**Студент: Моисеенков И. П.**

**Группа: М8О-208Б-19**

**Номер по списку: 21**

**«СИСТЕМЫ ПРОГРАММИРОВАНИЯ»**

**Курсовая работа 2021.**

**Часть 2.**

**Для заданного в Лабораторной №8 диалекта языка МИКРОЛИСП разработайте семантический анализатор, применяя методику Лабораторной №10, Правила SemanticRules.rtf и MessageForms.rtf**

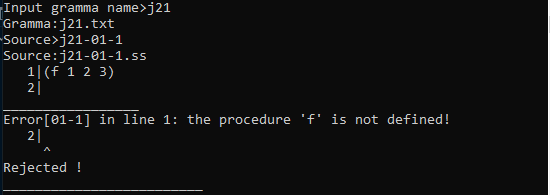
**Шаблон файла semantics.cpp создайте с помощью приложения Make-semantics.cpp .**

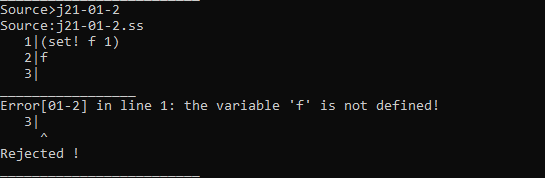
**Разработайте сценарии тестирования алгоритмов анализа.**

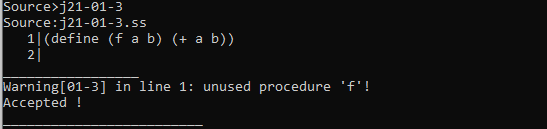
**Перечень документов в отчете.**

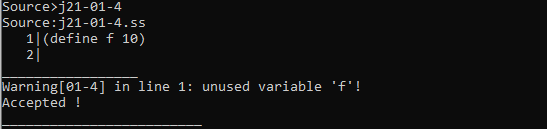
**Вариант грамматики: j21**

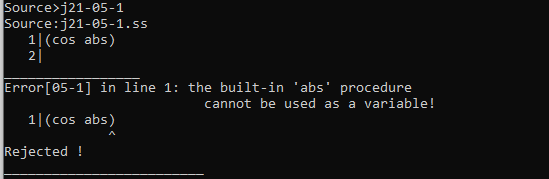
**Скриншоты всех тестов, упорядоченные по номерам продукций и сообщений.**

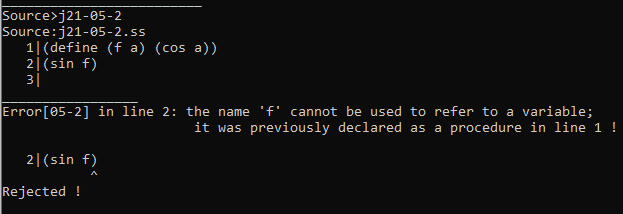
****

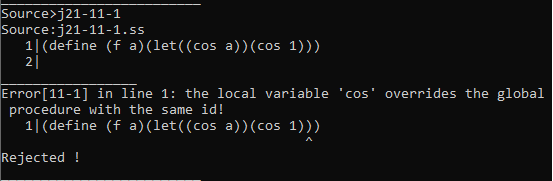
****

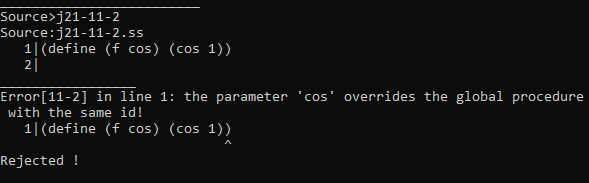
****

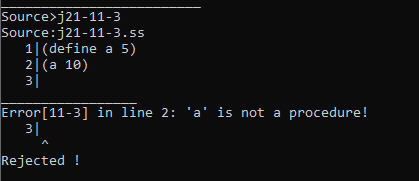
****

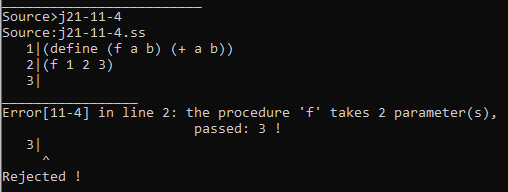
****

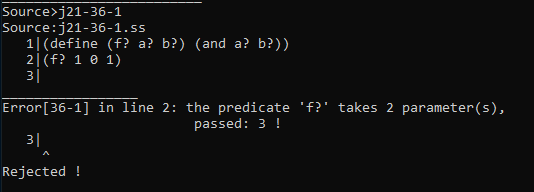
****

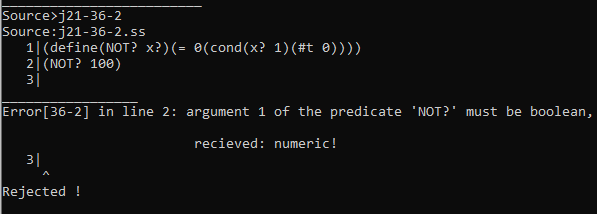
****

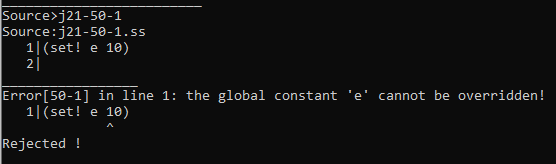
****

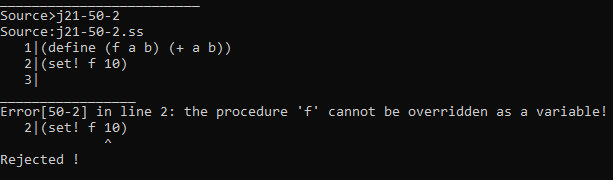
****

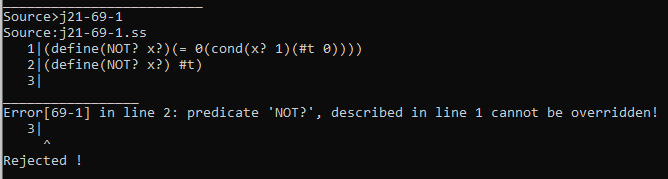
****

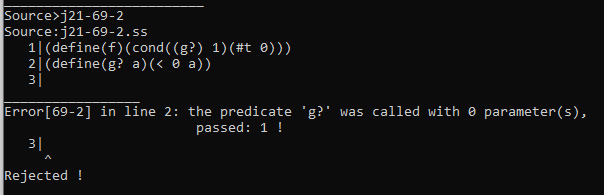
****

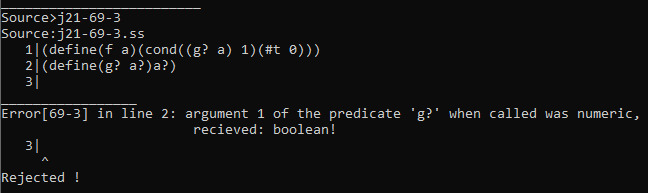
****

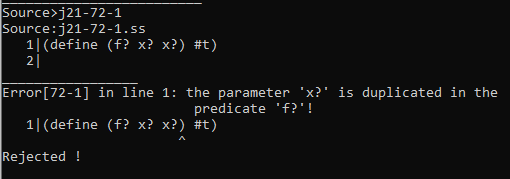
****

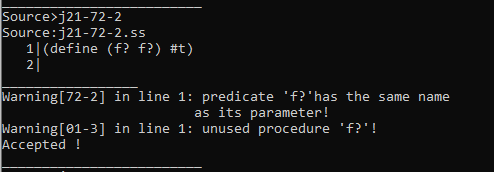
****

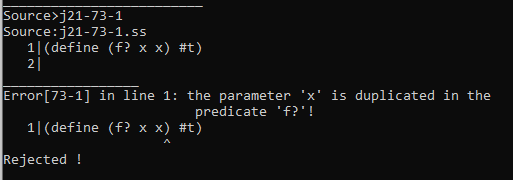
****

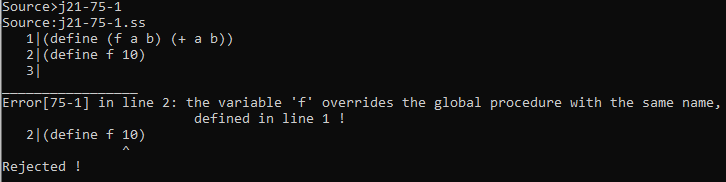
****

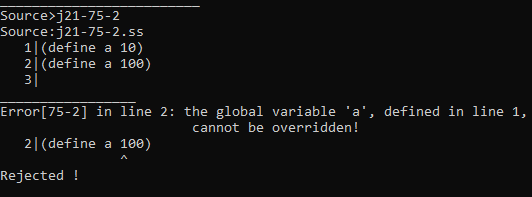
****

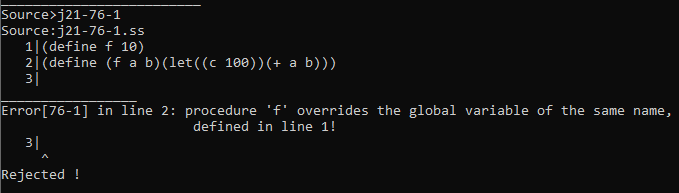
****

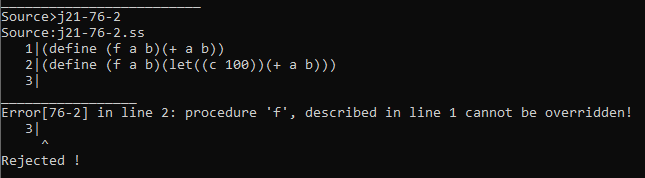
****

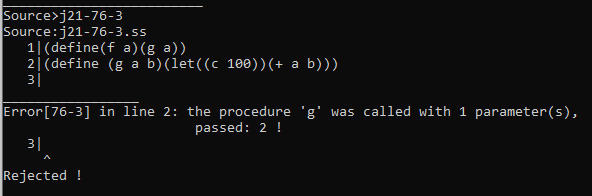
****

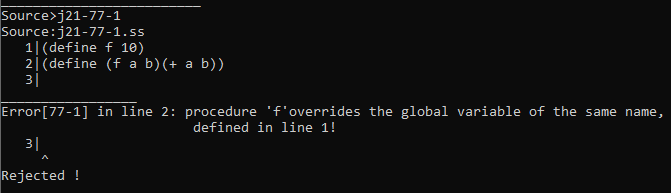
****

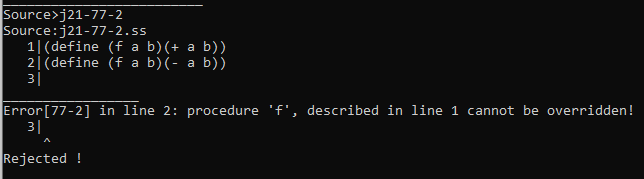
****

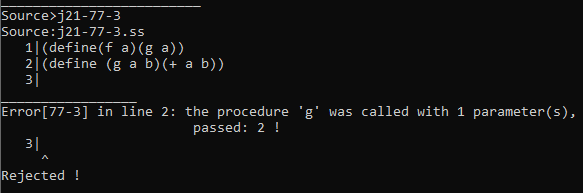
****

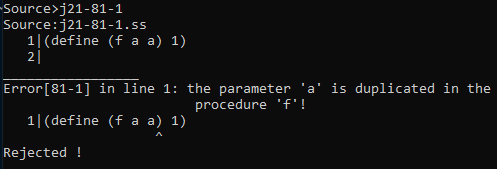
****

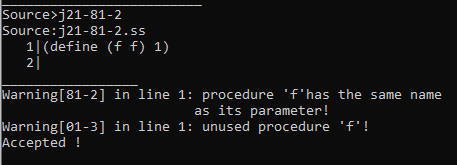
****

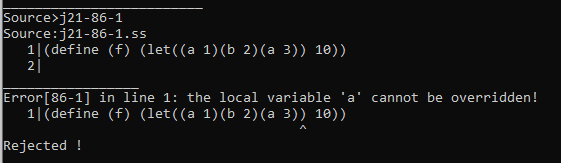
****

****

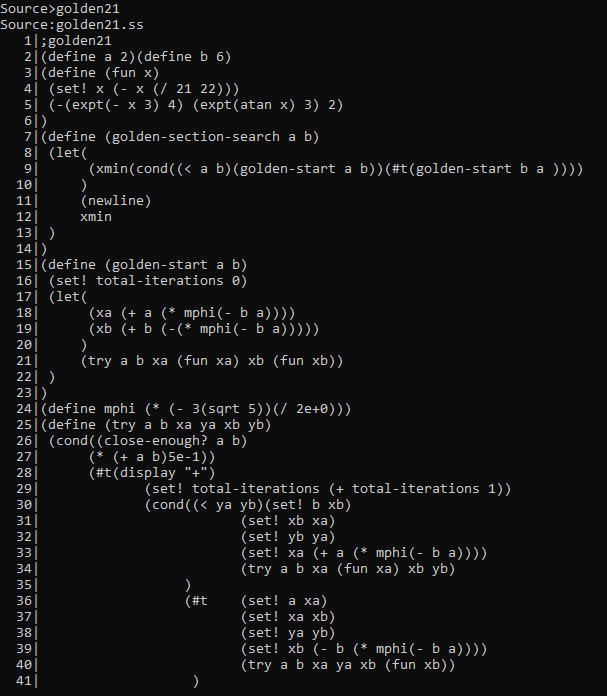
****

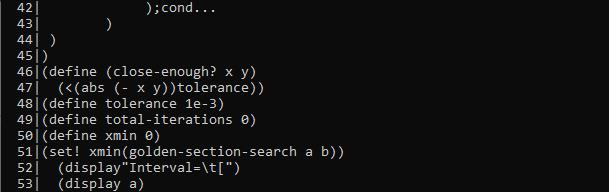
****

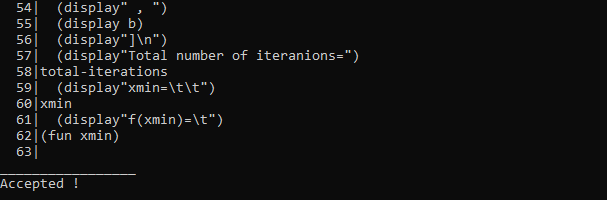
****

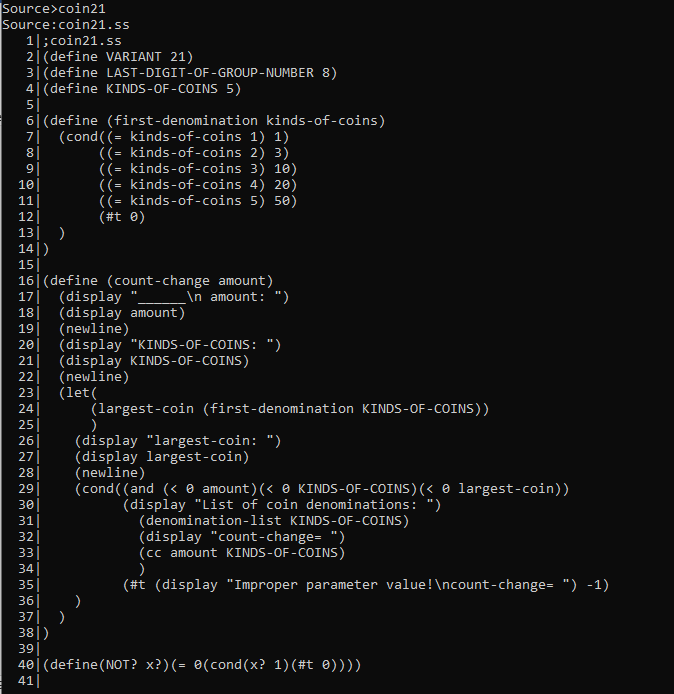
****

**Полные скриншоты анализа своих вариантов программ golden21 и coin21**

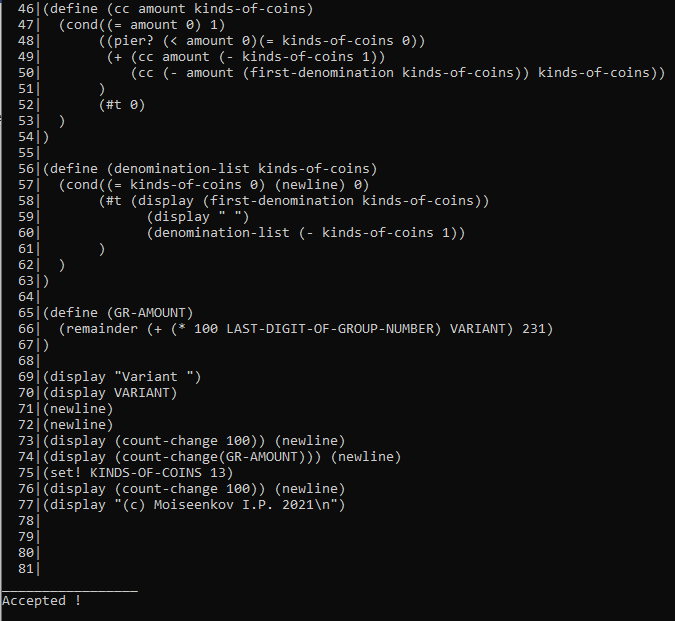
****

****

****

****

****

****

**Распечатка файла semantics.cpp.**

**/\* $j21 \*/  
#include "semantics.h"  
using namespace std;  
void tSM::init(){  
 globals.clear();  
 locals.clear();  
 params.clear();  
 scope = 0;**

**// константы:  
 globals["e"] = tgName(VAR | DEFINED | BUILT);  
 globals["pi"] = tgName(VAR | DEFINED | BUILT);  
  
 // предопределенные процедуры:  
 globals["abs"] = tgName(PROC | DEFINED | BUILT, "", 1);  
 globals["atan"] = tgName(PROC | DEFINED | BUILT, "", 1);  
 globals["cos"] = tgName(PROC | DEFINED | BUILT, "", 1);  
 globals["exp"] = tgName(PROC | DEFINED | BUILT, "", 1);  
 globals["expt"] = tgName(PROC | DEFINED | BUILT, "", 2);  
 globals["log"] = tgName(PROC | DEFINED | BUILT, "", 1);  
 globals["quotient"] = tgName(PROC | DEFINED | BUILT, "", 2);  
 globals["remainder"] = tgName(PROC | DEFINED | BUILT, "", 2);  
 globals["sin"] = tgName(PROC | DEFINED | BUILT, "", 1);  
 globals["sqrt"] = tgName(PROC | DEFINED | BUILT, "", 1);  
 globals["tan"] = tgName(PROC | DEFINED | BUILT, "", 1);  
 globals["display"] = tgName(PROC | DEFINED | BUILT, "", 1);  
 globals["newline"] = tgName(PROC | DEFINED | BUILT, "", 1);  
}  
int tSM::p01(){ // S -> PROG  
 bool error = false;  
 for (auto it = globals.begin(); it != globals.end(); ++it) {  
 if (it->second.test(PROC) && it->second.test(USED) && !it->second.test(DEFINED)) {  
 ferror\_message += "Error[01-1] in line " + it->second.line + ": the procedure '" +  
 it->first + "' is not defined!\n";  
 // процедура 'f' не определена  
 // the procedure 'f' is not defined  
 error = true;  
 }  
 else if (it->second.test(VAR) && it->second.test(USED) && !it->second.test(DEFINED)) {  
 ferror\_message += "Error[01-2] in line " + it->second.line + ": the variable '" +  
 it->first + "' is not defined!\n";  
 // переменная 'f' не определена  
 // the variable 'f' is not defined  
 error = true;  
 }  
 else if (it->second.test(PROC) && !it->second.test(USED)  
 && it->second.test(DEFINED) && !it->second.test(BUILT)) {  
 ferror\_message += "Warning[01-3] in line " + it->second.line + ": unused procedure '" +  
 it->first + "'!\n";  
 // неиспользуемая процедура 'f'  
 // unused procedure 'f'  
 }  
 else if (it->second.test(VAR) && !it->second.test(USED)  
 && it->second.test(DEFINED) && !it->second.test(BUILT)) {  
 ferror\_message += "Warning[01-4] in line " + it->second.line + ": unused variable '" +  
 it->first + "'!\n";  
 // неиспользуемая переменная 'f'  
 // unused variable 'f'  
 }  
 }  
 if (error) {  
 return 1;  
 }  
 return 0;  
}  
int tSM::p02(){ // PROG -> CALCS  
 return 0;}  
int tSM::p03(){ // PROG -> DEFS  
 return 0;}  
int tSM::p04(){ // PROG -> DEFS CALCS  
 return 0;}  
int tSM::p05(){ // E -> $id  
 string name = S1->name;  
 switch (scope) {  
 case 2:  
 if (locals.count(name)) {  
 // переменная есть в локальной области видимости -> все ок  
 break;  
 }  
 case 1:  
 if (params.count(name)) {  
 // переменная есть среди параметров функции -> все ок  
 break;  
 }  
 default:  
 tgName &ref = globals[name];  
 if (ref.empty()) {  
 // переменная встретилась впервые -> создаем ее и помечаем как использованную  
 ref = tgName(VAR | USED, S1->line);  
 break;  
 }  
 if (ref.test(VAR)) {  
 // переменная уже встречалась до этого -> помечаем как использованную  
 ref.set(USED);  
 break;  
 }  
 // это не переменная  
 if (ref.test(BUILT)) {  
 ferror\_message +=  
 "Error[05-1] in line " + S1->line + ": the built-in '"  
 + name +  
 "' procedure \n\t\t\t cannot be used as a variable!\n";  
 // встроенную процедуру 'abs' нельзя использовать в качестве переменной  
 // the built-in 'abs' procedure cannot be used as a variable  
 return 1;  
 }  
  
 ferror\_message +=  
 "Error[05-2] 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;  
 }  
 return 0;  
}  
int tSM::p06(){ // E -> $int  
 return 0;}  
int tSM::p07(){ // E -> $dec  
 return 0;}  
int tSM::p08(){ // E -> AREX  
 return 0;}  
int tSM::p09(){ // E -> COND  
 return 0;}  
int tSM::p10() { // E -> CPROC  
 return 0;}  
int tSM::p11() { // CPROC -> HCPROC )  
 string name = S1->name;  
 switch (scope) {  
 case 2:  
 if (locals.count(name)) {  
 // в локальной области видимости есть переменная с таким же именем -> ошибка  
 ferror\_message +=  
 "Error[11-1] in line " + S1->line + ": the local variable '"  
 + name +  
 "' overrides the global procedure with the same id!\n";  
 // локальная переменная 'f' перекрывает глобальную процедуру с тем же идентификатором  
 // the local variable 'f' overrides the global procedure with the same id  
 return 1;  
 }  
 case 1:  
 if (params.count(name)) {  
 // среди параметров функции есть переменная с таким же именем -> ошибка  
 ferror\_message +=  
 "Error[11-2] in line " + S1->line + ": the parameter '"  
 + name +  
 "' overrides the global procedure with the same id!\n";  
 // параметр 'f' перекрывает глобальную процедуру с тем же идентификатором  
 // the parameter 'f' overrides the global procedure with the same id  
 return 1;  
 }  
 default:  
 tgName &ref = globals[name];  
 if (ref.empty()) {  
 // процедура встретилась впервые -> добавляем и помечаем как использованную  
 ref = tgName(PROC | USED, S1->line, S1->count);  
 }  
 if (ref.test(VAR)) {  
 // встретили переменную вместо процедуры -> ошибка  
 ferror\_message +=  
 "Error[11-3] in line " + S1->line + ": '" + name +  
 "' is not a procedure!\n";  
 // 'f' не является процедурой  
 // 'f' is not a procedure  
 return 1;  
 }  
 if (ref.arity != S1->count) {  
 // количество аргументов не совпадает с количеством переданных параметров -> ошибка  
 ferror\_message +=  
 "Error[11-4] in line " + S1->line + ": the procedure '"  
 + name + "' takes " + Uint\_to\_str(ref.arity) + " parameter(s),\n\t\t\tpassed: " +  
 Uint\_to\_str(S1->count) + " !\n";  
 // процедура 'f' принимает 2 параметра, передано: 3  
 // the procedure 'f' takes 2 parameters, passed: 3  
 return 1;  
 }  
 // помечаем как использованную  
 ref.set(USED);  
 }  
 return 0;  
}  
int tSM::p12(){ // HCPROC -> ( $id  
 S1->types = 0;  
 S1->name = S2->name;  
 S1->count = 0;  
 return 0;  
}  
int tSM::p13(){ // HCPROC -> HCPROC E  
 ++S1->count;  
 return 0;  
}  
int tSM::p14(){ // AREX -> HAREX E )  
 return 0;}  
int tSM::p15(){ // HAREX -> ( AROP  
 return 0;}  
int tSM::p16(){ // HAREX -> HAREX E  
 return 0;}  
int tSM::p17(){ // AROP -> +  
 return 0;}  
int tSM::p18(){ // AROP -> -  
 return 0;}  
int tSM::p19(){ // AROP -> \*  
 return 0;}  
int tSM::p20(){ // AROP -> /  
 return 0;}  
int tSM::p21(){ // COND -> ( cond BRANCHES )  
 return 0;}  
int tSM::p22(){ // BRANCHES -> CLAUS  
 return 0;}  
int tSM::p23(){ // BRANCHES -> CLAUS BRANCHES  
 return 0;}  
int tSM::p24(){ // CLAUS -> ( BOOL CLAUSB )  
 return 0;}  
int tSM::p25(){ // CLAUSB -> E  
 return 0;}  
int tSM::p26(){ // CLAUSB -> INTER CLAUSB  
 return 0;}  
int tSM::p27(){ // STR -> $str  
 return 0;}  
int tSM::p28(){ // STR -> SIF  
 return 0;}  
int tSM::p29(){ // SIF -> ( if BOOL STR STR )  
 return 0;}  
int tSM::p30(){ // BOOL -> $bool  
 return 0;}  
int tSM::p31(){ // BOOL -> $idq  
 return 0;}  
int tSM::p32(){ // BOOL -> REL  
 return 0;}  
int tSM::p33(){ // BOOL -> OR  
 return 0;}  
int tSM::p34(){ // BOOL -> AND  
 return 0;}  
int tSM::p35(){ // BOOL -> CPRED  
 return 0;}  
int tSM::p36() { // CPRED -> HCPRED )  
 string name = S1->name;  
 tgName &ref = globals[name];  
 if (ref.empty()) {  
 // предикат встретился впервые -> добавляем и помечаем как использованный  
 ref = tgName(PROC | USED, S1->line, S1->count);  
 return 0;  
 }  
 if (ref.arity != S1->count) {  
 // количество аргументов не совпадает с количеством переданных параметров -> ошибка  
 ferror\_message +=  
 "Error[36-1] in line " + S1->line + ": the predicate '"  
 + name + "' takes " + Uint\_to\_str(ref.arity) + " parameter(s),\n\t\t\tpassed: " +  
 Uint\_to\_str(S1->count) + " !\n";  
 // предикат 'f?' принимает 2 параметра, передано: 3  
 // the predicate 'f?' takes 2 parameters, passed: 3  
 return 1;  
 }  
 if (ref.types != S1->types) {  
 // типы аргументов не совпадают с типами переднаных параметров -> ошибка  
 int bad\_param\_num = -1;  
 for (int i = 0; i < S1->count; ++i) {  
 int type1 = ref.types & (1 << i);  
 int type2 = S1->types & (1 << i);  
 if (type1 != type2) {  
 bad\_param\_num = i;  
 break;  
 }  
 }  
 ferror\_message +=  
 "Error[36-2] in line " + S1->line + ": argument " + Uint\_to\_str(bad\_param\_num + 1) +  
 " of the predicate '" + name + "' must be " +  
 ((ref.types & (1 << bad\_param\_num)) == 0 ? "numeric" : "boolean") + ",\n\t\t\trecieved: " +  
 ((S1->types & (1 << bad\_param\_num)) == 0 ? "numeric" : "boolean") + "!\n";  
 // аргумент 3 предиката 'f?' должен быть числовым, получен: булевский  
 // argument 3 of the predicate 'f?' must be numeric, received: boolean  
 return 1;  
 }  
 // помечаем как использованный  
 globals[S1->name].set(USED);  
 return 0;  
}  
int tSM::p37(){ // HCPRED -> ( $idq  
 S1->name = S2->name;  
 S1->count = 0;  
 return 0;  
}  
int tSM::p38(){ // HCPRED -> HCPRED ARG  
 S1->types |= S2->types << S1->count;  
 ++S1->count;  
 return 0;  
}  
int tSM::p39(){ // ARG -> E  
 S1->types = 0;  
 return 0;  
}  
int tSM::p40(){ // ARG -> BOOL  
 S1->types = 1;  
 return 0;  
}  
int tSM::p41(){ // REL -> ( = E E )  
 return 0;}  
int tSM::p42(){ // REL -> ( < E E )  
 return 0;}  
int tSM::p43(){ // OR -> HOR BOOL )  
 return 0;}  
int tSM::p44(){ // HOR -> ( or  
 return 0;}  
int tSM::p45(){ // HOR -> HOR BOOL  
 return 0;}  
int tSM::p46(){ // AND -> HAND BOOL )  
 return 0;}  
int tSM::p47(){ // HAND -> ( and  
 return 0;}  
int tSM::p48(){ // HAND -> HAND BOOL  
 return 0;}  
int tSM::p49(){ // SET -> HSET E )  
 return 0;}  
int tSM::p50(){ // HSET -> ( set! $id  
 string name = S3->name;  
 switch (scope) {  
 case 2:  
 if (locals.count(name)) {  
 // переменная есть в локальной области видимости -> все ок  
 break;  
 }  
 case 1:  
 if (params.count(name)) {  
 // переменная есть среди параметров -> все ок  
 break;  
 }  
 default:  
 tgName &ref = globals[name];  
 if (ref.empty()) {  
 // переменная встретилась впервые -> создаем и помечаем как использованную  
 ref = tgName(VAR | USED, S1->line);  
 break;  
 }  
 if (ref.test(VAR) && ref.test(BUILT)) {  
 // пытаемся переопределить глобальную константу -> ошибка  
 ferror\_message +=  
 "Error[50-1] in line " + S1->line + ": the global constant '" +  
 name +  
 "' cannot be overridden!\n";  
 // глобальную константу 'pi' нельзя переопределить  
 // the global constant 'pi' cannot be overridden  
 return 1;  
 }  
 if (ref.test(PROC)) {  
 // пытаемся присвоить значение процедуре -> ошибка  
 ferror\_message +=  
 "Error[50-2] in line " + S1->line + ": the procedure '" +  
 name +  
 "' cannot be overridden as a variable!\n";  
 // процедура 'f' не может быть переопределена как переменная  
 // the procedure 'f' cannot be overridden as a variable  
 return 1;  
 }  
 }  
 // помечаем переменную, как использованную  
 globals[name].set(USED);  
 return 0;  
}  
int tSM::p51(){ // DISPSET -> ( display E )  
 return 0;}  
int tSM::p52(){ // DISPSET -> ( display BOOL )  
 return 0;}**

**int tSM::p53(){ // DISPSET -> ( display STR )  
 return 0;}  
int tSM::p54(){ // DISPSET -> ( newline )  
 return 0;}  
int tSM::p55(){ // DISPSET -> SET  
 return 0;}  
int tSM::p56(){ // INTER -> DISPSET  
 return 0;}  
int tSM::p57(){ // INTER -> E  
 return 0;}  
int tSM::p58(){ // CALCS -> CALC  
 return 0;}  
int tSM::p59(){ // CALCS -> CALCS CALC  
 return 0;}  
int tSM::p60(){ // CALC -> E  
 return 0;}  
int tSM::p61(){ // CALC -> BOOL  
 return 0;}  
int tSM::p62(){ // CALC -> STR  
 return 0;}  
int tSM::p63(){ // CALC -> DISPSET  
 return 0;}  
int tSM::p64(){ // DEFS -> DEF  
 return 0;}  
int tSM::p65(){ // DEFS -> DEFS DEF  
 return 0;}  
int tSM::p66(){ // DEF -> PRED  
 return 0;}  
int tSM::p67(){ // DEF -> VAR  
 return 0;}  
int tSM::p68(){ // DEF -> PROC  
 return 0;}  
int tSM::p69(){ // PRED -> HPRED BOOL )  
 string name = S1->name;  
 tgName& ref = globals[name];  
 if (ref.empty()) {  
 // предикат определяется впервые -> создаем учетную запись  
 ref = tgName(PROC | DEFINED, S1->line, S1->count, S1->types);  
 // и возвращаемся обратно в глобальную область видимости  
 scope = 0;  
 params.clear();  
 return 0;  
 }  
 if (ref.test(DEFINED)) {  
 // предикат уже был определен -> ошибка  
 ferror\_message +=  
 "Error[69-1] in line " + S1->line + ": predicate '" +  
 name +  
 "', described in line " + ref.line + " cannot be overridden!\n";  
 // предикат 'f?', описанный в строке 1, не может быть переопределен  
 // predicate 'f?', described in line 1, cannot be overridden  
 return 1;  
 }  
 if (ref.test(USED) && ref.arity != S1->count) {  
 // количество аргументов не совпадает с количеством переданных параметров -> ошибка  
 ferror\_message +=  
 "Error[69-2] in line " + S1->line + ": the predicate '"  
 + name + "' was called with " + Uint\_to\_str(ref.arity) + " parameter(s),\n\t\t\tpassed: " +  
 Uint\_to\_str(S1->count) + " !\n";  
 // предикат 'f?' был вызван с 2 параметрами, передано: 3  
 // predicate 'f?' was called with 2 parameters, passed: 3  
 return 1;  
 }  
 if (ref.test(USED) && ref.types != S1->types) {  
 // типы аргументов не совпадают с типами переднаных параметров -> ошибка  
 int bad\_param\_num = -1;  
 for (int i = 0; i < S1->count; ++i) {  
 int type1 = ref.types & (1 << i);  
 int type2 = S1->types & (1 << i);  
 if (type1 != type2) {  
 bad\_param\_num = i;  
 break;  
 }  
 }  
 ferror\_message +=  
 "Error[69-3] in line " + S1->line + ": argument " + Uint\_to\_str(bad\_param\_num + 1) +  
 " of the predicate '" + name + "' when called was " +  
 ((ref.types & (1 << bad\_param\_num)) == 0 ? "numeric" : "boolean") + ",\n\t\t\trecieved: " +  
 ((S1->types & (1 << bad\_param\_num)) == 0 ? "numeric" : "boolean") + "!\n";  
 // аргумент 3 предиката 'f?' при вызове был числовым, получен: булевский  
 // argument 3 of the predicate 'f?' when called was numeric, received: boolean  
 return 1;  
 }  
 // помечаем предикат как определенный  
 ref.set(DEFINED);  
 // и меняем область видимости  
 scope = 0;  
 params.clear();  
 return 0;  
}  
int tSM::p70(){ // HPRED -> PDPAR )  
 scope = 1;  
 return 0;  
}  
int tSM::p71(){ // PDPAR -> ( define ( $idq  
 S1->name = S4->name;  
 S1->count = 0;  
 return 0;  
}  
int tSM::p72(){ // PDPAR -> PDPAR $idq  
 if (params.count(S2->name)) {  
 ferror\_message +=  
 "Error[72-1] in line " + S2->line + ": the parameter '"  
 + S2->name +  
 "' is duplicated in the\n\t\t\tpredicate '"  
 + S1->name + "'!\n";  
 // в предикате 'f?' дублируется параметр 'x?'  
 // the parameter 'x?' is duplicated in the predicate 'f?'  
 return 1;  
 }  
  
 if (S2->name == S1->name) {  
 ferror\_message +=  
 "Warning[72-2] in line " + S2->line + ": predicate '"  
 + S1->name +  
 "'has the same name \n"  
 "\t\t\tas its parameter!\n";  
 // у предиката 'f?' такое же имя, как у его параметра  
 // predicate 'f?' has the same name as its parameter  
 }  
 // добавляем параметр в список  
 S1->types |= 1 << S1->count;  
 params.insert(S2->name);  
 ++S1->count;  
 return 0;  
}  
  
int tSM::p73(){ // PDPAR -> PDPAR $id  
 if (params.count(S2->name)) {  
 ferror\_message +=  
 "Error[73-1] in line " + S2->line + ": the parameter '"  
 + S2->name +  
 "' is duplicated in the\n\t\t\tpredicate '"  
 + S1->name + "'!\n";  
 // в предикате 'f?' дублируется параметр 'x'  
 // the parameter 'x' is duplicated in the predicate 'f'  
 return 1;  
 }  
 // добавляем параметр в список  
 S1->types |= 0 << S1->count;  
 params.insert(S2->name);  
 ++S1->count;  
 return 0;  
}  
int tSM::p74(){ // VAR -> VARDCL E )  
 return 0;}  
int tSM::p75() { // VARDCL -> ( define $id  
 string name = S3->name;  
 tgName& ref = globals[name];  
 if (ref.empty()) {  
 // переменная до этого не была определена -> создаем новую учетную запись  
 ref = tgName(VAR | DEFINED, S3->line);  
 return 0;  
 }  
 if (ref.test(PROC)) {  
 // уже есть процедура с таким идентификатором -> ошибка  
 ferror\_message +=  
 "Error[75-1] in line " + S3->line + ": the variable '"  
 + S3->name +  
 "' overrides the global procedure with the same name, \n\t\t\tdefined in line "  
 + ref.line + " !\n";  
 // переменная 'x' перекрывает глобальную процедуру с таким же названием, определенную в строке 2  
 // the variable 'x' overrides the global procedure with the same name, defined in line 2  
 return 1;  
 }  
 if (ref.test(DEFINED)) {  
 // переменная с таким именем уже определена -> ошибка  
 ferror\_message +=  
 "Error[75-2] in line " + S3->line + ": the global variable '"  
 + S3->name +  
 "', defined in line " + ref.line + ", \n\t\t\tcannot be overridden!\n";  
 // глобальная переменная 'x', определенная в строке 1, не может быть переопределена  
 // the global variable 'x', defined in line 1, cannot be overridden  
 return 1;  
 }  
 // помечаем как определенную  
 ref.set(DEFINED);  
 return 0;  
}  
int tSM::p76(){ // PROC -> HPROC BLOCK )  
 string name = S1->name;  
 tgName& ref = globals[name];  
 if (ref.empty()) {  
 // процедура определяется впервые -> создаем учетную запись  
 ref = tgName(PROC | DEFINED, S1->line, S1->count, S1->types);  
 // и возвращаемся обратно в глобальную область видимости  
 scope = 0;  
 params.clear();  
 return 0;  
 }  
 if (ref.test(VAR)) {  
 // определена переменная с таким идентификатором -> ошибка  
 ferror\_message +=  
 "Error[76-1] in line " + S1->line + ": procedure '" +  
 name +  
 "' overrides the global variable of the same name,\n\t\t\tdefined in line "  
 + ref.line + "!\n";  
 // процедура 'f' перекрывает глобальную переменную с тем же именем, определенную в строке 1  
 // procedure 'f' overrides the global variable of the same name, defined in line 1  
 return 1;  
 }  
 if (ref.test(DEFINED)) {  
 // процедура уже была определена -> ошибка  
 ferror\_message +=  
 "Error[76-2] in line " + S1->line + ": procedure '" +  
 name +  
 "', described in line " + ref.line + " cannot be overridden!\n";  
 // процедура 'f', описанная в строке 1, не может быть переопределена  
 // procedure 'f', described in line 1, cannot be overridden  
 return 1;  
 }  
 if (ref.test(USED) && ref.arity != S1->count) {  
 // количество аргументов не совпадает с количеством переданных параметров -> ошибка  
 ferror\_message +=  
 "Error[76-3] in line " + S1->line + ": the procedure '"  
 + name + "' was called with " + Uint\_to\_str(ref.arity) + " parameter(s),\n\t\t\tpassed: " +  
 Uint\_to\_str(S1->count) + " !\n";  
 // процедура 'f' была вызвана с 2 параметрами, передано: 3  
 // procedure 'f' was called with 2 parameters, passed: 3  
 return 1;  
 }  
 // помечаем предикат как определенный  
 ref.set(DEFINED);  
 // и меняем область видимости  
 scope = 0;  
 params.clear();  
 return 0;  
}  
int tSM::p77(){ // PROC -> HPROC E )  
 string name = S1->name;  
 tgName& ref = globals[name];  
 if (ref.empty()) {  
 // процедура определяется впервые -> создаем учетную запись  
 ref = tgName(PROC | DEFINED, S1->line, S1->count, S1->types);  
 // и возвращаемся обратно в глобальную область видимости  
 scope = 0;  
 params.clear();  
 return 0;  
 }  
 if (ref.test(VAR)) {  
 // определена переменная с таким идентификатором -> ошибка  
 ferror\_message +=  
 "Error[77-1] in line " + S1->line + ": procedure '" +  
 name +  
 "'overrides the global variable of the same name,\n\t\t\tdefined in line "  
 + ref.line + "!\n";  
 // процедура 'f' перекрывает глобальную переменную с тем же именем, определенную в строке 1  
 // procedure 'f' overrides the global variable of the same name, defined in line 1  
 return 1;  
 }  
 if (ref.test(DEFINED)) {  
 // процедура уже была определена -> ошибка  
 ferror\_message +=  
 "Error[77-2] in line " + S1->line + ": procedure '" +  
 name +  
 "', described in line " + ref.line + " cannot be overridden!\n";  
 // процедура 'f', описанная в строке 1, не может быть переопределена  
 // procedure 'f', described in line 1, cannot be overridden  
 return 1;  
 }  
 if (ref.test(USED) && ref.arity != S1->count) {  
 // количество аргументов не совпадает с количеством переданных параметров -> ошибка  
 ferror\_message +=  
 "Error[77-3] in line " + S1->line + ": the procedure '"  
 + name + "' was called with " + Uint\_to\_str(ref.arity) + " parameter(s),\n\t\t\tpassed: " +  
 Uint\_to\_str(S1->count) + " !\n";  
 // процедура 'f' была вызвана с 2 параметрами, передано: 3  
 // procedure 'f' was called with 2 parameters, passed: 3  
 return 1;  
 }  
 // помечаем предикат как определенный  
 ref.set(DEFINED);  
 // и меняем область видимости  
 scope = 0;  
 params.clear();  
 return 0;  
}  
int tSM::p78(){ // HPROC -> PCPAR )  
 // точка анализа входит в тело процедуры  
 scope = 1;  
 return 0;  
}  
int tSM::p79(){ // HPROC -> HPROC INTER  
 return 0;}  
int tSM::p80(){ // PCPAR -> ( define ( $id  
 S1->name = S4->name;  
 S1->count = 0;  
 return 0;  
}  
int tSM::p81(){ // PCPAR -> PCPAR $id  
 if (params.count(S2->name)) {  
 ferror\_message +=  
 "Error[81-1] in line " + S2->line + ": the parameter '"  
 + S2->name +  
 "' is duplicated in the\n\t\t\tprocedure '"  
 + S1->name + "'!\n";  
 // в процедуре 'f' дублируется параметр 'x'  
 // the parameter 'x?' is duplicated in the procedure 'f?'  
 return 1;  
 }  
  
 if (S2->name == S1->name) {  
 ferror\_message +=  
 "Warning[81-2] in line " + S2->line + ": procedure '"  
 + S1->name +  
 "'has the same name \n"  
 "\t\t\tas its parameter!\n";  
 // у процедуры 'f' такое же имя, как у ее параметра  
 // procedure 'f' has the same name as its parameter  
 }  
 // добавляем параметр в список  
 S1->types |= 0 << S1->count;  
 params.insert(S2->name);  
 ++S1->count;  
 return 0;  
}  
int tSM::p82(){ // BLOCK -> HBLOCK E )  
 locals.clear();  
 return 0;  
}  
int tSM::p83(){ // HBLOCK -> BLVAR )  
 scope = 2;  
 return 0;  
}  
int tSM::p84(){ // HBLOCK -> HBLOCK INTER  
 return 0;}  
int tSM::p85(){ // BLVAR -> ( let ( LOCDEF  
 // встретили первую локальную переменную  
 S1->count = 1;  
 S1->name = S4->name;  
 locals.insert(S4->name);  
 return 0;  
}  
int tSM::p86(){ // BLVAR -> BLVAR LOCDEF  
 string name = S2->name;  
 if (locals.count(name)) {  
 // уже есть локальная переменая с таким идентификатором -> ошибка  
 ferror\_message +=  
 "Error[86-1] in line " + S2->line + ": the local variable '"  
 + S1->name +  
 "' cannot be overridden!\n";  
 // локальная переменная 'x' не может быть переопределена  
 // the local variable 'x' cannot be overridden  
 return 1;  
 }  
 locals.insert(name);  
 ++S1->count;  
 return 0;  
}  
int tSM::p87(){ // LOCDEF -> ( $id E )  
 S1->name = S2->name;  
 return 0;  
}  
//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
int tSM::p88(){return 0;} int tSM::p89(){return 0;}   
int tSM::p90(){return 0;} int tSM::p91(){return 0;}   
int tSM::p92(){return 0;} int tSM::p93(){return 0;}   
int tSM::p94(){return 0;} int tSM::p95(){return 0;}   
int tSM::p96(){return 0;} int tSM::p97(){return 0;}   
int tSM::p98(){return 0;} int tSM::p99(){return 0;}   
int tSM::p100(){return 0;} int tSM::p101(){return 0;}   
int tSM::p102(){return 0;} int tSM::p103(){return 0;}   
int tSM::p104(){return 0;} int tSM::p105(){return 0;}   
int tSM::p106(){return 0;} int tSM::p107(){return 0;}   
int tSM::p108(){return 0;} int tSM::p109(){return 0;}   
int tSM::p110(){return 0;}**