МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора LLH-2019»

Выполнил студент Лойко Любарт Геннадьевич

(Ф.И.О.)

Руководитель проекта преп.-стаж. Котович Дмитрий Витальевич

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-стаж. Котович Дмитрий Витальевич

(учен. степень, звание, должность, подпись, Ф.И.О.)

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-стаж. Котович Дмитрий Витальевич

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2019

**Уводзіны**

Імператыўная мова праграмавання LLH-2019 з’яўляецца працэдурнай, слаба тыпізаванай, кампіліруемай.

Імператыўнае праграмаванне -гэта парадыгма праграмавання (стыль напісання зыходнага кода кампутарнай праграмы), для якой характэрна наступнае:

- у зыходным кодзе праграмы запісваюцца інструкцыі (каманды);

- інструкцыі павінны выконвацца паслядоўна;

- дадзеныя, якія атрымліваюць пры выкананні папярэдніх інструкцый, могуць чытацца з памяці наступнымі інструкцыямі;

- дадзеныя, атрыманыя пры выкананні інструкцыі, могуць запісвацца ў памяць.

Працэдурны мова праграмавання - мова высокага ўзроўню, у якім выкарыстоўваецца метад разбіцця праграм на асобныя звязаныя паміж сабой модулі - падпраграмы (працэдуры, блокі і функцыі).

# Транслятар - гэта комплекс асобных праграм, якія дазваляюць пераўтвараць зыходны код на адной мове праграмавання ў зыходны код на іншай мове праграмавання.

# Класічны транслятар складаецца з наступных частак:

# - лексічны аналізатар;

# - сінтаксічны аналізатар;

# - семантычны аналізатар;

# - генератар кода, або інтэрпрэтатар.

# Усе часткі транслятара, узаемадзейнічаючы паміж сабой, апрацоўваюць ўваходных тэкст і будуюць для яго эквівалентны тэкст на байт-кодзе.

# 1 Спецыфікацыя мовы праграмавання

* 1. **Характарыстыка мовы праграмавання**

Імператыўная мова праграмавання LLH-2019 з’яўляецца працэдурнай, слаба тыпізаванай, кампіліруемай, з статычнай тыпізацыяй. Падтрымлівае працэдурную парадыгму праграмавання.

* 1. **Алфавіт мовы**
  2. **Ужываемыя сепаратары**

Сепаратары служаць для раздзялення аперацый мовы. Сепаратары, ужываемыя у мове праграмавання LLH-2019 прыведзены ў табліцы 1.1.

Табліца 1.1 – Ужываемыя сепаратары

|  |  |
| --- | --- |
| Сепаратар | Назчначэнне сепаратара |
| ; | Раздзяленне інструкцый |
| { } | Праграмны блок |
| ( ) | Параметры функцыі. Умова. Заданне прыярэтэту аперацый |
| “ ” (прабел) | Радзяленне слоў |
| , | Раздзяленне параметраў функцый |
| [] | Індэкс масіва |

* 1. **Ужываемыя кадзіроўкі**
  2. **Тыпы дадзеных**

Дапускаецца выкарыстанне фундаментальных тыпаў дадзеных, прыведзеных у табліцы 1.2. Тыпы дадзеных карыстальнікаў не падтрымліваюцца.

Табліца 1.2 – Фундаментальныя тыпы дадзеных

|  |  |
| --- | --- |
| Тип данных | Описание |
| integer | З’яўляецца знакавым пералічаемым тыпам дадзеных (2 байта).  Прыдназначаны для арыфметычных і лагічных аперацый над цэлымі лікамі. Ініцыялізацыя па змаўчанні: 0. Максімальнае значэнне 216-1  Прадугледжаны наступныя арыфметычныя аперацыі:  + - бінарная аперацыя складання (лік + лік);  - - бінарная аперацыя розніцы (лік - лік);  \* - бінарная аперацыя здабытку (лік \* лік);  / - бінарная аперацыя дзялення (лік / лік);  > - бінарная лагічная аперацыя параўнання (лік / лік);  <- бінарная лагічная аперацыя параўнання (лік / лік);  ? - бінарная лагічная аперацыя роўнасці (лік / лік);  ! - бінарная лагічная аперацыя няроўнасці (лік / лік);  & - бінарная аперацыя лагічная памнажэння (лік / лік);  | - бінарная аперацыя лагічная складання (лік / лік);  % - бінарная аперацыя атрымання астатка ад дзялення (лік % лік). |
| string | З’яўляецца ўказальнікам на радок (256 байт). Выкарыстоўваецца для працы з сімваламі, кожны сімвал займае 1 байт. Максімальная колькасць сімвалаў – 256. Ініцыялізацыя па змаўчанні: адрас пачатку масіва;  Прадугледжаны наступныя аперацы:  [] – бінарная аперацыя атрымання адраса элемента радка (радок[лік]).  ^- унарная аперацыя атрымання значэнне па ячэйцы масіва (^адрас). |
| char | З’яўляецца цэлалікавым тыпам дадзеных(1 байт). Выкарыстоўваецца для працы з сімваламі а таксама для арыфметычных і лагічных аперацый над цэлымі лікамі. Ініцыялізацыя па змаўчанні: 0;  Прадугледжаны наступныя арыфметычныя аперацыі:  + - бінарная аперацыя складання (лік + лік);  - - бінарная аперацыя розніцы (лік - лік);  \* - бінарная аперацыя здабытку (лік \* лік);  / - бінарная аперацыя дзялення (лік / лік);  > - бінарная лагічная аперацыя параўнання (лік / лік);  <- бінарная лагічная аперацыя параўнання (лік / лік);  ? - бінарная лагічная аперацыя роўнасці (лік / лік);  ! - бінарная лагічная аперацыя няроўнасці (лік / лік);  & - бінарная аперацыя лагічная памнажэння (лік / лік);  | - бінарная аперацыя лагічная складання (лік / лік);  % - бінарная аперацыя атрымання астатка ад дзялення (лік % лік). |
| array | З’яўляецца ўказальнікам на масіў цэлых ліка(адвольны памер, але не меньш 4 байт). Выкарыстоўваецца для працы з масівамі цэлых лікаў. Ініцыялізацыя па змаўчанні: адрас пачатку масіва; Прадугледжаны наступныя аперацы:  [] – бінарная аперацыя атрымання адраса элемента радка (радок[лік]).  ^- унарная аперацыя атрымання значэнне па ячэйцы масіва (^адрас). |
| sarray | З’яўляецца ўказальнікам на масіў, кожны элемент якога з’яўляецца радком з 256 сімвалаў(адвольны памер, кратны 256 байт). Выкарыстоўваецца для працы з масівам радкоў. Ініцыялізацыя па змаўчанні: адрас пачатку масіва; Прадугледжаны наступныя аперацы:  [] – бінарная аперацыя атрымання адраса элемента радка (радок[лік]).  ^- унарная аперацыя атрымання значэнне па ячэйцы масіва (^адрас). |

* 1. **Пераўтварэнні тыпаў дадзеных**

Цэлалікавыя тыпы дадзеных неяўна пераводзяцца да іншых цэлалікавых тыпаў дадзеных (з магчымай стратай дадзеных у выпадку прывядзення ў тып з меньшым аб’ёмам). Для пераўтварэння радковых і цэлалікавых тыпаў прадугледжаныя стандартныя функцыі intTOChar(лік).

* 1. **Ідэнтыфікатары**

Ідэнтыфікатары могуць выступаць у якасці імёнаў функцый, імёнаў параметраў, імёнаў зменных. Зарэзерваваныя ідэнтыфікатары не прадугледжаныя.

Імя ідэнтыфікатара складаецца па наступных правілах:

- складаецца з малых літар лацінскага алфавіту і лічбаў [a..z], [0..9].

- даўжыня ідэнтыфікатара не павінна перавышаць 10 знакаў. Пры перавышэнні максімальна дапушчальнай даўжыні даўжыня ідэнтыфікатара усякаецца да 10.

Рэгулярны выраз:

Дзе *А* – рэгулярны выраз для ідэнтыфікатара,

*l* – мноства літар,

*d* –мноста лічбаў.

Регклярное выражнение + матиматич формат задания

Прыклады няправільных ідэнтыфікатараў:

-імя (сімвалы не належаць лацінскаму алфавіту)

-123name (ідэнтыфікатар не можа пачынацца з лічбы)

-id name (ідэнтыфікатар не можа ўтрымліваць знак прабела)

-iDname (ідэнтыфікатар не можа ўтрымліваць вялікія літары)

-I,d\_n@m.=e+$ (ідэнтыфікатар не можа ўтрымліваць знакаў прыпынку, ніжніх падкрэсліванняў, сімвалаў аперацый і т.д.);

-idnamenamen (даўжыня ідэнтыфікатара больш 10 сімвалаў, ён будзе ўсечаны)

Прыклады правільных ідэнтыфікатараў:

-idname

-idname123

* 1. **Літэралы**

Прадугледжаны лікавыя, радковыя і сімвальныя літэралы. Правілы запісу прыведзены ў табліцы 1.3.

Табліца 1.3 – Праілы запісу літэралаў

|  |  |
| --- | --- |
| Тып літэрала | Апісанне літэрала |
| Лікавыя, дзесяцічныя | Утрымліваюць толькі лічбы [0..9] без дробавай часткі. Максімальнае значэнне 216-1 |
| Лікавыя, шаснадцацірычныя | Утрымліваюць лічбы [0..9] і сімвалы лацінскага алфавіту [a..f] без дробавай часткі. Заканчваюцца на сімвал ‘h’. Максімальнае значэнне 216-1 |
| Лікавыя, васьмірычныя | Утрымліваюць лічбы [0..7] без дробавай часткі. Заканчваюцца на сімвал ‘o’. Максімальнае значэнне 216-1 |
| Лікавыя, дваічныя | Утрымліваюць лічбы [0..1] без дробавай часткі. Заканчваюцца на сімвал ‘b’. Максімальнае значэнне 216-1 |
| Радковыя | Сімвалы, змешчаныя ў двайных кавычках “”, лік якіх не прывышае 255. Даступныя ўсе знакі знакі прыпынку, сімвалы ангельскага і рускага алфавітаў акрамя двайных кавычак “”. Пры выхадзе за межы дапушчальнасці, выводзіцца адпаведная памылка. |
| сімвальныя | Сімвал, змешчаны ў адзінарных кавычках ‘’. Даступныя ўсе знакі знакі прыпынку, сімвалы ангельскага і рускага алфавітаў акрамя адзінарных кавычак ‘’. |

Прыклады няправільных літэралаў:

-1\_3-2p4a (лікавы літэрал не можа ўтраымліваць літар, калі ён не шаснадцацірычны)

-321b (для дваічнага літэрала дапусцімы толькі сімвалы: 1, 0.

-2398o (для васьмірычнага літэрала не дапусцімы лічбы больш за 7)

-32ao (для васьмірычнага літэрала не дапусцімы літары)

-32оo (для васьмірычнага літэрала толькі апошні сімвал павінен быць о)

-“strin”g” (радковы літэрал не мода ўтрымліваць “)

-‘char’ (сімвальны літэрал не можа ўтрымліваць больш аднаго сімвала ў кавычках)

-‘’’ (сімвальны літэрал не можа складацца з адзінарнай кавычкі ‘)

Прыклады правільных літэралаў:

-1234

-1001

-1234h

-1234o

-1234adfh

-1001b

-1001o

* 1. **Аб’яўленне дадзеных і вобласць бачнасці**

У мове праграмавання LLH-2019 неабходна аб’явіць зменную да яе выкарыстання. Вобласцю бачнасці зменнай з'яўляецца функцыя, у якой яна вызначана. Вобласць бачнасці пабудавана па прынцыпе C++ (зверху ўніз). Для аб'явы зменных выкарыстоўваецца наступная канструкцыя:

declare  <Тып дадзеных> <ідэнтыфікатар>;

* 1. **Ініцыялізацыя дадзеных**

Пры аб'яўленні зменнай дапускаецца ініцыялізацыя дадзеных. Пры гэтым зменнай будзе прысвоена значэнне літэрала або ідэнтыфікатара, які стаяў справа ад знака роўнасці. Аб'ектамі-ініцыялізатарамі могуць быць ідэнтыфікатары, літэралы, выказванні і вяртаемыя значэнні функцый для лікавых тыпаў дадзеных. Для array і sarray ініцыялізацыя абавязковая і аб'ектамі-ініцыялізатарамі могуць быць толькі лікавыя літэралы, якія азначаюць колькасць элементаў масіва. Для string ініцыялзацыя забараняецца, па змаўчанні значэнне зменнай задаецца як “” (пусты радок). Пры аб'яўленні без ініцыялізацыі прадугледжаны значэнні па змаўчанні: значэнне 0 для цэлалікавага тыпу дадзеных.

declare integer intid = 1001b;

declare integer intid = ‘a’;

declare char charid = ‘a’;

declare char charid = 10h;

declare string str;

declare array arr = 5;

declare sarray sarr = 6;

* 1. **Інструкцыі мовы**

У мове праграмавання LLH-2019 ужываюцца інструкцыі, прадстаўленыя ў табліцы 1.4.

|  |  |
| --- | --- |
| Інтсрукцыя мовы | Сінтаксіс |
| Аб’яўленне зменнай | declare <тып дадзеных> <ідэнтыфікатар>;  declare <тып дадзеных> <ідэнтыфікатар> = <літэрал>;  declare <тып дадзеных> <ідэнтыфікатар> = <выраз>; |
| Аб’яўленне функцыі | < тып дадзеных >  function <ідэнтыфікатар>([параметры])  {<цела функцыі>;  return [<выраз>];}; |
| Прысваенне | integer:  <ідэнтыфікатар> = < літэрал >;  <ідэнтыфікатар> = < выраз >;  <ідэнтыфікатар 1> = <ідэнтыфікатар 2>;  char:  <ідэнтыфікатар> = < літэрал >;  <ідэнтыфікатар> = < выраз >;  <ідэнтыфікатар 1> = <ідэнтыфікатар 2>; |
| Галоўная функцыя | main  {  <інструкцыі мовы>;  return ;  } |
| Выклік функцый | <ідэнтыфікатар функцыі> ([параметры]+); |
| Вяртанне з функцыі | return <ідэнтыфікатар>;  return <літэрал>;  return <выраз>;  return; |

Табліца 1.4 - Інструкцыі мовы

* 1. **Аперацыі мовы**

Унарная аперацыя атрымання значэнне па адрасе “^” ўжывальная толькі для масіваў. Астатнія аперацыі ў мове праграмавання ужывальныя выключна да лікавых тыпаў дадзеных. Аперацыі прыведзеныя ў табліцы 1.5.

Камутатыўнасць – кам.

Дыстрыбутыўнасць – дыс.

Асацыятыўнасць – ас.

|  |  |
| --- | --- |
| Тып | Аперацыі |
| бінарныя арыфметычныя. Дапусцімыя толькі лікавыя тыпы аперандаў. Вяртаемае значэнне лікавага тыпу | () – прыярытэтнасць выканання (прыярытэт-1)  + ­­­ ̶ складанне (прыярытэт-4) кам., ас.  - ̶ розніца (прыярытэт-4) кам., ас.  \* ̶ здабытак (прыярытэт-3)кам., дыс., ас.  / ̶ дзяленне (прыярытэт-3) дыс.  % - астатак ад дзялення (прыярытэт-3) дыс.  = - прысваенне (прыярытэт – 7) |
| бінарныя лагічныя.  Дапусцімыя толькі лікавыя тыпы аперандаў. Вяртаемае значэнне лікавага тыпу | | - лагічнае складанне (прыярытэт-6)  & - дагічны здабытак (прыярытэт-6) |
| Бінарныя аперацыі параўнання. Дапусцімыя толькі лікавыя тыпы аперандаў. Вяртаемае значэнне лікавага тыпу | > - параўнанне аператараў (прыярытэт-5)  < - параўнанне аператараў (прыярытэт-5)  ? – роўнасць аператаў (прыярытэт-5)  ! – няроўнасць аператараў (прыярытэт-5) |
| Для працы з масівамі. Дапусцімыя толькі аперанды-ўказальнікі на масівы | ^ - атрыманне значэнне па адрасе (прыярытэт-3)  []- атрыманне адраса элемента масіва (прыярытэт-2) |

Табліца 1.5 - Аперацыі мовы

Аперацыі параўнання і лагічныя аперацыі вяртаюць 1 калі ўмова выконваецца, і 0 калі ўмова не выконваецца.

Масівы трэба выкарыстоўваць толькі з аперацый [] для атрымання адраса элемента. Перад масівамі таксама трэба выкарыстоўваць аперыцю “^” для атрымання значэння па адрасе, калі масіў не з’яўляецца параметрам выклікаемай функцыі.

Адрасныя аперацыі а таксама аперацыі з радкамі недапусцімыя.

парадак

* 1. **Выразы і іх вылічэнні**

Прадугледжаны наступныя правілы складання выразаў:

- выразы запісваюцца да ўводу сепаратара ";";

- разглядаюцца злева направа;

- для змены прыярытэту аперацыі выкарыстоўваюцца круглыя дужкі ();

- рэалізацыя выразаў адбываецца з дапамогай зваротнага польскага запісу.

- Параметры функцый перадаюцца зправа налева.

Выразы дапускаюць выкарыстанне вяртаемых значэнняў функцый а таксама выкарыстанне значэнняў элементаў масіваў.

У выразах могуць выкарыстоўвацца ўсе даступныя аперацыі.

**1.14 Праграмныя канструкцыі мовы**

Праграмныя канструкцыі прадстаўлены ў табліцы 1.7.

|  |  |
| --- | --- |
| Канструкцыя | Прадстаўленне ў мове |
| Галоўная функцыя | main  "{"  <інструкцыі мовы>  return;  "}" |
| Функцыя | <тып дадзеных> function <ідэнтыфікатар> " (" [параметры] ")"  "{"  < інструкцыі мовы >  return [<выраз>];  "}" |
| Кропка ўвахода | main |
| Цыкл | while(<выраз>) "{ "<інтсрукцыі>[break;] [<інструкцыі>] "}" |
| Ўмова | if " (" <выраз> ")" "{"  <інструкцыі>  }[else if " ("<выраз>")" "{" <інструкцыі> "}" ]  [else "{" <інструкцыі> "}" ] |

Табліца 1.7 - Праграмныя канструкцыі мовы.

У мове праграмавання LLH-2019 дапусцімы канструкцыі функцый, умоў і цыклаў. Кожная функцыя павінна мець ключавое слова return, якое вяяртанне з цела функцыі. Пасля return павінен быць выраз, калі тып функцыі не зададзены як void. Канструкцыя цыкла пачынаецца з ключавога слова while, пасля чаго ідзе ўмова выканання цела цыкла. Толькі ў целе цыкла можа выкарыстоўвацца ключавое слова break, якое абазначае датэрміновы выхад з цыкла.

* 1. **Вобласць бачнасці ідэнтыфікатараў**

Усе ідэнтыфікатары, абвешчаныя ўнутры функцыі, з'яўляюцца лакальнымі. Глабальных зменных, абвешчаных па-за межамі якога-небудзь блок, мова не дапускае. Параметры бачныя толькі ўнутры функцыі, у якой абвешчаныя.

* 1. **Семантычныя праверкі**

Пералік семантычных праверак, прадугледжаных мовай, прыведзены ў табліцы 1.8.

|  |  |
| --- | --- |
| Нумар | Правіла |
| 1 | Ключавое слова break можа выкарыстоўвацца толькі ўнутры цела цыкла |
| 2 | Прывышэнне памера лікавых, сімвальных і радковых літэралаў |
| 3 | Паўторнае аб’яўленне ідэнтыфікатараў |
| 4 | Выкарыстанне ідэнтыфікатараў без аб’яўлення |
| 5 | Пасля дэклярацыі масіва павінны быць указаны памер масіва |
| 6 | Забаронена выкарыстанне масіва без указання індэкса |
| 7 | Забаронена ініцыялізацыя радка |
| 8 | Пасля назвы функцыі павінны адкрывацца дужкі, для ўказання параметраў |
| 9 | Забаронена выкарыстоўваць літэрал пустога радка |
| 10 | Забаронена выкарыстоўваць глабальныя пераменныя і літэралы |
| 11 | Толькі функцыі могуць мець тып дадзеных void |

Табліца 1.8 - Пералiк семантычных праверак

* 1. **Размеркаванне аператыўнай памяі на этапе выканання**

Трансліраваны код выкарыстоўвае дзве вобласці памяці. У сегмент канстант уваходзяць ўсе літэралы, выкарыстоўваемыя ў зыходным кодзе. Ўсе зменныя і параметры функцый захоўваюцца ў стэку толькі пад час выканання функцый, у якіх яны аб’ялены, і знішчаюцца са стэка як толькі функцыя аб’яўлення заканчвае сваё выкананне. Закошт гэтага вызначаецца функцыянальная вобласць бачнасці, а таксама з-за таго, што кожна зменнай ў адпаведнасць ставіцца імя функцыі, ў якой зменная была аб’ялена, што дазваляе выкарыстоўваць аднолькавыя па назве пераменныя ў розных функцыях. Такім чынам у аператыўнай памяці хаваюцца толькі дадзеныя выконваемых функцый, што значна эканоміць аператыўную памяць.

* 1. **Стандартная бібліятэка і яе склад**

У мове LLH-2019 прысутнічае стандартная бібліятэка, якая паключаецца аўтаматычна пры трансляцыі зыходнага кода ў ассэмблер. Змест стандартнай бібліятэкі прадстаўлена у табліцы 1.9.

|  |  |
| --- | --- |
| Функцыя | Апісанне |
| void function printStr(string) | Выводзіць радок у кансоль |
| void function printLine() | Друкуе ў каноль сімвал пераноса радка |
| void strCopy(string dest,string src) | Капіруе ў радок dest радок src |
| string intToChar(integer) | Вяртае лікавы параметр ў выглядзе радка |
| void strConcat(string dest,string src) | Дабаўляе ў канец радка dest радок src |
| integer readNum() | Вяртае лік, уведзены з кансолі |
| string readLine() | Вяртае радок, уведзены з кансолі |
| integer rnd() | Вяртае выпадковы лік ў дыяпазано ад 216 -1 да -216 |
| integer \_pow(integer deg,integer num) | Узводзіць num у ступень deg, вяртае значэнне num |

Табліца 1.9 – Змест стандартнай бібліятэкі

* 1. **Увод і вывад дадзеных**

Для вывада радка выкарыстоўваецца функцыя стандартнай бібліятэкі printStr, для вывада ліка, яго трэба перавесці ў радок, што можна зрабіць з дапамогай функцыі стандартнай бібліятэкі intToChar(), пасля чаго атрыманы радок можна будзе вывесці.

Увод дадзеных ажжыцяўляецца з дапамагай функцый стандартнай бібліятэкі readLine() і readNum() для ўвода радка і ліка адпаведна.

* 1. **Кропка ўваходу**

Кропкай ўвахода з’яўляецца ключавое слова main. Яно не можа адсутнічаць. Зыходны код павінен мець хаця б адну функцыю, ў гэтым выпадку адзінай функцыя з’яўляецца main,што правяраецца на сінтаксічнай фазе транслятара.

* 1. **Прэпрацэссар**

У мове LLH-2019 прэпрацэссар адсутнічае.

* 1. **Пагадненне аб выкліках**

Выкарыстоўваецца пагадненне аб выкліках stdcall, якое прадугледжвае перадачу параметраў функцыі зправа налева, вызваленне стэка выклікаемай функцыяй. Вяртаемае значэнне функцыі памяшчаецца ў рэгістр eax.

* 1. **Аб’ектны код**

Код, на мове праграмавання LLH-2019 трансліруецца ў мову ассэмблер.

* 1. **Класіфікацыя паведамленняў транслятара**

Класіфікацыя ажыццяўляецца згодна з нумарам памылкі.

Сэрыі памылак:

* 0-100 – сістэмныя памылкі
* 101-110 – памылкі параметраў
* 111-200 – памылкі адкрыцця файлаў
* 201-300 – лексічныя і семантычныя памылкі
* 601-700 – сінтаксічныя памылкі
  1. **Кантрольны прыклад**

Кантрольны прыклад, напісаны на мове LVV-2019, прадстаўлены ў дадатку А.

**Глава 2. Структура транслятара**

**2.1 Кампаненты транслятара, іх назначэнне і прынцыпы ўзаемадзеяння**

Транслятар ажыццяўляе пераўтварэнне праграмы, прадстаўленай на адной з моваў праграмавання, у праграму на іншай мове. Схема транслятара прадстаўлена на схеме 1.1.



Схема 1.1 – Транслятар

Транслятар зкладаецца з наступных кампанентаў:

-Лексічны аналізатар. Правярае зыходны код на дапусцімасць сімвалаў, і фарміруе з яго табліцу лексем і ідэнтыфікатараў.

-Сінтаксічны аналізатар. Правярае структуру зыходнага кода і яго сінтаксіс на падставе атрыманых ад лексічнага аналізатара лексем. У выпадку паспяховага разбора кода фарміруе дрэва разбора.

-Семантычны аналізатар. Правярае сэнсавую правільнасць канструкцый зыходнага кода на падставе атрыманых ад лексічнага аналізатара табліц лексем і ідэнтыфікатараў.

-Генератар кода. Пры паспяховым аналізе зыходнага тэкста, пачынаецца генерацыя кода, на падставе дапоўненых табліц лексем і ідэнтыфікатараў. На выхадзе атрымліваецца код на іншай мове праграмавання, які можа быць пераўтвораны ў аб’ектны файл.

**2.2 Пералічэнне уваходных параметраў транслятара**

|  |  |  |
| --- | --- | --- |
| Уваходны параметр | Апісанне | Значэнне па змаўчанні |
| /in:<імя\_файла> | Уваходна файл з зыходным кодам з пашырэннем .txt | няма |
| /log:<імя\_файла> | Файл для запісу інфармацыі аб аналізе зыходнага кода і яго трансляцыі | <імя\_файла>.log.txt |
| /out:<імя\_файла> | Файл, у які запісваецца атрыманы ў ходзе трансляцыі код на ассэмблер | <імя\_файла>.out.asm |
| /syn:<значэнне> | Калі параметр не роўны нулю ў файл пратакола будзе запісвацца дадатковая інфармацыя аб ходзе сінтаксічнага аналіза | 0 |
| /pol:<значэнне> | Калі параметр не роўны нулю ў файл пратакола будзе запісвацца дадатковая інфармацыя пераводзе выразаў у польскую натацыю | 0 |

**2.3** **Пратаколы, фарміруемыя транслятарам**

|  |  |
| --- | --- |
| Пратакол | Апісанне |
| файл параметра /log: | Утрымлівае інфармацыю аб ходзе трансляцыі |
| выходны файл параметра /out: | Утрымлівае трансліраваны на мову ассэмблера код |
|  |  |

# Глава 3. Распрацоўка лексічнага аналізатара

## 3.1 Структура лексічнага аналізатара

Лексічны аналізатар - праграма, якая прымае на ўваход тэкст (паслядоўнасць знакаў з алфавіту) і якая разбівае яго на падрадкі (лексемы) у адпаведнасці з некаторым наборам рэгулярных выразаў.



## 3.2 **Кантроль уваходных сімвалаў**

Кожнаму сімвалу табліцы ASCII адпавядае значэнне ў масіве прыведзенаму ў дадатку А, якое вызначае дапушчальнасць адпаведнага сімвала.

## 3.3 Выдаленне залішніх сімвалаў

Лішнімі сімваламі з’яўляюцца сімвалы прабелу, табуляцыі і пераноса радка**.** На этапе стварэння абыходзяцца паслядоўна сімвалы зыходнага кода. Пры сустрэчы лішніх сімвалаў яны не заносяцца ў табліцу лексем і абыход сімвалаў працягваецца.

## 3.4 Пералічэнне ключавых слоў

Ключавыя словы і іх лексемы прыведзены у табліцы

|  |  |  |
| --- | --- | --- |
| Зыходнае слова | Лексема | Рэгулярны выраз |
| main | m |  |
| function | f |  |
| integer | t |  |
| void | t |  |
| string | t |  |
| char | t |  |
| array | t |  |
| sarray | t |  |
| declare | d |  |
| while | w |  |
| break | b |  |
| if | k |  |
| else | a |  |
| return | r |  |
| <памылка> | е |  |
| ( ) | ( ) |  |
| [ ] | [ ] |  |
| { } | { } |  |
| ; | ; |  |
| = | = |  |
| + | v |  |
| - | v |  |
| \* | v |  |
| / | v |  |
| % | v |  |
| ^ | v |  |
| ! | v |  |
| > | v |  |
| < | v |  |
| ? | v |  |
| , | , |  |
| <літэрал радка> | l | “(<лічба>+<літара>)+” |
| <літэрал дзесяцічнага ліка> | l | (<лічба>)+ |
| <літэрал дваічнага ліка> | l | (1+0)+b |
| <літэрал васьмірычнага ліка> | l | (1+2+3+4+5+6+7+0)+o |
| <літэрал шаснадцацірычнага ліка> | l | (<лічбы>+<a..f>)+h |
| <літэрал сімвала> | l | ‘<літара>’ |
| <ідэнтыфікатар> | i | <літара>(<літара>+<лічба>)+ |

## 3.5 Асноўныя структуры дадзеных

Асноўныя структуры, выкарыстаныя для пабудовы табліц лексем і ідэнтыфікатараў прадстаўлены ў дадатку А. Лексема ўтрымлівае інфармацыю аб сваім становішчы, індэкс адпавядаючаму ёй ідэнтыфікатару, а таксама сістэму злічэння. Элемент табліцы ідэнтыфікатараў ўтрымлівае індэкс ў табліцы лексем, тып дадзеных, тып ідэнтыфікатара, функцыя, якой ідэнтыфікатар належыць а таксама значэнне, калі ідэнтыфікатар з’яўляецца літэралам. Апрача ідэнтыфікатараў На этапе лексічнага аналіза ствараецца табліца функцый, якая ўтрымлівае тып вяртаемага значэння функцый, яе параметры і лакальныя зменныя.

## 3.6 Структура і пералічэнне паведамленняў лексічнага аналізатара

Спіс дапусцімых памылак на лексічны аналізатары прыведзены ў табліцы

|  |  |
| --- | --- |
| код памылкі | тэкст памылкі |
| 200 | maximum number of lexem |
| 201 | maximum number of id |
| 204 | lexic error |
| 215 | errors count reached the limit |
| 217 | parametrs count reached the limit |
| 218 | functions count reached the limit |

## 3.7 Прынцып апрацоўкі памылак

Пры паспяховым аналізе зыходнага тэкста лексічны аналізатар выводзіць атрыманую табліцу лексем, з указаннем нумароў радкоў. Пры паведамленні аб памылцы выводзіцца радок памылковай лексемы, пачынаючы адлік з 1, пазіцыя пачатку памылковай лексемы ў радку, пачынаючы з 0, код і тэкст памылкі. Пры з’яўденні памылкі яна запісваецца ў адпаведны спіс і праца аналізатара працягваецца да тых пор, пакуль колькасць памылак ў спісе не дасягне рэўнай адметкі, вызначынай параметрам. У гэтым выпадку праца лексічнага аналізатара спыняецца і выводзяцца памылкі, захаваныя ў спісе. У выпадку, калі спіс не перапоўніўся аналізатар працягвае працу да канца, пасля чаго спыняецца праца транслятара а атрыманыя памылкі выводзяцца.

## 3.8 Параметры лексічнага аналізатара

Параметрамі лексічнага аналізатара з’яўляецца файл, куды будзе запісвацца інфармацыя аб ходзе аналіза, зыходны тэкст, які будзе апрацоўвацца аналізатарам і максімальная колькасць памылак, пры якой аналізатар будзе працягваць працу.

**3.9 Алгарытм лексічнага аналізатара**

Алагарытм лексічнага аналізатара засноўваецца на пасімвальным абыходзе зыходнага тэкста з мэтай выдзялення лексем, шляхам параўанання словаў зыходнага тэкса з ключавымі словамі мовы LLH-2019, альбо шляхам распазнавання прынадлежнасці словаў рэгулярным граматыкыкам.

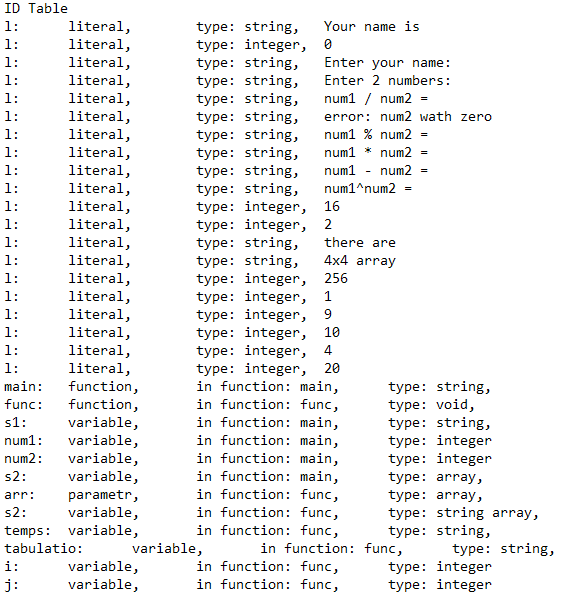
Адбываецца абыход пасімвальна зыходнага тэксту праграммы. Пры сустрэчы сімвалаў-сепаратараў адбываецца апрацоўка атрыманага слова, а менавіта: праверка на ключавыя словы, літэралы і ідэнтыфікатар. Калі ўсе праверкі завяршыліся памылкай, аналізатар змяшчае лексему памылкі, пасля чаго працягвае работу. Пры апрацоўцы слова таксама апрацоўваецца сепаратар, які стаіць пасля яго і ў залежнасці ад сепаратара ён будзе прадстаўлены ў выглядзе лексемы, альбо будзе прапушчаны. Табліца ідэнтыфікатараў ствараецца шляхам абыходу табліцы лексем, з вызначэннем тыпа, значэння і пазіцыі ідэнтыфікатара. кот

**3.10 Кантрольны прыклад**

Табліца лексем ўтрымлівае нумар радка ў зыходным кодзе, пасля чаго лексемы адпавядаючыя зыходнаму коду гэтага радка. Атрыманая на этапе лексічнага аналіза табліца лексем для кантрольнага прыклада паказана на малюнку

|  |  |
| --- | --- |
|  |  |

Малюнак

Табліца ідэнтыфікатараў паказвае назву ідэнтыфікатара, тып ідэнтыфікатара, адпаведны яму тып дадзеных і значэне па змаўчанні, калі яно ёсць. Табліца ідэнтыфікатараў паказана на малюнку 

Табліца функцый адлюстроўвае назву функцыі, вяртаемы тып, колькасць пераменных, аб’яўленых у функцыі і колькасць параметраў функцыі. Табліца функцый паказана на малюнку



# Глава 4. Распрацоўка сінтаксічнага аналізатара

## 4.1 Структура сінтаксічнага аналізатара

## Сінтаксічны аналізатар служыць для аналіза сінтаксічных канструкцый мовы. Ён сочыць за тым, каб ўсе канструкцыі выкарыстоўваліся ў правільным кантэксце. Для апісання мовы праграмавання выкарыстоўваюцца кантэксна-свабодная граматыка, якой і карыстаецца сінтаксічны аналізатар для праверкі зыходнага тэксту. На ўваход сінтаксічны аналізатар прымае табліцу лексем, а таксама параметры сваёй работы, такія як колькасць захоўваемых памылак і падрабязнасць вываду паведамленняў аналізу.

## 4.2 Кантэксна-свабодная граматыка, апісваючая сінтаксіс

У сінтаксічным аналізатары транслятара мовы LLH-2019 выкарыстоўваецца кантэкстна-cвабодная граматыка , дзе T-мноства тэрмінальных сімвалаў якое складаецца з усіх дапусцімых сімвалаў алфавіта мовы, N-мноства нетэрмінальных сімвалаў, P-мноства правілаў мовы, S - пачатковы сімвал граматыкі, які з'яўляецца нетэрміналам.

Гэтая граматыка мае нармальную форму Грэйбах, так яна не мае лямба-пераходаў, недасяганльных сімвалаў і леварэкусріўных правілаў.

Граматыка мовы LLH-2019 прадстаўлена ў дадатку Б.

TS-тэрмінальныя сімвалы, якімі з'яўляюцца сепаратары, знакі арыфметычных аперацый і некаторыя малыя літары.

NS-нетэрмінальныя сімвалы, прадстаўленыя некалькімі загалоўнымі літарамі лацінскага алфавіту. і якія абазначаюць правіла граматыкі.

|  |  |  |
| --- | --- | --- |
| Правіла | Ланцужкі правіла | Апісанне |
| S | m{N;r;};  m{N;r;}S  tf(F){N;rE;};S  tf(F){N;rE;};  tf(){N;rE;};S  tf(){N;rE;}; | Агульная структура праграммы. Правіла вызначае колькі функцый ў зыходным тэксце і ў якім парадку яны размяшчаюцца. |
| N | dti;  dti;N  dti=E;  dti=E;N  i=E;  i=E;N  i();  i();N  i(E);  i(E);N  i(EW);  i(EW);N  k(E){N}  k(E){N}N  k(E){N}C  k(E){N}CN  b;  b;N  w(E){N}  w(E){N}N  i[E]=E;  i[E]=E;N  rE;N;  r;N | Правіла апісвае магчымыя канструкцыі ў целе блока |
| E | i  l  (E)  (E)M  i()  i()M  iM  lM  i(E)  i(E)M  i(EW)  i(EW)M  i[E]  vi[E]  i[E]M  vi[E]M | Магчымыя выразы |
| M | vE  vEM | магчымыя працягі выразу |
| F | ti  ti,F | параметры дыкляруемай функцыі |
| W | ,E  ,EW | параметры выклікаемай функцыі |
| C | ak(E){N}  ak(E){N}C  a{N} | умоўныя канструкцыі, якія могуць ісці пасля асноўнай умовы if |

## 4.3 Пабудова канечнага магазіннага аўтамата

Аўтамат з магазіннай памяццю можа быць апісаны як



Дзе *Q*-мноства становішчаў аўтамата,

*V*-алфавіт уваходных (тэрмінальных) сімвалаў

*Z*-алфавіт магазінных (нетэрмінальных) сімвалаў

-функцыя пераходаў аўтамата

*q0*-пачатковы стан аўтамата

*z0*-пачатковы стан магазіна (маркер дна стэка)

*F-*мноства канечных станаў аўтамата.

Алгарытм работы аўтамата наступны:

-аўтамат чытае першы сімвал з зыходнай стужкі і робіць зрух стужкі, альбо чытае пусты сімвал, пры гэтым стужка застаецца нязменнай.

-у адпаведнасці з цякучым імгненным станам аўтамата, сімвалам стужкі і верхнім сімвалам магазіна шукаецца правіла. Калі правіла знойдзена, то верхні сімвал магазіна замяняецца на ланцужок, адпавядаючы знойдзенаму правілу, і праца аўтамата працягваецца. У выпадку, калі падыходзячае правіла не было знойдзена, аўтамат спыняе працу.

-праца аўтамата працягваецца да памылкі, альбо пакуль у магазіне і ў стужцы не застанецца пусты радок.



Мноства імгненых станаў аўтамата.

## 4.4 Асноўныя структуры дадзеных

Асноўныя структуры дадзеных прыведзеныя у дадатку Б.

## 4.5 Апісанне алгарытма сінтаксічнага разбору

Алгартым працы сінтаксічнага аналізатара наступны: чытаецца сімвал з стэка аўтамата і зыходнай стужкі. Далей магчымы выпадкі:

-У выпадку, калі гэты сімвал з’яўляецца маркерам дна стэка, зыходны тэкст паспяхова разабраны, праца аналізатара закончана.

-У выпадку, калі два сімвала супадаюць стужка і стэк зрухваюцца на адну пазіцую і аналізатар працягвае работу з пачатку.

-У выпадку, калі сімвалы не супадаюць і яны абодва з’яўляюцца тэрміналамі то з’яўляецца адпаведная памылка і аўтамат вяртаецца да мінулага захаванага становішча.

-У выпадку, калі верхні сімвал стэка з’яўляцца нетэрмінальным, то захоўваецца цякучы стан аўтамата і шукаецца адпаведны ланцужок правіла ў граматыцы мовы. Калі ланцужок знойдзены, то аўтамат працягвае працу, калі ланцужок не знойдзены, з’яўляецца паведамленне аб памылцы і аўтамат вяртаецца да мінулага захаванага становішча. Калі мінулага становішча няма, гэта азначае, што зыходны тэкст не правільны, праца аўтамата спыняецца і выводзяцца апошнія памылкі.

Агульная схема працы сінтаксічнага аналізатара прадстаўлена на малюнку 

## 4.6 Структура і пералічэнне паведамленняў сінтакічнага аналізатара

## На этапе сінтаксічнага аналіза могуць з’яўляцца паведамленні аб ходзе разбора ланцужка зыходных лексем, а таксама памылкі пры няўдалым разборы тэкста. Пры з’яленні памылкі, у адпаведнасці з правілам, у якім узнікла памылка, памылка будзе мець адпаведны код.

Пры кожным захаванні становішча аўтамата і пры кожным вяртанні да мінулага становішча сінтаксічны аналізатар, ў залежнасці ад дэтальнасці выводзімымых паведамленняў, будзе выводзіць цякучы стан аўтамата, які ўтрымлівае цякую стужку сімвалаў, цякучы стэк і выбраны ланцужок правіла для разбора.

|  |  |
| --- | --- |
| умова ўзнікнення паведамлення | тэкст паведамлення |
| Не разабралася правіла граматыкі з нетэрмінальнага сімвала S | incorrect structure of programm |
| Не разабралася правіла граматыкі з нетэрмінальнага сімвала N | error in body structure |
| Не разабралася правіла граматыкі з нетэрмінальнага сімвала E | error in expression |
| Не разабралася правіла граматыкі з нетэрмінальнага сімвала E | error in expression |
| Не разабралася правіла граматыкі з нетэрмінальнага сімвала F | error in declared function's parametors |
| Не разабралася правіла граматыкі з нетэрмінальнага сімвала W | error in executed function's parametors |
| Не разабралася правіла граматыкі з нетэрмінальнага сімвала C | error in condition structure |
| Не знайшлося правіла для цякучага становішча аўтамата | No rule |
| Не знайшлося падыходзячага ланцужка дадзенага правіла для цякучага становішча аўтамата | No chain |
| сімвал вяршыні стэка няроўны верхняму сімвалу стужкі | incorrect symbol |

## 4.7 Параметры сінтаксічнага аналізатара і рэжымы яго работы

У якасці параметраў сінтаксічнаму аналізатару перадаецца табліца лексем зыходнага кода, граматыка мовы, максімальная колькасць памылак для вывада, файл для вывада інфармацыі аб ходзе аналіза і дакладнасць выводзімай інфармацыі. Пачынаючы работу, аналізатар абыходзіць зыходную табліцу лексем, правяраючы яе па зыходным правілам граматыкі і ў залежнасці ад зыходнай ступені дакладнасці паведамленняў запісвае ў зыходны файл інфармацыю. Пры вялікай дакладнасці выводзімых паведамленняў аналізатар будзе паведамляць аб кожным выбраным ланцужку правіла грамытыкі, аб захаванні становішча аўтамата і аб вяртанняў да мінулых становішчаў, пасля чаго аналізатар выводзіць дрэва разбора, пры ўдалым разборы тэксту, альбо памылкі, у вападку няўдалага разбору. Калі ступень дакладнасці мінімальная, то выводзіцца толькі дрэва разбора альбо спіс памылак у адпаведнасці з вынікам работы.

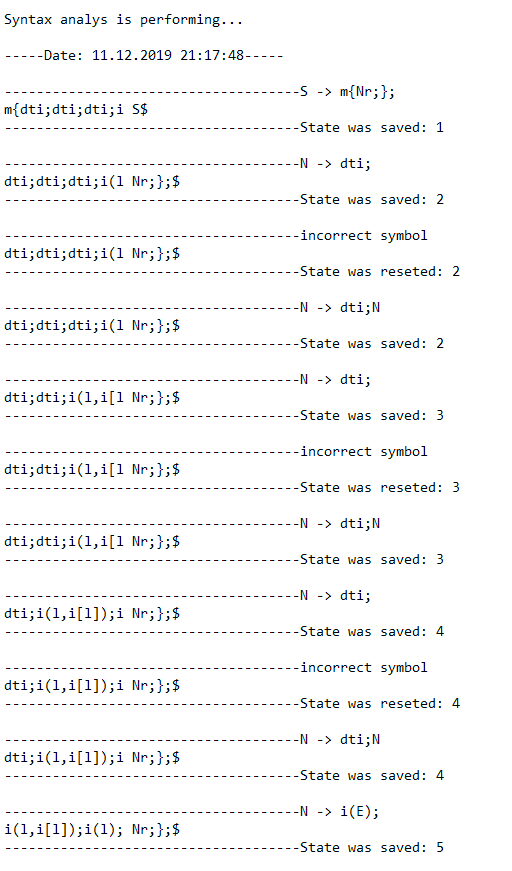
## 4.8 Прынцып апрацоўкі памылак

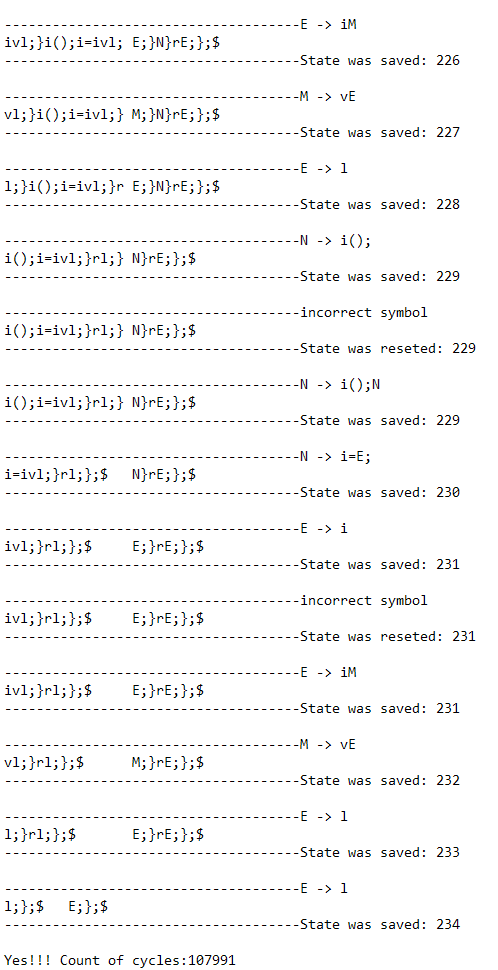
## У ходзе разбору зыходнага тэкста, сінтаксічны аналізатар захоўвае становішчы аўтамата, якія найбольш блізкія да завяршэння аналіза. Колькасць гэтых становішчаў абмежаваная і гэтае абмежаванне можа ўказвацца ў зыходных параметрах. Пры няўдалым разбору тэкста аналізатар выводзіць апошнія найбольш паспяховыя становішчы аўтамата і адпавядаючыя ім паведамленні. Парадак

## 4.9 Кантрольны прыклад

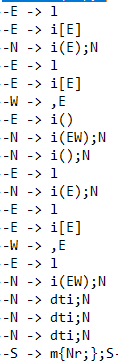
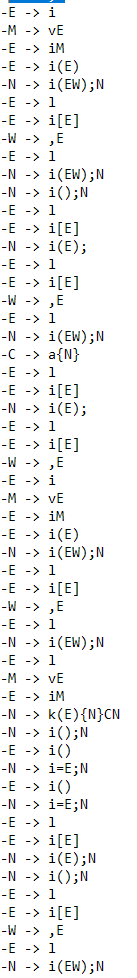
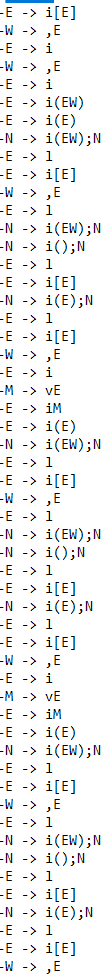
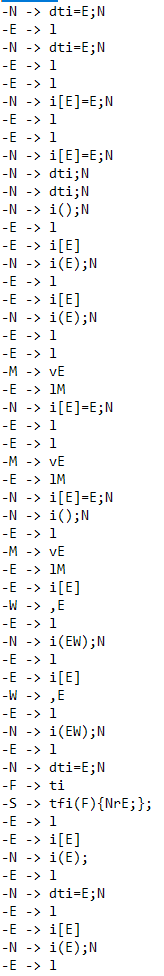
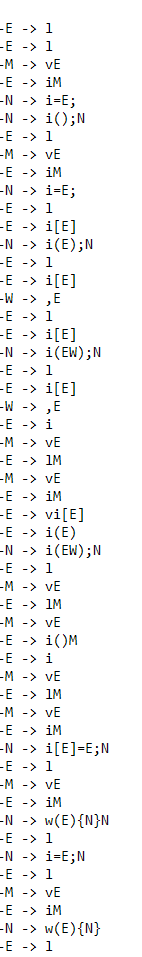
Пратакол сінтаксічнага аналіза паказвае імгненныя становішчы аўтамата, а менавіта: цякучая стужка з лексем, цякучы стэк аўтамата, правіла, якое будзе ўжывацца альбо памылка на цякучым стане аўтамата і нумар становішча аўтамата якое захоўваецца, альбо да якога аўтамат вяртаецца.

Пачатак пратакола сінтаксічнага аналіза паказаны на малюнку



Завяршэнне пратакола сінтаксічнага аналізатара паказана на малюнку 

Выніковае дрэва разбору паказана на малюнку

Малюнак

# Глава 5. Распрацоўка семантычнага аналізатара

## 5.1 Структура семантычнага аналізатара

Асноўная задача семантычнага аналіза – гэта праверка семантычных правіл. На гэтым этапе могуць ажыццяўляцца праверкі на правільнае выкарыстанне асобных канструкцый, якія не былі ўключаны ў папярэднія фазы аналізу. Семантычны аналіз можа быць арганізаваны як асобны этап, альбо часткова прысутнічаць у іншых фазах аналізу. Схема працы семантычнага аналізатара прыведзена на малюнку



## 5.2 Функыі семантычнага аналізатара

|  |  |  |
| --- | --- | --- |
| Фаза | Правіла | Алгарытм праверкі пры абыходзе табліцы лексем |
| лексічны аналіз | Ключавое слова break можа выкарыстоўвацца толькі ўнутры цела цыкла | Пры абыходзе табліцы лексем кожны праграмны блок запісваецца ў стэк, і пры сустрэчы break ў стэку павінен мецца блок цыкла |
| лексічны аналіз | Прывышэнне памера лікавых, сімвальных і радковых літэралаў | Пры сустрэчы літэрала, яго значэнне параўноўваецца з максімальна дапушчальнымі |
| лексічны аналіз | Паўторнае аб’яўленне ідэнтыфікатараў | Пры знаходжанні новага ідэнтыфікатара правяраецца табліца ідэнтыфікатараў на паўтарэнне дэклярацыі |
| лексічны аналіз | Выкарыстанне ідэнтыфікатараў без аб’яўлення | Пры знаходжанні новага ідэнтыфікатара Правяраецца табліца ідэнтыфікатараў наяўнасць адпаведнага ідэнтыфікатара |
| лексічны аналіз | Пасля дэклярацыі масіва павінны быць указаны памер масіва | Пасля дэклярацыі масіва павінен знаходзіцца лікавы літэра |
| семантычны аналіз | Забаронена выкарыстанне масіва без указання індэкса | Пры знаходжанні ідэнтыфікатара масіва пасля яго павінна знаходзіцца адкрываючая квадратная дужка |
| лексічны аналіз | Забаронена ініцыялізацыя радка | Пасля дэклярацыі радка не павінна знаходзіцца лексемы прысвойвання |
| семантычны аналіз | Пасля назвы функцыі павінны адкрывацца дужкі, для ўказання параметраў | Пры знаходжанні ідэнтыфікатара функцыі, пасля яго павінна знаходзіцца адкрываючая дужка |
| лексічны аналіз | Забаронена выкарыстоўваць літэрал пустога радка | Кожны радковы літэрал параўноўваецца з пустым радком |
| лексічны аналіз | Забаронена выкарыстоўваць глабальныя пераменныя і літэралы | Пры знаходжанні ідэнтыфікатара правяраецца наяўнасць незакрытага цела функцыі |
| лексічны аналіз | Толькі функцыі могуць мець тып дадзеных void | Пры заданні тыпаў ідэнтыфікатарам правяраецца тып |

## 5.3 Структура і пералічэнне паведамленняў семантычнага аналізатара

Спіс паведамленняў семантычных праверак прадстаўлены ў табліцы

|  |  |
| --- | --- |
| код памылкі | тэкст памылкі |
| 202 | incorrect id's declaration syntax |
| 203 | id must be declarated, or wrong declaration was appeard |
| 208 | wrong type of literal |
| 209 | double declaration |
| 213 | invalid array using |
| 214 | empty string |
| 216 | global identificators and literals are forbidden |
| 219 | literal size reached the limit |
| 220 | variable or parametr can't have type void |

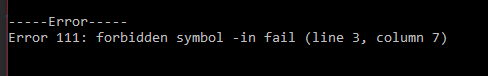
## 5.4 Прынцып апрацоўкі памылак

Пры выяўленні памылак яны заносяцца ў спіс, да тых пор, пакуль колькасць памылак ў спісе не дасягне зададзенай уваходным параметрам велічыні. Разам з тэкстам памылкі і яе нумарам выводзіцца пазіцыя лексемы, ў якой памылка ўзнікла.

## 5.5 Катрольны прыклад

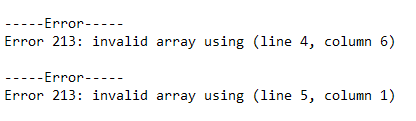
Памылка выкарыстання зменнай разам з кодам, які выклікаў памылку да яе абўялення паказаны на малюнку





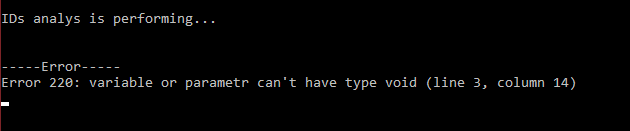
Памылка неправільнага выкарыстання масіваў, з-за не ўказання індэкса, разам з кодам, які прывёў да памылкі прыведзены на малюнку





Памылка выкарыстання зменнай з тыпам дадзеных void разам з кодам, які прывёў да памылкі паказаны на малюнку





# Глава 6. Пераўтварэнне выразаў

## 6.1 Выразы дапускаемыя мовай праграмавання

У мове праграмавання LLH-2019 аперандамі выразаў могуць быць любыя тыпы дадзеных, акрамя ўказальнікаў на масівы. Усе тыпы дадзеных пераводзяцца ў лікавы фармат, перад выкананнем аперацыі, пасля чаго атрыманы вынік пераводзіцца ў патрэбны тып дадзеных. Зпяршая выконваюцца аперацыі ў дужках, пасля чаго выконваюцца самыя высокія па прыярытэту аперацыі. Аперацыі выконваюцца злева на права, у выпадку калі прыярытэт аперацый аднолькавы. У якасці аперандаў дапускаецца выкарыстанне значэнняў функый. Тыповыя выразы з кантрольнага прыклада прыведзены на малюнку

## 6.2 Польскі запіс і прынцып яго пабудовы

Польскі запіс – гэта спосаб запісу арыфметыных выразаў, які зручна выкарыстоўваць з-за адсутнасці дужак. У такой форме запісу выразы значна лягчэй перакладаць на мову асэблер. У гэтай форме запісу аперацыя стаіць не паміж яе аперандамі, як ў звычайнай інфікснай форме запісу, а пасля іх. У мове праграмавання LLH-2019 акрамя арыфметычных аперацый у польскі запіс пераводзяцца таксама таксама выклік функцый і звяртанне да масіва.

Алгарытм пераўтварэння выразу ў польскі запіс наступны:

-зыходны радок паслядоўна правяраецца злева направа

-пры сустрэчы аперанда ён запісваецца ў выходны радок

-пры сустрэчы аперацыі са стэка ў выходны радок прыбіраюцца ўсе аперацыі пачынаючы з канца стэка з большым альбо роўным прыярытэтам да тых пор, пакуль у вяршыні стэка не апынецца коска, адкрываючая дужка, адкрываючая квадратная дужка альбо пакуль стэк не будзе пустым

-пры сустрэчы выкліка функцыі альбо масіва яны апрацоўваюцца як аперацыі толькі з рознымі колькасцямі аперандаў

-калі зыходны радок зкончыўся аперацыі са стэка паслядоўна прыбіраюцца ў выходны радок

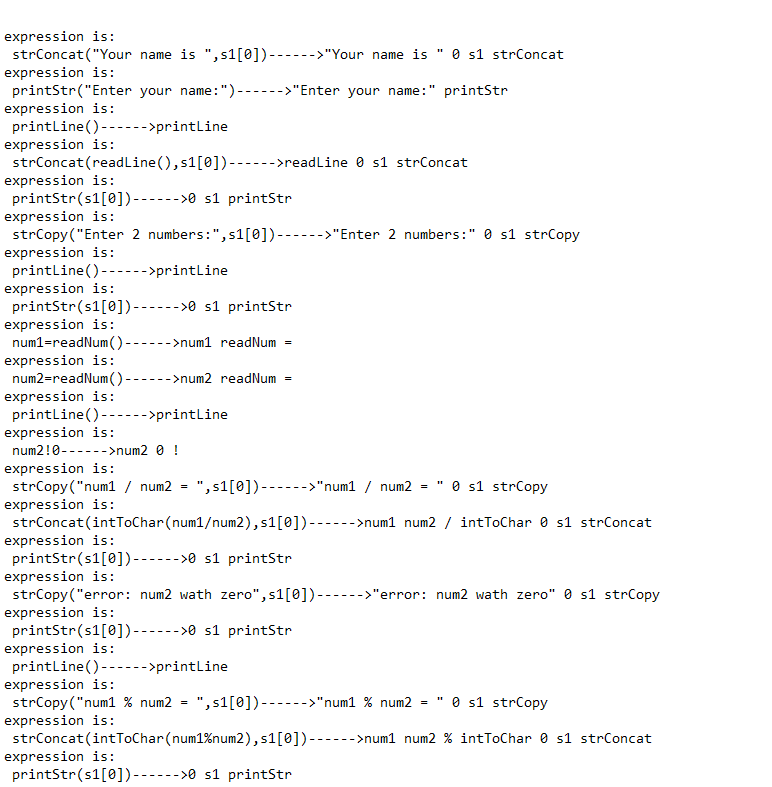
Прыклад разбору аднаго з выразаў кантрольнага прыклада паказаны на малюнку 

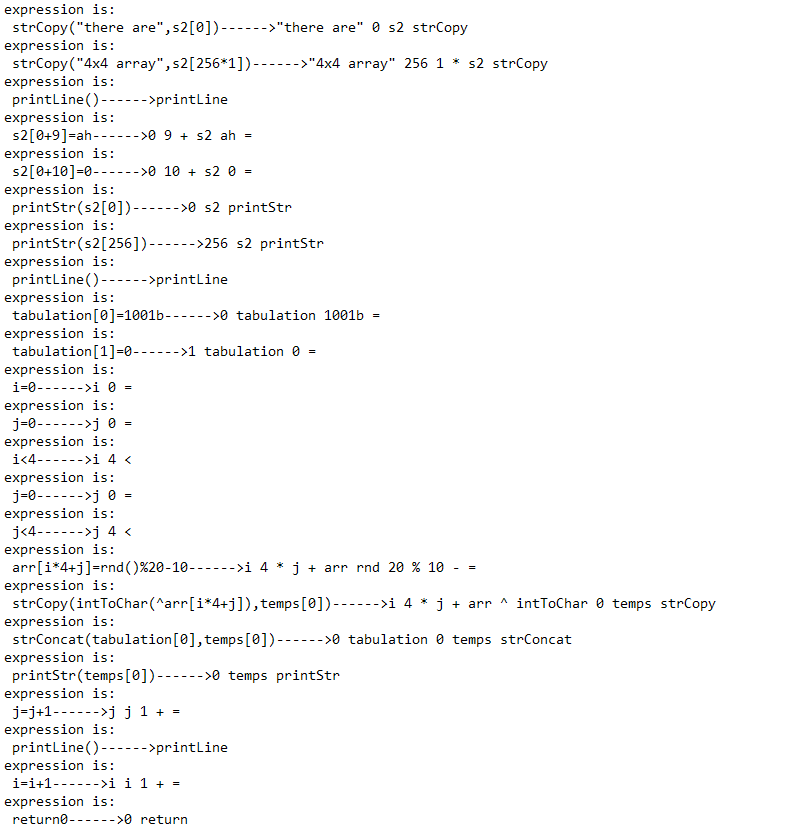
Злева размяшчаецца запіс у інфікснай форму, зправа – у постфікснай. У польскім запісе ўсе арыфметныя аперацыія бяруць два апошніх значэння ў якасці сваіх аперандаў, аперацыя атрымання значэння па адрасе “^” а таксама атрамання элемента масіва бяруць ў якасці аперанда толькі аошняе значэнне, гэтыя аперацыі з’яўляюцца ўнарнымі. Выклік функцыі бярэ столькі значэнняў у якасці аперандаў, колькі функцыя мае параметраў.

## 6.3 Праграмная рэалізацыя апрацоўкі выразаў

Рэалізацыя пераўтварэння выраза ў польскі запіс паказана ў дадатку В. Каб атрымліваць інфармацыю аб пераводзе ў польскі запіс кожнага выразу ўваходны параметр bool additionalInfo задаецца як true. Гэты параметр можа быць зададзены карыстальнікам ў якасці параметра транслятара, для гэта трэба ўказаць ключ /pol:

## 6.4 Кантрольны прыклад

Пачатак паведамленняў пратакола, дзе паказваецца пераўтварэнне выразаў у польскую натацыу паказаны на малюнку 

Канец паведамленняў пратакола, дзе паказваецца пераўтварэнне выразаў у польскую натацыу паказаны на малюнку 

# Глава 7. Генерацыя кода

## 7.1 Структура генератара кода

Генерацыя кода – апошня фаза працы транслятара. На гэтай фазе адбываецца ўтварэнне выніковага кода на патрэбнай мове. У транслятары LLH-2019 выходным з’яўляецца код на мове асэмблера. Схема работы генератара кода паказана на малюнку



## 7.2 Прадстаўленне тыпаў дадзеных у аператыўнай памяці

Усе пераменныя, аб’яўленыя ў зыходным кодзе, выкарыстоўваюцца як лакальныя пераменныя функцыі, у якой яны аб’яўлены. Такім чынам дадзеныя пераменных заносяцца ў аператыўную памяць толькі пад час выканання функцыі, у якой яны аб’яўлены. Калі функцыя пачынае працу, яна аўтаматычна выдзяляе ў стэку патрэбную для ўсіх пераменных і параметраў памяць, якую пасля завяршэння функцыі ачышчае, вяртаючы ўказальнік стэка ў мінулую пазіцую.Усе літэралы зыходнага кода хаваюцца ў секцыі канстант, для таго, каб аднолькавыя па значэнні літэралы не аб’яўляліся некалькі разоў для функцый, дзе яны вызначаны. Адпаведнасць тыпаў дадзеных мовы LLH-2019 да тыпаў дадзеных мовы асэмблера паказана ў табліцы

|  |  |  |
| --- | --- | --- |
| LLH-2019 | Assembler | Займаемы памер |
| integer | word | 2 байта |
| char | byte | 1 байт |
| string | ptr byte | 4 байта на ўказальнік + 256 байтаў на масіў |
| sarray | ptr byte | 4 байта на ўказальнік + 256\*<памер масіва> байтаў на масіва |
| array | ptr word | 4 байта на ўказальнік + 2\*<памер масіва> байтаў на масіва |

## 7.3 Статычная бібліятэка

Склад функцый статычнай бібліятэкі прыведзены ў табліцы. Функцыі статычнай бібліятэкі рэалізаваны на мове C++ і загадзя зкампіляваны ў файл з пашырэннем .lib. Статычная бібліятэка падключаецца да праграммы на этапе кампаноўкі выходнага кода транслятара на мове assembler цераз дырэктыву includelib.

## 7.4 Асаблівасці алгарытма генерацыі кода

Агульная схема алгарытма генератара кода паказана на малюнку 

Генератар кода абыходзіць лексемы, захоўваючы інфармацыю аб блоках цыклу і ўмоў. Для кожнай лексемы запісваецца ў радок фрагмент кода на мове assembler у адпаведнасці з лексемай, папярэднімі ёй лексемамі і блокамі, ў якіх лексема знаходзіцца.

## 7.5 Уваходныя параметры генератара кода

У якасці ўваходных параметраў генератар кода прымае табліцу лексем, функцый, ідэнтыфікатараў і файл, ў які будзе запісана інфармацыя аб ходзе генерацыі.

## 7.6 Кантрольны прыклад

Пачатак і канец кода асэмблера ў выніку генерацыі кантрольнага прыклада паказаны на малюнку

|  |  |
| --- | --- |
|  |  |

# Глава 8. Тэставанне транслятара

## 8.1 Агульныя палажэнні

## 8.2 Вынікі тэставання

|  |  |  |
| --- | --- | --- |
| Код, які выклікаў памылку | Паведамленне памылкі | Этап трансляцыі |
|  | -----Error-----  Error 203: id must be declarated, or wrong declaration was appeard (line 4, column 9) | стварэнне табліцы ідэнтыфікатараў |
|  | -----Error-----  Error 202: incorrect id's declaration syntax (line 3, column 8) | стварэнне табліцы ідэнтыфікатараў |
|  | -----Error-----  Error 204: lexic error (line 3, column 8) | стварэнне табліцы лексем |
|  | -----Error-----  Error 209: double declaration (line 4, column 16) | стварэнне табліцы ідэнтыфікатараў |
|  | -----Error-----  Error 211: using of addres, instead of a value (line 5, column 6)  -----Error-----  Error 211: using of addres, instead of a value (line 6, column 7) | семантычны аналіз |
|  | -----Error-----  Error 212: there must be opening bracket (line 4, column 7) | семантычны аналіз |
|  | -----Error-----  Error 213: invalid array using (line 4, column 1) | семантычны аналіз |
|  | -----Error-----  Error 213: invalid array using (line 3, column 16)  -----Error-----  Error 213: invalid array using (line 4, column 15) | стварэнне табліцы ідэнтыфікатараў |
|  | -----Error-----  Error 214: empty string (line 4, column 9) | стварэнне табліцы ідэнтыфікатараў |
|  | -----Error-----  Error 219: literal size reached the limit (line 3, column 23) | стварэнне табліцы ідэнтыфікатараў |
|  | -----Error-----  Error 220: variable or parametr can't have type void (line 3, column 14) | стварэнне табліцы ідэнтыфікатараў |
|  | -----Error-----  Error 221: unexpected break (line 2, column 2) | Генерацыя кода |

# Заключение

# Дадатак А

Дапушчальныя ўваходны сімвалы выкананыя ў выглядзе масіва, дзе T – абазначае дапусцімы сімвал, F – недапусцімы сімвал.

const int TABLE[] = {

/\* (0)\*/IN::F, /\*\_ (1)\*/IN::F, /\*\_ (2)\*/IN::F, /\*\_ (3)\*/IN::F, /\*\_ (4)\*/IN::F, /\*\_ (5)\*/IN::F, /\*\_ (6)\*/IN::F, /\* (7)\*/IN::F, /\* (8)\*/ IN::F, /\* (9)\*/IN::F,IN::F,IN::F,IN::F, /\*(13)\*/IN::F, /\*\_ (14)\*/IN::F, /\*\_ (15)\*/IN::F, /\*

/\*\_ (16)\*/IN::F, /\*\_ (17)\*/IN::F, /\*\_ (18)\*/IN::F, /\*\_ (19)\*/IN::F, /\*\_ (20)\*/IN::F, /\*\_ (21)\*/IN::F, /\*\_ (22)\*/IN::F, /\*\_ (23)\*/IN::F, /\*\_ (24)\*/IN::F, /\*\_ (25)\*/IN::F, /\*27\*/ IN::F, /\*\_ (27)\*/IN::F, /\*\_ (28)\*/IN::F, /\*\_ (29)\*/IN::F, /\*\_ (30)\*/IN::F,

/\*\_ (31)\*/IN::F, /\* (32)\*/IN::F, /\*! (33)\*/IN::T, /\*" (34)\*/IN::T, /\*# (35)\*/IN::F, /\*$ (36)\*/IN::F, /\*% (37)\*/IN::T, /\*& (38)\*/IN::T, /\*' (39)\*/IN::F, /\*( (40)\*/IN::T, /\*) (41)\*/IN::T, /\*\* (42)\*/IN::T, /\*+ (43)\*/IN::T, /\*, (44)\*/IN::T, /\*- (45)\*/IN::T,

/\*. (46)\*/IN::F, /\*/ (47)\*/IN::T, /\*0 (48)\*/IN::T, /\*1 (49)\*/IN::T, /\*2 (50)\*/IN::T, /\*3 (51)\*/IN::T, /\*4 (52)\*/IN::T, /\*5 (53)\*/IN::T, /\*6 (54)\*/IN::T, /\*7 (55)\*/IN::T, /\*8 (56)\*/IN::T, /\*9 (57)\*/IN::T, /\*: (58)\*/IN::T, /\*; (59)\*/IN::T, /\*< (60)\*/IN::T,

/\*= (61)\*/IN::T, /\*> (62)\*/IN::T, /\*? (63)\*/IN::T, /\*@ (64)\*/IN::F, /\*A (65)\*/IN::T, /\*B (66)\*/IN::T, /\*C (67)\*/IN::T, /\*D (68)\*/IN::T, /\*E (69)\*/IN::T, /\*T (70)\*/IN::T, /\*G (71)\*/IN::T, /\*H (72)\*/IN::T, /\*I (73)\*/IN::T, /\*J (74)\*/IN::T, /\*K (75)\*/IN::T,

/\*L (76)\*/IN::T, /\*M (77)\*/IN::T, /\*N (78)\*/IN::T, /\*O (79)\*/IN::T, /\*P (80)\*/IN::T, /\*Q (81)\*/IN::T, /\*R (82)\*/IN::T, /\*S (83)\*/IN::T, /\*T (84)\*/IN::T, /\*U (85)\*/IN::T, /\*V (86)\*/IN::T, /\*W (87)\*/IN::T, /\*X (88)\*/IN::T, /\*Y (89)\*/IN::T, /\*Z (90)\*/IN::T,

/\*[ (91)\*/IN::T, /\*\ (92)\*/IN::T, /\*] (93)\*/IN::T, /\*^ (94)\*/IN::T, /\*\_ (95)\*/IN::F, /\*` (96)\*/IN::T, /\*a (97)\*/IN::T, /\*b (98)\*/IN::T, /\*c (99)\*/IN::T, /\*d (100)\*/IN::T, /\*e (101)\*/IN::T, /\*f (102)\*/IN::T, /\*g (103)\*/IN::T, /\*h (104)\*/IN::T, /\*i (105)\*/IN::T,

/\*j (106)\*/IN::T, /\*k (107)\*/IN::T, /\*l (108)\*/IN::T, /\*m (109)\*/IN::T, /\*n (110)\*/IN::T, /\*o (111)\*/IN::T, /\*p (112)\*/IN::T, /\*q (113)\*/IN::T, /\*r (114)\*/IN::T, /\*s (115)\*/IN::T, /\*t (116)\*/IN::T, /\*u (117)\*/IN::T, /\*v (118)\*/IN::T, /\*w (119)\*/IN::T, /\*x (120)\*/IN::T,

/\*y (121)\*/IN::T, /\*z (122)\*/IN::T, /\*{ (123)\*/IN::T, /\*| (124)\*/IN::T, /\*} (125)\*/IN::T, /\*~ (126)\*/IN::F, /\* (127)\*/IN::F, /\*? (128)\*/IN::F, /\*? (129)\*/IN::F, /\*' (130)\*/IN::F, /\*? (131)\*/IN::F, /\*" (132)\*/IN::F, /\*: (133)\*/IN::F, /\*┼ (134)\*/IN::F, /\*╪ (135)\*/IN::F,

/\*? (136)\*/IN::F, /\*% (137)\*/IN::T, /\*? (138)\*/IN::F, /\*< (139)\*/IN::T, /\*? (140)\*/IN::F, /\*? (141)\*/IN::F, /\*? (142)\*/IN::F, /\*? (143)\*/IN::F, /\*? (144)\*/IN::F, /\*' (145)\*/IN::F, /\*' (146)\*/IN::F, /\*" (147)\*/IN::F, /\*" (148)\*/IN::F, /\* (149)\*/IN::F, /\*- (150)\*/IN::F,

/\*- (151)\*/IN::F, /\*? (152)\*/IN::F, /\*F (153)\*/IN::F, /\*? (154)\*/IN::F, /\*> (155)\*/IN::F, /\*? (156)\*/IN::F, /\*? (157)\*/IN::F, /\*? (158)\*/IN::F, /\*? (159)\*/IN::F, /\*  (160)\*/IN::F, /\*Ў (161)\*/IN::F, /\*ў (162)\*/IN::F, /\*? (163)\*/IN::F, /\*¤ (164)\*/IN::F, /\*? (165)\*/IN::F,

/\*│ (166)\*/IN::F, /\*\_ (167)\*/IN::F, /\*Ё (168)\*/IN::F, /\*c (169)\*/IN::F, /\*Є (170)\*/IN::F, /\*< (171)\*/IN::F, /\*┐ (172)\*/IN::F, /\*- (173)\*/IN::F, /\*R (174)\*/IN::F, /\*Ї (175)\*/IN::F, /\*° (176)\*/IN::F, /\*+ (177)\*/IN::F, /\*? (178)\*/IN::F, /\*? (179)\*/IN::F, /\*? (180)\*/IN::F,

/\*ч (181)\*/IN::F, /\*\_ (182)\*/IN::F, /\*· (183)\*/IN::F, /\*ё (184)\*/IN::F, /\*№ (185)\*/IN::F, /\*є (186)\*/IN::F, /\*> (187)\*/IN::F, /\*? (188)\*/IN::F, /\*? (189)\*/IN::F, /\*? (190)\*/IN::F, /\*ї (191)\*/IN::F, /\*А (192)\*/IN::F, /\*Б (193)\*/IN::F, /\*В (194)\*/IN::F, /\*Г (195)\*/IN::F,

/\*Д (196)\*/IN::F, /\*Е (197)\*/IN::F, /\*Ж (198)\*/IN::F, /\*З (199)\*/IN::F, /\*И (200)\*/IN::F, /\*Й (201)\*/IN::F, /\*К (202)\*/IN::F, /\*Л (203)\*/IN::F, /\*М (204)\*/IN::F, /\*Н (205)\*/IN::F, /\*О (206)\*/IN::F, /\*П (207)\*/IN::F, /\*Р (208)\*/IN::F, /\*С (209)\*/IN::F, /\*Т (210)\*/IN::F,

/\*У (211)\*/IN::F, /\*Ф (212)\*/IN::F, /\*Х (213)\*/IN::F, /\*Ц (214)\*/IN::F, /\*Ч (215)\*/IN::F, /\*Ш (216)\*/IN::F, /\*Щ (217)\*/IN::F, /\*Ъ (218)\*/IN::F, /\*Ы (219)\*/IN::F, /\*Ь (220)\*/IN::F, /\*Э (221)\*/IN::F, /\*Ю (222)\*/IN::F, /\*Я (223)\*/IN::F, /\*а (224)\*/IN::F, /\*б (225)\*/IN::F,

/\*в (226)\*/IN::F, /\*г (227)\*/IN::F, /\*д (228)\*/IN::F, /\*е (229)\*/IN::F, /\*ж (230)\*/IN::F, /\*з (231)\*/IN::F, /\*и (232)\*/IN::F, /\*й (233)\*/IN::F, /\*к (234)\*/IN::F, /\*л (235)\*/IN::F, /\*м (236)\*/IN::F, /\*н (237)\*/IN::F, /\*о (238)\*/IN::F, /\*п (239)\*/IN::F, /\*р (240)\*/IN::F,

/\*с (241)\*/IN::F, /\*т (242)\*/IN::F, /\*у (243)\*/IN::F, /\*ф (244)\*/IN::I, /\*х (245)\*/IN::F, /\*ц (246)\*/IN::F, /\*ч (247)\*/IN::F, /\*ш (248)\*/IN::F, /\*щ (249)\*/IN::F, /\*ъ (250)\*/IN::F, /\*ы (251)\*/IN::F, /\*ь (252)\*/IN::F, /\*э (253)\*/IN::F, /\*ю (254)\*/IN::F, /\*я (255)\*/IN::F,

};

Структура для элемента табліцы лексем:

struct Entry //запіс у табліцы

{

COUNTSYSTEM countSys = DEC; //сістэма злічэння

char lexema[LEXEMA\_FIXSIZE]; //лексема

int sn; //нумар радка у зыходным кодзе

int idxTI; //індэкс у табліцы ід

int line; //радок у зыходным тэксце

int col; //нумар калонкі ў зыходным тэксце

bool isAssignedId = false; //ці стаіць лексема злева ад "="

};

Струкура для табліцы лексем:

struct LexTable //табліца

{

int maxsize; //максімальны памер табліцы

int size; //цякучы памер табліцы

Entry\* table; //масіў з лексем

};

Структура для вызначэння функцыі

enum IDDATATYPE{INT=1,

STR=2, //радок

VOID=3, //функцыя не вяртае значэння

ARRAY, //масіў лікаў

ARRAY\_STR, //масіў радкоў

CHAR //сімвал

}; //тып дадзеных

enum IDTYPE {V=1,F=2,P=3,L=4}; //пераменная, функцыя, параметр, літэрал

struct FuncDefenition {

IDDATATYPE returnType;

int params[PARMS\_MAX\_COUNT]; //індэксы ў табліцы ідэнтыфікатараў параметраў

int locals[TI\_MAXSIZE]; //індэксы ў табліцы ідэнтыфікатараў лакальных зменных

std::string name; //імя функцыі

short curLocals=0; //індэкс апошняй пераменнай

short curParams=0; //індэкс апошняга параметра

short startOfFunc = -1; //лексема, з якой пачынаецца апісанне функцыі

}; //структура для захоўвання функцый

Структура для элемента табліцы ідэнтыфікатараў

struct Entry {

int idxfirstLE; //індэкс у табліцы лексем

char id[ID\_MAXSIZE]; //значэнне

LT::COUNTSYSTEM countSystem = LT::DEC; //сістэма злічэння

IDDATATYPE iddatatype; //тып дадзеных

IDTYPE idtype; //тып ідэнтыфікатара

FuncDefenition \*funcId;//функцыя, у якой вызначаны ід

std::string AssemblerName; //імя якое просвоена ідэнтыфікатару ў асэмблеры

union

{

int vint; //лікавае значэнне

char cint; //сімвальнае значэнне

struct

{

char len; //даўжыня радка

char str[TI\_STR\_MAXSIZE-1]; //значэнне радка

} vstr;

} value; //значэнне ідэнтыфікатара

};

Структура для табліцы ідэнтыфікатараў

struct IdTable //табліца

{

int maxsize; //максімальны памер табліцы

int size; //цякучы памер табліцы

Entry\* table; //указальнік на масіў элеметнаў

int countOfStandartFuncs = 9; //колькасць стандартных функцый

FuncDefenition \*\*funcs; //масіў функцый

int funcCount=0; //колькасць карыстальніцкіх функцый

};

# Дадатак Б

Структуры для правіл граматыкі.

struct Rule { //правіла у граматыцы

GRBHALPHABET nn; //нэтэрмінал(левы сімвал)

int idError;

short size; //колькасць правых частваў правіла

Chain\* chains;

Rule(GRBHALPHABET pnn, int idError, short psize, Chain c, ...);

char \*getCRule(short nchain); //атрымаць правіла ў выглядзе радка

short getNextChain(

GRBHALPHABET t, Rule::Chain& pchain, short j

); //атрымаць наступны за j ланцуг або -1

};

struct Chain { //правая частва правіла

short size; //даўжыня

GRBHALPHABET\* nt; //ланцуг тэрміналаў і нетэрміналаў

Chain() {size=0;nt=0;};

Chain(short psize, GRBHALPHABET s, ...);

char\* getCChain(); //атрымаць правую чатску правіла

static GRBHALPHABET T(char t){return GRBHALPHABET(t);}; //трэмінал

static GRBHALPHABET N(char n) { return -GRBHALPHABET(n);}; //не тэрмінал

static bool isT(GRBHALPHABET s) { return s > 0; };

static bool isN(GRBHALPHABET s) { return !isT(s); };

static char alphabet\_to\_char(GRBHALPHABET s)

{

return isT(s)?char(s):char(-s);

}

}\*

struct Greibach {

short size; //колькасць правіл

GRBHALPHABET startN; //стартавы сімвал

GRBHALPHABET stbottomT; //дно стэка

Rule\* rules;//правілы

Greibach() {

short size = 0;

startN = 0;

stbottomT = 0;

rules = 0;

};

Greibach(

GRBHALPHABET pstartN,

GRBHALPHABET pstbottomT,

short psize,

Rule r, ...

);

short getRule(

GRBHALPHABET pnn,

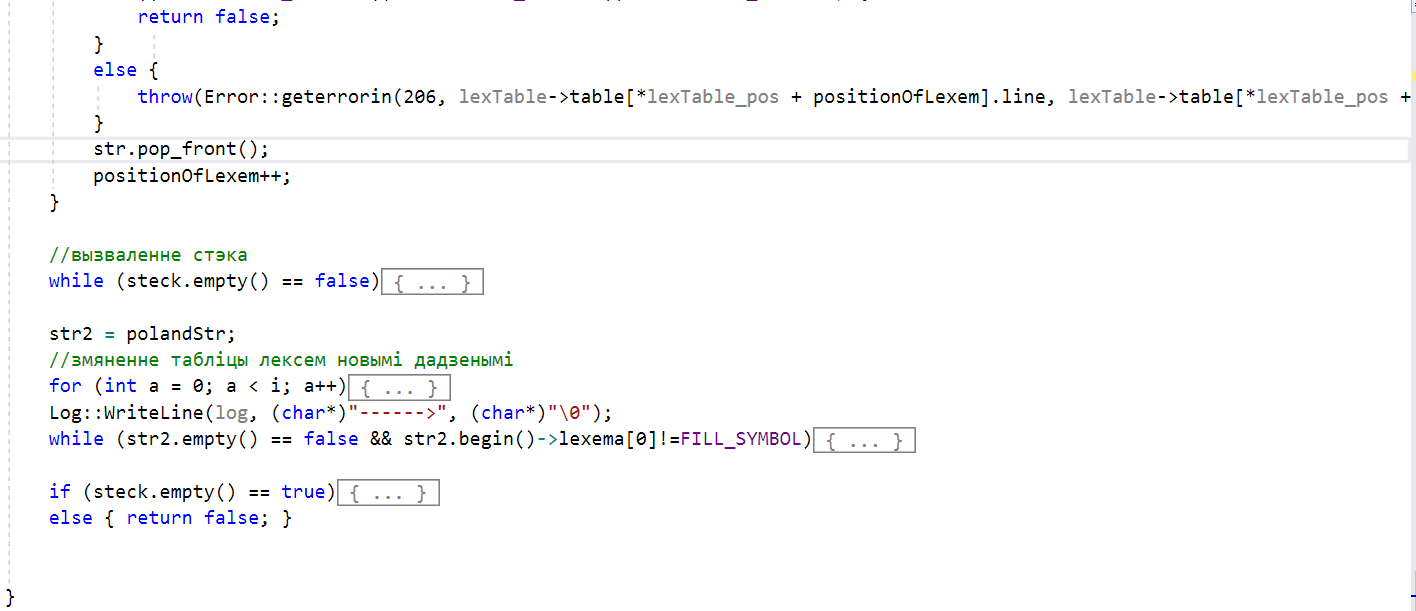
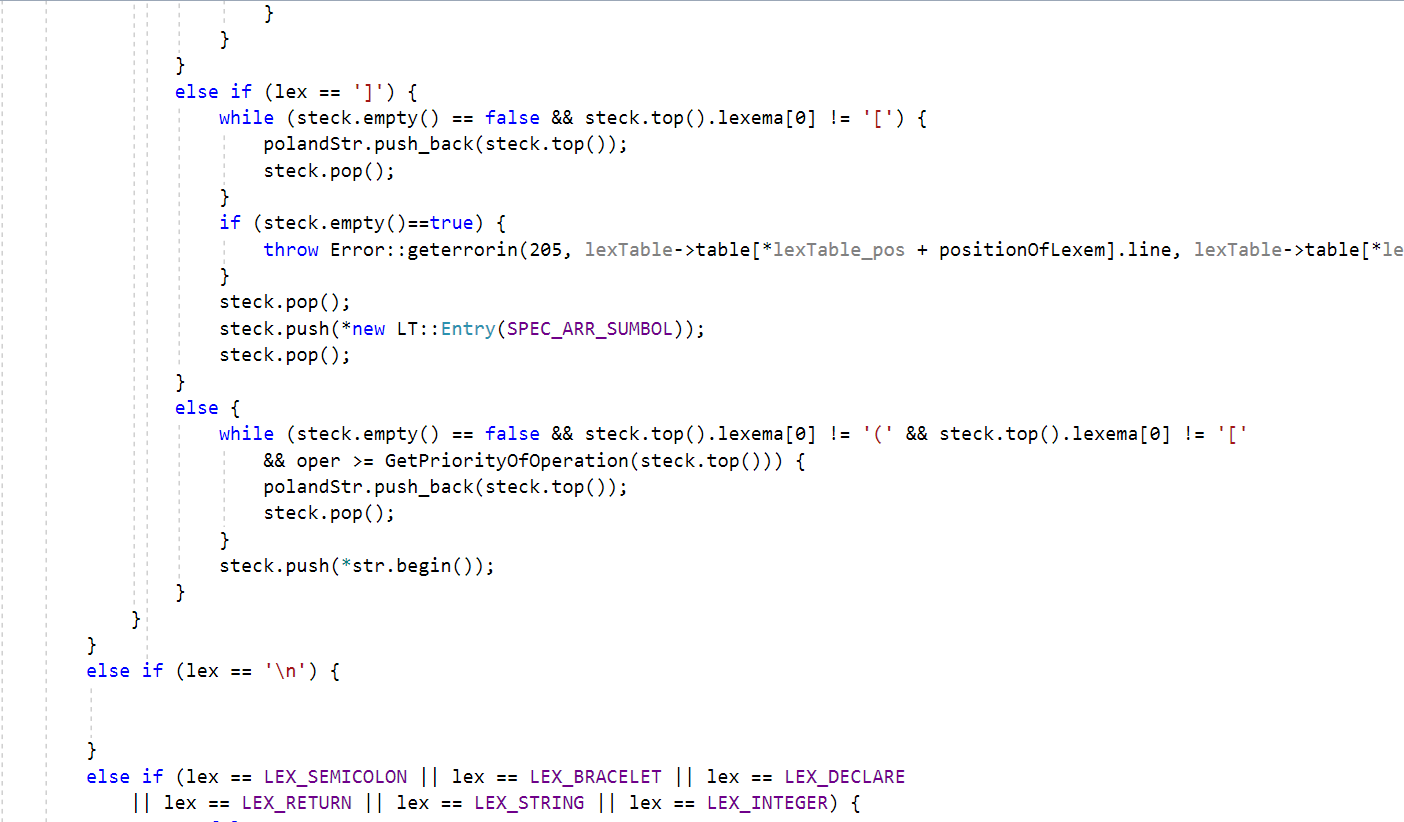
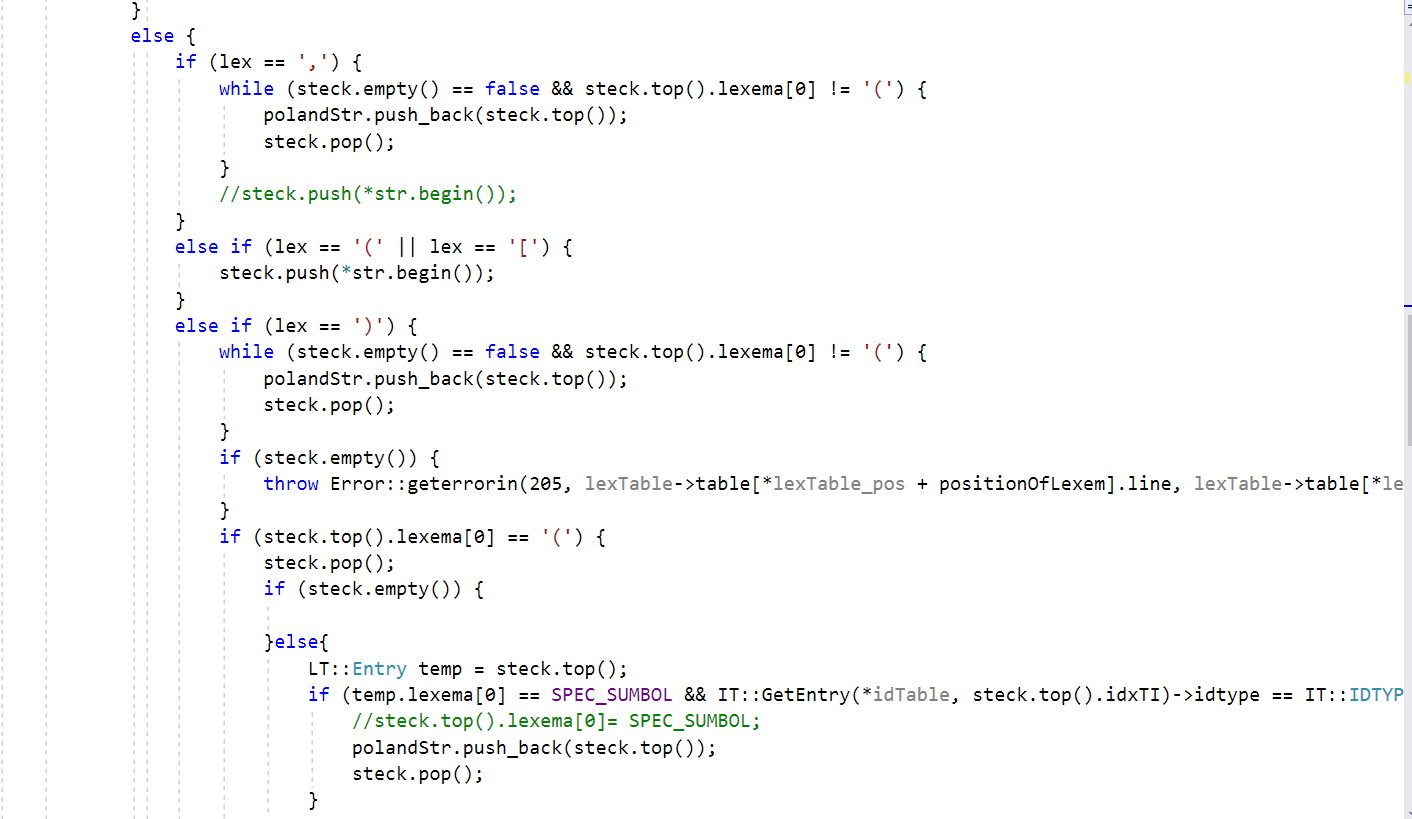
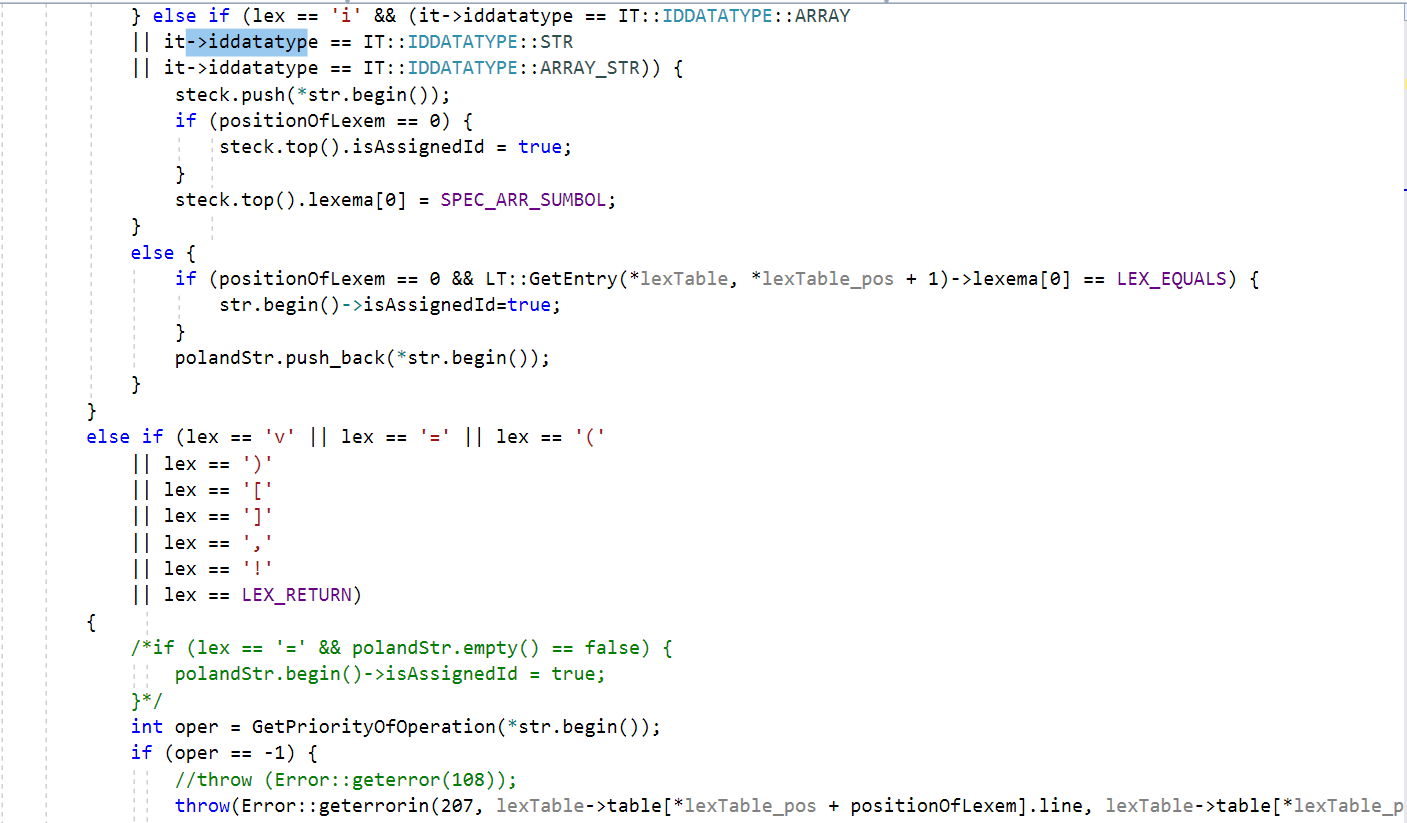
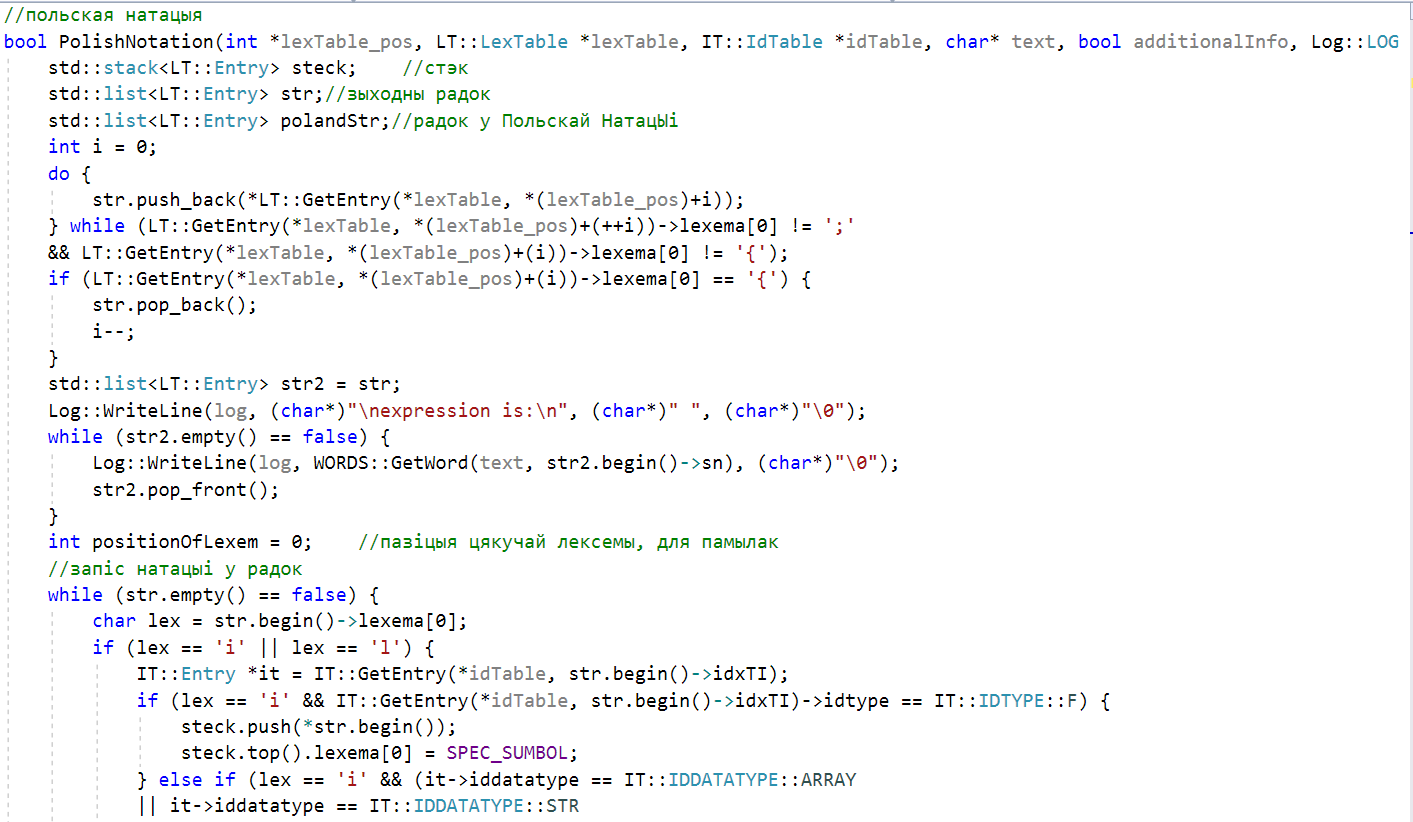
Rule& prule

); //атрмаць нумар правіла, альбо -1

Rule getRule(short n); //атрмаць правіла з нумара

};

**Дадатак В**



# Дадатак Г

# Дадатак Д

# Дадатак Е

# Литература