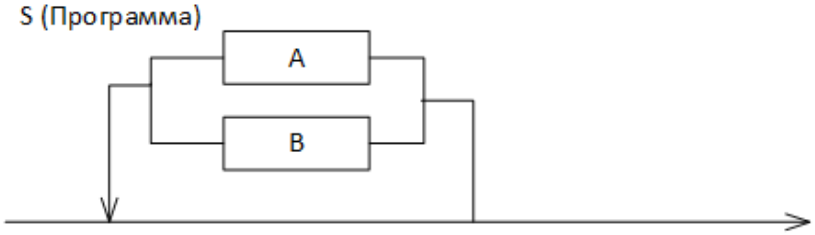
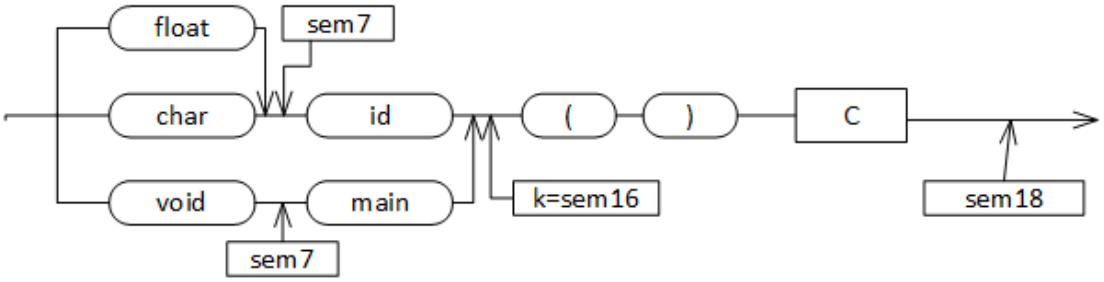
Синтаксические диаграммы с разметкой семантических подпрограмм:

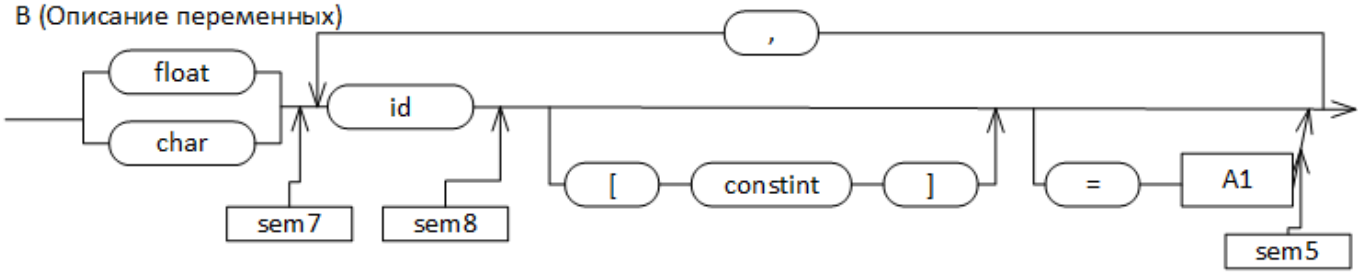
S (Программа)



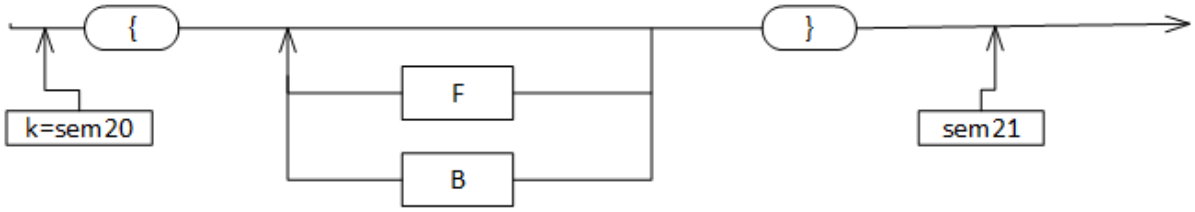
A (Описание функции)



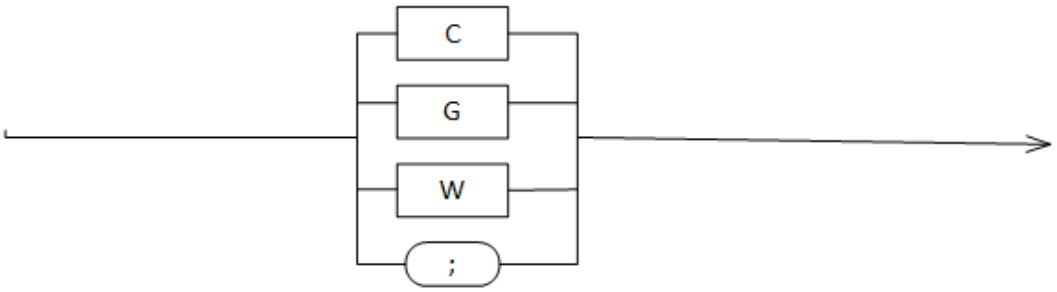
B (Описание переменных)



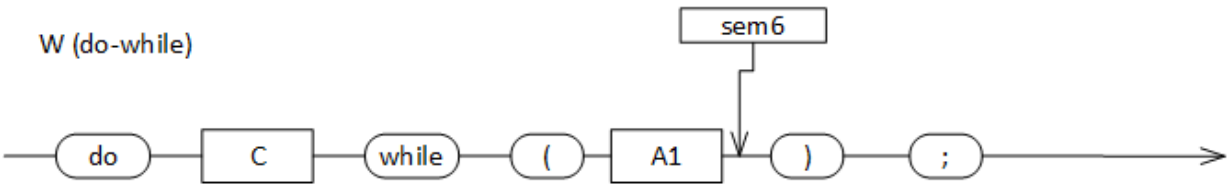
C (Блок)



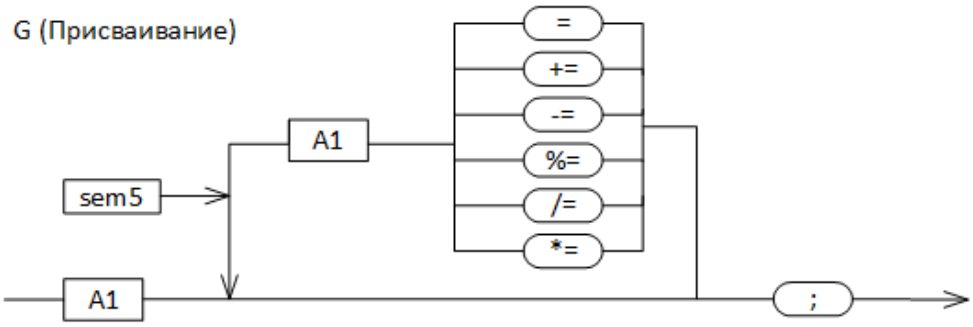
F (Один оператор)



W (do-while)

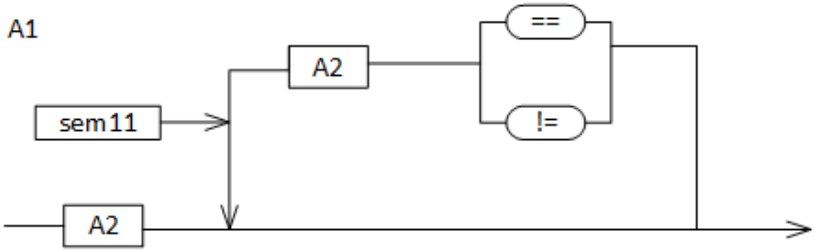


G (Присваивание)

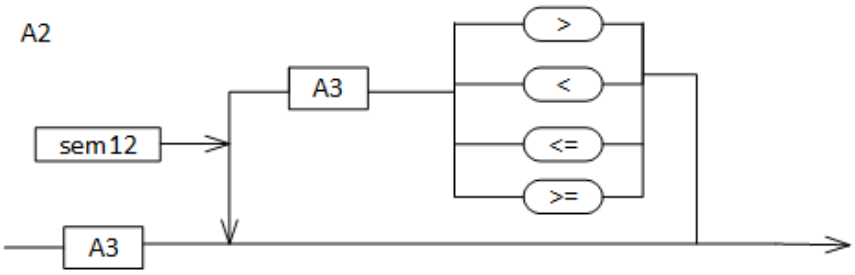


A1-A5 (Арифметика и сравнение)

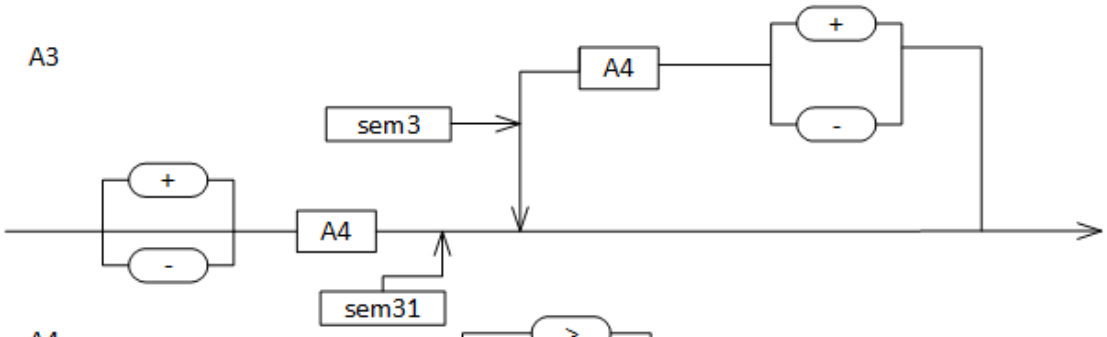
A1



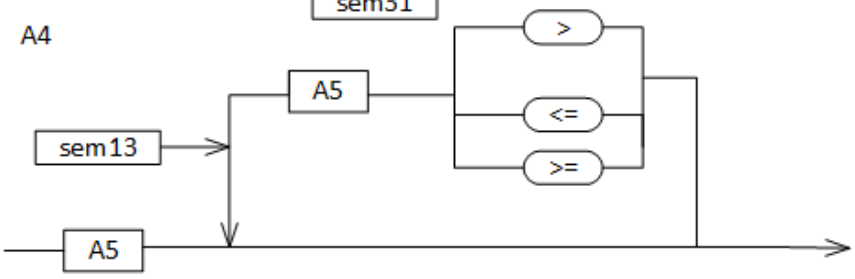
A2



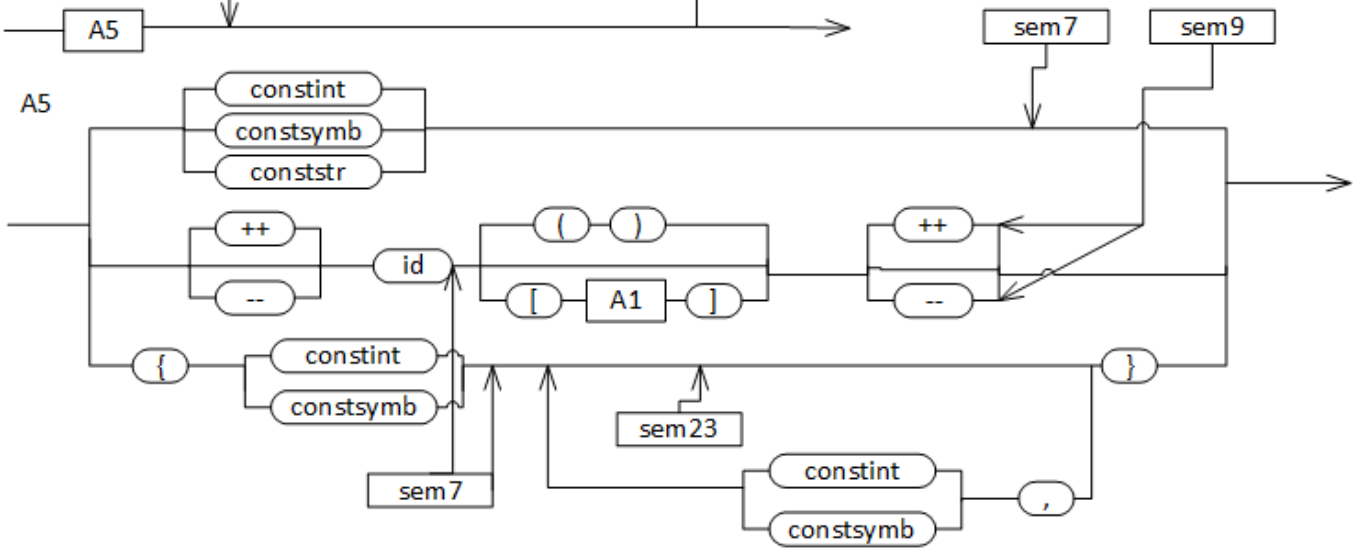
A3



A4



A5



Sem31(t) – проверяет допустимость унарной операции на данными типа t;

Sem3(t,g) – вычисляет тип результата операции над t и g;

Sem6(t) – проверяет, является ли t выражением;

Sem16(t,lex) – заносит в таблицу идентификатор lex в качестве имени функции, возвращающей тип t;

Sem18(k) – восстанавливает значение указателя таблицы по значению k;

Sem11(g,t) – приведение типа, результат операции == или !=;

Sem12(g,t) – приведение типа, результат операции >, <, <= или >=;

Sem13(g,t) – приведение типа, результат операции *, /, %;

Sem5(g,t) – проверка возможности приведения типов g и t;

Sem7(t,lex) – определение типа t;

Sem8(id,t) – добавление в таблицу id с типом t;

Sem20(id) – заносим в дерево id как тип. Создаем правую вершину и сдвигаем туда указатель для дальнейшего занесения информации;

Sem21(k) – восстанавливает указатель на вершину k;

Sem9(t) – проверяет, можно ли применять к типу t ++ или --;

Sem23(t) – увеличивает количество заполненных элементов массива и сравнивает это количество с размерностью массива. Если размерность < количества элементов, то выводим ошибку.

Реализация метода рекурсивного спуска с проверкой контекстных условий

diagram.h:

```
#pragma once
#ifndef __DIAGRAM
#define __DIAGRAM
#include "semant.h"
using namespace std;

class TDiagram
{
private:
	Tscaner *sc;
	Tree *root;
public:
	static int numpar;
	static DATA_TYPE dtt;
	TDiagram(Tscaner * s, Tree *r) {
		sc=s;
		root = r;
	}
	~TDiagram(){}
	void S(); //программа
	void A(); //описание функции
	void B(); //описание переменных
	void E(DATA_TYPE typ); //список переменных
	void C(); //блок+операторы
	void G(); //присваивание
	void F(); //один оператор
	void W(); //do-while
	//Арифметика и сравнение
	DATA_TYPE A1(bool init); //уровень приоритета 1
	DATA_TYPE A2(bool init); //уровень приоритета 2
	DATA_TYPE A3(bool init); //уровень приоритета 3
	DATA_TYPE A4(bool init); //уровень приоритета 4
	DATA_TYPE A5(bool init); //уровень приоритета 5
};
#endif
```

diagram.cpp:

```
#include "defs.h"
#include "scaner.h"
#include "diagram.h"

int TDiagram::numpar = 0;
DATA_TYPE TDiagram::dtt = TYPE_UNKNOWN;

//Программа
void TDiagram::S(){
	cout<<"S in "<<sc->getLine()<<endl;
	//программа
	TypeLex l;
	int t;
	int uk1;
	int line1;

	uk1=sc->getUk();
	line1=sc->getLine();
	t=sc->scaner(l);

	while(t!=Teof){

		sc->setUk(uk1);
		sc->setLine(line1);
```

```
		if (t==Tvoid) A();
		else{
			if (t==Tfloat || t==Tchar) {
				t=sc->scaner(l);t=sc->scaner(l);
				if (t==Tid)
				{
					t=sc->scaner(l);
					if(t==TLBKT){
						sc->setLine(line1);
						sc->setUk(uk1);
						A();
					}
					else{
						sc->setLine(line1);
						sc->setUk(uk1);
						B();
					}
				}
				else sc->printError("Ожидался идентификатор", l);
			}
			else sc->printError("Ожидался модификатор void, float или char", l);
		}

		uk1=sc->getUk();
		line1=sc->getLine();
		t=sc->scaner(l);

	}

	cout<<"S out "<<sc->getLine()<<endl;

	system("cls");
	printf("-------------------------------------------------------------------------------
\n");
	printf("|  ID           УЗЕЛ  ТИП РАЗМЕР |  ID   ЛЕВЫЙ ПОТОМОК |  ID  ПРАВЫЙ ПОТОМОК
|\n");
	printf("-------------------------------------------------------------------------------
\n");
	root->Print();
	printf("-------------------------------------------------------------------------------
\n");
	printf("ТИПЫ:\n 1 - не определен\n 2 - char\n 3 - float\n 4 - функция\n 7 - массив char\n
8 - массив float\n");
	printf("-------------------------------------------------------------------------------
\n");
}

//Описание функции
void TDiagram::A(){
	cout<<"A in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1;
	int line1;

	uk1 = sc->getUk();
	t = sc->scaner(l);
	if((t != Tfloat)&&(t != Tchar)&&(t != Tvoid)){
		sc->printError("Ожидался модификатор типа char,float или void",l);
	}
	if((t==Tfloat)||(t==Tchar))
	{
		DATA_TYPE typ = root->SEM7(l,sc);
		t=sc->scaner(l);
		if(t != Tid)
			sc->printError("Ожидался идентификатор",l);
```

```
	}
	else if(t==Tvoid){
		t=sc->scaner(l);
		if(t != Tmain)
			sc->printError("Ожидался идентификатор main",l);
	}
	Tree *k = root->SemInclude(l,l,TYPE_FUNCT,sc); //внесем функцию

	t=sc->scaner(l);
	if(t != TLBKT){
		sc->printError("Ожидался символ (",l);
	}

	t=sc->scaner(l);
	if(t != TRBKT){
		sc->printError("Ожидался символ )",l);
	}

	C();//Блок

	root->SetCur(k);

	t=sc->scaner(l);
	cout<<"A out "<<sc->getLine()<<endl;
}

//Описание переменных
void TDiagram::B(){
	cout<<"B in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1;
	int line1;

	uk1 = sc->getUk();
	t = sc->scaner(l);
	if((t != Tfloat)&&(t != Tchar)){
		sc->printError("Ожидался модификатор типа char или float",l);
	}
	DATA_TYPE typ = root->SEM7(l,sc);

	E(typ);

	cout<<"B out "<<sc->getLine()<<endl;
}

//Список переменных

void TDiagram::E(DATA_TYPE typ){
	cout<<"E in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1,uk11;
	int line1;

	do{
		uk1 = sc->getUk();
	//	uk11 = uk1;
		t = sc->scaner(l);
		if(t!=Tid){
			sc->printError("Ожидался идентификатор",l);
		}

		Tree *v = root->SemInclude(l,l,typ,sc);

		uk1 = sc->getUk();
		t = sc->scaner(l);
```

```
		numpar=1;
		if(t==TsLBKT)
		{
			t = sc->scaner(l);
			if(t!=Tconstint){
				sc->printError("Ожидалась целая константа",l);
			}

			root->SemSetParam(v,atoi(l));

			numpar=atoi(l);

			if(root->SemType(v)==TYPE_CHAR) {root->SemSetType(v,TYPE_CHAR_ARRAY);dtt =
TYPE_CHAR_ARRAY;}
			if(root->SemType(v)==TYPE_FLOAT) {root->SemSetType(v,TYPE_FLOAT_ARRAY);dtt =
TYPE_FLOAT_ARRAY;}

			t = sc->scaner(l);
			if(t!=TsRBKT){
				sc->printError("Ожидался символ ]",l);
			}
			t = sc->scaner(l);
		}
		if(t==Tsave)
		{
			A1(true);
			dtt = TYPE_UNKNOWN;
			uk1 = sc->getUk();
			t = sc->scaner(l);
		}
		dtt=TYPE_UNKNOWN;
	}while(t==Tcom);

	if(t != Tsemi){
		sc->printError("Ожидался символ ;",l);
	}

	cout<<"E out "<<sc->getLine()<<endl;
}

//Блок
void TDiagram::C(){
	cout<<"C in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1;
	int line1;

	Tree *k = root->SemInclude("-EMPTY-","-EMPTY-", TYPE_UNKNOWN,sc); //Внесем функцию

	t=sc->scaner(l);
	uk1 = sc->getUk();
	line1 = sc->getLine();
	if(t!=TfLBKT) sc->printError("Ожидался символ {",l);
	while(t!= TfRBKT && t!= Teof){
		sc->setUk(uk1);
		sc->setLine(line1);
		t=sc->scaner(l);
		if((t==Tchar)||(t==Tfloat)){
			sc->setLine(line1);
			sc->setUk(uk1);
			B();
		}
		else{
			sc->setLine(line1);
			sc->setUk(uk1);
```

```
                F();
            }

            uk1=sc->getUk();
            line1=sc->getLine();
            sc->setUk(uk1);
            sc->setLine(line1);
            t=sc->scaner(l);
        }
        if(t==Teof)sc->printError("Ожидался символ }",l);

        sc->setUk(uk1);
        sc->setLine(line1);

        root->SetCur(k);

        cout<<"C out "<<sc->getLine()<<endl;
}

//Один оператор
void TDiagram::F(){
        cout<<"F in "<<sc->getLine()<<endl;
        TypeLex l;
        int t;
        int uk1;
        int line1;

        uk1=sc->getUk();
        line1=sc->getLine();
        t=sc->scaner(l);
        if(t==TfLBKT){
            sc->setLine(line1);
            sc->setUk(uk1);
            C();
            t=sc->scaner(l);
        }
        else{
            if(t==Tdo){
                sc->setLine(line1);
                sc->setUk(uk1);
                W();
            }
            else
            {
                if(t==Tsemi);
                else {
                    sc->setLine(line1);
                    sc->setUk(uk1);
                    G();
                }
            }
        }

        cout<<"F out "<<sc->getLine()<<endl;
}

//Присваивание
void TDiagram::G(){
        cout<<"G in "<<sc->getLine()<<endl;
        TypeLex l;
        int t;
        int uk1;
        int line1;

        DATA_TYPE ttyp = TYPE_UNKNOWN;
        DATA_TYPE gtyp;
```

```
	int check = 1;
	do{
		if(check%2 != 0) ttyp = A1(false);
		else
		{
			gtyp = A1(false);
			ttyp = root->SEM13(ttyp,gtyp,t,sc);
		}
		uk1=sc->getUk();
		line1=sc->getLine();
		t=sc->scaner(l);
		check++;

	}while((t==Tsubeq)||(t==Tsave)||(t==Tsumeq)||(t==Tmuleq)||(t==Tdiveq)||(t==Tmodeq));

	if(t!=Tsemi) sc->printError("Ожидался символ ;",l);
	sc->setUk(uk1);
	sc->setLine(line1);


	cout<<"G out "<<sc->getLine()<<endl;
}

//Do-While
void TDiagram::W(){
	cout<<"W in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1;
	int line1;

	t = sc->scaner(l);
	if(t!=Tdo){
		sc->printError("ожидалось ключевое слово do",l);
	}

	C();

	line1 = sc->getLine();
	uk1 = sc->getUk();
	t = sc->scaner(l);
	t = sc->scaner(l);
	if(t!=Twhile){
		sc->printError("ожидалось ключевое слово while",l);
	}

	t = sc->scaner(l);
	if(t!=TLBKT){
		sc->printError("ожидался символ (",l);
	}

	A1(false);

	t = sc->scaner(l);
	if(t!=TRBKT){
		sc->printError("ожидался символ )",l);
	}

	t = sc->scaner(l);
	if(t!=Tsemi){
		sc->printError("ожидался символ ;",l);
	}

	uk1=sc->getUk();
	line1=sc->getLine();

	cout<<"W out "<<sc->getLine()<<endl;
```

```
}

//Арифметика и сравнение
DATA_TYPE TDiagram::A1(bool init){
	cout<<"A1 in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1;
	int line1;

	DATA_TYPE ttyp = TYPE_UNKNOWN;
	DATA_TYPE gtyp;


	ttyp = A2(init);
	uk1=sc->getUk();
	line1=sc->getLine();
	t=sc->scaner(l);

	while ((t==Teq) || (t==Tneq)){
		gtyp = A2(init);
		ttyp = root->SEM13(ttyp,gtyp,t,sc);

		uk1=sc->getUk();
		line1=sc->getLine();
		t=sc->scaner(l);
	}
	sc->setUk(uk1);
	sc->setLine(line1);
	cout<<"A1 out "<<sc->getLine()<<endl;
	return ttyp;
}

DATA_TYPE TDiagram::A2(bool init){
	cout<<"A2 in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1;
	int line1;

	DATA_TYPE ttyp= TYPE_UNKNOWN;
	DATA_TYPE gtyp;

	ttyp = A3(init);

	uk1=sc->getUk();
	line1=sc->getLine();
	t=sc->scaner(l);

	while ((t==Tmore) || (t==Tless)||(t==Tmoreeq) || (t==Tlesseq)){

		gtyp = A3(init);
		ttyp = root->SEM13(ttyp,gtyp,t,sc);

		uk1=sc->getUk();
		line1=sc->getLine();
		t=sc->scaner(l);
	}
	sc->setUk(uk1);
	sc->setLine(line1);

	cout<<"A2 out "<<sc->getLine()<<endl;

	return ttyp;
}
```

```
DATA_TYPE TDiagram::A3(bool init){
	cout<<"A3 in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1;
	int line1;
	DATA_TYPE ttyp= TYPE_UNKNOWN;
	DATA_TYPE gtyp;
	bool flagy = false;

	uk1=sc->getUk();
	line1=sc->getLine();
	t=sc->scaner(l);
	if(t!=Tsum&&t!=Tsub){
		sc->setUk(uk1);
		sc->setLine(line1);
	}
	else flagy = true;


	ttyp = A4(init);

	if(flagy){//ПРОВЕРИМ ПРИМЕНИМОСТЬ УНАРКИ
		root->SEM9(ttyp,sc);
	}

	uk1=sc->getUk();
	line1=sc->getLine();
	t=sc->scaner(l);

	while ((t==Tsum) || (t==Tsub)){
		gtyp = A4(init);
		ttyp = root->SEM13(ttyp,gtyp,t,sc);

		uk1=sc->getUk();
		line1=sc->getLine();
		t=sc->scaner(l);
	}

	sc->setUk(uk1);
	sc->setLine(line1);

	cout<<"A3 out "<<sc->getLine()<<endl;
	return ttyp;
}

DATA_TYPE TDiagram::A4(bool init){
	cout<<"A4 in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1;
	int line1;
	DATA_TYPE ttyp= TYPE_UNKNOWN;
	DATA_TYPE gtyp;

	ttyp = A5(init);
	uk1=sc->getUk();
	line1=sc->getLine();
	t=sc->scaner(l);

	while ((t==Tmul) || (t==Tdiv)|| (t==Tmod)){
		gtyp = A5(init);
		ttyp = root->SEM13(ttyp,gtyp,t,sc);

		uk1=sc->getUk();
		line1=sc->getLine();
		t=sc->scaner(l);
```

```
	}
	sc->setUk(uk1);
	sc->setLine(line1);

	cout<<"A4 out "<<sc->getLine()<<endl;
	return ttyp;
}

DATA_TYPE TDiagram::A5(bool init){
	cout<<"A5 in "<<sc->getLine()<<endl;
	TypeLex l;
	int t;
	int uk1;
	int line1;
	bool flag = false;
	bool flag1 = false;
	DATA_TYPE dtyp,ttyp;
	uk1=sc->getUk();
	line1=sc->getLine();
	t=sc->scaner(l);
	if(t==Tconststr || t==Tconstint || t==Tconstsymb){
		if(t==Tconstint) dtyp = TYPE_CHAR;
		if(t==Tconstsymb)dtyp = TYPE_CHAR;
		if(t==Tconststr) dtyp = TYPE_CHARST;

		if(init==true)
		{
			if(t==Tconstint && (dtt==TYPE_FLOAT_ARRAY || dtt==TYPE_CHAR_ARRAY)) sc-
>printError("Необходима инициализация с помощью {...}",l);
			if(t==Tconstsymb && (dtt==TYPE_FLOAT_ARRAY || dtt==TYPE_CHAR_ARRAY)) sc-
>printError("Необходима инициализация с помощью {...}",l);
			if(t==Tconststr && dtt!=TYPE_CHAR_ARRAY) sc->printSemError("Нельзя
использовать char* для инициализации");
		}
	}
	else{
		if(t==TfLBKT){
			if(init!=true) sc->printError("Ожидалось выражение ",l);
			t=sc->scaner(l);
			if(t!=TfRBKT)
			{
				if(t!=Tconstint && t!=Tconstsymb) sc->printError("Ожидалась целая,
символьная или строковая константа ",l);
				t=sc->scaner(l);
				int check1 = 0;
				while(t!=TfRBKT)
				{
					if(t!=Tcom) sc->printError("Ожидался символ ,",l);
					check1++;
					t=sc->scaner(l);
					if(t!=Tconstint && t!=Tconstsymb) sc->printError("Ожидался
идентификатор ",l);
					t=sc->scaner(l);
				};
				if(check1>=numpar)sc->printSemError("Слишком много значений
инициализатора");
			}
		}
		else{
			bool flagy = false;

			uk1=sc->getUk();
			line1=sc->getLine();

			if((t==Tinc) || (t==Tdec))
			{
				t=sc->scaner(l);
```

```
                        flagy=true;
                }
                if(t!=Tid) sc->printError("Ожидался идентификатор",l);

                Tree *typ = root->SemGetType(l,sc);
                dtyp = typ->getDT();

                uk1=sc->getUk();
                line1=sc->getLine();

                bool ffff=false;
                if(dtyp==TYPE_FUNCT)ffff=true;
                t=sc->scaner(l);
                if(t==TLBKT)
                {
                        if(ffff==false)sc->printSemError("Выражение перед скобками должно
иметь тип функции");
                        t=sc->scaner(l);
                        if(t!=TRBKT) sc->printError("Одилался символ )",l);
                        flag1=true;
                }
                else
                {
                        if(t==TsLBKT)
                        {
                                if(dtyp!=TYPE_FLOAT_ARRAY && dtyp!=TYPE_CHAR_ARRAY)sc-
>printSemError("Выражение не является массивом");
                                dtyp = A1(init);
                                if(dtyp == TYPE_CHARST) sc->printSemError("Выражение должно
быть численного типа");
                                t=sc->scaner(l);
                                if(t!=TsRBKT) sc->printError("Ожидался символ ]",l);
                                flag1=true;
                        }
                }
                if(ffff==true && flag1!=true)
                {
                        sc->printSemError("После выражения, обозначающего функцию должны быть
скобки");
                }
                if(dtyp==TYPE_FLOAT_ARRAY && flag1!=true)
                {
                        sc->printSemError("Не указан элемент массива");
                }
                if(flag1==true)
                {
                        uk1=sc->getUk();
                        line1=sc->getLine();
                        t=sc->scaner(l);
                }
                if(flagy){
                        dtyp=root->SEM9(dtyp,sc);
                }
                if((t!=Tinc)&&(t!=Tdec))
                {
                        sc->setLine(line1);
                        sc->setUk(uk1);
                }
                else{
                        //Проверка типа
                                dtyp=root->SEM9(dtyp,sc);
                }
            }
        }
        return dtyp;
        cout<<"A5 out "<<sc->getLine()<<endl;
}
```

semant.h:

```
#pragma once
#include"defs.h"
#include"scaner.h"

enum DATA_TYPE {TYPE_UNKNOWN = 1, TYPE_CHAR,
	TYPE_FLOAT, TYPE_FUNCT, TYPE_VOID, TYPE_ARRAY, TYPE_CHAR_ARRAY, TYPE_FLOAT_ARRAY,
TYPE_CHARST
};

class Tree // элемент семантической таблицы
{
	TypeLex id; // идентификатор объекта
	DATA_TYPE DataType; // тип значения
	int size;
	Tree * Up, * Left, * Right;
	int number;
	static int ns;
	// родитель, левый и правый потомок
	public:
	static Tree * Cur; // текущий элемент дерева
// ФУНКЦИИ ОБРАБОТКИ БИНАРНОГО ДЕРЕВА
	Tree(Tree * l, Tree * r, Tree * u, TypeLex aid,DATA_TYPE aDataType);
	Tree();
	~Tree();
	void SetLeft (Tree * Data);
	void SetRight(Tree * Data);
	void SetLeft (TypeLex aid,DATA_TYPE aDataType);
	void SetRight(TypeLex aid,DATA_TYPE aDataType);
	Tree * FindUp (Tree * From, TypeLex id);
	Tree * FindUpOneLevel (Tree * From, TypeLex id);
	Tree * FindUp (TypeLex id);

	Tree * Top ();

	Tree * FindRightLeft (Tree * From, TypeLex id);
	Tree * FindRightLeft (TypeLex id);

	void Print(void);

	// СЕМАНТИЧЕСКИЕ ПОДПРОГРАММЫ
	void SetCur(Tree * a) ;    // установить текущий узел дерева
	Tree * GetCur(void);// получить значение текущего узла дерева
	Tree * SemInclude(TypeLex a,TypeLex b, DATA_TYPE t,Tscaner *s);// занесение идентификатора
a в таблицу с типом t
	void SemSetParam(Tree* Addr, int num);
	void SemSetType(Tree *Addr, DATA_TYPE t);// установить тип t для переменной по адресу Addr
	Tree * SemGetType(TypeLex a,Tscaner *s); // найти в таблице переменную с именем a и вернуть
ссылку на соответствующий элемент дерева
	Tree * SemGetStruct(TypeLex a,Tscaner *s);     // найти в таблице функцию с именем a и
вернуть ссылку на соответствующий элемент дерева
	Tree * SemGetFunct(TypeLex a,Tscaner *s); //функцию найти
	int DupControl(Tree *Addr, TypeLex a);// проверка идентификатора a на повторное описание
внутри блока
	DATA_TYPE SEM7(TypeLex t,Tscaner *s);//Определяет тип лексемы t
	DATA_TYPE SemType(Tree *Addr);
	DATA_TYPE SEM13(DATA_TYPE fir,DATA_TYPE sec,int a,Tscaner *s);//Приведение типов для
*/дел/%/+/-/==/!=
	DATA_TYPE SEM9(DATA_TYPE a,Tscaner *s);//Определяет возможность применения инкремента
	DATA_TYPE getDT();
	int getNum();
};
```

semant.cpp:

```
#pragma once
#include "defs.h"
#include "semant.h"

#define max(a,b) a<b? b : a

int Tree::ns;

Tree* Tree::Cur=(Tree*)NULL;

Tree::Tree (Tree * l, Tree * r, Tree * u, TypeLex aid,DATA_TYPE aDataType){// конструктор создает
узел с заданными связями и данными
	strcpy(id,aid);
	DataType = aDataType;
	size = -1;
	Up=u; Left=l; Right=r; // установили ссылки

	number = ++ns;
}
Tree::Tree (void){// конструктор создает новый узел с пустыми связями и данными
	DataType = TYPE_UNKNOWN;
	Up=NULL; Left=NULL; Right=NULL;
	strcpy(id,"-EMPTY-");
	size = -1;
	ns = 0;
	number = 0;
}
void Tree::SetLeft (Tree * Data){// назначить левого потомка от текущей вершины this

	Left=Data; // связали this с вершиной
}
void Tree::SetRight(Tree * Data){// назначить правого потомка от текущей вершины this

	Right=Data; // связали this с  вершиной
}

void Tree::SetLeft (TypeLex aid,DATA_TYPE aDataType){// создать левого потомка от текущей вершины
this
	Tree *a = new Tree(NULL,NULL,this,aid,aDataType);
	Left = a;

}
void Tree::SetRight(TypeLex aid,DATA_TYPE aDataType){// создать правого потомка от текущей
вершины this

	Tree *a = new Tree(NULL,NULL,this,aid,aDataType);
	Right = a;
}

Tree* Tree::FindUp(TypeLex id){// поиск данных в дереве, начиная от текущей вершины this до его
корня вверх по связям
	return FindUp(this, id);
}
Tree* Tree::FindUp(Tree * From, TypeLex id){// поиск данных в дереве от заданной вершины From до
его корня вверх по связям
	Tree * i=From; // текущая вершина поиска
	while( (i!=NULL) && (strcmp(id, i->id)!=0)){
		i=i->Up; // поднимаемся наверх по связям
	}
	return i;
}

Tree * Tree::FindRightLeft(TypeLex id){// поиск прямых потомков текущей вершины this
	return FindRightLeft(this, id);
}
```

```
Tree * Tree::FindRightLeft(Tree * From, TypeLex id){// поиск прямых потомков заданной вершины
From
	Tree *i=From->Right; // текущая вершина поиска
	while( (i!=NULL) &&(memcmp(id, i->id, max(strlen(i->id),strlen(id)))!=0) ){
		i==i->Left;// обходим только соседей по левым связям
	}

	return i;
}
Tree * Tree::FindUpOneLevel(Tree * From, TypeLex id){// Поиск элемента id вверх по дереву от
текущей вершины From.Поиск осуществляется на одном уровне вложенности по левым связям
	Tree * i=From; // текущая вершина поиска
	if(i->Up == NULL) return NULL;

	while( (i!=NULL) &&( i->Up->Right != i)){
		if (memcmp(id, i->id, max(strlen(i->id),strlen(id)))==0){
			return i; // нашли совпадающий идентификатор
		}
		i=i->Up; // поднимаемся наверх по связям
		if(i->Up == NULL) return NULL;
	}
	return NULL;
}
Tree* Tree::Top(){// поиск данных в дереве от заданной вершины From до его корня вверх по связям
	Tree* i = Cur;
	while( (i!=NULL)){
		i=i->Up; // поднимаемся наверх по связям
	}
	return i;
}
void Tree::Print (void){
	printf("| %3d  %15s  %d  %5d |", number,id,DataType,size);
	if (Left !=NULL) printf(" %3d %15s |", Left->number, Left->id );
	else printf("                    - |");
	if (Right!=NULL) printf(" %3d %15s |", Right->number, Right->id );
	else printf("                    - |");
	printf("\n");
	if (Left!=NULL) Left->Print();
	if (Right!=NULL&&Right->DataType!=TYPE_FUNCT) Right->Print();
}
void Tree::SetCur(Tree * a){// установить текущий узел дерева
	Cur=a;
}

Tree * Tree::GetCur(void){// получить значение текущего узла дерева
	return Cur;
}
Tree * Tree::SemInclude(TypeLex a,TypeLex b, DATA_TYPE t,Tscaner *s){// занесение идентификатора
a в таблицу с типом t
	if (DupControl(Cur, a)){

		s->printError("Повторное описание идентификатора ",a);
	}
	Tree * v;
	if (t!=TYPE_FUNCT){
		Cur->SetLeft (a,t); // сделали вершину - переменную
		Cur = Cur->Left;

		return Cur;
	}
	else
	{
		Cur->SetLeft (a,t); // сделали вершину - функцию
		Cur = Cur->Left;
		v=Cur; // это точка возврата после выхода из функции
		Cur->SetRight ("-EMPTY-",TYPE_UNKNOWN); // сделали пустую вершину
		Cur = Cur->Right;
```

```
		return v;
	}
}
void Tree::SemSetType(Tree* Addr, DATA_TYPE t){// установить тип t для переменной по адресу Addr
	Addr->DataType=t;
}

void Tree::SemSetParam(Tree* Addr, int num)
// установить число формальных параметров n для функции по адресу Addr
{
Addr->size=num;
}

//Указатель на вершину с переменной
Tree * Tree::SemGetType(TypeLex a,Tscaner *s){// найти в таблице переменную с именем a и вернуть
ссылку на соответствующий элемент дерева
	Tree *v=FindUp(Cur, a);
	if (v==NULL){
		s->printError("Отсутствует описание идентификатора ",a);
	}	return v;
}
Tree * Tree::SemGetFunct(TypeLex a, Tscaner *s)
// Найти в таблице функцию с именем a и вернуть ссылку на элемент дерева
{
	Tree * v=FindUp(Cur, a);
	if (v==NULL)
		s->printError("Отсутствует описание функции ",a);
	if (v->DataType!=TYPE_FUNCT)
		s->printError("Не является функцией идентификатор ",a);
	return v;
}

//Опред. типа
DATA_TYPE Tree::SEM7(TypeLex a,Tscaner *s){
	if(strcmp(a,"char") ==0) return TYPE_CHAR;
	if(strcmp(a,"float") ==0) return TYPE_FLOAT;
	if(strcmp(a,"void") ==0) return TYPE_VOID;

	return TYPE_UNKNOWN; //Тип не определен
}
//Приведение типов для */дел/%/+/-/==/!=/=/-=/+=/</<= и тд
DATA_TYPE Tree::SEM13(DATA_TYPE fir,DATA_TYPE sec,int a,Tscaner *s){
	if(fir==TYPE_UNKNOWN||sec==TYPE_UNKNOWN){
			s->printSemError("Невозможно совершить операцию с такими операндами");	}
	else{
		if(a == Tmod){
			if(fir==TYPE_FLOAT||sec==TYPE_FLOAT) s->printSemError("Невозможно выполнить
операцию % с операндом типа float");
			if(fir==TYPE_CHARST||sec==TYPE_CHARST) s->printSemError("Невозможно выполнить
операцию % с операндом типа char*");

			return TYPE_CHAR;
		}
		else{

			if((a==Tmore) || (a==Tless)||(a==Tmoreeq) ||
(a==Tlesseq)||(a==Teq)||(a==Tneq)){
				if((fir==TYPE_CHARST && sec!=TYPE_CHARST && sec!=TYPE_CHAR_ARRAY) ||
(fir!=TYPE_CHARST && sec==TYPE_CHARST && fir!=TYPE_CHAR_ARRAY)) s->printSemError("Невозможно
выполнить операцию сравнения с операндом типа char*");
				if(fir==TYPE_CHARST && sec==TYPE_CHARST)
					return TYPE_CHARST;
				return TYPE_CHAR;
			}
		if((a==Tsubeq)||(a==Tsave)||(a==Tsumeq)||(a==Tmuleq)||(a==Tdiveq)||(a==Tmodeq)){
	if((fir==TYPE_CHARST || sec==TYPE_CHARST)) s->printSemError("Невозможно выполнить операцию
присваивания с операндом типа char*");
```

```
	if(fir==TYPE_FUNCT)s->printSemError("Невозможно присвоить значение функции");
				return TYPE_CHAR;
			}
			if((fir==TYPE_CHARST && sec!=TYPE_CHARST) || (fir!=TYPE_CHARST &&
sec==TYPE_CHARST)) s->printSemError("Невозможно выполнить операцию с операндом типа char*");
			if(fir==TYPE_CHARST || sec==TYPE_CHARST)
				return TYPE_CHARST;


			if(fir==TYPE_FLOAT||sec==TYPE_FLOAT){
				return TYPE_FLOAT;
			}
			return TYPE_CHAR;
		}
	}
	return fir;
}
int Tree::getNum(){
	return size;
}
DATA_TYPE Tree::getDT(){
	return DataType;
}
DATA_TYPE Tree::SEM9(DATA_TYPE a,Tscaner *s){
	if(a==TYPE_UNKNOWN || a==TYPE_CHARST){
		s->printError("невозможно применить унарную операцию","\n");
	}
	return a;
}
DATA_TYPE Tree::SemType(Tree* Addr){
	if(Addr->DataType==TYPE_ARRAY) return TYPE_ARRAY;
	if(Addr->DataType==TYPE_CHAR) return TYPE_CHAR;
	if(Addr->DataType==TYPE_FLOAT) return TYPE_FLOAT;
	if(Addr->DataType==TYPE_CHAR_ARRAY) return TYPE_CHAR_ARRAY;
	if(Addr->DataType==TYPE_FLOAT_ARRAY) return TYPE_FLOAT_ARRAY;
	if(Addr->DataType==TYPE_UNKNOWN) return TYPE_UNKNOWN;
	if(Addr->DataType==TYPE_FUNCT) return TYPE_FUNCT;
	if(Addr->DataType==TYPE_VOID) return TYPE_VOID;
}
int Tree::DupControl(Tree* Addr, TypeLex a){// Проверка идентификатора a на повторное описание
внутри блока. Поиск осуществляется вверх от вершины Addr.
	if(a == "-EMPTY-") return 0;
	if (FindUpOneLevel(Addr, a)==NULL){
		return 0;
	}
	return 1;
}
```