

Учебный тренажер Mlispsem.

Руководство по применению.

Семантический анализатор проверяет программу на соответствие Правилам семантики, перечисленным в SemanticRules.rtf.

В тренажере реализована объектно-ориентированная модель семантического анализатора МИКРОЛИСПа. В основе модели лежит тот же базовый класс tBC, что и в модели транслятора. Специфика семантического анализа отражена в производном классе tSM. Определение класса содержится в файле semantics.h.

```
// semantics.h 2021
#ifndef SEMAN_H
#define SEMAN_H
#include "base-compiler.h"

class tSM:public tBC{
public:
//конструктор
tSM(const char* grammar_name) :tBC(grammar_name),
scope(0){}
private:
Свойства глобального имени
enum PropFlags{
PROC = 1, //процедура
VAR = 2, //переменная
DEFINED = 4, //определена
USED = 8, //использована
BUILT =16 //встроенная
};
```

Имена в глобальной области видимости характеризуются набором параметров, объединенных в учетную запись
struct tgName{

```
int properties;
```

Множество битовых флагов заданных в перечислении *PropFlags*. Каждый флаг это степень двойки. Двоичное число 11110 определяет множество из четырех флагов. Такое число можно задать выражением *VAR|DEFINED|USED|BULT*.

`std::string line;`

Заданный в формате *string* номер строки исходного текста, в которой анализатор впервые «познакомился» с именем и создал для него учетную запись.

`int arity;`

Арность - это количество параметров процедуры. Для переменных арность равна 0.

`int types;`

Сигнатура типов параметров процедуры - это упорядоченное множество битов. В двоичном представлении числа *types* *i*-й справа бит соответствует *i*-му параметру процедуры. Бит 0 обозначает числовой тип, бит 1 – булевский. Значение 110 показывает, что первый параметр числовой, а второй и третий булевские.

***types* со значением 0 показывает, что все параметры числовые.**

Максимальная арность, которую может корректно обработать эта модель анализатора, равна 32, а точнее $8 * \text{sizeof}(\text{int})$.

```
// конструктор  
tgName(int bprop=0,  
std::string bline=std::string(),  
int barity=0,int btypes=0):  
properties(bprop),  
line(bline),  
arity(barity),  
types(btypes){}
```

Конструктор можно вызвать с четырьмя, тремя, двумя, одним и вообще без аргументов. В последнем случае это будет конструктор «по умолчанию».

```
// функции  
bool test(int aprop){
```

```
    return (properties & aprop)==apropr;
}
```

Функция проверяет наличие одновременно всех флагов, заданных аргументом. Например, `test(PROC|DEFINED)`, что эквивалентно `test(PROC) && test(DEFINED)`.

```
bool empty(){
    return properties==0;
}
```

Функция выявляет «пустую» учетную запись, созданную конструктором «по умолчанию».

```
void set(int aprop){
    properties |= aprop;
}
```

Функция добавляет в учетную запись заданный набор флагов.

```
}; // struct tgName
```

Таблица имен разделена на три части: Таблицу глобальных имен, Таблицу параметров и Таблицу локальных имен.

Такое деление возможно, поскольку в МИКРОЛИСПе нет вложенных процедур и блоков.

```
typedef std::map<std::string,tgName>tGlobal;
```

Таблица глобалтых имен представлена ассоциативным массивом пар (<имя>,<учетная запись>).

```
typedef std::set<std::string>tNameSet;
```

Таблицы параметров и локальных имен не содержат учетных записей. Анализатор регистрирует в них только факт принадлежности имени соответствующей области видимости.

```
// Таблица глобальных имен
tGlobal      globals;
```

```
// Таблица локальных имен
tNameSet     locals;
```

```
// Таблица параметров
tNameSet      params;
```

```
int scope;
```

Текущая точка анализа программы:

0 - вне процедуры;

1 - внутри процедуры;

2 - внутри тела блока.

```
protected:
```

```
void init();
```

Функция вызывается перед началом анализа нового текста.

```
int p01(); int p02(); int p03(); int p04();
// ...
};
#endif
```

Рассмотрим реализацию семантического анализатора для языка грамматики ms.

```
# $ms
```

```
  $id  $idq  $dec  $int
```

```
$bool  $str  (    )
```

```
  +    -    *    /
```

```
  <    =    >    <=
```

```
  >=   and   not   or
```

```
cond  else  if   let
```

```
define display newline  set!
```

```
#
```

```
  S -> PROG #1
```

```
  PROG -> CALCS #2 |
```

```
    DEFS #3 |
```

```
    DEFS CALCS #4
```

```
  CALCS -> CALC #5 |
```

```
    CALCS CALC #6
```

```
  CALC -> E #7
```

```
    E -> $int #8 |
```

```
    $id #9
```

```
  DEF -> PROC #10
```

```

DEFS -> DEF #11 |
    DEFS DEF #12
PROC -> HPROC E ) #13
HPROC -> PCPAR ) #14
PCPAR -> ( define ( $id #15 |
    PCPAR $id #16

```

Продукция #9 порождает числовое выражение-переменную.

Рассмотрим алгоритм семантического анализа использования переменной, инкапсулированный в эту продукцию (файл semantics.cpp).

```

int tSM::p09(){ //    E -> $id
    string name = S1->name;

```

Имя переменной записано в S1->name. Это лексема токена \$id.

В S1->line записан номер строки исходного текста, в которой лексический анализатор обнаружил токен. Значение имеет тип std::string.

```

switch(scope){
    case 2:if(locals.count(name))break;

```

Если переменная находится в теле блока и является локальной, то ошибки нет.

Метод std::set::count вернет 1, если name содержится в контейнере, и 0, если отсутствует.

```

    case 1:if(params.count(name))break;

```

Если переменная находится в теле процедуры и является параметром, то ошибки нет.

```

    default:tgName& ref = globals[name];

```

Оператор индексации ищет name в Таблице глобальных имен и возвращает ссылку на учетную запись.

Если имени нет в Таблице, оператор индексации контейнера тар создаст новую пару с «пустой» учетной записью.

```

    if(ref.empty()){

```

Первое применение еще не объявленной переменной считается ее объявлением.

В учетной записи нет свойства DEFINED.

```

        ref = tgName(VAR|USED, S1->line);
        break;

```

```
} //if(ref.empty())...
```

Имя найдено в Таблице глобальных имен.

```
if(ref.test(VAR)){
    ref.set(USED);
    break;
}
```

пате не является именем переменной, а значит это имя процедуры, которое неправильно используется для ссылки на переменную.

```
if(ref.test(BUILT)){
```

Анализатор накапливает сообщения об ошибках в переменной `string::ferror_message`. Она определена в базовом классе `tBC` и доступна всем продукциям анализатора.

```

ferror_message+=
    "Error[09-1] in line "+ S1->line +": the built-in '"
    +name+
    "' procedure \n\t\t\t cannot be used as a
le!\n";

```

//встроенную процедуру 'abs' нельзя использовать в качестве переменной

```
//the built-in 'abs' procedure cannot be used as a variable
```

В комментарии записан пример сообщения на русском, а затем перевод на английский, выполненный Яндексом.

```
return 1;
}
```

error_message+=

```
"Error[09-2] in line "+ S1->line +": the name '"
+name+
```

" cannot be used to refer to a variable;\n"+

```
"\t\t\tit was previously declared as a procedure in  
line "+ ref.line +" !\n";
```

//имя 'f' нельзя использовать для ссылки на

переменную, в строке 1 оно ранее объявлено как процедура

```
//the name 'f' cannot be used to refer to a variable; it was
previously declared as a procedure in line 1
```

```
return 1;
```

```
} //switch...
```

Анализатор не обнаружил ошибок.

Здесь можно разместить код для синтеза целевой программы.

```
return 0;}
```

Командный интерфейс тренажера такой же, как у Mlispgen.

Input grammar name>ms

Grammar:ms.txt

Source>e

Source:temp.ss

```
1|e
```

```
2|
```

Accepted !

Source>ms-09-1

Source:ms-09-1.ss

```
1|; E -> $id [1]
```

```
2|abs 0
```

```
3|
```

**Error[09-1] in line 2: the built-in 'abs' procedure
cannot be used as a variable!**

```
2|abs 0
```

```
^
```

Rejected !

Source>ms-16-2

Source:ms-16-2.ss

```
1|; PCPAR -> PCPAR $id [2]
```

```
2|(define(f f) f)
```

```
3|
```

**Warning[16-2] in line 2: procedure 'f' has the same name
as its parameter!**

Accepted !
