**Студент: Куликов А.В.**
**Группа: 208**
**Номер по списку: 9**

# «СИСТЕМЫ ПРОГРАММИРОВАНИЯ»
## Курсовая работа 2019.
## Часть 2.

**Для языка МИКРОЛИСП на базе класса tSM сконструировать семантический анализатор, реализующий правила, записанные в файле SemanticRules19.rtf**

**Для каждого алгоритма анализа разработать сценарий тестирования, покрывающий все ветвления алгоритма.**
**Имя тестового файла должно содержать имя продукции атрибутов, которой предназначен тест.**

**Анализатор протестировать с помощью приложения mlispsem, настроенного на грамматику mlisp19.**

**В отчете представить:**
**- текст задания;**
**- распечатку файла semantics.cpp;**
**- блок-схемы алгоритмов анализа, аннотированные именами тестовых файлов;**
**- протоколы тестирования;**
**- выводы.**

**Распечатка файла code-gen.cpp.**

```
>
/* $mlisp19 */
#include "semantics.h"
#include "semempty.cpp"

#include <iostream>
#include <sstream>

using namespace std;

const string numEx = "[numeric expression]";
const string boolEx = "[boolean expression]";
```

```
// вызывается при начале обработки нового текста
программы
void tSM::init() {
   globals.clear();
   locals.clear();
   params.clear();
   scope = 0;
   //константы:
   globals["e"] =
      tgName(VAR | DEFINED | BUILT);
   globals["pi"] =
      tgName(VAR | DEFINED | BUILT);

   // элементарные процедуры:
   globals["remainder"] =
      tgName(PROC | DEFINED | BUILT, 2);
   globals["quotient"] =
      tgName(PROC | DEFINED | BUILT, 2);
   globals["abs"] =
      tgName(PROC | DEFINED | BUILT, 1);
   globals["expt"] =
      tgName(PROC | DEFINED | BUILT, 2);
   globals["log"] =
      tgName(PROC | DEFINED | BUILT, 1);
   globals["atan"] =
      tgName(PROC | DEFINED | BUILT, 1);
   globals["sqrt"] =
      tgName(PROC | DEFINED | BUILT, 1);
   return;
}
int tSM::p01() {  //      S -> PROG
   bool error = false;

   vector<string> undefProc;
   vector<string> undefVar;

   vector<string> unusedProc;
   vector<string> unusedVar;

   for (tGlobal::iterator it = globals.begin();
```

```
     it != globals.end();
     ++it) {
    const string &name = it->first;
    tgName &ref = it->second;

    if (ref.test(PROC)) {
        if (!ref.test(DEFINED)) {
            undefProc.push_back(name);
            error = true;
        }
        if (!ref.test(USED) && !ref.test(BUILT)) {
            unusedProc.push_back(name);
        }
    }
    else if (ref.test(VAR)) {
        if (!ref.test(DEFINED)) {
            undefVar.push_back(name);
            error = true;
        }
        if (!ref.test(USED) && !ref.test(BUILT)) {
            unusedVar.push_back(name);
        }
    }
}

ostringstream buf;

auto formBuf = [&buf](const vector<string> &vec) {
    int count = vec.size();
    for (int i = 0; i < count; i++) {
        buf << vec[i];
        if (i != count - 1)
            buf << ", ";
    }
    buf << endl;
};

if (!undefProc.empty()) {
    buf << "[!]Undefined procedures: ";
    formBuf(undefProc);
}
```

```
   if (!undefVar.empty()) {
      buf << "[!]Undefined variables: ";
      formBuf(undefVar);
   }
   if (!unusedProc.empty()) {
      buf << "[?]Unused procedures: ";
      formBuf(unusedProc);
   }
   if (!unusedVar.empty()) {
      buf << "[?]Unused variables: ";
      formBuf(unusedVar);
   }

   ferror_message = buf.str();
   if (error){
      return 1;
   }

   return 0;
}

int tSM::p11() {  //       E -> $id
   const string &name = S1->name;
   tgName &ref = globals[name];

   ostringstream buf;
   buf << "[!]Numeric expression evaluation: '" << S1->name
<< "' ";

   bool isProc = ref.test(PROC);

   auto setDef = [](tgName &r) {
      if (r.empty()) {
         r = tgName(VAR | USED);
      }
      else {
         r.set(USED);
      }
   };

   if (scope == 0) {
```

```
      if(isProc){ // p11-1
         buf <<  "is procedure not a variable !";
         ferror_message = buf.str();
         return 1;
      }
      setDef(ref); // p11-2
   }

   if(scope == 1){
      if(!params.count(name)){
         if(isProc){ // p11-4
            buf << "is procedure not a variable !";
            ferror_message = buf.str();
            return 1;
         }
         setDef(ref); // p11-5
      } // p11-3
   }

   if(scope == 2){
      if(!locals.count(name)){
         if (!params.count(name)) {
            if (isProc) { // p11-8
               buf << "is procedure not a variable !";
               ferror_message = buf.str();
               return 1;
            }
            setDef(ref); // p11-9
         } // p11-7
      } // p11-6
   }

   return 0;
}
int tSM::p45() {  //   CPROC -> HCPROC )
   string name = S1->name;
   int count = S1->count;
   if (scope > 1) {              // внури тела let
      if (locals.count(name)) {  // локальное имя
                                 //p45-1.ss
         ferror_message =
```

```
            "[!]Procedure application:"
            " local variable '" +
            name +
            "' shadows the procedure!";
         return 1; // 1 если ошибка
      }                        // if locals ...
   }                           // if scope ...
   if (scope > 0) {               // внутри процедуры
      if (params.count(name)) {  // имя параметра
                               //p45-2.ss
         ferror_message =
            "[!]Procedure application:"
            " parameter '" +
            name +
            "' shadows the procedure!";
         return 1;
      }  // if params...
   }     // if scope...

   // do {} while(false) для удобства, чтобы можно было
пользоваться break'ами.
   do {
      //  найти имя в глобальной таблице
      tgName& ref = globals[name];
      if (ref.empty()) {  //неизвестное имя
                        //       создать новую учетную запись
                        // cout << name << " USED WITH " << count
<< endl;
         ref = tgName(PROC | USED, count);
         break;
      }

      // имя найдено
      if (!ref.test(PROC)) {  //не процедура
                           //p45-3.ss
         ferror_message =
            "[!]Procedure application:"
            "  '" +
            name +
            "' is not a procedure!";
         return 1;
```

```
      }

      if (ref.arity != count) {  //число аргументов
                              //              не равно числу параметров
         std::ostringstream buf;
         buf << "[!]Procedure application: '" << name << "' "
            //p45-4.ss
            << (ref.test(DEFINED) ? "expects "  // процедура
                                          //                                уже
определена
                                          //p45-5.ss

                                 // процедура еще не определена, но
уже вызывалась ранее
                                 : "has been called already\n\t with ")
            << ref.arity << " argument"
            << (ref.arity != 1 ? "s" : "")
            << ", given: " << count << " !";
         ferror_message = buf.str();
         return 1;
      }
      // ошибок нет
      ref.set(USED);  //имя использовано
   } while (false);
   return 0;
}
int tSM::p46() {  //  HCPROC -> ( $id
   S1->name = S2->name;
   S1->count = 0;
   return 0;
}
int tSM::p47() {  //  HCPROC -> HCPROC E
   ++S1->count;
   return 0;
}
int tSM::p49() {  //    BOOL -> $idq
   tgName &ref = globals[S1->name];
   const string &name = S1->name;

   ferror_message =
      "[!]Boolean expression evaluation: '" + S1->name + "' ";
```

```
    if (!params.count(name)) { // p49-1
        ferror_message +=
            "is inaccessible in this scope: [param] !";
        return 1;
    }
    // p49-2
    return 0;
}
int tSM::p55() {  //   CPRED -> HCPRED ) // non-checked
    string name = S1->name;
    int count = S1->count;
    int types = S1->types;

    if (scope > 0) {
        if (params.count(name)) { // p55-1
            ferror_message =
                "[!]Predicate application:"
                " parameter '" +
                name +
                "' shadows the predicate!";
            return 1;
        }
    }

    do {
        //  найти имя в глобальной таблице
        tgName& ref = globals[name];
        if (ref.empty()) { // p55-2
            ref = tgName(PROC | USED, count, types);
            break;
        }

        if (ref.arity != count) { // p55-3
            std::ostringstream buf;
            buf << "[!]Predicate application: '" << name << "' "
                //p45-4.ss
                << (ref.test(DEFINED) ? "expects "
                                      : "has been called already\n\t with ")
                << ref.arity << " argument"
                << (ref.arity != 1 ? "s" : "")
```

```
            << ", given: " << count << " !";
         ferror_message = buf.str();
         return 1;
      }

      if(ref.types != types){ // p55-4
         int diffInd;
         for(int i = 0; i < sizeof(types); i++){
            if ((ref.types & (1 << i)) != (types & (1 << i))){
               diffInd = i;
               break;
            }
         }

         int type = (ref.types & (1 << diffInd));

         std::ostringstream buf;
         buf << "[!]Predicate application: '" << name << "' "
            << (ref.test(DEFINED) ? "expects "
                                  : "has been called already\n\t with ")
            << (type ? boolEx : numEx) << " on position " <<
diffInd
            << ", given: " << (!type ? boolEx : numEx) << " !";
         ferror_message = buf.str();
         return 1;
      }

      ref.set(USED);
   } while (false);

   // p55-5
   return 0;
}
int tSM::p56() {  //  HCPRED -> ( $idq
   S1->name = S2->name;
   S1->types = 0; // non-checked
   S1->count = 0;
   return 0;
}
int tSM::p57() {  //  HCPRED -> HCPRED ARG // non-checked
```

```
    S1->types |= (S2->types << S1->count);
    S1->count++;
    return 0;
}
int tSM::p58() {  //     ARG -> E
    S1->types = 0; // non-checked
    return 0;
}
int tSM::p59() {  //     ARG -> BOOL
    S1->types = 1; // non-checked
    return 0;
}
int tSM::p74() {  //     SET -> ( set! $id E )
    const string &name = S3->name;
    tgName &ref = globals[name];

    bool isProc = ref.test(PROC);
    bool isBuilt = ref.test(BUILT);

    ostringstream buf;
    buf << "[!]Assignment operation: '"
    << name << "' is ";

    if (isBuilt) {
        buf << "built-in ";
        if (isProc) {
            buf << "procedure";
        }
        else {
            buf << "constant";
        }
    }
    else if(isProc) {
        buf << "procedure";
    }

    auto setDef = [](tgName &r) {
        if (r.empty()) {
            r = tgName(VAR | USED);
        } else {
            r.set(USED);
```

```
      }
   };

   if (scope == 0) {
      if (isProc || isBuilt) {  // p74-1
         ferror_message = buf.str();
         return 1;
      }
      setDef(ref); // p74-2
   }

   if (scope == 1) {
      if (!params.count(name)) {
         if (isProc || isBuilt) {  // p74-3
            ferror_message = buf.str();
            return 1;
         }
         setDef(ref); // p74-4
      }
      // p74-5
   }

   if (scope == 2) {
      if (!locals.count(name)) {
         if (!params.count(name)) {
            // cout << "here" << endl;
            if (isProc || isBuilt) { // p74-6
               ferror_message = buf.str();
               return 1;
            }
            setDef(ref); // p74-7
         }
         // p74-8
      }
      // p74-9
   }

   return 0;
}

int tSM::p87() {  //    PRED -> HPRED BOOL )
```

```
    scope = 0;
    params.clear();
    return 0;
}
int tSM::p88() {  //   HPRED -> PDPAR )
    scope = 1;
    const string &name = S1->name;
    tgName &ref = globals[name];
    int arity = S1->count;
    int types = S1->types;

    if (ref.empty()) { // p88-1
        ref = tgName(PROC | DEFINED, arity, types);
        return 0;
    }

    std::ostringstream buf;

    buf << "[!]Predicate definition: predicate '"
        << name << "' ";

    if (ref.test(DEFINED)) { // p88-2
        buf << "has been aldeady defined";
        ferror_message = buf.str();
        return 1;
    }

    if (ref.test(USED)) {
        if (ref.arity != S1->count) { // p88-3
            buf << "has been already called\n\t with "
                << ref.arity << " arguments"
                << ", given: " << S1->count << " !";
            ferror_message = buf.str();
            return 1;
        }

        if (S1->types != ref.types) { // p88-4
            int diffInd;
            for (int i = 0; i < sizeof(S1->types); i++) {
                if ((ref.types & (1 << i)) != (S1->types & (1 << i))) {
                    diffInd = i;
```

```
                break;
            }
        }

        int type = ref.types & (1 << diffInd);

        buf << "has been already called\n\t with "
            << (type ? boolEx : numEx)
            << " on position " << diffInd
            << ", given: " << (!type ? boolEx : numEx) << " !";

        ferror_message = buf.str();
        return 1;
      }
      // p88-5
   }

   ref.set(DEFINED);
   return 0;
}
int tSM::p89() {  //   PDPAR -> ( define ( $idq
   S1->types = 0;
   S1->name = S4->name; // non-checked
   return 0;
}
int tSM::p90() {  //   PDPAR -> PDPAR $idq
   if (params.count(S2->name)) { //p90-1
      ferror_message =
         "[!]Procedure definition: in '" + S1->name +
         "' duplicate parameter identifier '" + S2->name + "'!";
      return 1;
   }
   params.insert(S2->name);
   S1->types |= (1 << S1->count);
   S1->count++; //p90-1
   return 0;
}
int tSM::p91() {  //   PDPAR -> PDPAR $id
   if (params.count(S2->name)) { //p91-1
      ferror_message =
         "[!]Procedure definition: in '" + S1->name +
```

```
            "' duplicate parameter identifier '" + S2->name + "'!";
        return 1;
    }
    params.insert(S2->name);
    S1->count++; //p91-2
    return 0;
}
int tSM::p92() {  //     VAR -> ( define $id CONST )
    const string &name = S3->name;
    tgName &ref = globals[name];

    if (ref.empty()) { // p92-1
        ref = tgName(VAR | DEFINED);
        return 0;
    }

    std::ostringstream buf;

    buf << "[!]Global variable definition: ";

    if (ref.test(PROC)) {
        buf << "procedure";
    } else {
        buf << "variable";
    }

    buf << " '" << name << "' ";

    if (ref.test(DEFINED)) { // p92-2
        buf << "has been aldeady defined";
        ferror_message = buf.str();
        return 1;
    }
    else if (ref.test(USED)) {
        if (ref.test(PROC)) { // p92-3
            buf << "has been already used";
            ferror_message = buf.str();
            return 1;
        }
    }
    ref.set(DEFINED); // p92-4
```

```
    return 0;
}
int tSM::p93() {  //    PROC -> HPROC LET )
    params.clear();
    scope = 0;
    return 0;
}
int tSM::p94() {  //    PROC -> HPROC E )
    params.clear();
    scope = 0;
    return 0;
}
int tSM::p95() {  //   HPROC -> PCPAR )

    scope = 1;
    const string &name = S1->name;
    tgName &ref = globals[name];
    int arity = S1->count;

    if (ref.empty()) { // p95-1
        ref = tgName(PROC | DEFINED, arity);
        return 0;
    }

    std::ostringstream buf;

    buf << "[!]Procedure definition: ";

    if(ref.test(PROC)){
        buf << "procedure";
    }
    else{
        buf << "variable";
    }

    buf << " '" << name << "' ";

    if(ref.test(DEFINED)){ // p95-2
        buf << "has been aldeady defined";
        ferror_message = buf.str();
        return 1;
```

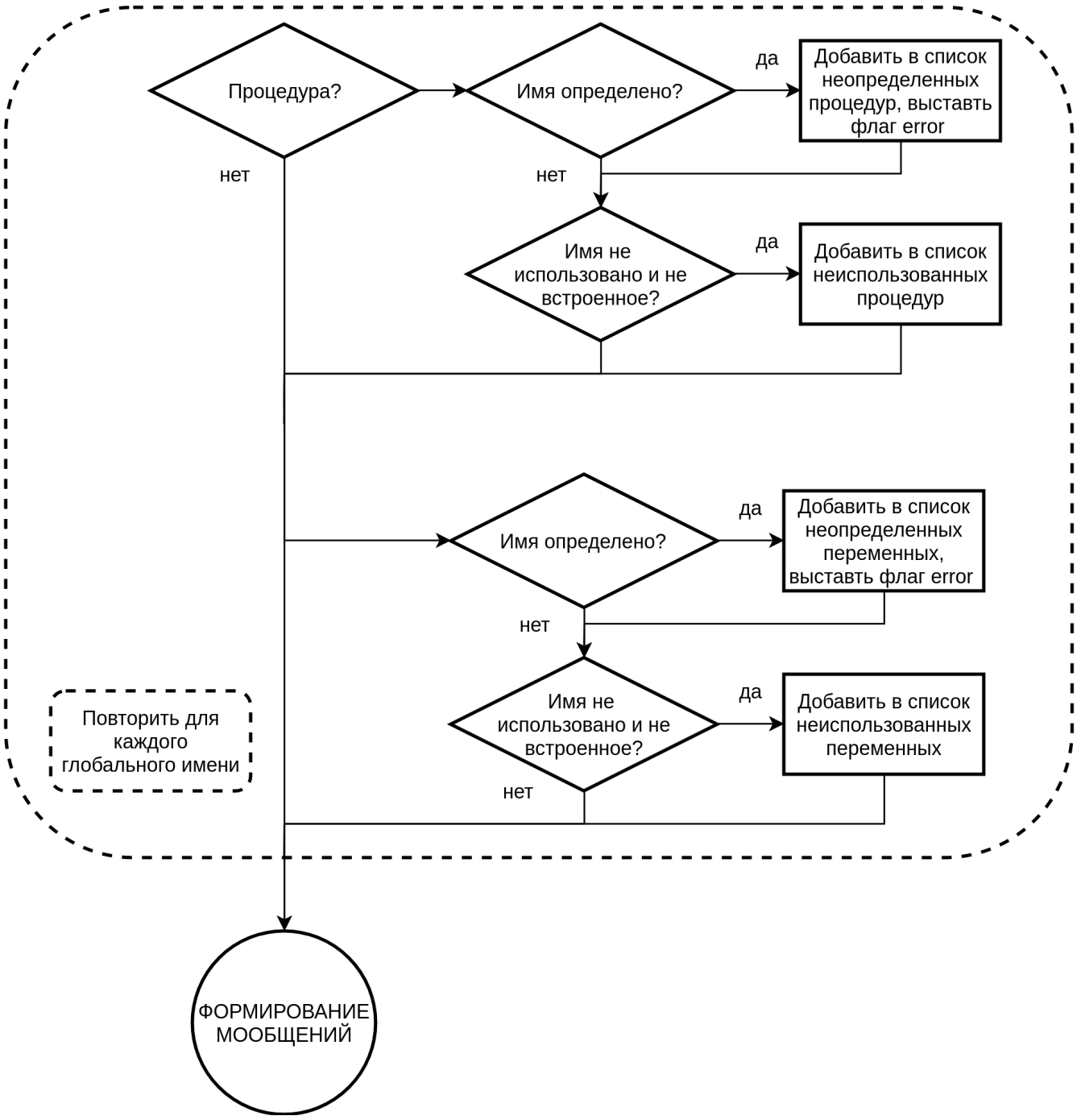
```
    }
    else if(ref.test(USED)){
        if (ref.test(PROC)) {
            if (ref.arity != arity) { // p95-3
                buf << "has been already called\n\t with "
                    << ref.arity << " arguments"
                    << ", given: " << S1->count << " !";
                ferror_message = buf.str();
                return 1;
            }
        }
        else{ // p95-4
            buf << "has been already used";
            ferror_message = buf.str();
            return 1;
        }
    }

    ref.set(DEFINED); // p95-5
    return 0;
}

int tSM::p97() {  //   PCPAR -> ( define ( $id
    S1->name = S4->name;
    S1->count = 0;
    return 0;
}

// проверка на повторяющиеся параметры
int tSM::p98() {  //   PCPAR -> PCPAR $id
    if (params.count(S2->name)) { //p98-1
        ferror_message =
            "[!]Procedure definition: in '" + S1->name +
            "' duplicate parameter identifier '" + S2->name + "'!";
        return 1;
    }
    params.insert(S2->name);
    ++S1->count;  //p98a
    return 0;
}
int tSM::p99() {  //    LET -> HLET E )
```

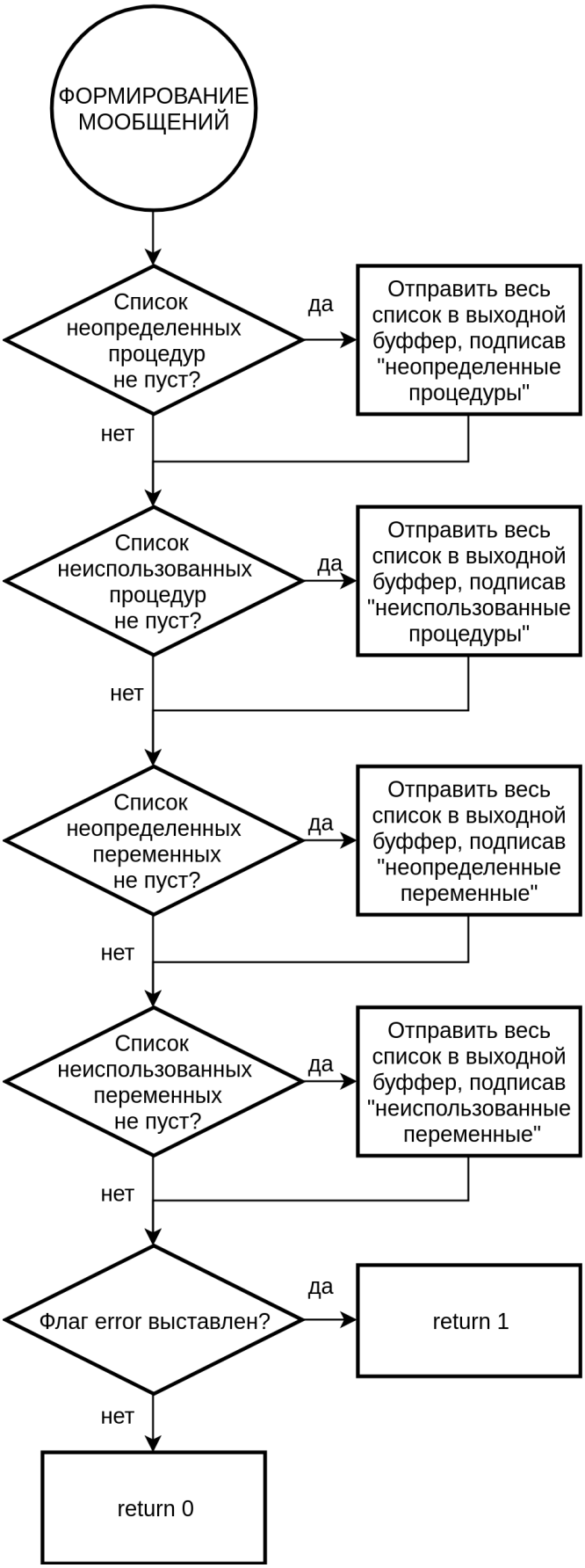
```
    locals.clear();
    return 0;
}
int tSM::p100() {  //    HLET -> LETLOC )
    scope = 2;
    return 0;
}
int tSM::p102() {  //  LETLOC -> ( let (
    locals.clear();
    return 0;
}
int tSM::p104() {  //  LETVAR -> ( $id E )
    if (locals.count(S2->name)) { //p104-1
        ferror_message =
            "[!]Local variable definition:"
            " duplicate local vatiable identifier '"
            + S2->name + "'!";
        return 1;
    }
    locals.insert(S2->name);  //p104-2
    return 0;
}
//____________________
```

**Блок-схемы алгоритов анализа:**

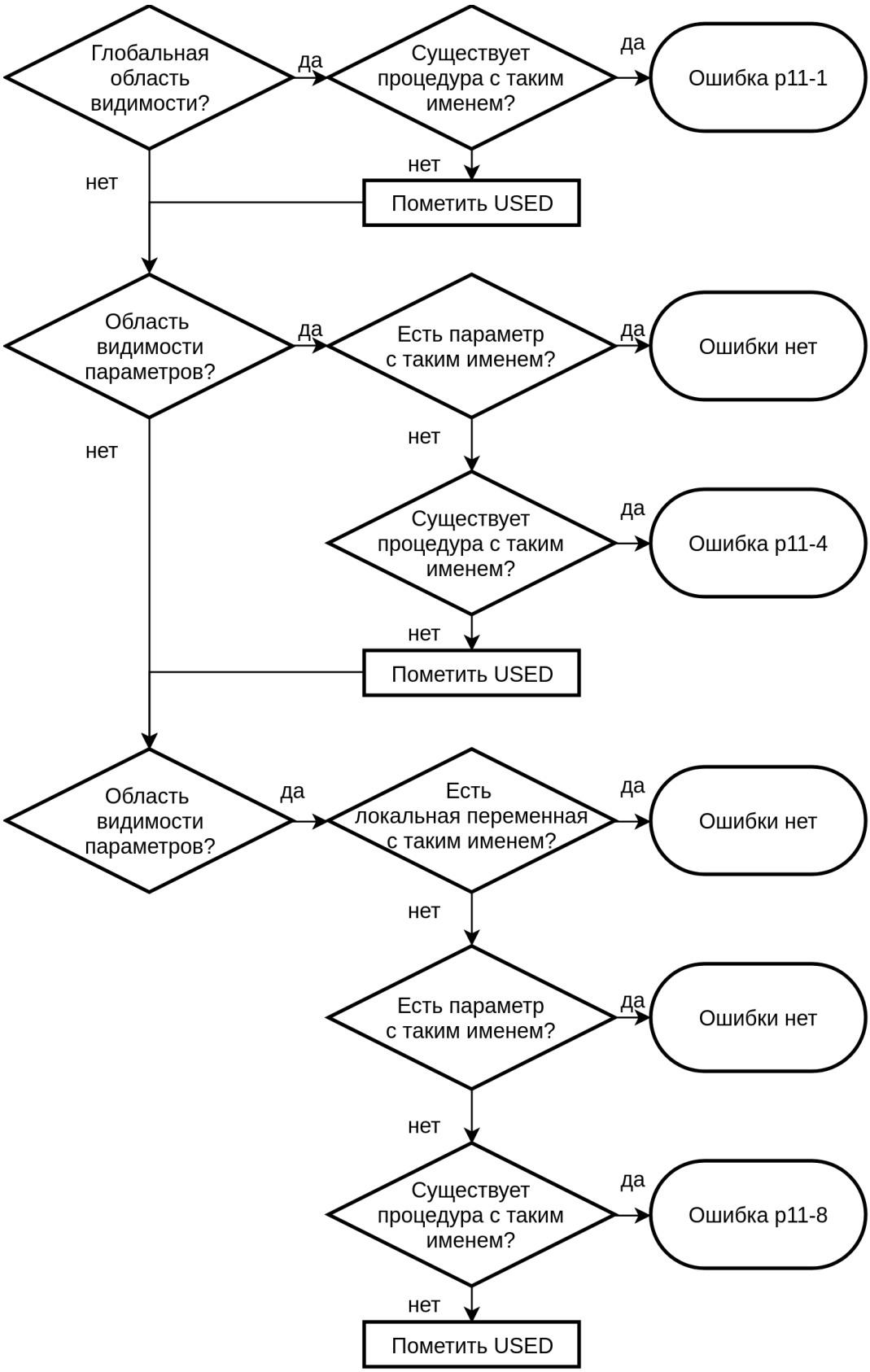
**p1**

>


Процедура?
Имя определено?
да
Добавить в список неопределенных процедур, выставть флаг error
нет
нет
Имя не использовано и не встроенное?
да
Добавить в список неиспользованных процедур
Имя определено?
да
Добавить в список неопределенных переменных, выставть флаг error
нет
Имя не использовано и не встроенное?
да
Добавить в список неиспользованных переменных
нет
Повторить для каждого глобального имени
ФОРМИРОВАНИЕ МООБЩЕНИЙ

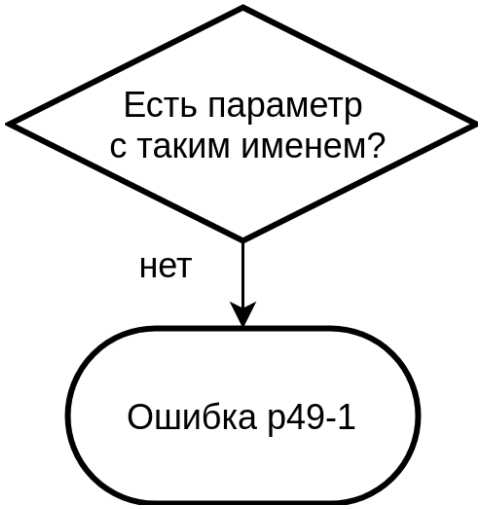

ФОРМИРОВАНИЕ МООБЩЕНИЙ
Список неопределенных процедур не пуст?
да
Отправить весь список в выходной буффер, подписав "неопределенные процедуры"
нет
Список неиспользованных процедур не пуст?
да
Отправить весь список в выходной буффер, подписав "неиспользованные процедуры"
нет
Список неопределенных переменных не пуст?
да
Отправить весь список в выходной буффер, подписав "неопределенные переменные"
нет
Список неиспользованных переменных не пуст?
да
Отправить весь список в выходной буффер, подписав "неиспользованные переменные"
нет
Флаг error выставлен?
да
return 1
нет
return 0

**p11**
**>**

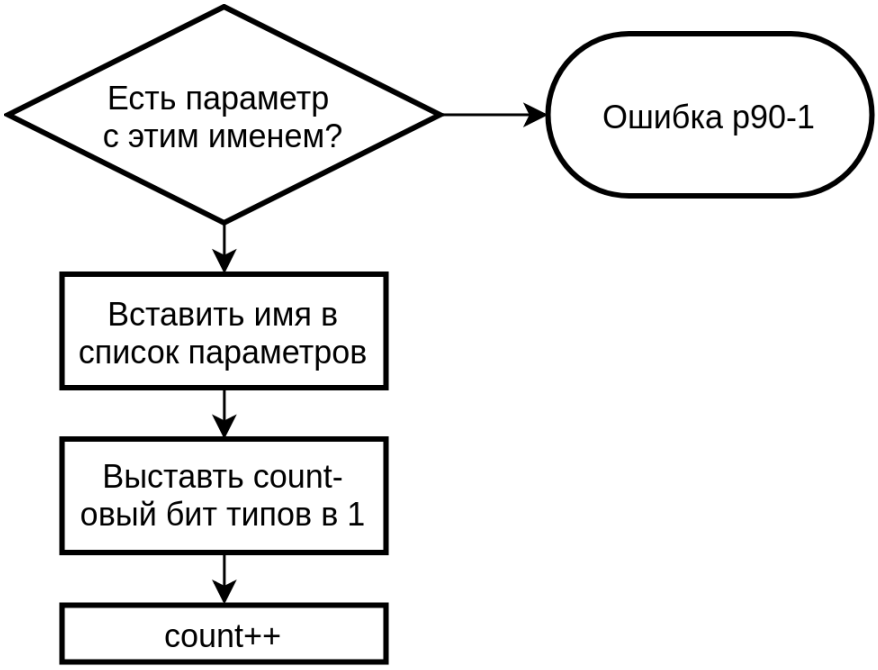

Глобальная область видимости?
да
Существует процедура с таким именем?
да
Ошибка p11-1
нет
нет
Пометить USED
Область видимости параметров?
да
Есть параметр с таким именем?
да
Ошибки нет
нет
нет
Существует процедура с таким именем?
да
Ошибка p11-4
нет
Пометить USED
Область видимости параметров?
да
Есть локальная переменная с таким именем?
да
Ошибки нет
нет
Есть параметр с таким именем?
да
Ошибки нет
нет
Существует процедура с таким именем?
да
Ошибка p11-8
нет
Пометить USED

**p45**
**> была дана в качестве образца, осталась без изменений.**

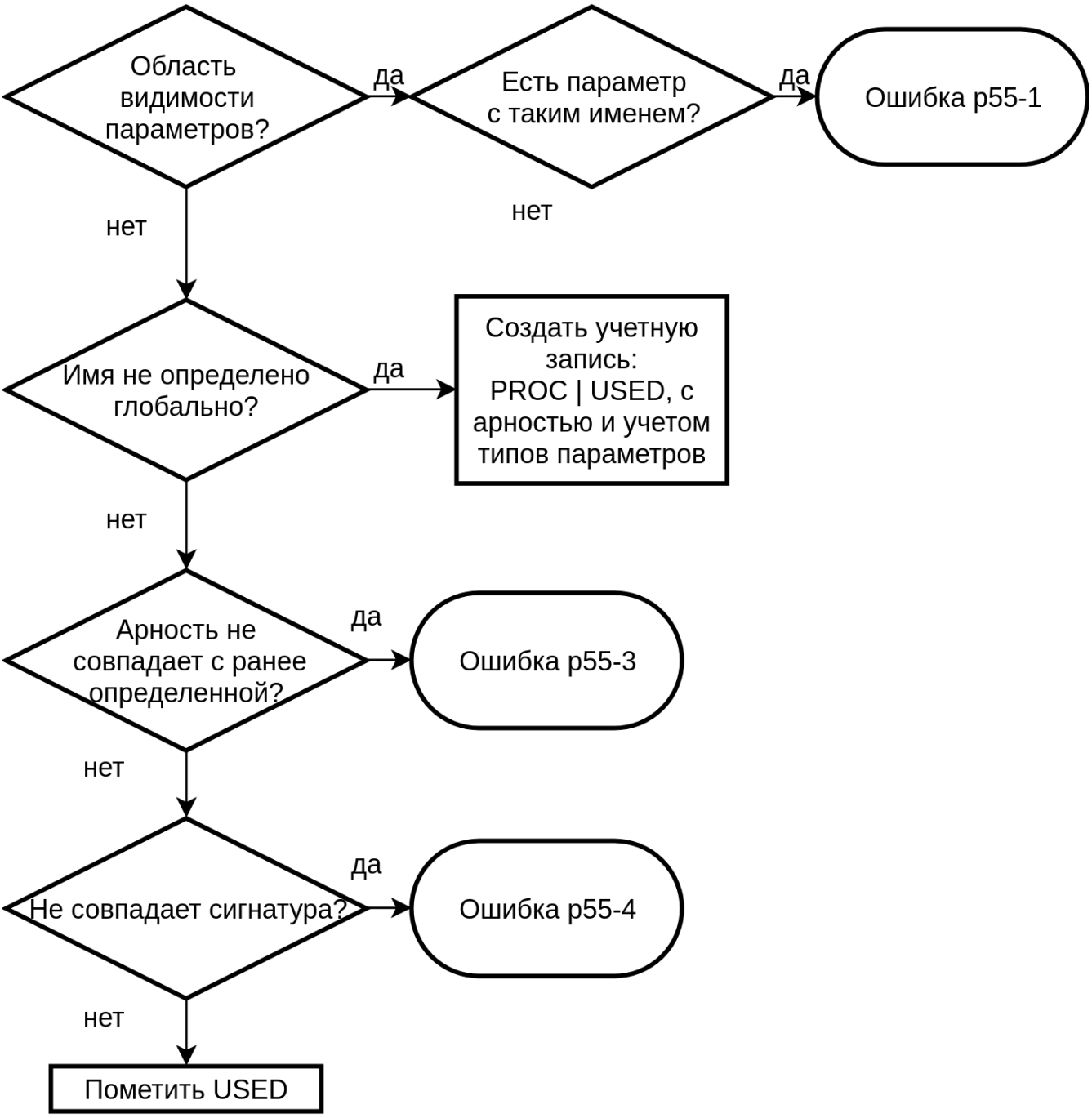
**P49**
**>**



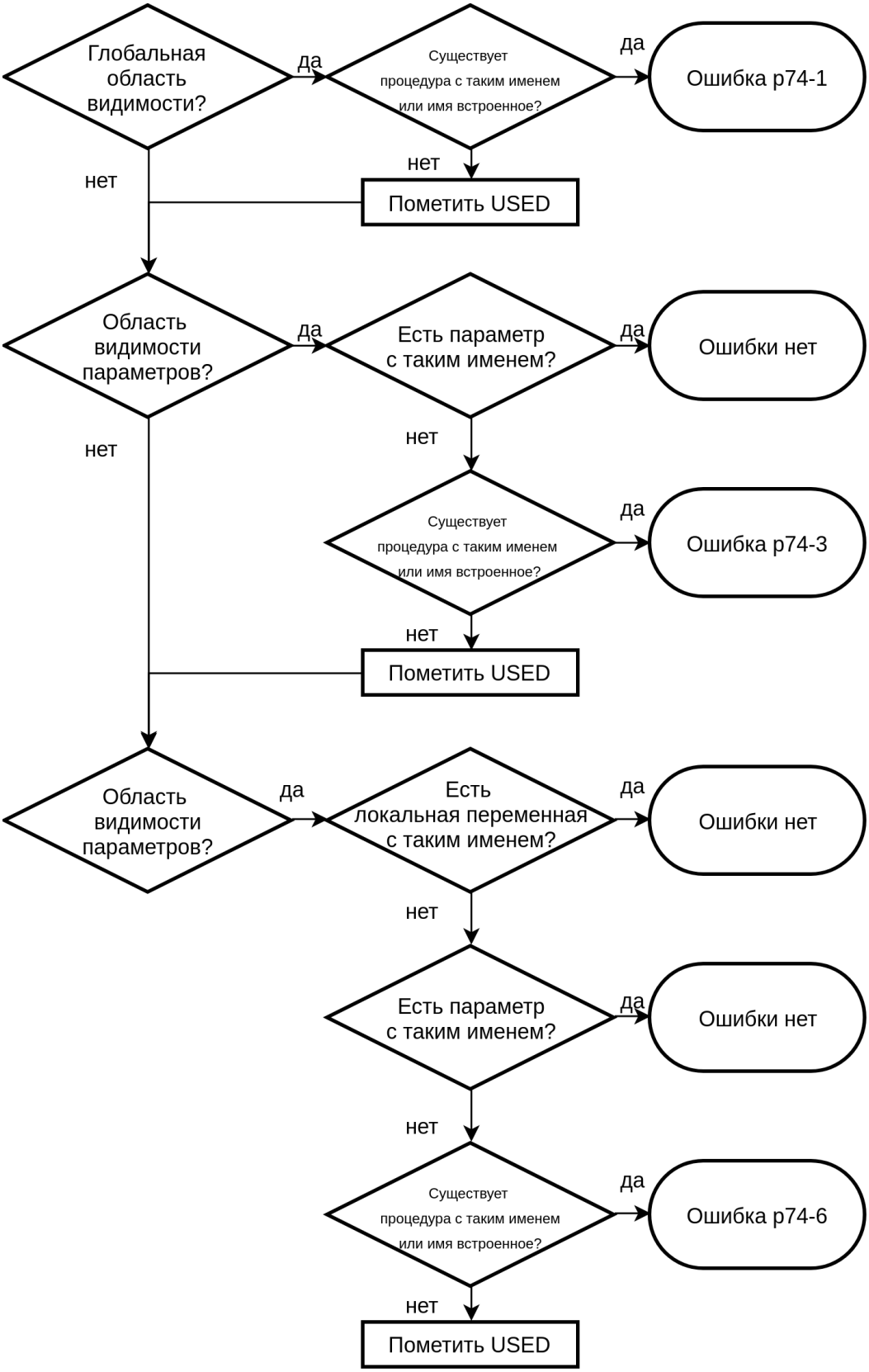

**p90**
**>**



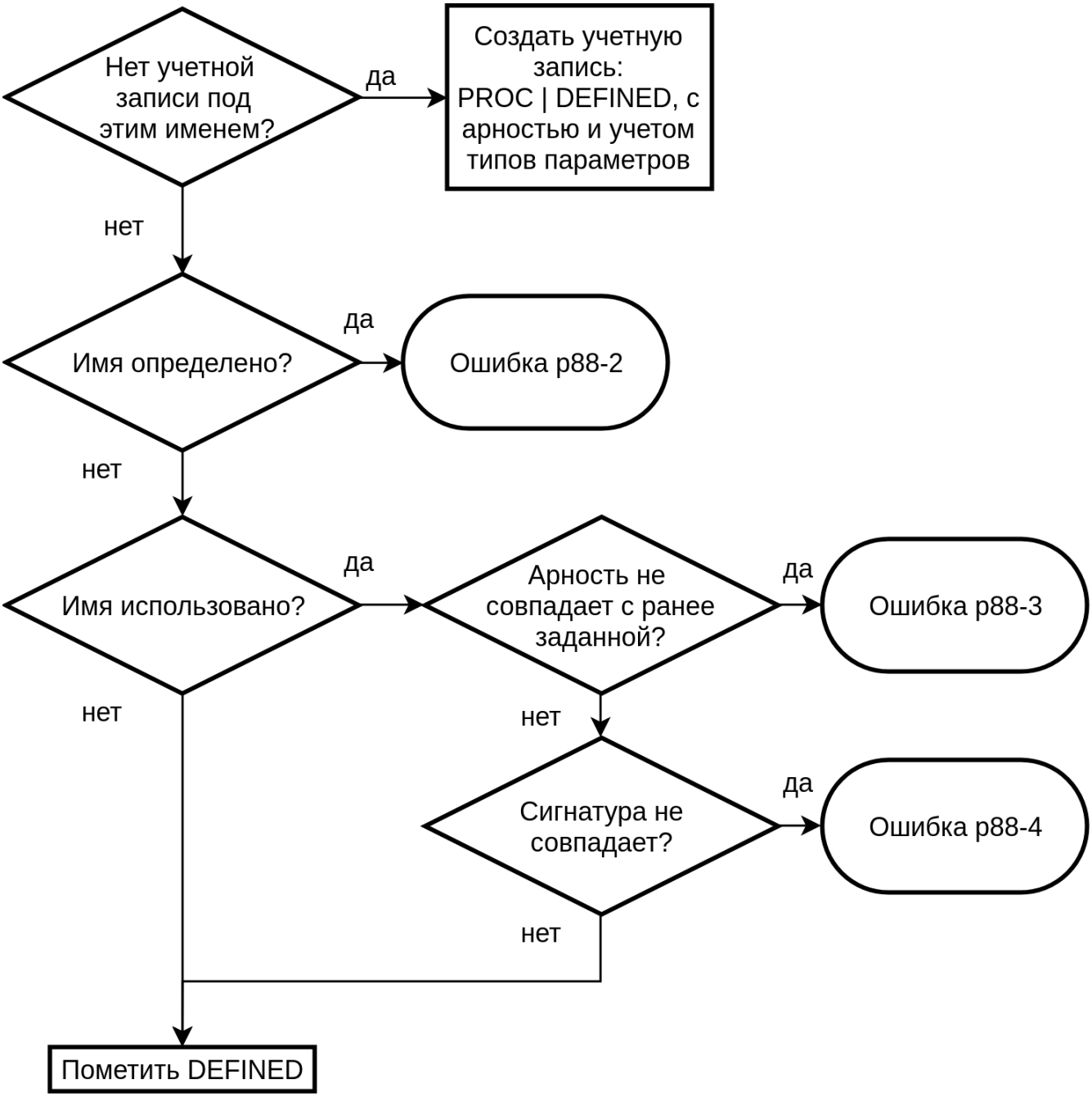
**p55**
**>**


Область видимости параметров?
да
Есть параметр с таким именем?
да
Ошибка p55-1
нет
нет
Имя не определено глобально?
да
Создать учетную запись: PROC | USED, с арностью и учетом типов параметров
нет
Арность не совпадает с ранее определенной?
да
Ошибка p55-3
нет
Не совпадает сигнатура?
да
Ошибка p55-4
нет
Пометить USED

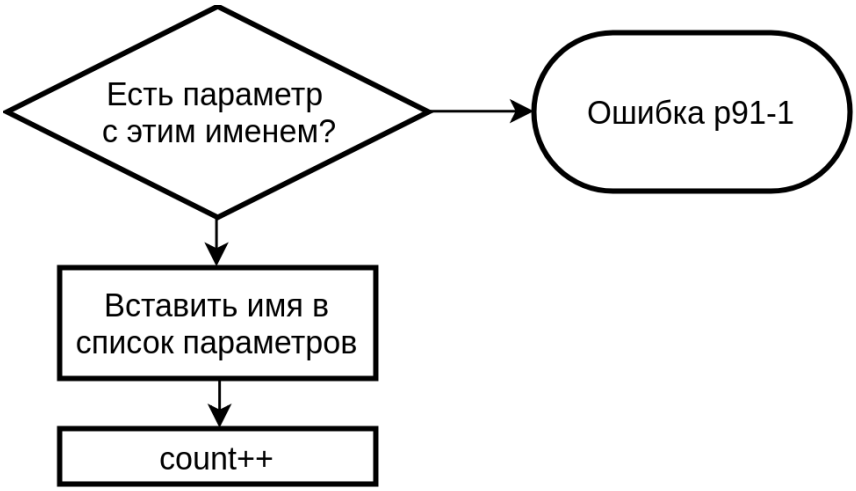
**p74**
**>**


Глобальная область видимости?
да
Существует процедура с таким именем или имя встроенное?
да
Ошибка p74-1
нет
нет
Пометить USED
Область видимости параметров?
да
Есть параметр с таким именем?
да
Ошибки нет
нет
нет
Существует процедура с таким именем или имя встроенное?
да
Ошибка p74-3
нет
Пометить USED
Область видимости параметров?
да
Есть локальная переменная с таким именем?
да
Ошибки нет
нет
Есть параметр с таким именем?
да
Ошибки нет
нет
Существует процедура с таким именем или имя встроенное?
да
Ошибка p74-6
нет
Пометить USED

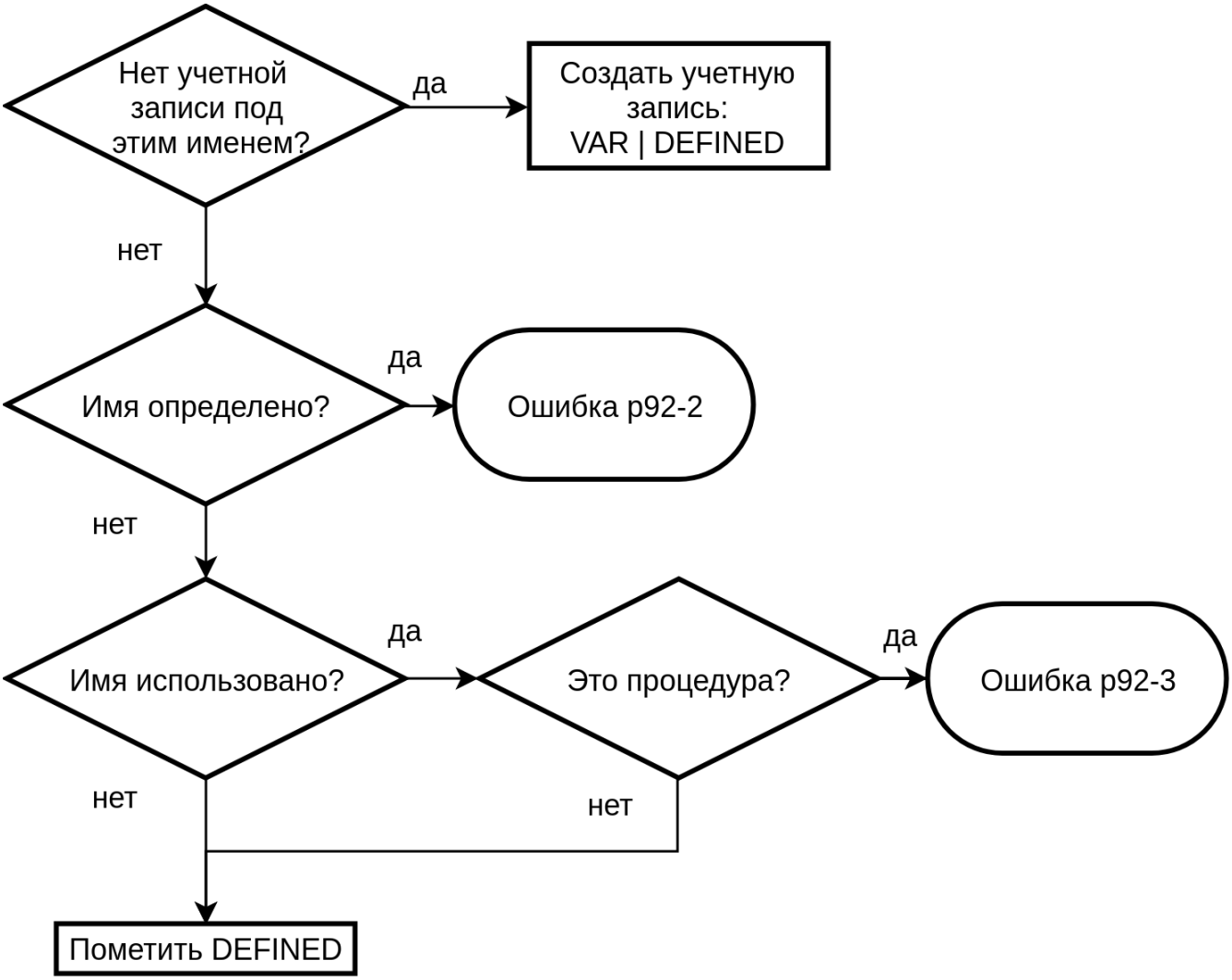
**p88**
**>**


Нет учетной записи под этим именем?
да
Создать учетную запись: PROC | DEFINED, с арностью и учетом типов параметров
нет
Имя определено?
да
Ошибка p88-2
нет
Имя использовано?
да
Арность не совпадает с ранее заданной?
да
Ошибка p88-3
нет
нет
Сигнатура не совпадает?
да
Ошибка p88-4
нет
Пометить DEFINED

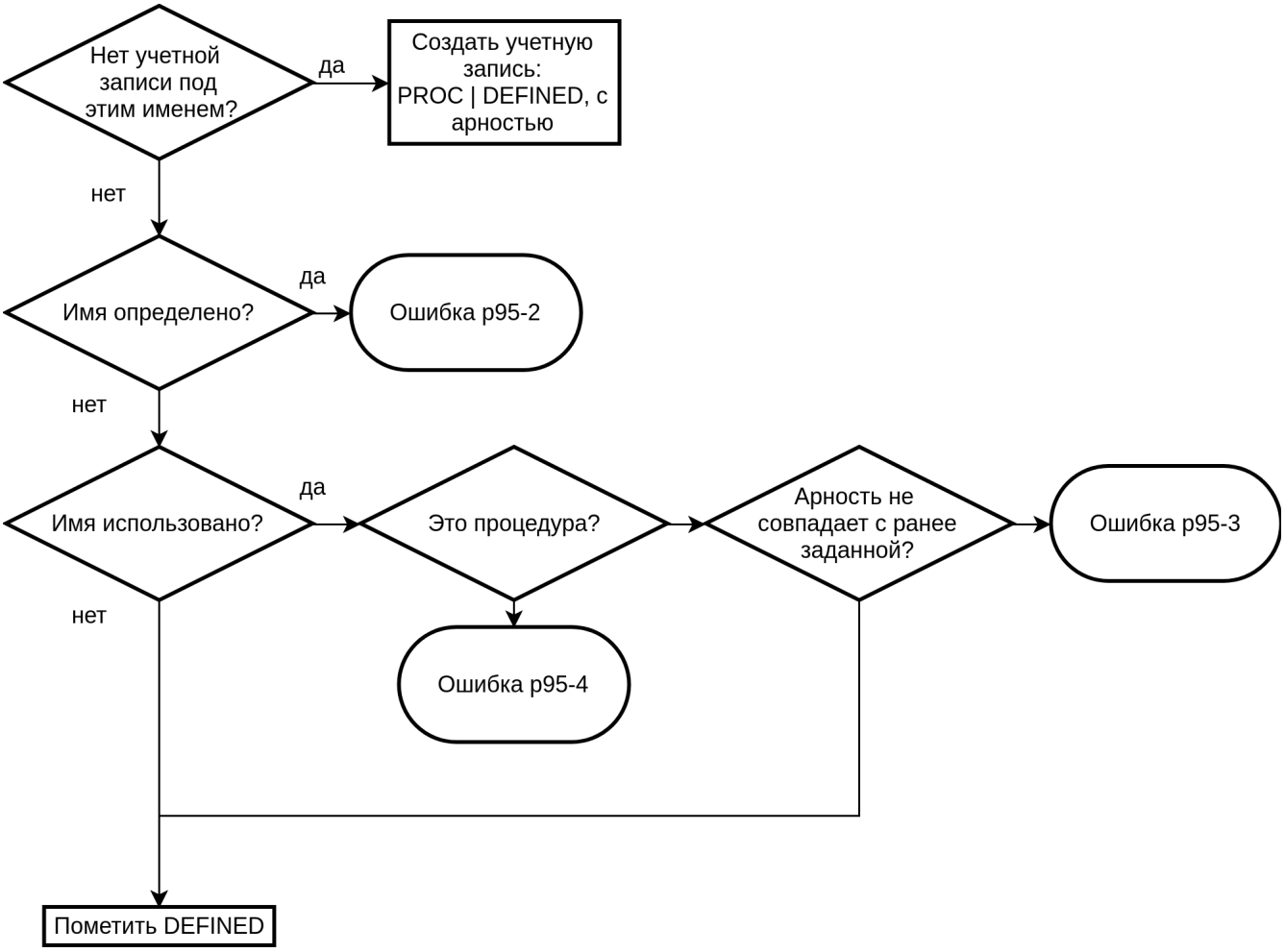
**p91**
**>**


Есть параметр
с этим именем?
Ошибка p91-1
Вставить имя в
список параметров
count++

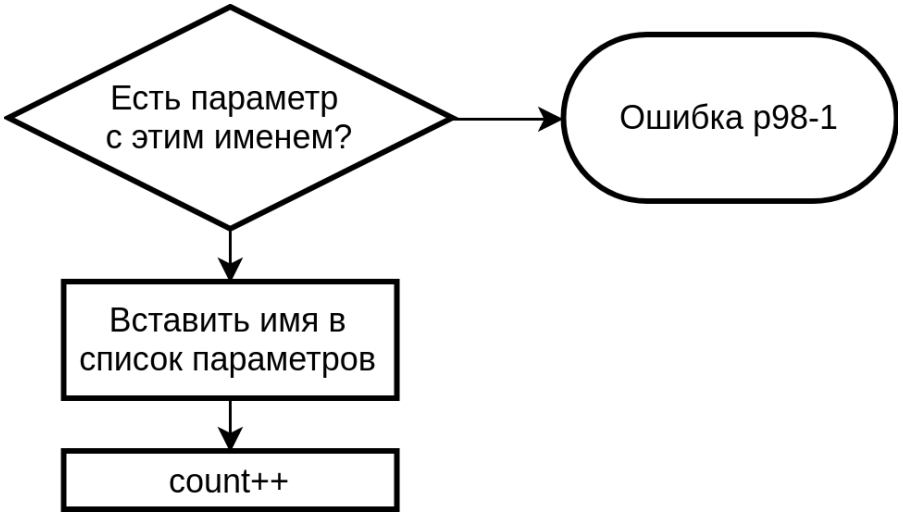
**p92**
**>**


Нет учетной
записи под
этим именем?
да
Создать учетную
запись:
VAR | DEFINED
нет
Имя определено?
да
Ошибка p92-2
нет
Имя использовано?
да
Это процедура?
да
Ошибка p92-3
нет
нет
Пометить DEFINED

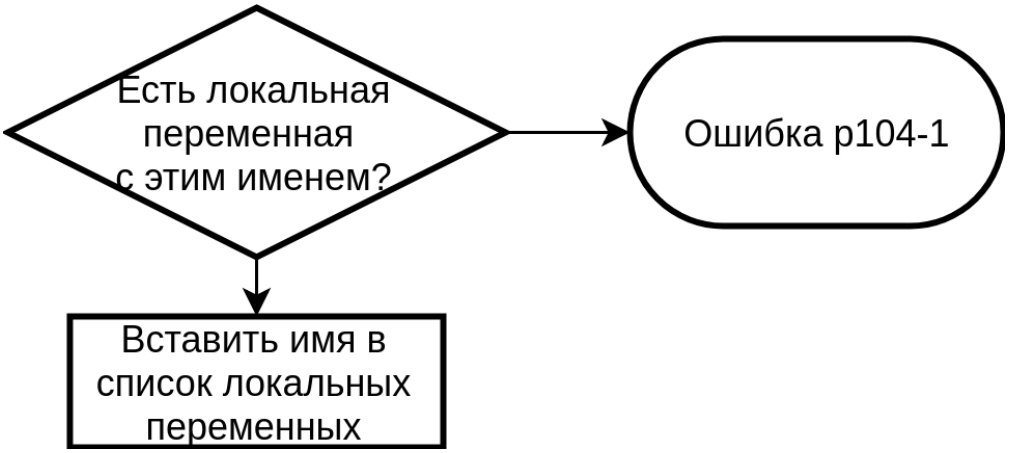
**p95**
**>**


Нет учетной записи под этим именем?
да
Создать учетную запись: PROC | DEFINED, с арностью
нет
Имя определено?
да
Ошибка p95-2
нет
Имя использовано?
да
Это процедура?
Арность не совпадает с ранее заданной?
Ошибка p95-3
нет
Ошибка p95-4
Пометить DEFINED

**p98**
**>**


Есть параметр с этим именем?
Ошибка p98-1
Вставить имя в список параметров
count++

**p104**
**>**




**Протоколы тестирования:**

**p1-1>**

```
Source>p1-1
Source:p1-1.ss
   1|(define a 4)
   2|(define b 6)
   3|
   4|(define (f x y) (* x a))
   5|(define (g x y) (f a y))
   6|

_________________
[?]Unused procedures: g
[?]Unused variables: b

Accepted!
_________________________
```

**p1-2>**

```
Source>p1-2
Source:p1-2.ss
   1|(define (f x y) (* x a))
   2|

_________________
[!]Undefined variables: a
[?]Unused procedures: f

   2|
      ^

Rejected!
_________________________
```

**p1-3>**

```
Source>p1-3
Source:p1-3.ss
   1|(define a 4)
   2|
   3|(define (g x y) (f a y))
   4|

________________
[!]Undefined procedures: f
[?]Unused procedures: g

   4|
     ^

Rejected!
_________________________
```

**p11-1>**

```
Source>p11-1
Source:p11-1.ss
   1|(define (f x y) (* x y))
   2|f
   3|

________________
[!]Numeric expression evaluation: 'f' is procedure not a variable !
   3|
     ^

Rejected!
_________________________
```

**p11-2>**

```
Source>p11-2
Source:p11-2.ss
   1|(define f 5)
   2|(define (g x y) (* x y))
   3|f
   4|

________________
[?]Unused procedures: g

Accepted!

_____________________
```

**p11-3>**

```
Source>p11-3
Source:p11-3.ss
   1|(define (f x y) x)
   2|(define (g x y) (* x y))
   3|(define (h f) f)
   4|
_________________
[?]Unused procedures: f, g, h

Accepted!
_________________________
```

**p11-4>**

```
_________________________
Source>p11-4
Source:p11-4.ss
   1|(define (f x y) x)
   2|(define (g x y) f)
   3|
_________________
[!]Numeric expression evaluation: 'f' is procedure not a variable !
   2|(define (g x y) f)
                    ^

Rejected!
_____________________
```

**p11-5>**

```
Source>p11-5
Source:p11-5.ss
   1|(define f 1)
   2|(define (g x y) f)
   3|
_________________
[?]Unused procedures: g

Accepted!
_________________________
```

**p11-6>**

```
Source>p11-6
Source:p11-6.ss
   1|(define (x a b) a)
   2|(define (g x y) (let ((x 6)) x))
   3|

_________________
[?]Unused procedures: g, x

Accepted!

_______________________
```

**p11-7>**

```
_________________________
Source>p11-7
Source:p11-7.ss
   1|(define (x a b) a)
   2|(define (g x y) (let ((h 6)) x))
   3|

_________________
[?]Unused procedures: g, x

Accepted!

_________________________
```

**p11-8>**

```
_________________________
Source>p11-8
Source:p11-8.ss
   1|(define (x a b) a)
   2|(define (g t y) (let ((h 6)) x))
   3|

_________________
[!]Numeric expression evaluation: 'x' is procedure not a variable !
   2|(define (g t y) (let ((h 6)) x))
                                  ^


Rejected!

_________________________
```

**p11-9>**

```
Source>p11-9
Source:p11-9.ss
   1|(define x 3)
   2|(define (g t y) (let ((h 6)) x))
   3|

_________________
[?]Unused procedures: g

Accepted!
```

**p45> Данная продукция, данная в качестве примера, не изменилась.**

**p49-1>**

```
Source>p49-1
Source:p49-1.ss
   1|(define (f? x? y?) q?)
   2|

_________________
[!]Boolean expression evaluation: 'q?' is inaccessible in this scope: [param] !
   1|(define (f? x? y?) q?)
                       ^

Rejected!
_________________________
```

**p49-2>**

```
Source>p49-2
Source:p49-2.ss
   1|(define (f? x? y?) x?)
   2|

_________________
[?]Unused procedures: f?

Accepted!
_________________________
```

**p55-1>**

```
Source>p55-1
Source:p55-1.ss
   1|(define (f? a? b?) (a? b?))
   2|
_________________
[!]Predicate application: parameter 'a?' shadows the predicate!
   1|(define (f? a? b?) (a? b?))
                            ^

Rejected!
_____________________
```

**p55-2>**

```
Source>p55-2
Source:p55-2.ss
   1|(define (g? x y) (f? x y))
   2|(define (f? a b) (= a b))
   3|
_________________
[?]Unused procedures: g?

Accepted!
_____________________
```

**p55-3>**

```
Source>p55-3
Source:p55-3.ss
   1|(define (f? a b) (= a b))
   2|(define (g? x y) (f? x y y))
   3|
_________________
[!]Predicate application: 'f?' expects 2 arguments, given: 3 !
   2|(define (g? x y) (f? x y y))
                             ^

Rejected!
```

**p55-4>**

```
_________________________
Source>p55-4
Source:p55-4.ss
   1|(define (f? a b c) (= a b))
   2|(define (g? x y) (f? x #t y))
   3|
_________________
[!]Predicate application: 'f?' expects [numeric expression] on position 1, given: [boolean expression] !
   2|(define (g? x y) (f? x #t y))
                          ^

Rejected!
```

**p55-5>**

```
_________________________
Source>p55-5
Source:p55-5.ss
   1|(define (f? a b c) (= a b))
   2|(define (g? x y) (f? x x y))
   3|
___________________
[?]Unused procedures: g?

Accepted!
_________________________
```

**p74-1>**

```
_________________________
Source>p74-1
Source:p74-1.ss
   1|(set! e 5)
   2|
_________________
[!]Assignment operation: 'e' is built-in constant
   2|
     ^

Rejected!
```

**p74-2>**

```
Source>p74-2
Source:p74-2.ss
   1|(define a 5)
   2|(set! a 6)
   3|

----------------

Accepted!
```

**p74-3>**

```
Source>p74-3
Source:p74-3.ss
   1|(define (g x) (* x x))
   2|(define (f x y)
   3|    (set! g 5)
   4|    x
   5|)
   6|

-----------------
[!]Assignment operation: 'g' is procedure
   4|    x
         ^

Rejected!
```

**p74-5>**

```
-------------------------
Source>p74-5
Source:p74-5.ss
   1|(define (x a b) 5)
   2|(define (f x y)
   3|    (set! x 7)
   4|    x
   5|)
   6|

-----------------
[?]Unused procedures: f, x

Accepted!
```

**p74-6>**

```
------------------------
Source>p74-6
Source:p74-6.ss
   1|(define (f x y)
   2|    (let ((t 6) (r 7))
   3|        (set! sqrt 5)
   4|        x
   5|    )
   6|)
   7|

-----------------
[!]Assignment operation: 'sqrt' is built-in procedure
   4|        x
            ^

Rejected!
```

**p74-7>**

```
------------------------
Source>p74-7
Source:p74-7.ss
   1|(define d 98)
   2|(define (f x y)
   3|    (let ((t 6) (r 7))
   4|        (set! d 5)
   5|        x
   6|    )
   7|)
   8|

-----------------
[?]Unused procedures: f

Accepted!
```

**p74-8>**

```
------------------------
Source>p74-8
Source:p74-8.ss
   1|(define (x a b) a)
   2|(define (f x y)
   3|    (let ((t 6) (r 7))
   4|        (set! x 5)
   5|        x
   6|    )
   7|)
   8|

-----------------
[?]Unused procedures: f, x

Accepted!

------------------------
```

**p74-9>**

```
------------------------
Source>p74-9
Source:p74-9.ss
   1|(define (t a b) a)
   2|(define (f x y)
   3|    (let ((t 6) (r 7))
   4|        (set! t 5)
   5|        x
   6|    )
   7|)
   8|

-----------------
[?]Unused procedures: f, t

Accepted!
```

**p88-1>**

```
________________________
Source>p88-1
Source:p88-1.ss
   1|(define (f? x y?) y?)
   2|

_________________
[?]Unused procedures: f?

Accepted!
```

**p88-2>**

```
________________________
Source>p88-2
Source:p88-2.ss
   1|(define (f? x y?) y?)
   2|(define (f? x y?) y?)
   3|

_________________
[!]Predicate definition: predicate 'f?' has been aldeady defined
   2|(define (f? x y?) y?)
                      ^

Rejected!
```

**p88-3>**

```
________________________
Source>p88-3
Source:p88-3.ss
   1|(define (g? x y?) (f? x #t y?))
   2|(define (f? x y?) y?)
   3|

_________________
[!]Predicate definition: predicate 'f?' has been already called
         with 3 arguments, given: 2 !
   2|(define (f? x y?) y?)
                      ^

Rejected!
```

**p88-4>**

```
------------------------
Source>p88-4
Source:p88-4.ss
   1|(define (g? x y?) (f? x #t y?))
   2|(define (f? x y? z) y?)
   3|

-----------------
[!]Predicate definition: predicate 'f?' has been already called
         with [boolean expression] on position 2, given: [numeric expression] !
   2|(define (f? x y? z) y?)
                        ^

Rejected!
```

**p88-5>**

```
-------------------------
Source>p88-5
Source:p88-5.ss
   1|(define (g? x y?) (f? x #t x))
   2|(define (f? x y? z) y?)
   3|

-----------------
[?]Unused procedures: g?

Accepted!
```

**p90-1>**

```
------------------------
Source>p90-1
Source:p90-1.ss
   1|(define (f? x? y x?) x?)
   2|

-----------------
[!]Procedure definition: in 'f?' duplicate parameter identifier 'x?'!
   1|(define (f? x? y x?) x?)
                       ^

Rejected!
```

**p90-2>**

```
_________________________
Source>p90-2
Source:p90-2.ss
   1|(define (f? x? y z?) x?)
   2|

_________________
[?]Unused procedures: f?

Accepted!
```

**p91-1>**

```
_________________________
Source>p91-1
Source:p91-1.ss
   1|(define (f? x y? x) y?)
   2|

_________________
[!]Procedure definition: in 'f?' duplicate parameter identifier 'x'!
   1|(define (f? x y? x) y?)
                     ^

Rejected!
```

**p91-2>**

```
_________________________
Source>p91-2
Source:p91-2.ss
   1|(define (f? x y? z) y?)
   2|

_________________
[?]Unused procedures: f?

Accepted!

_______________________
```

**p92-1>**

```
________________________
Source>p92-1
Source:p92-1.ss
   1|(define a 5)
   2|

_________________
[?]Unused variables: a

Accepted!
```

**p92-2>**

```
________________________
Source>p92-2
Source:p92-2.ss
   1|(define a 5)
   2|(define a 5)
   3|

_________________
[!]Global variable definition: variable 'a' has been aldeady defined
   3|
     ^

Rejected!
```

**p92-3>**

```
Source>p92-3
Source:p92-3.ss
   1|(define (f x y) (a 5))
   2|(define a 5)
   3|

_________________
[!]Global variable definition: procedure 'a' has been already used
   3|
     ^

Rejected!
```

**p92-4>**

```
Source>p92-4
Source:p92-4.ss
   1|(define (f x y) a)
   2|(define a 5)
   3|

_________________
[?]Unused procedures: f

Accepted!
```

**p95-1>**

```
_________________________
Source>p95-1
Source:p95-1.ss
   1|(define (f x y) (* x y))
   2|

_________________
[?]Unused procedures: f

Accepted!
```

**p95-2>**

```
Source>p95-2
Source:p95-2.ss
   1|(define f 4)
   2|(define (f x y) (* x y))
   3|

_________________
[!]Procedure definition: variable 'f' has been aldeady defined
   2|(define (f x y) (* x y))
                    ^

Rejected!
```

**p95-3>**

```
________________________
Source>p95-3
Source:p95-3.ss
   1|(define (g x y) (f x y 10))
   2|(define (f x y) (* x y))
   3|

_________________
[!]Procedure definition: procedure 'f' has been already called
         with 3 arguments, given: 2 !
   2|(define (f x y) (* x y))
                    ^

Rejected!
```

**p95-4>**

```
________________________
Source>p95-4
Source:p95-4.ss
   1|(define (g x y) f)
   2|(define (f x y) (* x y))
   3|

_________________
[!]Procedure definition: variable 'f' has been already used
   2|(define (f x y) (* x y))
                    ^

Rejected!
```

**p95-5>**

```
________________________
Source>p95-5
Source:p95-5.ss
   1|(define (g x y) (f x y))
   2|(define (f x y) (* x y))
   3|

_________________
[?]Unused procedures: g

Accepted!
```

**p98> Данная продукция, данная в качестве примера, не изменилась.**

**p104-1>**

```
Source>p104-1
Source:p104-1.ss
   1|(define (f x y)
   2|    (let
   3|        ((t 5) (r 6) (t 1))
   4|        t
   5|    )
   6|)
   7|

_________________
[!]Local variable definition: duplicate local vatiable identifier 't'!
   3|        ((t 5) (r 6) (t 1))
                         ^

Rejected!
```

**p104-2>**

```
_________________________
Source>p104-2
Source:p104-2.ss
   1|(define (f x y)
   2|    (let
   3|        ((t 5) (r 6) (q 1))
   4|        t
   5|    )
   6|)
   7|

_________________
[?]Unused procedures: f

Accepted!
```

**Выводы по проделанной работе**
**>**

**Каждый язык программирования имеет четко заданные семантические соглашения, которые не могут быть проверены на этапе синтаксического разбора в виду сложности реализации такого алгоритма, или же просто выходящие за рамки синтаксических соглашений. Именно для этого и предназначен семантический анализатор.**

**В ходе выполнения курсового проекта был разработан подобный семантический анализатор языка МИКРОЛИСП.**

**Для каждого алгоритма  анализа составлен набор тестов, проверяющих его корректность и полноту.**

**Так же для наиболее сложных из них составлена блок-схема, призванная облегчить понимание алгоритма.**

**Сам процесс работы оказался сложнее, чем в первой части курсовой работы, возможно, из-за обилия случаев, каждый из которых нужно расписать отдельно и по возможности учесть все. Так же алгоритмы довольно сильно взаимосвязаны между собой, и изменения в одном, могут повлечь изменения в нескольких других. Это усложняло процесс тестирования.**

**Стоит отметить, что в настоящее время существуют довольно продвинутые синтаксические анализаторы. Например,  IDE CLion может распознавать бесконечную рекурсию и бесконечные циклы, что пригодилось даже на протяжении данного курса при изучении основ МИКРОЛИСПа и ручной трансляции программ с него, на C++.**

**В результате получен готовый прототип семантического анализатора языка МИКРОЛИСП, реализующий (вроде бы) все заданные семантические правила. По крайней мере, довольно плотное тестирование во время и после разработки серьезных проблем не выявило.**

**Задание второй части курсового проекта выполнено в полном объеме.**