
    Результат работы: возвращает текущий верхний свободный элемент ПОЛИЗА
  - **void print()**
    Параметры функции: нет
    Результат работы: выводит элемерты ПОЛИЗА

Для удобной работы с полизом также используются функции:

- **PolizElement GetPolizElement(std::string str)**

  Параметры функции: строка, являющаяся именем рассматриваемого объекта (переменной, или массива)

  Результат работы: Функция просматривает все элементы TID'a. При совпадении названия объекта *str* с элементом TID'a возвращает его тип, предварительно преобразовав в тип элемента ПОЛИЗА, для последующей удобной записи в ПОЛИЗ.

Для оперативной работы с ПОЛИЗОМ также использовался класс Lex

Класс, определяющий элемент ПОЛИЗА. Создание элементов ПОЛИЗА, в том числе меток, происходит за счет создания объекта вышеупомянутого класса. Пример создания метки:

```
Lex LABEL1(LABEL, POLIZ.get _free());
```

Класс Lex содержит следующие поля:

- **PolizElement t_ex**
- **std::string name**
- **double v_lex_int**
- **std::string v_lex_string**
- **char v_lex_char**
- **std::vector<double> v_lex_vector_int**
- **std::vector<char> v_lex_vector_char;**
- **std::vector<std::string> v_lex_vector_string;**

В зависимости от вариативности выполняемых функций, поля представлены в виде всех используемых типов данных, при этом для экономиии ресурсов и размерности кода, типы даннных int, double, bool хранятся в языке с++ в перремнной типа double. Класс содержит 10 конструкторов, упроощающих процедуру создания объекта.

- **перечисление type __of __object Является перечислением всех типов данных, используемых в языке, а именно:**

  - ***Integer***
  - ***Line***
  - ***Valid***
  - ***Bool***
  - ***Nothing***
  - ***NoChar***
  - ***Void***

- **перечисление Condition**

  Содержит все состояния, необходимые для корректной работы лексического анализатора согласно существующей грамматике

- **перечисление PolizElement**

  Содержит все возможные объекты полиза, в том числе необходимые переходы и метки

- **Функция lexical__analyz**

  Представляет собой реализацию лексического анализатора. Представлена ввиде цикла, считывающего символы до конца файла. Поддерживает состояния. В зависимости от текущего символа происходит переход в новое состояние. Если считанный символ не является пробелом и не является символом '?', отвечающим за комментарии, он добавляется в вектор типа Token *tockens*. В этом векторе хранятся все лексемы, рассмоpенные на этапе лексического анализа. Для удобного обращения предусмотрены функции:

  - **Token get__tocken()**

    Параметры функции: нет

    Результат работы: возвращает текущую лексему. При выходе за границы выкидывает ошибку.

- **TID**

  Таблица идентификаторов анализируемой программы. Представлен std::vector<table> TID Для семантического анализа были использованы некторые функции, позволяющие работать с TID, а именно:

  type__of__object Check__ID
  Параметры функции: имя переменной
  Результат работы: при нахождении переменной в области видимости и наличия объявления переменной вернет type__of__object.
  bool Check__Array(std::string str)
  Параметры функции: имя переменной
  Результат работы: определение является ли заданный объект массивом и возвращение резульата *true* или *false*

  Отдельно представлен TID для работы с функциями.

  - **type__of__object Check__ID__F(std::string str)**
    Параметры функции: строка, представляющая имя проверяемого объекта (предполагаемое имя функции)
    Результат работы: определяет по входной строке, существует ли функция с таким именем. Возвращает в зависимости от результата *true* или *false*.

  - **std::vector<std::pair<type__of__object, std::pair<std::string, bool»> GetParam eters(std::string function__name)**

    Параметры функции: строка, представляющая имя проверяемого объекта (имя рассматриваемой функции)

    Результат работы: возвращает vector параметров рассматриваемой функции. В возвращаемом vector'е хранятся пары из типа параметра и пары, хранящей имя параметра и переменную bool, обозначающую является ли параметр массивом.

  - **bool CheckIsParametersBeen(std::string function__name, std::string comparison __name)**
    Параметры функции: первый параметр - строка (*name*) - является имянем рассматриваемой функции, второй параметр - строка (*comparison__name*) - имя проверяемого параметра
    Результат работы: функция выполняет поиск в TID'е объекта (функции) с заданным именем. Далее просматриваются все параметры заданной функции. Если находится совпадение с *comparison__name* возвращается *false*, иначе - *true*. В тексте программы функция используется для проверки множественной инициализации некоторого объекта среди параметров функций.

  Пример кода для просмотра элементов TID:

```
for (it = tid_list.begin(); it != tid_list.end(); ++it) {
     for (int j = 0; j < it->TID.size(); ++j) {
          if (it->TID[j].name == str) {
                    ....
           }
    }
}
```

Описание работы системы исключений

При работе синтаксического анализатора при нахождении несоотвествия кода пользователя грамматике языка происходит выбрасывание исключений. В функции analyze() находятся try-блоки, в которых размещен код, осуществляющий вывод сообщения об ошибке. Исключение генерируется в блоке throw, который срабатывает при несовпадении рассматриваемой лексемы и лексемы грамматики, или же при остутствии необходимой лексемы. Пример части кода:

```
if (type != Integer) {
        throwWrong(c_number, type);
}
catch (const WrongNumber& ex) {
std::cout << "In the line << ex.str << "a number was expected!" <<
std::endl << "Received: "<< ex.now_lex << ".";
return;
}
```

Наши контакты

Почта: *valeriab7002@gmail.com, mviktoriapro@gmail.com*
Телефон: *+7 (777)-777-77-77*