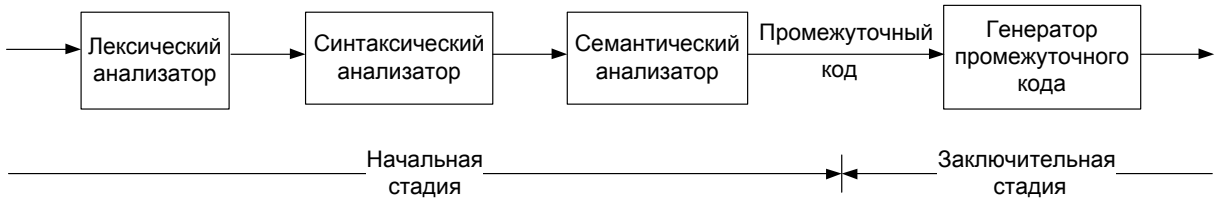
Лекция 22
БГТУ, ФИТ, ПОИТ, 3 семестр, Языки программирования

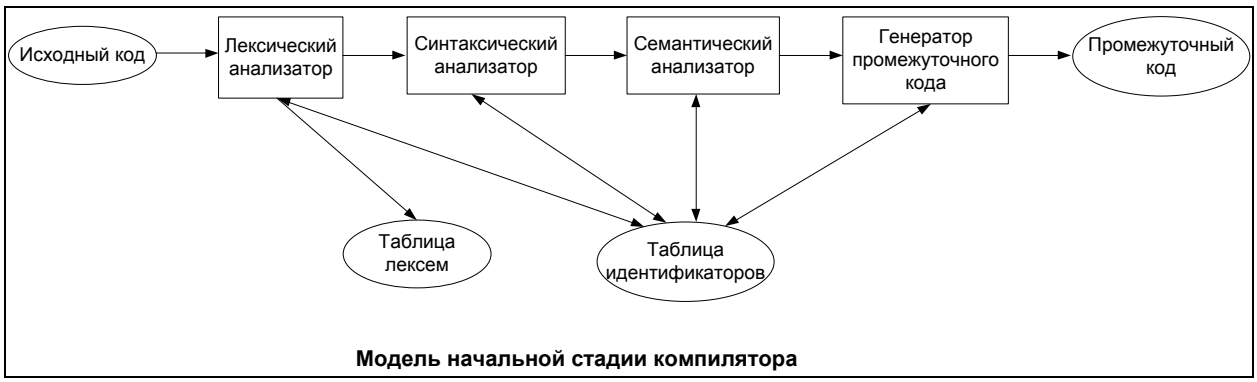
Генерация промежуточного кода

1. Логическая структура транслятора



Начальная, анализирующая часть транслятора (или front end) отвечает за лексический, синтаксический и семантический анализ исходной программы и порождает промежуточное представление. При изменении входного языка фронтальная часть может быть заменена независимо от других частей компилятора.

Заключительная, синтезирующая часть (back end) не зависит от входного языка и может быть изменена для другой целевой машины.



Модель начальной стадии компилятора

На начальной стадии компилятора анализируется исходная программа и создается промежуточное представление, из которого на заключительной стадии генерируется целевой код.

2. **Семантический анализ и подготовка к генерации кода:** назначение семантического анализа, этапы семантического анализа.

Входные данные для семантического анализа:

- таблица идентификаторов;
- дерево разбора – результат разбора синтаксических конструкций входного языка.

Основные действия семантического анализатора:

1) проверка соблюдения в исходной программе семантических правил входного языка;
2) дополнение внутреннего представления программы в компиляторе операторами и действиями, неявно предусмотренными семантикой входного языка;
3) проверка элементарных семантических (смысловых) норм языка программирования.

1). Проверка соблюдения *семантических правил* входного языка - сопоставление входных цепочек программы с требованиями семантики входного языка программирования.

Примеры семантических правил:

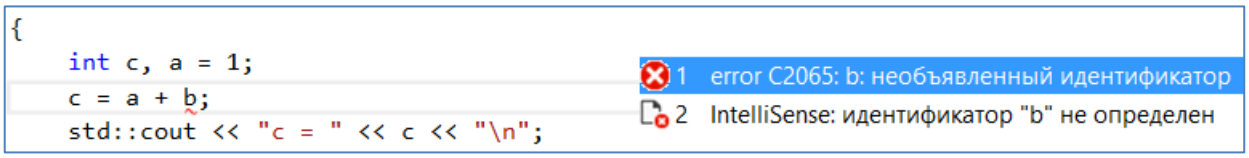
- каждый идентификатор должен быть описан только один раз (с учетом блочной структуры описаний);
- все операнды в выражениях и операциях должны иметь типы, допустимые для данного выражения или операции;
- типы переменных в выражениях должны быть согласованы между собой;
- при вызове процедур и функций число и типы фактических параметров должны быть согласованы с числом и типами формальных параметров.

Пример. Оператор языка C++:

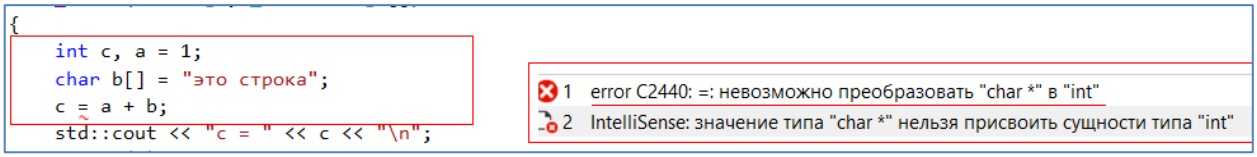
```
a = b + c;
```

это правильный оператор.

Если хотя бы один из идентификаторов не описан, то это ошибка:



Не допускается, чтобы один из идентификаторов был числовыми, а другой — строковым:



2). *Дополнение внутреннего представления программы операторами* и действиями неявно предусмотренными семантикой входного языка.

Операторы языка C++	Выполняемые операции
a = b + c;	- операция сложения; - операция присваивания результата.

int c = 1;
float b = 2.5;
double a;
a = b + c;
	- преобразование целочисленной переменной с в формат чисел с плавающей точкой;
	- сложение двух чисел с плавающей точкой;
	- преобразование результата в число с плавающей точкой удвоенной точности;
	- присвоение результата переменной a.

3). *Проверка элементарных смысловых норм* языков программирования, напрямую не связанных с входным языком, — это сервисная функция, которую предоставляют большинство современных компиляторов.

Примеры соглашений:

- каждая переменная или константа должна хотя бы один раз использоваться в программе;
- каждая переменная должна быть определена до ее первого использования при любом ходе выполнения программы;
- переменной должно всегда предшествовать присвоение ей какого-либо значения;
- результат функции должен быть определен при любом ходе ее выполнения;
- каждый оператор в исходной программе должен иметь возможность хотя бы один раз выполниться;
- операторы условия и выбора должны предусматривать возможность пути выполнения программы по каждой из своих ветвей;
- операторы цикла должны предусматривать возможность завершения цикла.

```
int f_test(int a) {
    int b, c;
    b = 0;
    c = 0;
    if (b = 1) { std::cout << "a = " << a << "\n"; return a; }
    c = a + b;
    std::cout << "c = " << c << "\n";
    }
int _tmain(int argc, _TCHAR* argv[])
{
    f_test(3);
    system("pause");
    return 0;
}
```

Сообщения компилятора (уровень предупреждений 4 (/W4)):

```
1>------ Сборка начата: проект: L22, Конфигурация: Debug Win32 ------
1>  L22.cpp
1>d:\adel\lplab\122\122\122.cpp(15): warning C4100: argv: неиспользованный формальный параметр
1>d:\adel\lplab\122\122\122.cpp(15): warning C4100: argc: неиспользованный формальный параметр
1>d:\adel\lplab\122\122\122.cpp(11): warning C4706: назначение в пределах условного выражения
1>d:\adel\lplab\122\122\122.cpp(14): warning C4715: f_test: значение возвращается не при всех путях выполнения
1>  L22.vcxproj -> D:\Adel\LPLab\L22\Debug\L22.exe
========== Сборка: успешно: 1, с ошибками: 0, без изменений: 0, пропущено: 0 ==========
```

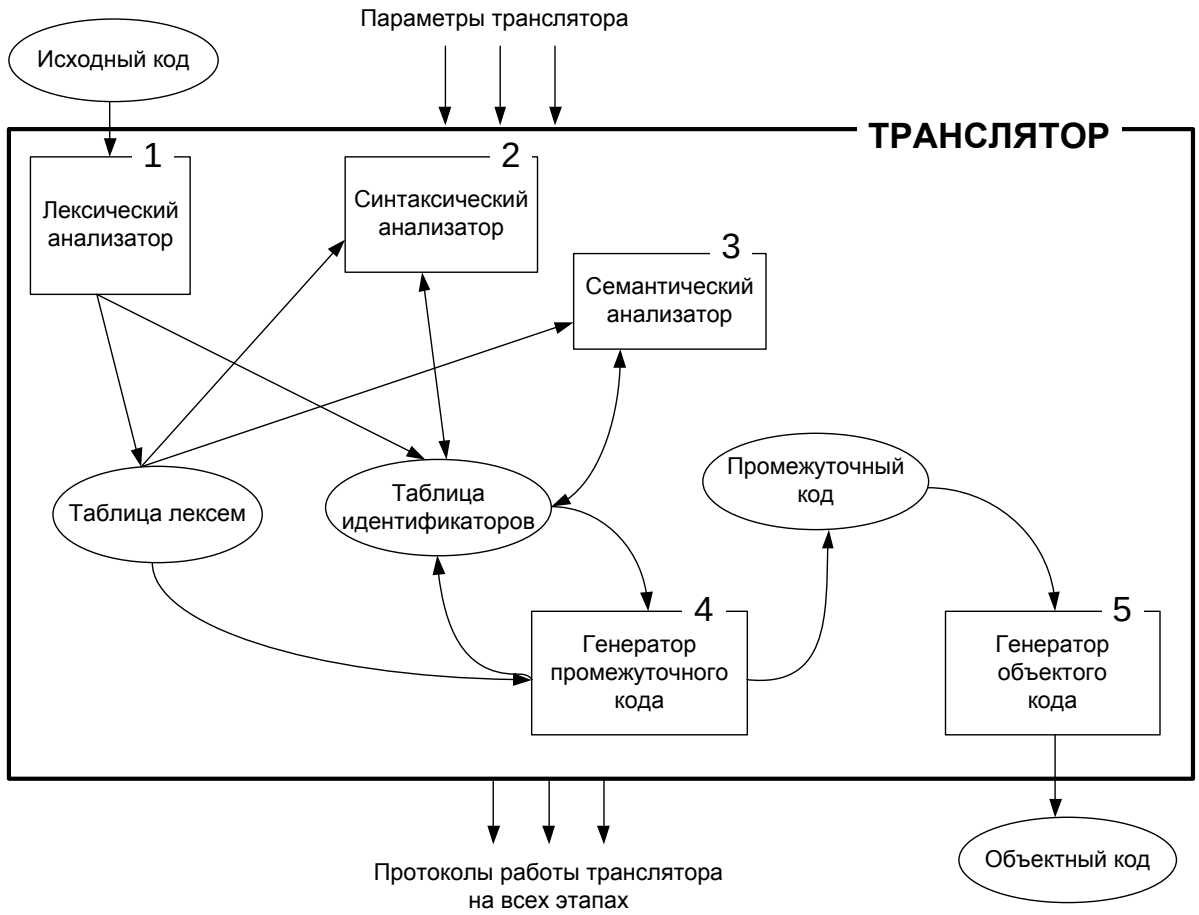
Результат выполнения:

```
a = 3
Для продолжения нажмите любую клавишу . . .
```

Семантика учебного компилятора для языка программирования svv-2015:

№	Правило
1	Наличие функции main
2	Усечение слишком длинных идентификаторов до 5 символов
3	Сначала осуществляется проверка на ключевые слова, а затем на идентификатор. Не допускаются идентификаторы совпадающие с ключевыми словами
4	Нет повторяющихся наименований функций
5	Нет повторяющихся объявлений идентификаторов
6	Предварительное объявление, применяемых функций
7	Предварительное объявление, применяемых идентификаторов.
8	Соответствие типов формальных и фактических параметров при вызове функций
9	Усечение слишком длинного значения string-литерала
10	Округление слишком большого значения integer-литерала
11	Если ошибка возникает на этапе лексического анализа, синтаксический анализ не выполняется
12	При возникновении ошибки в процессе лексического анализа, ошибочная фраза игнорируется (предполагается, что ее нет) и осуществляется попытка разбора следующей фразы. Граница фразы, любой сепаратор (пробел, скобка, запятая, точка с запятой и пр.)
13	Если 3 подряд фразы не разобраны, то работа транслятора останавливается
14	При возникновении ошибки в процессе синтаксического анализа, ошибочная фраза игнорируется (предполагается, что ее нет) и осуществляется попытка разбора следующей фразы. Граница фразы – точка с запятой.

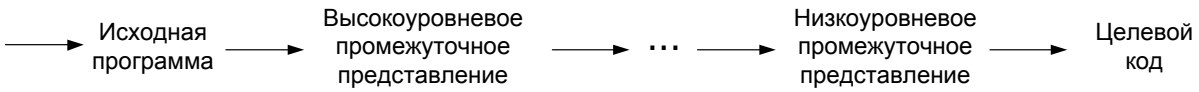
3. **Структура транслятора**



4. **Генерация промежуточного кода.**

Промежуточный код: код удобный для генерации объектного кода. Перед генерацией кода необходимо преобразовать выражения (сначала получить польскую запись, затем сгенерировать дополнительный код).

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы в цепочку символов выходного языка.



В процессе трансляции программы на некотором исходном языке в код для заданной целевой машины компилятор может построить последовательность промежуточных представлений. Высокоуровневые представления близки к исходному языку, а низкоуровневые – к целевому коду.

Способы внутреннего представления программ

Формы внутреннего представления программ:

- списочные структуры, представляющие синтаксические дерево;
- многоадресный код с явно именуемым результатом (тетрады);
- многоадресный код с неявно именуемым результатом (триады);
- обратная (постфиксная) польская запись операций;
- ассемблерный код или машинные команды.

Синтаксические деревья

Это структура, представляющая собой результат работы синтаксического анализатора и отражающая синтаксис конструкций входного языка.

Многоадресный код с явно именуемым результатом (тетрады)

Тетрады – форма записи операций из четырех составляющих: операция, два операнда и результат операции.

<операция>(<операнд1>,<операнд2>,<результат>).

Пример. Запись выражения A:=B*C+D-B*10 в виде тетрад:

1. * (B, C, T1)
2. + (Tl, D, T2)
3. * (B, 10,T3)
4. – (T2, T3, T4)
5. := (T4, 0, A)

где идентификаторы T1, T2, T3,T4 обозначают временные переменные.

Многоадресный код с неявно именуемым результатом (триады)

Триады – форма записи операций из трех составляющих: операция и два операнда.

<операция>(<операнд1>,<операнд2>)

Пример. Запись выражения A:=B*C+D-B*10 в виде триад:

1. * (B, C)
2. + (^1, D)
3. * (B, 10)
4. – (^2, ^3)
5. := (A, ^4)

Знак ^ означает ссылку операнда одной триады на результат другой.

Обратная польская запись операций

Обратная (постфиксная) польская запись — удобная форма записи операций и операндов для вычисления выражений. Эта форма предусматривает, что знаки операций записываются после операндов.

Вычисление выражений с помощью обратной польской записи

Вычисление выражений в обратной польской записи выполняется с помощью стека. Выражение просматривается в порядке слева направо, и встречающиеся в нем элементы обрабатываются по следующим правилам:

1. Если встречается операнд, то он помещается в вершину стека.
2. Если встречается знак унарной операции, то операнд выбирается с вершины стека, операция выполняется и результат помещается в вершину стека.
3. Если встречается знак бинарной операции, то два операнда выбираются с вершины стека, операция выполняется и результат помещается в вершину стека.

Вычисление выражения заканчивается, когда достигается конец записи выражения.

Ассемблерный код и машинные команды

Команды ассемблера представляют собой форму записи машинных команд

Внутреннее представление программы зависимо от архитектуры вычислительной системы, на которую ориентирован результирующий код.

5. **Пример:**польская запись.

tfi(ti,ti)
{
dti;
i=iv(ivi);
ri;
};
tfi(ti,ti)
{
dti;
dtfi(ti,ti,ti);
i=i(i,l,l)vi;
ri;
};
m
{
dti;
dti;
dti;
dti;
dti;
dti;
dtfi(ti);
i=i;
i=l;
i=l;
i=l;
i=i(i,i);
i=i(i,i);
pl;
pi;
pi;
pi(i);
rl;
};
	tfi(ti,ti)
	{
	dti;
	i= iiivv;
	ri;
	};
	tfi(ti,ti)
	{
	dti;
	dtfi(ti,ti,ti);
	i=ill@$_3$iv;
	ri;
	};
	m
	{
	dti;
	dti;
	dti;
	dti;
	dti;
	dti;
	dtfi(ti);
	i=i;
	i=l;
	i=l;
	i=l;
	i=ii@$_2$;
	i=ii@$_2$;
	pl;
	pi;
	pi;
	p**i@$_1$**;
	rl;
	};

6. **Пример:** генерация дополнительного кода.

tfi(ti,ti)
{
dti;
 	
i= iiivv;
ri;
};
tfi(ti,ti)
{
dti;
dtfi(ti,ti,ti);
i=ill@$_3$iv;
ri;
};
m
{
dti;
dti;
dti;
dti;
dti;
dti;
dtfi(ti);
i=i;
i=l;
i=l;
i=l;
 	
i=ii@$_2$;
i=ii@$_2$;
pl;
pi;
pi;
p**i@$_1$**;
rl;
};
	tfi(ti,ti)
	{
	dti;
	dti;
	i=iiv;
	i=iiv;
	ri;
	};
	tfi(ti,ti)
	{
	dti;
	dtfi(ti,ti,ti);
	dti;
	dti;
	i=ill@$_3$;
	i=iiv;
	ri;
	};
	m
	{
	dti;
	dti;
	dti;
	dti;
	dti;
	dti;
	dtfi(ti);
	i=i;
	i=l;
	i=l;
	i=l;
	i=ii@$_2$;
	i=ii@$_2$;
	pl;
	pi;
	pi;
	pi(i);
	rl;
	};

7. **Тетрады:** пример

Тетрады: операция (операнд1, операнд2, результат3)

start(p1,p2,p3)	Создать таблицу ссылок p1 = null p2 = null p3 = адрес таблицы ссылок
entry(p1,p2,p3)	Поместить ссылку на локальную функцию в таблицу ссылок. Инициализировать счетчик стека. p1 = адрес таблицы ссылок p2 = имя p3 = адрес ссылки
mentry(p1,p2,p3)	Поместить ссылку на главную локальную функцию в таблицу ссылок. Инициализировать счетчик стека. p1 = адрес таблицы ссылок p2 = имя p3 = адрес ссылки
ext(p1,p2,p3)	поместить ссылку на внешнюю функцию в таблицу ссылок p1 = адрес таблицы ссылок p2 = null p3 = адрес ссылки
stackaddr(p1,p2,p3)	Вычислить адрес в стеке p1 = смещение p2 = null p3 = адрес
push(p1,p2,p3)	Записать в стек p1 = длина p2 = значение p3 = адрес в стеке (null)
pop(p1,p2,p3)	Сдвинуть стек p1 = количество байт p2 = null p3 = null
+(p1,p2,p3)	Вычислить сумму двух integer-значений, результат поместить в стек p1 = значение 1 p2 = значение 2 p3 = адрес результата в стеке
*(p1,p2,p3)	Вычислить произведение двух integer-значений, результат поместить в стек p1 = значение 1 p2 = значение 2 p3 = адрес результата в стеке

cont(p1,p2,p3)	Конкатенация двух string-значений, результат поместить в стек p1 = значение 1 p2 = значение 2 p3 = адрес результата в стеке
str(p1,p2,p3)	Сформировать строку p1 = длина p2 = null p3 = адрес результата в стеке
store(p1,p2,p3)	Скопировать данные p1 = адрес источника p2 = адрес получателя p3 = null
callstd(p1,p2,p3)	Вызов внешней функции p1 = адрес в таблице ссылок p2 = null p3 = null
callloc(p1,p2,p3)	Вызов локальной функции p1 = адрес в таблице ссылок p2 = null p3 = null
prints(p1,p2,p3)	Писать строку в стандартный вывод p1 = строка p2 = null p3 = null
printi(p1,p2,p3)	Писать string-значение в стандартный вывод p1 = string-значение p2 = null p3 = null
goto(p1,p2,p3)	Переход по адресу p1 = адрес p2 = null p3 = null

		start(null,null,start0)

integer function
fi(integer x, integer y)
{
	tfi(ti,ti)
	{
		entry(start0,
		‘fi’,
		entryfi)
		
		stackaddr(0, null, fi01)
		stackaddr(4, null, fi02) // rc
		stackaddr(8, null, fix)
		stackaddr(12, null, fiy)

declare integer z;	dti;	push(4,0,fiz)
z= x*(x+y);	**dti;**	push(4,0,fi03)

	i=iiv;	+(fix,fiy,fi04) //push
		store(fi04,fi03,null)

	i=iiv;	*(fix,f03,fi05) //push
		store(fi05,fiz,null)

return z;};	ri};	store(fiz,fi02,null) pop(16, null, null) goto(fi01,null,null)

string function
fs(string a, string b) {
	tfi(ti,ti)
	{
		entry(start0,
		‘fs’,
		entryfs)
		
		stackaddr(0, null, fs01) // ret
		stackaddr(4, null, fs02) // rc
		stackaddr(8, null, fsa)
		stackaddr(12, null, fsb)

declare string c;	dti;	str(0, null,fs05) // new 1+255
		push(4,fs05,fsc) // адрес

declare string function substr(string a, integer p, integer n);	dtfi(ti,ti,ti);	ext(start0 ,'substr@s@i@i', fs04) //push

c = substr(a, 1,3)+ b;	**dti;**	str(0, null,fs06) // new 1+255
		push(4,fs06,fs07) // адрес

	dti;	str(0, null,fs08) // new 1+255
		push(4,fs06,fs09) // адрес

	i=ill@$_3$;	push(4,fs10, null) push(4,3,null) push(4,1,null) push(4,fsa,null) callstd(fs04,null,null) fs10:store(fs10,fs07,null) pop(16,null, null)
	i=iiv;	cont(fs07,fsb,fs11) store(fs11,fsc,null)
return c; };	ri;};	store(fsc,fc02,null) pop(10, null, null) goto(fs01,null,null)

main
{
	m
	{
		mentry(start0, null, null)
		stackaddr(0, null, main01) // ret
		stackaddr(4, null, main02) // rc

declare integer x;	dti;	push(4,0,mainx)
declare integer y;	dti;	push(4,0,mainy)

declare integer z;	dti;	push(4,0,mainz)

declare string sa;	dti;	str(0, null,main03) // new 1+255
		push(4, main01, mainsa) // адрес

declare string sb;	dti;	str(0, null,main04) // new 1+255
		push(4, main04, mainsb) // адрес

declare string sc;	dti;	str(0, null,main05) // new 1+255
		push(4, main05, mainsc) // адрес

declare integer function strlen(string p);	dtfi(ti);	ext(start0,'strlen@s', main04) //push
x = 1;	i=l;	store(1,mainx,null)
y = 5;	i=l;	store(5,mainy,null)
sa = '1234567890';	i=l;	store('1234567890',mainsa,null)
sb = '1234567890';	i=l;	store('1234567890',mainsa,null)

z = fi(x,y);	**i=ii@$_2$;**	push(4,main07,null) //ret
		push(4,0,main08) //rc
		push(4,3,null)
		push(4,1,null)
		callloc(entryfi,null,null)
		main05:store(main08,mainz,null)
		pop(16,null, null)

sc = fs(sa,sb);	**i=ii@$_2$;**	push(4,main09,null) //ret
		push(4,0,main10) //rc
		push(4,mainsb,null)
		push(4,mainsa,null)
		callloc(entryfs,null,null)
		main07:store(main10,mainsc,null)
		pop(16,null, null)

print 'контрольный пример';	pl;	prints('контрольный пример',null,null)

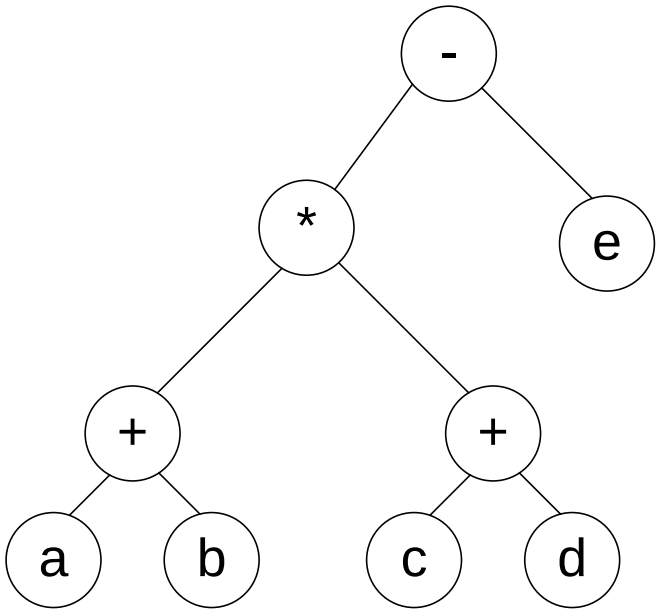
print z;	pi;	printi(mainz,null,null)
print sc;	pi;	prints(mainsc,null,null)

print strlen(sc);	pi@$_1$;	push(4, main11,null) //ret
		push(4,0,main12) //rc
		push(4,mainsc,null)
		callstd(main04,null,null)
		main11: pop(12,null, null)
		printi(main12, null, null)

return 0; };	rl; };	store(0,main02,null) goto(main01,null,null)

8. Вычисление выражений с помощью обратной польской записи (обход графа: слева направо, сначала листья, затем корень)

(a+b)*(c+d) - e



ab+cd+*e-

9. Вычисление за один просмотр

0	ab+cd+*e-	R_1=a
1	R_1b+cd+*e-	R_2=b
2	R_1R_2+cd+*e-	R_3= R_1+R_2
3	R_3cd+*e-	R_4=c
4	R_3R_4d+*e-	R_5=d
5	$R_3R_4R_5$+*e-	R_6= R_4+R_5
6	R_3R_6*e-	R_6= R_4+R_5
7	R_3R_6*e-	R_7= R_3*R_6
8	R_7e-	R_8=e
9	R_7R_8-	R_9= R_7-R_8
10	R_9	

Определим приоритет операций:

Приоритет	Операция
1	(
1	)
2	+
2	-
3	*
3	/

Алгоритм построения:

- исходная строка: выражение;
- результирующая строка: польская запись;
- стек: пустой;
- исходная строка просматривается слева направо;
- операнды переносятся в результирующую строку в порядке их следования;
- операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;
- операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
- отрывающая скобка помещается в стек;
- закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;
- по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

10. Пример.

Исходная строка	Результирующая строка	Стек
$(a+b)*(c+d)-e$		
$a+b)*(c+d)-e$		(
$+b)*(c+d)-e$	a	(
$b)*(c+d)-e$	a	+(
$)*(c+d)-e$	ab	+(
$*(c+d)-e$	$ab+$	
$(c+d)-e$	$ab+$	*
$c+d)-e$	$ab+$	(*
$+d)-e$	$ab+c$	(*
$d)-e$	$ab+c$	+(*
$)-e$	$ab+cd$	+(*
$-e$	$ab+cd+$	*
e	$ab+cd+*$	-
	$ab+cd+*e$	-
	$ab+cd+*e-$	

Стек организован по принципу LIFO.

11. Алгоритм построения польской записи

Легко расширить алгоритм так, чтобы он обрабатывал выражения, содержащие вызовы функций, элементы массива, другие виды скобок.

Приоритет	Операция
0	(
0	)
1	,
2	+
2	-
3	*
3	/
4	[
4	]

Алгоритм построения:

- исходная строка: выражение;
- результирующая строка: польская запись;
- стек: пустой;
- исходная строка просматривается слева направо;
- операнды переносятся в результирующую строку в порядке их следования;
- операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;
- операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
- запятая не помещается в стек, если в стеке операции, то все выбираются в строку;
- отрывающая скобка помещается в стек;
- закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;
- квадратная закрывающая скобка выталкивает все до открывающей и генерирует @n (индекс n указывает число операндов, разделенных запятыми);
- по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

ыИсходная строка	Результирующая строка	Стек
$a * \left(b + \left[[c,d] + e,g\right]\right) - k/[e,f]$		
$* \left(b + \left[[c,d] + e,g\right]\right) - k/[e,f]$	a	
$\left(b + \left[[c,d] + e,g\right]\right) - k/[e,f]$	a	$*$
$b + \left[[c,d] + e,g\right]) - k/[e,f]$	a	$* ($
$+\left[[c,d] + e,g\right]) - k/[e,f]$	ab	$* ($
$\left[[c,d] + e,g\right]) - k/[e,f]$	ab	$* (+$
$[c,d] + e,g]) - k/[e,f]$	ab	$* (+[$
$c,d] + e,g]) - k/[e,f]$	ab	$* (+[[$
$,d] + e,g]) - k/[e,f]$	abc	$* (+[[$
$d] + e,g]) - k/[e,f]$	abc	$* (+[[$
$] + e,g]) - k/[e,f]$	$abcd$	$* (+[[$
$+e,g]) - k/[e,f]$	$abcd@_2$	$* (+[$
$e,g]) - k/[e,f]$	$abcd@_2$	$* (+[+$
$,g]) - k/[e,f]$	$abcd@_2e$	$* (+[+$
$g]) - k/[e,f]$	$abcd@_2e +$	$* (+[$
$]) - k/[e,f]$	$abcd@_2e + g$	$* (+[$
$) - k/[e,f]$	$abcd@_2e + g@_2$	$* (+$
$-k/[e,f]$	$abcd@_2e + g@_2 +$	$*$
$k/[e,f]$	$abcd@_2e + g@_2 +*$	$-$
$/[e,f]$	$abcd@_2e + g@_2 +* k$	$-$
$[e,f]$	$abcd@_2e + g@_2 +* k$	$-/$
$e,f]$	$abcd@_2e + g@_2 +* k$	$-/[$
$,f]$	$abcd@_2e + g@_2 +* k\mathrm{e}$	$-/[$
$f]$	$abcd@_2e + g@_2 +* k\mathrm{e}$	$-/[$
$]$	$abcd@_2e + g@_2 +* k\mathrm{e}\, f$	$-/[$
	$abcd@_2e + g@_2 +* k\mathrm{e}\, f@_2$	$-/$
	$abcd@_2e + g@_2 +* k\mathrm{e}\, f@_2/$	$-$
	$abcd@_2e + g@_2 +* k\mathrm{e}\, f@_2/-$	