1. Задания для лексического анализа

Во всех задачах приняты следующие обозначения:

1) слова, выделенные жирным шрифтом (например, **модуль**) являются ключевыми словами рассматриваемых языков;

2) всё, заключённое в квадратные скобки зелёного цвета (`[...]`) является необязательным;

3) то, что заключено в фигурные скобки зелёного цвета (`{...}`), может повторяться любое число раз, в том числе и ни разу;

4) круглые скобки зелёного цвета (`(...)`) служат для группировки конструкций;

5) знак `|` зелёного цвета означает „или“;

6) запись ид обозначает идентификатор;

7) всё прочее, что имеет чёрный цвет, является либо знаком операции, либо разделителем языка из задачи.

Задача 1.1. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **модуль** *ид* {[**использует** *списид*] [**предоставляет** *списид*] *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**перем** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**тип** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
массив[[*выр*]{,[*выр*]}]*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**большое**|**маленькое**}(**цел**|**вещ**|**беззн**|**лог**|**симв**|**строка**)|**ничто**|**байт**|**цел8**|
цел16|**цел32**|**цел64**|**цел128**|**беззн8**|**беззн16**|**беззн32**|**беззн64**|**беззн128**|
вещ32|**вещ64**|**вещ128**|**лог8**|**лог16**|**лог32**|**лог64**|**симв8**|**симв16**|**симв32**|
строка8|**строка16**|**строка32**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**функция** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**значение** *списид*:*простейшее_определение_типа*|
ссылка *списид*:*простейшее_определение_типа*|
конст ссылка *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**перечисление** *ид*{*списид*}
структура →**структура** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**функция** *ид* *сигнатура_функции*{*тело_блока*}
прототип →**функция** *ид* *сигнатура_функции*;
константы →**конст** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$[?$выр_0$:$выр_0$]
$выр_0$ →$выр_1${(||**!**||^^|**!**^^)$выр_1$}
$выр_1$ →$выр_2${(&&|**!**&&)$выр_2$}
$выр_2$ →{**!**}$выр_3$

выр$_3$ →*выр*$_4${(<|>|<=|>=|=|!=)*выр*$_4$}
выр$_4$ →*выр*$_5${(|~|^|~^)*выр*$_5$}
выр$_5$ →*выр*$_6${(&|~&|≪|≫|⋙)*выр*$_6$}
выр$_6$ →{~}*выр*$_7$
выр$_7$ →*выр*$_8${(+|−)*выр*$_8$}
выр$_8$ →*выр*$_9${(*|/|%|/.)*выр*$_9$}
выр$_9$ →*выр*$_{10}$[(**|**.)*выр*$_9$]
выр$_{10}$ →[+|−|#]*выр*$_{11}$
выр$_{11}$→*выр*$_{12}$[##*выр*$_{12}$]
выр$_{12}$ →*целое*|*вещественное*|**истина**|**ложь**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**массив**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**если** *выр* **то**{*тело_блока*}{**инес** *выр* **то**{*тело_блока*}}[**иначе**{*тело_блока*}]**всё**
цикла →[<:*ид*:>](**для** *ид* **из** *выр*{*тело_блока*}|**для** *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}|
пока *выр*{*тело_блока*}|**повторяй**{*тело_блока*}**покуда** *выр*)
присваивания →*имя* := *выр*
перехода →**выйди** [**из** *ид*]|**продолжи** [*ид*]|**возврат** [*выр*]
ввода →**ввод**(*списимён*)
вывода →**вывод**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**разбор** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**иначе**{*тело_блока*}]}
образец →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0*osосьм_цифра*('?*восьм_цифра*)*
двоичное →0(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →0(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →0|1|2|3|4|5|6|7|8|9
восьм_цифра →0|1|2|3|4|5|6|7
двоичн_цифра →0|1
шестн_цифра →0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?
символ_порядка →E|e|S|s|D|d|Q|q
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.2. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **module** *ид* {[**import** *списид*] [**export** *списид*] *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**var** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**type** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
array[[*выр*]{,[*выр*]}]*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**long**|**short**}(**int**|**float**|**uint**|**bool**|**char**|**string**)|**void**|**byte**|**int8_t**|**int16_t**|
int32_t|**int64_t**|**int128_t**|**uint8_t**|**uint16_t**|**uint32_t**|**uint64_t**|
uint128_t|**float32_t**|**float64_t**|**float128_t**|**bool8_t**|**bool16_t**|**bool32_t**|
bool64_t|**char8_t**|**char16_t**|**char32_t**|**string8_t**|**string16_t**|**string32_t**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**fn** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**val** *списид*:*простейшее_определение_типа*|
ref *списид*:*простейшее_определение_типа*|
const ref *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**enum** *ид*{*списид*}
структура →**struct** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**fn** *ид* *сигнатура_функции*{*тело_блока*}
прототип →**fn** *ид* *сигнатура_функции*;
константы →**const** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
$выр \to выр_0\,[?выр_0{:}выр_0]$
$выр_0 \to выр_1\{(\texttt{||}|\texttt{!||}|\texttt{\^{}\^{}}|\texttt{!\^{}\^{}})выр_1\}$
$выр_1 \to выр_2\{(\texttt{\&\&}|\texttt{!\&\&})выр_2\}$
$выр_2 \to \{\texttt{!}\}выр_3$
$выр_3 \to выр_4\{(\texttt{<}|\texttt{>}|\texttt{<=}|\texttt{>=}|\texttt{=}|\texttt{!=})выр_4\}$
$выр_4 \to выр_5\{(\texttt{|}|\texttt{\~{}|}|\texttt{\^{}}|\texttt{\~{}\^{}})выр_5\}$
$выр_5 \to выр_6\{(\texttt{\&}|\texttt{\~{}\&}|\ll|\gg|\ggg)выр_6\}$
$выр_6 \to \{\texttt{\~{}}\}выр_7$
$выр_7 \to выр_8\{(+|-)выр_8\}$
$выр_8 \to выр_9\{(\texttt{*}|\texttt{/}|\texttt{\%}|\texttt{/.})выр_9\}$
$выр_9 \to выр_{10}\,[(\texttt{**}|\texttt{**.})выр_9]$
$выр_{10} \to [+|-|\texttt{\#}]выр_{11}$
$выр_{11} \to выр_{12}\,[\texttt{\#\#}выр_{12}]$

выр$_{12}$ →*целое*|*вещественное*|**true**|**false**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**array**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**if** *выр* **then**{*тело_блока*}{**elif** *выр* **then**{*тело_блока*}}[**else**{*тело_блока*}]**endif**
цикла →[<:*ид*:>](**for** *ид* : *выр*{*тело_блока*}|**for** *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}|
while *выр*{*тело_блока*}|**repeat**{*тело_блока*}**until** *выр*)
присваивания →*имя* := *выр*
перехода →**break** [*ид*]|**continue** [*ид*]|**return** [*выр*]
ввода →**read**(*списимён*)
вывода →**print**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**match** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**else**{*тело_блока*}]}
образец →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):
целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0o*восьм_цифра*('?*восьм_цифра*)*
двоичное →0(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →0(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →0|1|2|3|4|5|6|7|8|9
восьм_цифра →0|1|2|3|4|5|6|7
двоичн_цифра →0|1
шестн_цифра →0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?
символ_порядка →E|e|S|s|D|d|Q|q
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.3. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **модуль** *ид* {**[использует** *списид*] **[предоставляет** *списид*] *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**перем** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**тип** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
массив`[`[*выр*]{,[*выр*]}`]`*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**большое**|**маленькое**}(**цел**|**вещ**|**беззн**|**лог**|**симв**|**строка**)|**ничто**|**байт**|**цел8**|
цел16|**цел32**|**цел64**|**цел128**|**беззн8**|**беззн16**|**беззн32**|**беззн64**|**беззн128**|
вещ32|**вещ64**|**вещ128**|**лог8**|**лог16**|**лог32**|**лог64**|**симв8**|**симв16**|**симв32**|
строка8|**строка16**|**строка32**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**функция** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**значение** *списид*:*простейшее_определение_типа*|
ссылка *списид*:*простейшее_определение_типа*|
конст ссылка *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**перечисление** *ид*{*списид*}
структура →**структура** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**функция** *ид* *сигнатура_функции*{*тело_блока*}
прототип →**функция** *ид* *сигнатура_функции*;
константы →**конст** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$`[`?$выр_0$:$выр_0$`]`
$выр_0$ →$выр_1${(`||`|`!||`|`^^`|`!^^`)$выр_1$}
$выр_1$ →$выр_2${(`&&`|`!&&`)$выр_2$}
$выр_2$ →{`!`}$выр_3$
$выр_3$ →$выр_4${(`<`|`>`|`<=`|`>=`|`=`|`!=`)$выр_4$}
$выр_4$ →$выр_5${(`|`|`~|`|`^`|`~^`)$выр_5$}
$выр_5$ →$выр_6${(`&`|`~&`|≪|≫|⋙)$выр_6$}
$выр_6$ →{`~`}$выр_7$
$выр_7$ →$выр_8${(+|−)$выр_8$}
$выр_8$ →$выр_9${(`*`|`/`|`%`|`/.`)$выр_9$}
$выр_9$ →$выр_{10}$`[`(`**`|`**.`)$выр_9$`]`
$выр_{10}$ →`[`+|−|`#``]`$выр_{11}$
$выр_{11}$→$выр_{12}$`[##`$выр_{12}$`]`
$выр_{12}$ →*целое*|*вещественное*|**истина**|**ложь**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_струкm*|*знач_массива*
знач_струкm →`[`|:*имя_модуля*::`]`*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* `<-` *выр*
знач_массива →**массив**`[:`*выр*,*элементарный_тип*`:]`{*списвыр*}
имя →`[`|:*имя_модуля*::`]` [*имя_типа*::*имя_части_типа*::]*ид*{(`[`*списвыр*`]`)|`[`*списвыр*`]`|@|.*ид*}
списвыр →*выр*{,*выр*}

имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →`[`*оператор*`]{;[`*оператор*`]}`
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**если** *выр* **то**{*тело_блока*}{**инес** *выр* **то**{*тело_блока*}}`[`**иначе**{*тело_блока*}`]`**всё**
цикла →`[<:`*ид*`:>](`**для** *ид* **из** *выр*{*тело_блока*}`[`**завершение**{*тело_блока*}`]`|
для *ид* `:=` *выр*,*выр*`[`,*выр*`]`{*тело_блока*}`[`**завершение**{*тело_блока*}`]`|
пока *выр*{*тело_блока*}|**повторяй**{*тело_блока*}**покуда** *выр*`)`
присваивания →*имя* `:=` *выр*
перехода →**выйди** `[`**из** *ид*`]`|**продолжи** `[`*ид*`]`|**возврат** `[`*выр*`]`
ввода →**ввод**`(`*списимён*`)`
вывода →**вывод**`(`*списвыр*`)`
списимён →*имя*{,*имя*}
разбора →**разбор** *выр*{*образец*`->{`*тело_блока*`}`{,*образец*`->{`*тело_блока*`}}[`**иначе**{*тело_блока*}`]}`
образец →`[|:`*имя_модуля*`::]`*имя_типа*`::`*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →*0овосьм_цифра*('?*восьм_цифра*)*
двоичное →*0*(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →*0*(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*
восьм_цифра →*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*
двоичн_цифра →*0*|*1*
шестн_цифра →*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?
символ_порядка →*E*|*e*|*S*|*s*|*D*|*d*|*Q*|*q*
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.4. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **module** *ид* {`[`**import** *списид*`]` `[`**export** *списид*`]` *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**var** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}

типы →**type** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
array[[*выр*]{,[*выр*]}]*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**long**|**short**}(**int**|**float**|**uint**|**bool**|**char**|**string**)|**void**|**byte**|**int8_t**|**int16_t**|
int32_t|**int64_t**|**int128_t**|**uint8_t**|**uint16_t**|**uint32_t**|**uint64_t**|
uint128_t|**float32_t**|**float64_t**|**float128_t**|**bool8_t**|**bool16_t**|**bool32_t**|
bool64_t|**char8_t**|**char16_t**|**char32_t**|**string8_t**|**string16_t**|**string32_t**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**fn** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**val** *списид*:*простейшее_определение_типа*|
ref *списид*:*простейшее_определение_типа*|
const ref *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**enum** *ид*{*списид*}
структура →**struct** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**fn** *ид* *сигнатура_функции*{*тело_блока*}
прототип →**fn** *ид* *сигнатура_функции*;
константы →**const** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$[?$выр_0$:$выр_0$]
$выр_0$ →$выр_1${(||/**!**||/^^/**!**^^)$выр_1$}
$выр_1$ →$выр_2${(&&/**!**&&)$выр_2$}
$выр_2$ →{**!**}$выр_3$
$выр_3$ →$выр_4${(</>/<=/>=/=/**!**=)$выр_4$}
$выр_4$ →$выр_5${(|/~|/^/~^)$выр_5$}
$выр_5$ →$выр_6${(&/~&/≪/≫/⋙)$выр_6$}
$выр_6$ →{~}$выр_7$
$выр_7$ →$выр_8${(+/−)$выр_8$}
$выр_8$ →$выр_9${(*/ /%/ /.)$выр_9$}
$выр_9$ →$выр_{10}$[(**/**.)$выр_9$]
$выр_{10}$ →[+/−/#]$выр_{11}$
$выр_{11}$→$выр_{12}$[##$выр_{12}$]
$выр_{12}$ →*целое*|*вещественное*|**true**|**false**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**array**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**if** *выр* **then**{*тело_блока*}{**elif** *выр* **then**{*тело_блока*}}[**else**{*тело_блока*}]**endif**

цикла →`[<:`*ид*`:>](`**for** *ид* `:` *выр*`{`*тело_блока*`}[`**else**`{`*тело_блока*`}]`|
for *ид* `:=` *выр*`,`*выр*`[,`*выр*`]{`*тело_блока*`}[`**else**`{`*тело_блока*`}]`|
while *выр*`{`*тело_блока*`}`|**repeat**`{`*тело_блока*`}`**until** *выр*`)`
присваивания →*имя* `:=` *выр*
перехода →**break** `[` *ид*`]`|**continue** `[`*ид*`]`|**return** `[`*выр*`]`
ввода →**read**`(`*списимён*`)`
вывода →**print**`(`*списвыр*`)`
списимён →*имя*`{,`*имя*`}`
разбора →**match** *выр*`{`*образец*`->{`*тело_блока*`}{,`*образец*`->{`*тело_блока*`}}[`**else**`{`*тело_блока*`}]}`
образец →`[|:`*имя_модуля*`::]`*имя_типа*`::`*имя_части_типа*`{..}`

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*(’?*дес_цифра*)*
восьмеричное →*0o восьм_цифра*(’?*восьм_цифра*)*
двоичное →*0*(b|B)*двоичн_цифра*(’?*двоичн_цифра*)*
шестнадцатеричное →*0*(x|X)*шестн_цифра*(’?*шестн_цифра*)*
дес_цифра →*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*
восьм_цифра →*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*
двоичн_цифра →*0*|*1*
шестн_цифра →*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*|a|b|c|d|e|f|A|B|C|D|E|F*
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?
символ_порядка →*E*|*e*|*S*|*s*|*D*|*d*|*Q*|*q*
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.5. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **модуль** *ид* {`[`**использует** *списид*`]` `[`**предоставляет** *списид*`]` *тело_модуля*}
списид →*ид*`{,`*ид*`}`
тело_модуля →`{`*переменные*|*типы*|*функция*|*прототип*|*константы*`}`
переменные →**перем** *списид*`:`*простейшее_определение_типа*`{;`*списид*`:`*простейшее_определение_типа*`}`
типы →**тип** *ид* = *определение_типа*`{;`*ид* = *определение_типа*`}`
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
массив`[[`*выр*`]{,[`*выр*`]}]`*простейшее_определение_типа*
элементарный_тип →`{@}(`*встроенный_тип*|*ид*`)`

встроенный_тип →{**большое**|**маленькое**}(**цел**|**вещ**|**беззн**|**лог**|**симв**|**строка**)|**ничто**|**байт**|**цел8**|
цел16|**цел32**|**цел64**|**цел128**|**беззн8**|**беззн16**|**беззн32**|**беззн64**|**беззн128**|
вещ32|**вещ64**|**вещ128**|**лог8**|**лог16**|**лог32**|**лог64**|**симв8**|**симв16**|**симв32**|
строка8|**строка16**|**строка32**

составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа

определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)

определение_указателя_на_функцию →**функция** *сигнатура_функции*

сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*

спис_форм_парам →*форм_парам*{,*форм_парам*}

форм_парам →**значение** *списид*:*простейшее_определение_типа*|
ссылка *списид*:*простейшее_определение_типа*|
конст ссылка *списид*:*простейшее_определение_типа*

определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}

часть_алгебр_типа →*перечисление*|*структура*

перечисление →**перечисление** *ид*{*списид*}

структура →**структура** *ид*{*спис_групп_полей*}

спис_групп_полей →*группа_полей*{;*группа_полей*}

группа_полей →*списид*:*простейшее_определение_типа*

функция →**функция** *ид сигнатура_функции*{*тело_блока*}

прототип →**функция** *ид сигнатура_функции*;

константы →**конст** *определение_константы*{,*определение_константы*}

определение_константы →*ид*:*простейшее_определение_типа* = *выр*

тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}

выр →$выр_0$[?$выр_0$:$выр_0$]

$выр_0$ →$выр_1${(||**!**|||^^|**!**^^)$выр_1$}

$выр_1$ →$выр_2${(&&|**!**&&)$выр_2$}

$выр_2$ →{**!**}$выр_3$

$выр_3$ →$выр_4${(<|>|<=|>=|=|**!**=)$выр_4$}

$выр_4$ →$выр_5${(|~|^|~^)$выр_5$}

$выр_5$ →$выр_6${(&|~&|≪|≫|⋙)$выр_6$}

$выр_6$ →{~}$выр_7$

$выр_7$ →$выр_8${(+|−)$выр_8$}

$выр_8$ →$выр_9${(*|/|%|/.)$выр_9$}

$выр_9$ →$выр_{10}$[(**|**.)$выр_9$]

$выр_{10}$ →[+|−|#]$выр_{11}$

$выр_{11}$→$выр_{12}$[##$выр_{12}$]

$выр_{12}$ →*целое*|*вещественное*|**истина**|**ложь**|*строковое*|(*выр*)|*имя*|*сост_знач*

сост_знач →*знач_структ*|*знач_массива*

знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}

значение_поля →*ид* <- *выр*

знач_массива →**массив**[:*выр*,*элементарный_тип*:]{*списвыр*}

имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}

списвыр →*выр*{,*выр*}

имя_модуля →*ид*

имя_типа →*ид*

имя_части_типа →*ид*

операторы →[*оператор*]{;[*оператор*]}

оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*

условный →**если** *выр* **то**{*тело_блока*}{**инес** *выр* **то**{*тело_блока*}}[**иначе**{*тело_блока*}]**всё**

цикла →[<:*ид*:>](**для** *ид* **из** *выр*{*тело_блока*}|**для** *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}|
пока *выр*{*тело_блока*}|**повторяй**{*тело_блока*}**покуда** *выр*)

присваивания →*имя* (:=|+:=|-:=|*:=|/:=|/.:=|**:=|**.:=|%:=|&:=|~&:=|≪:=|≫:=||:=|~|:=|^:=|~^:=|
||:=|**!**||:=|^^:=|**!**^^:=|&&:=|**!**&&:=|⋙:=) *выр*

перехода →**выйди** [**из** *ид*]|**продолжи** [*ид*]|**возврат** [*выр*]

ввода →**ввод**(*списимён*)

вывода →**вывод**(*списвыр*)

списимён →*имя*{,*имя*}
разбора →**разбор** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**иначе**{*тело_блока*}]}
образец →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):
целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0o*восьм_цифра*('?*восьм_цифра*)*
двоичное →0(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →0(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →0|1|2|3|4|5|6|7|8|9
восьм_цифра →0|1|2|3|4|5|6|7
двоичн_цифра →0|1
шестн_цифра →0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?
символ_порядка →E|e|S|s|D|d|Q|q
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`, а для хранения вещественного — значение типа `__float128`.

Задача 1.6. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*
программа → **module** *ид* {[**import** *списид*] [**export** *списид*] *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**var** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**type** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
array[[*выр*]{,[*выр*]}]*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**long**|**short**}(**int**|**float**|**uint**|**bool**|**char**|**string**)|**void**|**byte**|**int8_t**|**int16_t**|
int32_t|**int64_t**|**int128_t**|**uint8_t**|**uint16_t**|**uint32_t**|**uint64_t**|
uint128_t|**float32_t**|**float64_t**|**float128_t**|**bool8_t**|**bool16_t**|**bool32_t**|
bool64_t|**char8_t**|**char16_t**|**char32_t**|**string8_t**|**string16_t**|**string32_t**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**fn** *сигнатура_функции*

сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**val** *списид*:*простейшее_определение_типа*|
ref *списид*:*простейшее_определение_типа*|
const ref *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**enum** *ид*{*списид*}
структура →**struct** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**fn** *ид сигнатура_функции*{*тело_блока*}
прототип →**fn** *ид сигнатура_функции*;
константы →**const** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$[?$выр_0$:$выр_0$]
$выр_0$ →$выр_1${(||!||^^|!^^)$выр_1$}
$выр_1$ →$выр_2${(&&|!&&)$выр_2$}
$выр_2$ →{!}$выр_3$
$выр_3$ →$выр_4${(<|>|<=|>=|=|!=)$выр_4$}
$выр_4$ →$выр_5${(|~|^|~^)$выр_5$}
$выр_5$ →$выр_6${(&|~&|≪|≫|⋙)$выр_6$}
$выр_6$ →{~}$выр_7$
$выр_7$ →$выр_8${(+|−)$выр_8$}
$выр_8$ →$выр_9${(*|/|%|/.)$выр_9$}
$выр_9$ →$выр_{10}$[(**|**.)$выр_9$]
$выр_{10}$ →[+|−|#]$выр_{11}$
$выр_{11}$→$выр_{12}$[##$выр_{12}$]
$выр_{12}$ →*целое*|*вещественное*|**true**|**false**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**array**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**if** *выр* **then**{*тело_блока*}{**elif** *выр* **then**{*тело_блока*}}[**else**{*тело_блока*}]**endif**
цикла →[<:*ид*:>](**for** *ид* : *выр*{*тело_блока*}|**for** *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}|
while *выр*{*тело_блока*}|**repeat**{*тело_блока*}**until** *выр*)
присваивания →*имя* (:=|+:=|-:=|*:=|/:=|/.:=|**:=|**.:=|%:=|&:=|~&:=|≪:=|≫:=||:=|~|:=|^:=|~^:=|
||:=|!||:=|^^:=|!^^:=|&&:=|!&&:=|⋙:=) *выр*
перехода →**break** [*ид*]|**continue** [*ид*]|**return** [*выр*]
ввода →**read**(*списимён*)
вывода →**print**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**match** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**else**{*тело_блока*}]}
образец →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0*овосьм_цифра*('?*восьм_цифра*)*

двоичное →*0(b|B)двоичн_ цифра('?двоичн_ цифра)**
шестнадцатеричное →*0(x|X)шестн_ цифра('?шестн_ цифра)**
дес_ цифра →*0|1|2|3|4|5|6|7|8|9*
восьм_ цифра →*0|1|2|3|4|5|6|7*
двоичн_ цифра →*0|1*
шестн_ цифра →*0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F*
вещественное →*целая_ часть(дробная_ часть)?(символ_ порядка(+|-)?порядок)?*
символ_ порядка →*E|e|S|s|D|d|Q|q*
целая_ часть →*десятичное*
дробная_ часть →*.десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.7. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **модуль** *ид* {*[***используует** *списид] [***предоставляет** *списид] тело_ модуля*}
списид →*ид{,ид}*
тело_ модуля →*{переменные|типы|функция|прототип|константы}*
переменные →**перем** *списид:простейшее_ определение_ типа{;списид:простейшее_ определение_ типа}*
типы →**тип** *ид* = *определение_ типа{;ид* = *определение_ типа}*
определение_ типа →*простейшее_ определение_ типа|составное_ определение_ типа*
простейшее_ определение_ типа →*элементарный_ тип|*
массив*[[выр]{,[выр]}]простейшее_ определение_ типа*
элементарный_ тип →*{@}(встроенный_ тип|ид)*
встроенный_ тип →*{***большое***|***маленькое***}(***цел***|***вещ***|***беззн***|***лог***|***симв***|***строка***)|***ничто***|***байт***|***цел8***|*
цел16|***цел32***|***цел64***|***цел128***|***беззн8***|***беззн16***|***беззн32***|***беззн64***|***беззн128***|*
вещ32|***вещ64***|***вещ128***|***лог8***|***лог16***|***лог32***|***лог64***|***симв8***|***симв16***|***симв32***|*
строка8|***строка16***|***строка32***
составное_ определение_ типа →*определение_ кортежа|определение_ указателя_ на_ функцию|*
определение_ алгебраического_ типа
определение_ кортежа →*(:элементарный_ тип{,элементарный_ тип}:)*
определение_ указателя_ на_ функцию →**функция** *сигнатура_ функции*
сигнатура_ функции →*(спис_ форм_ парам):элементарный_ тип*
спис_ форм_ парам →*форм_ парам{,форм_ парам}*
форм_ парам →**значение** *списид:простейшее_ определение_ типа|*
ссылка *списид:простейшее_ определение_ типа|*
конст ссылка *списид:простейшее_ определение_ типа*
определение_ алгебраического_ типа →*часть_ алгебр_ типа{.|.часть_ алгебр_ типа}*
часть_ алгебр_ типа →*перечисление|структура*
перечисление →**перечисление** *ид{списид}*

структура →**структура** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**функция** *ид сигнатура_функции*{*тело_блока*}
прототип →**функция** *ид сигнатура_функции*;
константы →**конст** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$ [?$выр_0$:$выр_0$]
$выр_0$ →$выр_1${(||!||^^|!^^)$выр_1$}
$выр_1$ →$выр_2${(&&|!&&)$выр_2$}
$выр_2$ →{!}$выр_3$
$выр_3$ →$выр_4${(<|>|<=|>=|=|!=)$выр_4$}
$выр_4$ →$выр_5${(|~|^|~^)$выр_5$}
$выр_5$ →$выр_6${(&|~&|≪|≫|⋙)$выр_6$}
$выр_6$ →{~}$выр_7$
$выр_7$ →$выр_8${(+|−)$выр_8$}
$выр_8$ →$выр_9${(*|/|%|/.)$выр_9$}
$выр_9$ →$выр_{10}$ [(**|**.)$выр_9$]
$выр_{10}$ →[+|−|#]$выр_{11}$
$выр_{11}$→$выр_{12}$ [##$выр_{12}$]
$выр_{12}$ →*целое*|*вещественное*|**истина**|**ложь**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**массив**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**если** *выр* **то**{*тело_блока*}{**инес** *выр* **то**{*тело_блока*}}[**иначе**{*тело_блока*}]**всё**
цикла →[<:*ид*:>](**для** *ид* **из** *выр*{*тело_блока*}[**завершение**{*тело_блока*}]|
для *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}[**завершение**{*тело_блока*}]|
пока *выр*{*тело_блока*}|**повторяй**{*тело_блока*}**покуда** *выр*)
присваивания →*имя* (:=|+:=|-:=|*:=|/:=|/.:=|**:=|**.:=|%:=|&:=|~&:=|≪:=|≫:=||:=|~|:=|^:=|~^:=|
||:=|!||:=|^^:=|!^^:=|&&:=|!&&:=|⋙:=) *выр*
перехода →**выйди** [**из** *ид*]|**продолжи** [*ид*]|**возврат** [*выр*]
ввода →**ввод**(*списимён*)
вывода →**вывод**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**разбор** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**иначе**{*тело_блока*}]}
образец →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0*восьм_цифра*('?*восьм_цифра*)*
двоичное →0(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →0(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →0|1|2|3|4|5|6|7|8|9
восьм_цифра →0|1|2|3|4|5|6|7
двоичн_цифра →0|1
шестн_цифра →0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?

символ_порядка →*E*|*e*|*S*|*s*|*D*|*d*|*Q*|*q*
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.8. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **module** *ид* {[**import** *списид*] [**export** *списид*] *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**var** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**type** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
array[[*выр*]{,[*выр*]}]*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**long**|**short**}(**int**|**float**|**uint**|**bool**|**char**|**string**)|**void**|**byte**|**int8_t**|**int16_t**|
int32_t|**int64_t**|**int128_t**|**uint8_t**|**uint16_t**|**uint32_t**|**uint64_t**|
uint128_t|**float32_t**|**float64_t**|**float128_t**|**bool8_t**|**bool16_t**|**bool32_t**|
bool64_t|**char8_t**|**char16_t**|**char32_t**|**string8_t**|**string16_t**|**string32_t**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**fn** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**val** *списид*:*простейшее_определение_типа*|
ref *списид*:*простейшее_определение_типа*|
const ref *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**enum** *ид*{*списид*}
структура →**struct** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**fn** *ид сигнатура_функции*{*тело_блока*}
прототип →**fn** *ид сигнатура_функции*;
константы →**const** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*

тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →*выр*$_0$[?*выр*$_0$:*выр*$_0$]
выр$_0$ →*выр*$_1${(|||!||||^^|!^^)*выр*$_1$}
выр$_1$ →*выр*$_2${(&&|!&&)*выр*$_2$}
выр$_2$ →{!}*выр*$_3$
выр$_3$ →*выр*$_4${(<|>|<=|>=|=|!=)*выр*$_4$}
выр$_4$ →*выр*$_5${(|~||^|~^)*выр*$_5$}
выр$_5$ →*выр*$_6${(&|~&|≪|≫|⋙)*выр*$_6$}
выр$_6$ →{~}*выр*$_7$
выр$_7$ →*выр*$_8${(+|−)*выр*$_8$}
выр$_8$ →*выр*$_9${(*|/|%|/.)*выр*$_9$}
выр$_9$ →*выр*$_{10}$[(**|**.)*выр*$_9$]
выр$_{10}$ →[+|−|#]*выр*$_{11}$
выр$_{11}$→*выр*$_{12}$[##*выр*$_{12}$]
выр$_{12}$ →целое|вещественное|**true**|**false**|строковое|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**array**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**if** *выр* **then**{*тело_блока*}{**elif** *выр* **then**{*тело_блока*}}[**else**{*тело_блока*}]**endif**
цикла →[<:*ид*:>](**for** *ид* : *выр*{*тело_блока*}[**else**{*тело_блока*}]|
for *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}[**else**{*тело_блока*}]|
while *выр*{*тело_блока*}|**repeat**{*тело_блока*}**until** *выр*)
присваивания →*имя* (:=|+:=|-:=|*:=|/:=|/.:=|**:=|**.:=|%:=|&:=|~&:=|≪:=|≫:=||:=|~|:=|^:=|~^:=|
||:=|!||:=|^^:=|!^^:=|&&:=|!&&:=|⋙:=) *выр*
перехода →**break** [*ид*]|**continue** [*ид*]|**return** [*выр*]
ввода →**read**(*списимён*)
вывода →**print**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**match** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**else**{*тело_блока*}]}
образец →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0o*восьм_цифра*('?*восьм_цифра*)*
двоичное →0(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →0(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →0|1|2|3|4|5|6|7|8|9
восьм_цифра →0|1|2|3|4|5|6|7
двоичн_цифра →0|1
шестн_цифра →0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?
символ_порядка →E|e|S|s|D|d|Q|q
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый

в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.9. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **модуль** *ид* {[**использует** *списид*] [**предоставляет** *списид*] *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**перем** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**тип** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
массив[[*выр*]{,[*выр*]}]*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**большое**|**маленькое**}(**цел**|**вещ**|**беззн**|**лог**|**симв**|**строка**)|**ничто**|**байт**|**цел8**|
цел16|**цел32**|**цел64**|**цел128**|**беззн8**|**беззн16**|**беззн32**|**беззн64**|**беззн128**|
вещ32|**вещ64**|**вещ128**|**лог8**|**лог16**|**лог32**|**лог64**|**симв8**|**симв16**|**симв32**|
строка8|**строка16**|**строка32**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**функция** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**значение** *списид*:*простейшее_определение_типа*|
ссылка *списид*:*простейшее_определение_типа*|
конст ссылка *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**перечисление** *ид*{*списид*}
структура →**структура** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**функция** *ид* *сигнатура_функции*{*тело_блока*}
прототип →**функция** *ид* *сигнатура_функции*;
константы →**конст** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$[?$выр_0$:$выр_0$]
$выр_0$ →$выр_1${(||!||^^|!^^)$выр_1$}
$выр_1$ →$выр_2${(&&|!&&)$выр_2$}
$выр_2$ →{!}$выр_3$
$выр_3$ →$выр_4${(<|>|<=|>=|=|!=)$выр_4$}
$выр_4$ →$выр_5${(|~|^|~^)$выр_5$}

выр$_5$ →*выр*$_6${(**&**|**~&**|≪|≫|⋙)*выр*$_6$}
выр$_6$ →{**~**}*выр*$_7$
выр$_7$ →*выр*$_8${(+|−)*выр*$_8$}
выр$_8$ →*выр*$_9${(*****|**/**|**%**|**/.**)*выр*$_9$}
выр$_9$ →*выр*$_{10}$[(*****|*****.**)*выр*$_9$]
выр$_{10}$ →[+|−|**#**]*выр*$_{11}$
выр$_{11}$→*выр*$_{12}$[**##***выр*$_{12}$]
выр$_{12}$ →целое|вещественное|**истина**|**ложь**|строковое|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* **<-** *выр*
знач_массива →**массив**[**:***выр*,*элементарный_тип* **:**]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**если** *выр* **то**{*тело_блока*}{**инес** *выр* **то**{*тело_блока*}}[**иначе**{*тело_блока*}]**всё**
цикла →[**<:***ид***:>**](**для** *ид* **из** *выр*{*тело_блока*}|**для** *ид* **:=** *выр*,*выр*[,*выр*]{*тело_блока*}|
пока *выр*{*тело_блока*}|**повторяй**{*тело_блока*}**покуда** *выр*|
паук{*случай*{,*случай*}[**иначе**{*тело_блока*}]})
случай →**при** *выр*{*тело_блока*}
присваивания →*имя* **:=** *выр*
перехода →**выйди** [**из** *ид*]|**продолжи** [*ид*]|**возврат** [*выр*]
ввода →**ввод**(*списимён*)
вывода →**вывод**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**разбор** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**иначе**{*тело_блока*}]}
образец →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0*восьм_цифра*('?*восьм_цифра*)*
двоичное →0(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →0(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →0|1|2|3|4|5|6|7|8|9
восьм_цифра →0|1|2|3|4|5|6|7
двоичн_цифра →0|1
шестн_цифра →0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?
символ_порядка →E|e|S|s|D|d|Q|q
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.10. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **module** *ид* {[**import** *списид*] [**export** *списид*] *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**var** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**type** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
array[[*выр*]{,[*выр*]}]*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**long**|**short**}(**int**|**float**|**uint**|**bool**|**char**|**string**)|**void**|**byte**|**int8_t**|**int16_t**|
int32_t|**int64_t**|**int128_t**|**uint8_t**|**uint16_t**|**uint32_t**|**uint64_t**|
uint128_t|**float32_t**|**float64_t**|**float128_t**|**bool8_t**|**bool16_t**|**bool32_t**|
bool64_t|**char8_t**|**char16_t**|**char32_t**|**string8_t**|**string16_t**|**string32_t**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**fn** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**val** *списид*:*простейшее_определение_типа*|
ref *списид*:*простейшее_определение_типа*|
const ref *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**enum** *ид*{*списид*}
структура →**struct** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**fn** *ид* *сигнатура_функции*{*тело_блока*}
прототип →**fn** *ид* *сигнатура_функции*;
константы →**const** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
$выр \to выр_0\,[?выр_0{:}выр_0]$
$выр_0 \to выр_1\{(\texttt{||}|\texttt{!||}|\texttt{\^{}\^{}}|\texttt{!\^{}\^{}})выр_1\}$
$выр_1 \to выр_2\{(\texttt{\&\&}|\texttt{!\&\&})выр_2\}$
$выр_2 \to \{\texttt{!}\}выр_3$
$выр_3 \to выр_4\{(\texttt{<}|\texttt{>}|\texttt{<=}|\texttt{>=}|\texttt{=}|\texttt{!=})выр_4\}$
$выр_4 \to выр_5\{(\texttt{|}|\texttt{\~{}|}|\texttt{\^{}}|\texttt{\~{}\^{}})выр_5\}$
$выр_5 \to выр_6\{(\texttt{\&}|\texttt{\~{}\&}|\ll|\gg|\ggg)выр_6\}$
$выр_6 \to \{\texttt{\~{}}\}выр_7$
$выр_7 \to выр_8\{(+|-)выр_8\}$
$выр_8 \to выр_9\{(\texttt{*}|\texttt{/}|\texttt{\%}|\texttt{/.})выр_9\}$
$выр_9 \to выр_{10}\,[(\texttt{**}|\texttt{**.})выр_9]$
$выр_{10} \to [+|-|\texttt{\#}]выр_{11}$
$выр_{11} \to выр_{12}\,[\texttt{\#\#}выр_{12}]$

выр$_{12}$ →*целое*|*вещественное*|**true**|**false**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**array**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**if** *выр* **then**{*тело_блока*}{**elif** *выр* **then**{*тело_блока*}}[**else**{*тело_блока*}]**endif**
цикла →[<:*ид*:>](**for** *ид* : *выр*{*тело_блока*}|**for** *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}|
while *выр*{*тело_блока*}|**repeat**{*тело_блока*}**until** *выр*|
spider{*случай*{,*случай*}[**else**{*тело_блока*}]})
случай →**case** *выр*{*тело_блока*}
присваивания →*имя* := *выр*
перехода →**break** [*ид*]|**continue** [*ид*]|**return** [*выр*]
ввода →**read**(*списимён*)
вывода →**print**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**match** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**else**{*тело_блока*}]}
образец →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0o*восьм_цифра*('?*восьм_цифра*)*
двоичное →0(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →0(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →0|1|2|3|4|5|6|7|8|9
восьм_цифра →0|1|2|3|4|5|6|7
двоичн_цифра →0|1
шестн_цифра →0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?
символ_порядка →E|e|S|s|D|d|Q|q
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.11. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **модуль** *ид* {`[`**использует** *списид*`]` `[`**предоставляет** *списид*`]` *тело_модуля*}
списид →*ид*{`,`*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**перем** *списид*`:`*простейшее_определение_типа*{`;`*списид*`:`*простейшее_определение_типа*}
типы →**тип** *ид* = *определение_типа*{`;`*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
массив`[`*выр*`]`{`,`*выр*`]`}*простейшее_определение_типа*
элементарный_тип →{`@`}(*встроенный_тип*|*ид*)
встроенный_тип →{**большое**|**маленькое**}(**цел**|**вещ**|**беззн**|**лог**|**симв**|**строка**)|**ничто**|**байт**|**цел8**|
цел16|**цел32**|**цел64**|**цел128**|**беззн8**|**беззн16**|**беззн32**|**беззн64**|**беззн128**|
вещ32|**вещ64**|**вещ128**|**лог8**|**лог16**|**лог32**|**лог64**|**симв8**|**симв16**|**симв32**|
строка8|**строка16**|**строка32**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(`:`*элементарный_тип*{`,`*элементарный_тип*}`:`)
определение_указателя_на_функцию →**функция** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*)`:`*элементарный_тип*
спис_форм_парам →*форм_парам*{`,`*форм_парам*}
форм_парам →**значение** *списид*`:`*простейшее_определение_типа*|
ссылка *списид*`:`*простейшее_определение_типа*|
конст ссылка *списид*`:`*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{`.`|`.`*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**перечисление** *ид*{*списид*}
структура →**структура** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{`;`*группа_полей*}
группа_полей →*списид*`:`*простейшее_определение_типа*
функция →**функция** *ид* *сигнатура_функции*{*тело_блока*}
прототип →**функция** *ид* *сигнатура_функции*`;`
константы →**конст** *определение_константы*{`,`*определение_константы*}
определение_константы →*ид*`:`*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →*выр*$_0$`[`?*выр*$_0$`:`*выр*$_0$`]`
выр$_0$ →*выр*$_1$`{((|||!|||^^|!^^)`*выр*$_1$`}`
выр$_1$ →*выр*$_2$`{((&&|!&&)`*выр*$_2$`}`
выр$_2$ →`{!}`*выр*$_3$
выр$_3$ →*выр*$_4$`{(<|>|<=|>=|=|!=)`*выр*$_4$`}`
выр$_4$ →*выр*$_5$`{(|~|^|~^)`*выр*$_5$`}`
выр$_5$ →*выр*$_6$`{(&|~&|≪|≫|⋙)`*выр*$_6$`}`
выр$_6$ →`{~}`*выр*$_7$
выр$_7$ →*выр*$_8$`{(+|-)`*выр*$_8$`}`
выр$_8$ →*выр*$_9$`{(*|/|%|/.)`*выр*$_9$`}`
выр$_9$ →*выр*$_{10}$`[(**|**.)`*выр*$_9$`]`
выр$_{10}$ →`[+|-|#]`*выр*$_{11}$
выр$_{11}$→*выр*$_{12}$`[##`*выр*$_{12}$`]`
выр$_{12}$ →*целое*|*вещественное*|**истина**|**ложь**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →`[|:`*имя_модуля*`::]`*имя_типа*`::`*имя_части_типа*{*значение_поля*{`,`*значение_поля*}}
значение_поля →*ид* `<-` *выр*
знач_массива →**массив**`[:`*выр*`,`*элементарный_тип*`:]`{*списвыр*}
имя →`[|:`*имя_модуля*`::]` `[`*имя_типа*`::`*имя_части_типа*`::]`*ид*{(`[`*списвыр*`]`)|`[`*списвыр*`]`|@|`.`*ид*}
списвыр →*выр*{`,`*выр*}

имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →`[`*оператор*`]{;[`*оператор*`]}`
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**если** *выр* **то**{*тело_блока*}{**инес** *выр* **то**{*тело_блока*}}`[`**иначе**{*тело_блока*}`]`**всё**
цикла →`[<:`*ид*`:>](`**для** *ид* **из** *выр*{*тело_блока*}`[`**завершение**{*тело_блока*}`]`|
для *ид* `:=` *выр*,*выр*`[`,*выр*`]`{*тело_блока*}`[`**завершение**{*тело_блока*}`]`|
пока *выр*{*тело_блока*}|
паук{*случай*{`,`*случай*}`[`**иначе**{*тело_блока*}`]}`)
случай →**при** *выр*{*тело_блока*}
присваивания →*имя* `:=` *выр*
перехода →**выйди** `[`**из** *ид*`]`|**продолжи** `[`*ид*`]`|**возврат** `[`*выр*`]`
ввода →**ввод**(*списимён*)
вывода →**вывод**(*списвыр*)
списимён →*имя*{`,`*имя*}
разбора →**разбор** *выр*{*образец*`->{`*тело_блока*`}`{`,`*образец*`->{`*тело_блока*`}}[`**иначе**{*тело_блока*}`]}`
образец →`[|:`*имя_модуля*`::]`*имя_типа*`::`*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):
целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →*0o**восьм_цифра*('?*восьм_цифра*)*
двоичное →*0*(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →*0*(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*
восьм_цифра →*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*
двоичн_цифра →*0*|*1*
шестн_цифра →*0*|*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*|a|b|c|d|e|f|A|B|C|D|E|F*
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?*
символ_порядка →*E*|*e*|*S*|*s*|*D*|*d*|*Q*|*q*
целая_часть →*десятичное*
дробная_часть →*.десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.12. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*
программа → **module** *ид* {`[`**import** *списид*`]` `[`**export** *списид*`]` *тело_модуля*}
списид →*ид*{`,`*ид*}

тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**var** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**type** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
array[[*выр*]{,[*выр*]}]*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**long**|**short**}(**int**|**float**|**uint**|**bool**|**char**|**string**)|**void**|**byte**|**int8_t**|**int16_t**|
int32_t|**int64_t**|**int128_t**|**uint8_t**|**uint16_t**|**uint32_t**|**uint64_t**|
uint128_t|**float32_t**|**float64_t**|**float128_t**|**bool8_t**|**bool16_t**|**bool32_t**|
bool64_t|**char8_t**|**char16_t**|**char32_t**|**string8_t**|**string16_t**|**string32_t**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**fn** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**val** *списид*:*простейшее_определение_типа*|
ref *списид*:*простейшее_определение_типа*|
const ref *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**enum** *ид*{*списид*}
структура →**struct** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**fn** *ид* *сигнатура_функции*{*тело_блока*}
прототип →**fn** *ид* *сигнатура_функции*;
константы →**const** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$[?$выр_0$:$выр_0$]
$выр_0$ →$выр_1${(|||**!**|||^^|**!**^^)$выр_1$}
$выр_1$ →$выр_2${(&&|**!**&&)$выр_2$}
$выр_2$ →{**!**}$выр_3$
$выр_3$ →$выр_4${(<|>|<=|>=|=|**!**=)$выр_4$}
$выр_4$ →$выр_5${(|~|^|~^)$выр_5$}
$выр_5$ →$выр_6${(&|~&|≪|≫|⋙)$выр_6$}
$выр_6$ →{~}$выр_7$
$выр_7$ →$выр_8${(+|−)$выр_8$}
$выр_8$ →$выр_9${(*|/|%|/.)$выр_9$}
$выр_9$ →$выр_{10}$[(**|**.)$выр_9$]
$выр_{10}$ →[+|−|#]$выр_{11}$
$выр_{11}$→$выр_{12}$[##$выр_{12}$]
$выр_{12}$ →целое|вещественное|**true**|**false**|строковое|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**array**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**if** *выр* **then**{*тело_блока*}{**elif** *выр* **then**{*тело_блока*}}[**else**{*тело_блока*}]**endif**

цикла →`[`<:*ид*:>`]` `(`**for** *ид* : *выр*{*тело_блока*}`[`**else**{*тело_блока*}`]`|
for *ид* := *выр*,*выр*`[`,*выр*`]`{*тело_блока*}`[`**else**{*тело_блока*}`]`|
while *выр*{*тело_блока*}|**repeat**{*тело_блока*}**until** *выр*
spider{*случай*{,*случай*}`[`**else**{*тело_блока*}`]`}`)`
случай →**case** *выр*{*тело_блока*}
присваивания →*имя* := *выр*
перехода →**break** `[` *ид*`]`|**continue** `[`*ид*`]`|**return** `[`*выр*`]`
ввода →**read**(*списимён*)
вывода →**print**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**match** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}`[`**else**{*тело_блока*}`]`}
образец →`[`|:*имя_модуля*::`]`*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0*восьм_цифра*('?*восьм_цифра*)*
двоичное →0(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →0(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →0|1|2|3|4|5|6|7|8|9
восьм_цифра →0|1|2|3|4|5|6|7
двоичн_цифра →0|1
шестн_цифра →0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_часть*(*дробная_часть*)?(*символ_порядка*(+|-)?*порядок*)?
символ_порядка →E|e|S|s|D|d|Q|q
целая_часть →*десятичное*
дробная_часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.13. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **модуль** *ид* {`[`**использует** *списид*`]` `[`**предоставляет** *списид*`]` *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**перем** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**тип** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
массив`[`[*выр*]{,[*выр*]}`]` *простейшее_определение_типа*

элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**большое**|**маленькое**}(**цел**|**вещ**|**беззн**|**лог**|**симв**|**строка**)|**ничто**|**байт**|**цел8**|
цел16|**цел32**|**цел64**|**цел128**|**беззн8**|**беззн16**|**беззн32**|**беззн64**|**беззн128**|
вещ32|**вещ64**|**вещ128**|**лог8**|**лог16**|**лог32**|**лог64**|**симв8**|**симв16**|**симв32**|
строка8|**строка16**|**строка32**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**функция** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**значение** *списид*:*простейшее_определение_типа*|
ссылка *списид*:*простейшее_определение_типа*|
конст ссылка *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**перечисление** *ид*{*списид*}
структура →**структура** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**функция** *ид сигнатура_функции*{*тело_блока*}
прототип →**функция** *ид сигнатура_функции*;
константы →**конст** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →*выр*$_0$ [?*выр*$_0$:*выр*$_0$]
выр$_0$ →*выр*$_1$ {(|||**!**|||^^|**!**^^)*выр*$_1$}
выр$_1$ →*выр*$_2$ {(&&|**!**&&)*выр*$_2$}
выр$_2$ →{**!**}*выр*$_3$
выр$_3$ →*выр*$_4$ {(<|>|<=|>=|=|**!**=)*выр*$_4$}
выр$_4$ →*выр*$_5$ {(|~|^|~^)*выр*$_5$}
выр$_5$ →*выр*$_6$ {(&|~&|≪|≫|⋙)*выр*$_6$}
выр$_6$ →{~}*выр*$_7$
выр$_7$ →*выр*$_8$ {(+|−)*выр*$_8$}
выр$_8$ →*выр*$_9$ {(*|/|%|/.)*выр*$_9$}
выр$_9$ →*выр*$_{10}$ [(**|**.)*выр*$_9$]
выр$_{10}$ →[+|−|#]*выр*$_{11}$
выр$_{11}$→*выр*$_{12}$ [##*выр*$_{12}$]
выр$_{12}$ →*целое*|*вещественное*|**истина**|**ложь**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**массив**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**если** *выр* **то**{*тело_блока*}{**инес** *выр* **то**{*тело_блока*}}[**иначе**{*тело_блока*}]**всё**
цикла →[<:*ид*:>](**для** *ид* **из** *выр*{*тело_блока*}|**для** *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}|
пока *выр*{*тело_блока*}|**повторяй**{*тело_блока*}**покуда** *выр*
паук{*случай*{,*случай*}[**иначе**{*тело_блока*}]})
случай →**при** *выр*{*тело_блока*}
присваивания →*имя* (:=|+:=|-:=|*:=|/:=|/.:=|**:=|**.:=|%:=|&:=|~&:=|≪:=|≫:=||:=|~|:=|^:=|~^:=|
||:=|**!**||:=|^^:=|**!**^^:=|&&:=|**!**&&:=|⋙:=) *выр*

перехода →**выйди** *[***из** *ид]*|**продолжи** *[ид]*|**возврат** *[выр]*
ввода →**ввод**(*списимён*)
вывода →**вывод**(*списвыр*)
списимён →*имя{,имя}*
разбора →**разбор** *выр{образец->{тело_блока}{,образец->{тело_блока}}[**иначе**{тело_блока}]}*
образец →*[|:имя_модуля::]имя_типа::имя_части_типа{..}*

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра('?дес_цифра)**
восьмеричное →*0o восьм_цифра('?восьм_цифра)**
двоичное →*0(b|B)двоичн_цифра('?двоичн_цифра)**
шестнадцатеричное →*0(x|X)шестн_цифра('?шестн_цифра)**
дес_цифра →*0|1|2|3|4|5|6|7|8|9*
восьм_цифра →*0|1|2|3|4|5|6|7*
двоичн_цифра →*0|1*
шестн_цифра →*0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F*
вещественное →*целая_часть(дробная_часть)?(символ_порядка(+|-)?порядок)?*
символ_порядка →*E|e|S|s|D|d|Q|q*
целая_часть →*десятичное*
дробная_часть →*.десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.14. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **module** *ид {[**import** списид] [**export** списид] тело_модуля}*
списид →*ид{,ид}*
тело_модуля →*{переменные|типы|функция|прототип|константы}*
переменные →**var** *списид:простейшее_определение_типа{;списид:простейшее_определение_типа}*
типы →**type** *ид = определение_типа{;ид = определение_типа}*
определение_типа →*простейшее_определение_типа|составное_определение_типа*
простейшее_определение_типа →*элементарный_тип|*
array*[[выр]{,[выр]}]простейшее_определение_типа*
элементарный_тип →*{@}(встроенный_тип|ид)*
встроенный_тип →***{long|short}(int|float|uint|bool|char|string)|void|byte|int8_t|int16_t|***
int32_t|int64_t|int128_t|uint8_t|uint16_t|uint32_t|uint64_t|
uint128_t|float32_t|float64_t|float128_t|bool8_t|bool16_t|bool32_t|
bool64_t|char8_t|char16_t|char32_t|string8_t|string16_t|string32_t

составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**fn** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**val** *списид*:*простейшее_определение_типа*|
ref *списид*:*простейшее_определение_типа*|
const ref *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*
перечисление →**enum** *ид*{*списид*}
структура →**struct** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**fn** *ид сигнатура_функции*{*тело_блока*}
прототип →**fn** *ид сигнатура_функции*;
константы →**const** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$ [?$выр_0$:$выр_0$]
$выр_0$ →$выр_1${(|||!||^^|!^^)$выр_1$}
$выр_1$ →$выр_2${(&&|!&&)$выр_2$}
$выр_2$ →{!}$выр_3$
$выр_3$ →$выр_4${(<|>|<=|>=|=|!=)$выр_4$}
$выр_4$ →$выр_5${(|~|^|~^)$выр_5$}
$выр_5$ →$выр_6${(&|~&|≪|≫|⋙)$выр_6$}
$выр_6$ →{~}$выр_7$
$выр_7$ →$выр_8${(+|−)$выр_8$}
$выр_8$ →$выр_9${(*|/|%|/.)$выр_9$}
$выр_9$ →$выр_{10}$ [(**|**.)$выр_9$]
$выр_{10}$ →[+|−|#]$выр_{11}$
$выр_{11}$→$выр_{12}$ [##$выр_{12}$]
$выр_{12}$ →*целое*|*вещественное*|**true**|**false**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**array**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**if** *выр* **then**{*тело_блока*}{**elif** *выр* **then**{*тело_блока*}}[**else**{*тело_блока*}]**endif**
цикла →[<:*ид*:>](**for** *ид* : *выр*{*тело_блока*}|**for** *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}|
while *выр*{*тело_блока*}|**repeat**{*тело_блока*}**until** *выр*
spider{*случай*{,*случай*}[**else**{*тело_блока*}]})
случай →**case** *выр*{*тело_блока*}
присваивания →*имя* (:=|+:=|-:=|*:=|/:=|/.:=|**:=|**.:=|%:=|&:=|~&:=|≪:=|≫:=||:=|~|:=|^:=|~^:=|
||:=|!||:=|^^:=|!^^:=|&&:=|!&&:=|⋙:=) *выр*
перехода →**break** [*ид*]|**continue** [*ид*]|**return** [*выр*]
ввода →**read**(*списимён*)
вывода →**print**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**match** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**else**{*тело_блока*}]}

образец →*[|:имя_модуля::]имя_типа::имя_части_типа{..}*

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное|восьмеричное|двоичное|шестнадцатеричное*
десятичное →*дес_цифра('?дес_цифра)**
восьмеричное →*0oвосьм_цифра('?восьм_цифра)**
двоичное →*0(b|B)двоичн_цифра('?двоичн_цифра)**
шестнадцатеричное →*0(x|X)шестн_цифра('?шестн_цифра)**
дес_цифра →*0|1|2|3|4|5|6|7|8|9*
восьм_цифра →*0|1|2|3|4|5|6|7*
двоичн_цифра →*0|1*
шестн_цифра →*0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F*
вещественное →*целая_часть(дробная_часть)?(символ_порядка(+|-)?порядок)?*
символ_порядка →*E|e|S|s|D|d|Q|q*
целая_часть →*десятичное*
дробная_часть →*.десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.15. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **модуль** *ид* {*[***использует** *списид]* *[***предоставляет** *списид]* *тело_модуля*}
списид →*ид{,ид}*
тело_модуля →*{переменные|типы|функция|прототип|константы}*
переменные →**перем** *списид:простейшее_определение_типа{;списид:простейшее_определение_типа}*
типы →**тип** *ид* = *определение_типа{;ид* = *определение_типа}*
определение_типа →*простейшее_определение_типа|составное_определение_типа*
простейшее_определение_типа →*элементарный_тип|*
массив*[[выр]{,[выр]}]простейшее_определение_типа*
элементарный_тип →*{@}(встроенный_тип|ид)*
встроенный_тип →*{***большое**|**маленькое***}(***цел**|**вещ**|**беззн**|**лог**|**симв**|**строка***)*|***ничто**|**байт**|**цел8**|*
***цел16**|**цел32**|**цел64**|**цел128**|**беззн8**|**беззн16**|**беззн32**|**беззн64**|**беззн128**|*
***вещ32**|**вещ64**|**вещ128**|**лог8**|**лог16**|**лог32**|**лог64**|**симв8**|**симв16**|**симв32**|*
строка8**|**строка16**|**строка32
составное_определение_типа →*определение_кортежа|определение_указателя_на_функцию|*
определение_алгебраического_типа
определение_кортежа →*(:элементарный_тип{,элементарный_тип}:)*
определение_указателя_на_функцию →**функция** *сигнатура_функции*
сигнатура_функции →*(спис_форм_парам):элементарный_тип*
спис_форм_парам →*форм_парам{,форм_парам}*

форм_ парам →**значение** *списид:простейшее_ определение_ типа*|
ссылка *списид:простейшее_ определение_ типа*|
конст ссылка *списид:простейшее_ определение_ типа*
определение_ алгебраического_ типа →*часть_ алгебр_ типа*{.|.*часть_ алгебр_ типа*}
часть_ алгебр_ типа →*перечисление*|*структура*
перечисление →**перечисление** *ид*{*списид*}
структура →**структура** *ид*{*спис_ групп_ полей*}
спис_ групп_ полей →*группа_ полей*{;*группа_ полей*}
группа_ полей →*списид:простейшее_ определение_ типа*
функция →**функция** *ид сигнатура_ функции*{*тело_ блока*}
прототип →**функция** *ид сигнатура_ функции*;
константы →**конст** *определение_ константы*{,*определение_ константы*}
определение_ константы →*ид:простейшее_ определение_ типа* = *выр*
тело_ блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$ [?$выр_0$:$выр_0$]
$выр_0$ →$выр_1${(||| **!**||| |^^| **!**^^)$выр_1$}
$выр_1$ →$выр_2${(&& **!**&&)$выр_2$}
$выр_2$ →{**!**}$выр_3$
$выр_3$ →$выр_4${(<|>|<=|>=|= **!**=)$выр_4$}
$выр_4$ →$выр_5${(|~|^|~^)$выр_5$}
$выр_5$ →$выр_6${(&|~&|≪|≫|⋙)$выр_6$}
$выр_6$ →{~}$выр_7$
$выр_7$ →$выр_8${(+|−)$выр_8$}
$выр_8$ →$выр_9${(*|/|%|/.)$выр_9$}
$выр_9$ →$выр_{10}$ [(**|**.)$выр_9$]
$выр_{10}$ →[+|−|#]$выр_{11}$
$выр_{11}$→$выр_{12}$ [##$выр_{12}$]
$выр_{12}$ →целое|вещественное|**истина**|**ложь**|строковое|(*выр*)|*имя*|*сост_ знач*
сост_ знач →*знач_ структ*|*знач_ массива*
знач_ структ →[|:*имя_ модуля*::]*имя_ типа*::*имя_ части_ типа*{*значение_ поля*{,*значение_ поля*}}
значение_ поля →*ид* <- *выр*
знач_ массива →**массив**[:*выр*,*элементарный_ тип*:]{*списвыр*}
имя →[|:*имя_ модуля*::] [*имя_ типа*::*имя_ части_ типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_ модуля →*ид*
имя_ типа →*ид*
имя_ части_ типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**если** *выр* **то**{*тело_ блока*}{**инес** *выр* **то**{*тело_ блока*}}[**иначе**{*тело_ блока*}]**всё**
цикла →[<:*ид*:>](**для** *ид* **из** *выр*{*тело_ блока*}[**завершение**{*тело_ блока*}]|
для *ид* := *выр*,*выр*[,*выр*]{*тело_ блока*}[**завершение**{*тело_ блока*}]|
пока *выр*{*тело_ блока*}|**повторяй**{*тело_ блока*}**покуда** *выр*
паук{*случай*{,*случай*}[**иначе**{*тело_ блока*}]})
случай →**при** *выр*{*тело_ блока*}
присваивания →*имя* (:=|+:=|-:=|*:=|/:=|/.:=|**:=|**.:=|%:=|&:=|~&:=|≪:=|≫:=||:=|~|:=|^:=|~^:=|
||:=|**!**||:=|^^:=|**!**^^:=|&&:=|**!**&&:=|⋙:=) *выр*
перехода →**выйди** [**из** *ид*]|**продолжи** [*ид*]|**возврат** [*выр*]
ввода →**ввод**(*списимён*)
вывода →**вывод**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**разбор** *выр*{*образец*->{*тело_ блока*}{,*образец*->{*тело_ блока*}}[**иначе**{*тело_ блока*}]}
образец →[|:*имя_ модуля*::]*имя_ типа*::*имя_ части_ типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_ цифра*('?*дес_ цифра*)*

восьмеричное →0o*восьм_ цифра*('?*восьм_ цифра*)*
двоичное →0(b|B)*двоичн_ цифра*('?*двоичн_ цифра*)*
шестнадцатеричное →0(x|X)*шестн_ цифра*('?*шестн_ цифра*)*
дес_ цифра →0|1|2|3|4|5|6|7|8|9
восьм_ цифра →0|1|2|3|4|5|6|7
двоичн_ цифра →0|1
шестн_ цифра →0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
вещественное →*целая_ часть*(*дробная_ часть*)?(*символ_ порядка*(+|-)?*порядок*)?
символ_ порядка →E|e|S|s|D|d|Q|q
целая_ часть →*десятичное*
дробная_ часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*

Задача 1.16. *С помощью генератора лексических анализаторов Мяука для приводимого ниже языка напишите и протестируйте лексический анализатор.*

программа → **module** *ид* {[**import** *списид*] [**export** *списид*] *тело_модуля*}
списид →*ид*{,*ид*}
тело_модуля →{*переменные*|*типы*|*функция*|*прототип*|*константы*}
переменные →**var** *списид*:*простейшее_определение_типа*{;*списид*:*простейшее_определение_типа*}
типы →**type** *ид* = *определение_типа*{;*ид* = *определение_типа*}
определение_типа →*простейшее_определение_типа*|*составное_определение_типа*
простейшее_определение_типа →*элементарный_тип*|
array[[*выр*]{,[*выр*]}]*простейшее_определение_типа*
элементарный_тип →{@}(*встроенный_тип*|*ид*)
встроенный_тип →{**long**|**short**}(**int**|**float**|**uint**|**bool**|**char**|**string**)|**void**|**byte**|**int8_t**|**int16_t**|
int32_t|**int64_t**|**int128_t**|**uint8_t**|**uint16_t**|**uint32_t**|**uint64_t**|
uint128_t|**float32_t**|**float64_t**|**float128_t**|**bool8_t**|**bool16_t**|**bool32_t**|
bool64_t|**char8_t**|**char16_t**|**char32_t**|**string8_t**|**string16_t**|**string32_t**
составное_определение_типа →*определение_кортежа*|*определение_указателя_на_функцию*|
определение_алгебраического_типа
определение_кортежа →(:*элементарный_тип*{,*элементарный_тип*}:)
определение_указателя_на_функцию →**fn** *сигнатура_функции*
сигнатура_функции →(*спис_форм_парам*):*элементарный_тип*
спис_форм_парам →*форм_парам*{,*форм_парам*}
форм_парам →**val** *списид*:*простейшее_определение_типа*|
ref *списид*:*простейшее_определение_типа*|
const ref *списид*:*простейшее_определение_типа*
определение_алгебраического_типа →*часть_алгебр_типа*{.|.*часть_алгебр_типа*}
часть_алгебр_типа →*перечисление*|*структура*

перечисление →**enum** *ид*{*списид*}
структура →**struct** *ид*{*спис_групп_полей*}
спис_групп_полей →*группа_полей*{;*группа_полей*}
группа_полей →*списид*:*простейшее_определение_типа*
функция →**fn** *ид сигнатура_функции*{*тело_блока*}
прототип →**fn** *ид сигнатура_функции*;
константы →**const** *определение_константы*{,*определение_константы*}
определение_константы →*ид*:*простейшее_определение_типа* = *выр*
тело_блока →{*переменные*|*типы*|*функция*|*прототип*|*константы*|*операторы*}
выр →$выр_0$ [?$выр_0$:$выр_0$]
$выр_0$ →$выр_1${(||/|!||/|^^/|!^^)$выр_1$}
$выр_1$ →$выр_2${(&&/|!&&)$выр_2$}
$выр_2$ →{!}$выр_3$
$выр_3$ →$выр_4${(</|>/|<=/|>=/|=/|!=)$выр_4$}
$выр_4$ →$выр_5${(|/|~|/|^/|~^)$выр_5$}
$выр_5$ →$выр_6${(&/|~&/|≪/|≫/|⋙)$выр_6$}
$выр_6$ →{~}$выр_7$
$выр_7$ →$выр_8${(+/|−)$выр_8$}
$выр_8$ →$выр_9${(*/|/|%/|/.)$выр_9$}
$выр_9$ →$выр_{10}$[(**/|**.)$выр_9$]
$выр_{10}$ →[+/|−/|#]$выр_{11}$
$выр_{11}$→$выр_{12}$[##$выр_{12}$]
$выр_{12}$ →*целое*|*вещественное*|**true**|**false**|*строковое*|(*выр*)|*имя*|*сост_знач*
сост_знач →*знач_структ*|*знач_массива*
знач_структ →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{*значение_поля*{,*значение_поля*}}
значение_поля →*ид* <- *выр*
знач_массива →**array**[:*выр*,*элементарный_тип*:]{*списвыр*}
имя →[|:*имя_модуля*::][*имя_типа*::*имя_части_типа*::]*ид*{([*списвыр*])|[*списвыр*]|@|.*ид*}
списвыр →*выр*{,*выр*}
имя_модуля →*ид*
имя_типа →*ид*
имя_части_типа →*ид*
операторы →[*оператор*]{;[*оператор*]}
оператор →*цикла*|*присваивания*|*перехода*|*ввода*|*вывода*|*условный*|*разбора*
условный →**if** *выр* **then**{*тело_блока*}{**elif** *выр* **then**{*тело_блока*}}[**else**{*тело_блока*}]**endif**
цикла →[<:*ид*:>](**for** *ид* : *выр*{*тело_блока*}[**else**{*тело_блока*}]|
for *ид* := *выр*,*выр*[,*выр*]{*тело_блока*}[**else**{*тело_блока*}]|
while *выр*{*тело_блока*}|**repeat**{*тело_блока*}**until** *выр*
spider{*случай*{,*случай*}[**else**{*тело_блока*}]})
случай →**case** *выр*{*тело_блока*}
присваивания →*имя* (:=/|+:=/|-:=/|*:=/|/:=/|/.:=/|**:=/|**.:=/|%:=/|&:=/|~&:=/|≪:=/|≫:=/||:=/|~|:=/|^:=/|~^:=/
||:=/|!||:=/|^^:=/|!^^:=/|&&:=/|!&&:=/|⋙:=) *выр*
перехода →**break** [*ид*]|**continue** [*ид*]|**return** [*выр*]
ввода →**read**(*списимён*)
вывода →**print**(*списвыр*)
списимён →*имя*{,*имя*}
разбора →**match** *выр*{*образец*->{*тело_блока*}{,*образец*->{*тело_блока*}}[**else**{*тело_блока*}]}
образец →[|:*имя_модуля*::]*имя_типа*::*имя_части_типа*{..}

Здесь целочисленные и вещественные литералы выглядят так (записаны соответствующие регулярные определения):

целое →*десятичное*|*восьмеричное*|*двоичное*|*шестнадцатеричное*
десятичное →*дес_цифра*('?*дес_цифра*)*
восьмеричное →0*восьм_цифра*('?*восьм_цифра*)*
двоичное →0(b|B)*двоичн_цифра*('?*двоичн_цифра*)*
шестнадцатеричное →0(x|X)*шестн_цифра*('?*шестн_цифра*)*
дес_цифра →0|1|2|3|4|5|6|7|8|9
восьм_цифра →0|1|2|3|4|5|6|7

двоичн_ цифра →*0|1*
шестн_ цифра →*0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F*
вещественное →*целая_ часть*(*дробная_ часть*)?(*символ_ порядка*(+|-)?*порядок*)?
символ_ порядка →*E|e|S|s|D|d|Q|q*
целая_ часть →*десятичное*
дробная_ часть →.*десятичное*
порядок →*десятичное*

Простой строковый литерал — это любая (в том числе и пустая) последовательность символов, заключённая в одинарные кавычки. Если в строковом литерале нужно указать саму одинарную кавычку, то эту кавычку нужно удвоить. Символьный литерал представляет собой либо заключённый в одинарные кавычки символ, либо символ, заданный своим кодом. Последнее представляет собой знак $, после которого идёт целое число, являющееся кодом символа. Число это может быть двоичным, восьмеричным, шестнадцатеричным, или десятичным Внутреннее представление строковых и символьных литералов — в кодировке Unicode, а именно, в её четырёхбайтовом варианте. Текст программы — только в кодировке UTF-8. Строковый литерал может представлять собой как простой строковый литерал, так и чередование последовательности заданнных своими кодами символов и простых строковых литералов.

Если перед строковым или символьным литералом не указано ничего, то считается, что это строковый литерал в UTF-32. То же происходит, если перед строковым литералом указано u32. Если указано u16, то строковый литерал — в кодировке UTF-16. Наконец, если указано u8 — то в кодировке UTF-8.

Идентификатор представляет собой последовательность букв, цифр, и знаков подчёркивания, и может начинаться либо с буквы, либо со знака подчёркивания. Под буквой понимается либо русская буква, либо латинская буква.

При реализации сканера для хранения целочисленного значения лексемы следует использовать значение типа `unsigned __int128`*, а для хранения вещественного — значение типа* `__float128`*.*