БЕЗОПАСНОСТЬ ЦИФРОВЫХ ТЕХНОЛОГИЙ. – 2021. – № 2 (101). – 9–19

АВТОМАТИЗАЦИЯ И УПРАВЛЕНИЕ ТЕХНОЛОГИЧЕСКИМИ ПРОЦЕССАМИ И ПРОИЗВОДСТВАМИ

УДК 004.43 DOI: 10.17212/2782-2230-2021-2-9-19

КОМПИЛЯТОР С ЯЗЫКА ПРОГРАММИРОВАНИЯ EL: УСОВЕРШЕНСТВОВАНИЕ И РАЗВИТИЕ*

Н.А. ЗЕЛЕНЧУК[1], Е.Д. ПРИСТАВКА[2], А.А. МАЛЯВКО[3]

[1] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, лаборант кафедры вычислительной техники. E-mail: nikita-zelenchuk@yandex.ru*
[2] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, лаборант кафедры вычислительной техники. E-mail: pristavka_katya@mail.ru*
[3] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, кандидат технических наук, доцент кафедры вычислительной техники. E-mail: a.maliavko@corp.nstu.ru*

Реализация нового многопарадигменного (функционально-императивного) языка программирования El, разработанного на кафедре вычислительной техники Новосибирского государственного технического университета, в виде компилятора связана с необходимостью поиска способов решения ряда сложных проблем. Текущая версия компилятора лишь частично реализует функциональные возможности языка и генерирует далеко не оптимальные исполняемые коды. В настоящей работе рассматриваются задачи эффективной компиляции El-программы с учетом необходимости реализации новых высокоуровневых структур данных (двусторонние списки, векторы со специальными формами доступа и ряд других) и управляющих структур языка, которые позволяют единообразно определять циклические и ветвящиеся вычислительные процессы, а также заложенных в языке механизмов для явного управления изменчивостью переменных. Кратко рассмотрены задачи совершенствования и развития компилятора, организованного по классической мультиплатформенной схеме, в которой front-end (лексический, синтаксический и семантический анализаторы) преобразует транслируемую программу в псевдокод единого формата, а в качестве back-end, превращающего псевдокод в исполняемый код для разных платформ, используется эффективная инфраструктура построения компиляторов LLVM. Выполнение всех возможных операций над элементами высокоуровневых структур данных (списки, кортежи, векторы), а также над числами произвольной точности вынесено в библиотеку поддержки времени выполнения

* Статья получена 14 апреля 2021 г.

и, соответственно, может быть глубоко оптимизировано. Для этой структуры сформулированы намеченные пути решения задачи разработки и улучшения компилятора путем глубокого реформирования и оптимизации цепочки преобразований транслируемой программы, реализуемой front-end. На начальном этапе планируется реализовать новый компилятор для двух платформ: Linux и Windows.

Ключевые слова: язык программирования, программа, компилятор, стандартная библиотека, высокоуровневые структуры данных, вектор, список, кортеж, цепочка преобразований, постфиксная форма записи, псевдокод, LLVM

ВВЕДЕНИЕ

Языки программирования принято условно делить на две большие группы, соответствующие императивной и функциональной парадигмам [1]. Особенностью первой является задание в программе точной последовательности действий, которую нужно выполнить для получения результата. Для второй характерно описание зависимостей между данными в виде функций, как правило, не хранящих внутренние состояния и не выполняющих изменения однажды вычисленных значений объектов программы. Достижение результата вычислений обеспечивается путем удовлетворения определенных зависимостей между исходными и промежуточными данными. Нередко эти парадигмы противопоставляются друг другу, однако в настоящее время имеется множество примеров их взаимопроникновения, т. е. реализации механизмов, присущих одной парадигме в языке, преимущественно относящемся к другой. Эту тенденцию принято называть мульипарадигменностью.

Разрабатываемый на кафедре вычислительной техники НГТУ функционально-императивный язык El [2] представляет новый подход к мультипарадигменности в сочетании с высокоуровневостью структур данных. Он во многом походит на функциональный язык Erlang [3], однако отличается от него рядом особенностей: возможностью управления изменяемостью переменных, расширенными высокоуровневыми структурами данных, возможностью явно задавать последовательно выполняемые вычисления в тех случаях, когда это необходимо.

Требования к современному программному обеспечению постоянно растут и требуют от программистов долгой и кропотливой работы над ним. Применение высокоуровневых структур данных языка El позволит разработчику не тратить время на написание собственного алгоритма формирования и обработки данных нужной структуры, а воспользоваться готовым решением без потерь производительности.

Новые структуры данных (двусторонние списки, специальные формы индексации векторов и др.) и новые управляющие конструкции языка El (в язы-

ке имеется единственный управляющий оператор by, позволяющий единообразно программировать циклы, переключатели и условные операторы) приводят к необходимости поиска и реализации в компиляторе новых эффективных алгоритмов поддержки императивно-функциональной парадигмы.

Компиляция El-программы – это цепочка преобразований представления программы [4] из исходного текста в объектный код для LLVM-компилятора. В процессе трансляции программа преобразуется вначале в последовательность токенов, затем в постфиксную запись этой последовательности, затем в псевдокод и, наконец, в объектный код.

1. ПОСТАНОВКА ЗАДАЧИ

Необходимо выполнить доработку существующей версии компилятора для реализации полного набора операций с высокоуровневыми структурами данных (списки, кортежи, векторы) в стандартной библиотеке El и оптимизации цепочки преобразований внутреннего представления El-программы в компиляторе.

Процессы развития и усовершенствования можно разбить на несколько подзадач, а именно:

1) переработка и оптимизация цепочки преобразований: исходный код – формирование постфиксной формы записи (далее – ПФЗ) – формирование псевдокода – формирование кода Low Level Virtual Machine intermediate representation (далее – LLVM IR);

2) структурирование и систематизация механизма преобразования;

3) усовершенствование и отладка механизма тестирования программ на языке El;

4) доработка расширения грамматики до полной для высокоуровневых структур данных – списков, кортежей, векторов – и формирование для них ПФЗ.

2. ВЫБОР СРЕДСТВ РЕАЛИЗАЦИИ

Для разработки компилятора с языка El используются:

1) языки программирования C, C++ [5];

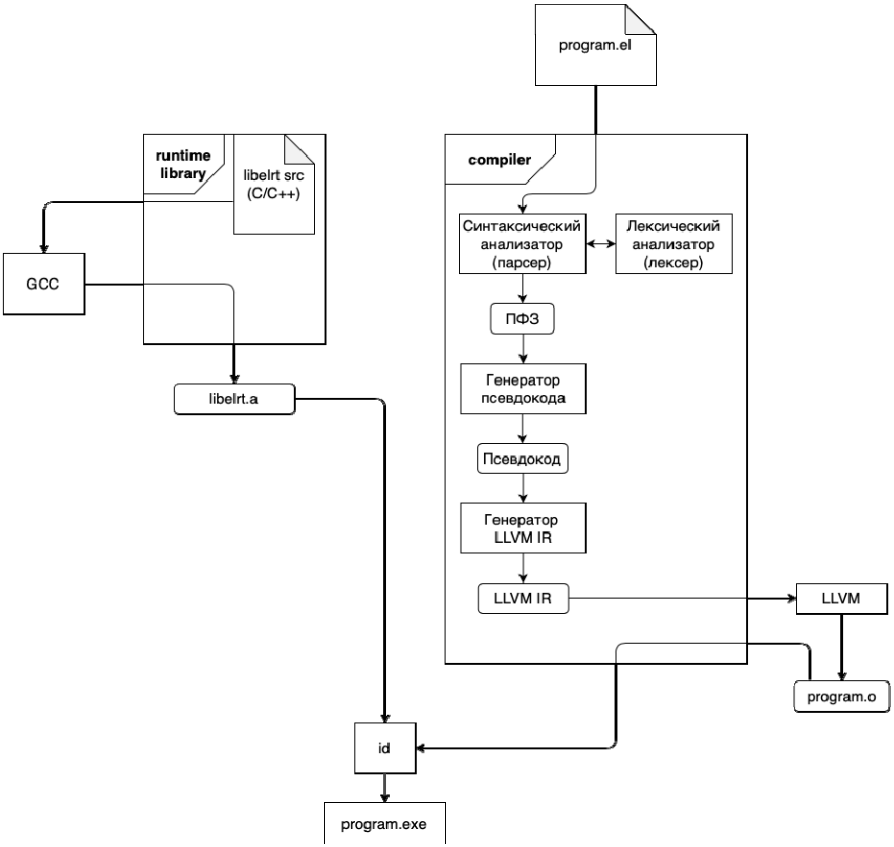
2) пакет программ автоматизации разработки трансляторов «ВебТрансЛаб», используемый в учебном процессе на кафедре вычислительной техники НГТУ [6];

3) кросс-платформенная IDE для C и C++ CLion [7];

4) инфраструктура проектирования компиляторов LLVM [8].

3. КОМПИЛЯТОР С ЯЗЫКА EL

На рисунке представлена структура компилятора языка El.



Структура компилятора языка El

The structure of the El language compiler

Работа компилятора организуется функцией main, которая обрабатывает параметры командной строки запуска, инициализирует необходимые классы и структуры данных, вызывает парсер (синтаксический анализатор, использующий лексический анализатор для получения очередных токенов по мере движения по правилам грамматики). Парсер формирует постфиксную форму

представления транслируемой программы во внутренних структурах данных. Если парсер завершился без обнаружения ошибок, то вызывается формирователь псевдокода, а затем – генератор ассемблерного кода для LLVM-компилятора. В случае, если все эти этапы также завершились без ошибок, выполняется запуск LLVM-компилятора, формирующего объектный модуль для целевой платформы, и, наконец, компилятора g++ [9] для сборки построенного объектного модуля с библиотекой времени выполнения. Если при выполнении какого-либо этапа были обнаружены критические ошибки, то последующие этапы компиляции не запускаются, вместо этого обеспечивается формирование диагностических сообщений об ошибках (при условии, что эти сообщения не формировались такими самостоятельными компонентами, как LLVM или g++ компиляторы).

3.1. ПОСЛЕДОВАТЕЛЬНОСТЬ ПРЕОБРАЗОВАНИЙ ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ ТРАНСЛИРУЕМОЙ ПРОГРАММЫ

Компиляция программы является длительным и сложным процессом. Этот процесс включает в себя сразу несколько этапов, результатом каждого из них является некоторое промежуточное представление транслируемой программы, с которым далее работает очередной этап компилятора [10]. Кратко опишем этапы следующей цепочки преобразований: формирование ПФЗ – формирование псевдокода – формирование кода LLVM IR.

Первым этапом преобразования исходного кода программы на языке El является обратная польская запись (далее – ПФЗ). Этот этап выполняется в процессе синтаксического разбора. ПФЗ – это такое промежуточное представление программы, в котором последовательность появления знаков любых операций совпадает с требуемой последовательностью их выполнения.

Исходная программа на языке El представляется последовательностью токенов, как и ПФЗ. Однако в ПФЗ отсутствуют скобки, которые меняют порядок выполнения операций, а также токены, которые требуются для разметки (разграничения) элементов программы по правилам языка. В ПФЗ токен может являться либо именем операнда, либо знаком операции.

Следующим за ПФЗ промежуточным представлением является псевдокод. Псевдокод представляет собой последовательность инструкций. Компилятор языка El использует в качестве псевдокода трехадресный код, в котором каждая инструкция содержит код операции, имена операндов и имя результата.

Псевдокод является промежуточной формой программы, которую строит транслятор для фиксации той последовательности операций, которую должен будет выполнять компьютер. Псевдокод в компиляторе El, как и постфиксная запись, представлен STL-контейнером vector, хранящим экземпляры PseudoCommand. Для преобразования ПФЗ в псевдокод используется следу-

ющий алгоритм: сперва создается пустой стек операндов, а затем из постфиксной записи извлекаются токены по одному, и если текущий токен является именем операнда, то он помещается в стек. Если текущей токен – знак операции, то из стека извлекаются операнды в количестве, равном арности операции. Если операция вырабатывает какой-либо результат, для него формируется имя и помещается в стек операндов для обработки следующими операциями. Совокупность знака операции, операндов и имени вырабатываемого результата образует одну инструкцию псевдокода. Генерация псевдокода происходит отдельными блочными структурами, так как исходный код программы значимо отличается от генерируемого объектного кода из-за определенных особенностей языка El. Сформированные блоки в дальнейшем достаточно просто объединить в нужной последовательности.

Заключительным этапом является генерация объектного кода. С целью поддержки множества различных целевых платформ компилятор языка программирования El использует средство поддержки разработки компиляторов LLVM. Полученный на предыдущих этапах трансляции псевдокод в последующем подвергается преобразованию в исходный код на языке ассемблера LLVM, после чего компилятор LLVM создает объектный модуль для целевой платформы, затем запускает компилятор LLVM. Он представляет собой последовательность инструкций для некой виртуальной машины и построен таким образом, чтобы из каждой инструкции можно было сформировать четко определенную последовательность инструкций для виртуальной машины LLVM [11].

При генерации кода LLVM первым делом в начало файла помещается блок деклараций. Сгенерированный LLVM-код содержит в себе вызовы библиотечных функций (см. следующий подраздел), поэтому компилятор LLVM должен знать их имена, сигнатуры и типы возвращаемого значения, также блок декларации содержит в себе объявление всех функций из библиотеки языка. После этого шага генерируется код функций модуля. Каждая скомпилированная функция принимает два аргумента:

1) указатель на переменную – кортеж, содержащий значения фактических аргументов;

2) указатель на переменную, в которую должно быть помещено возвращаемое значение.

3.2. ВЫСОКОУРОВНЕВЫЕ СТРУКТУРЫ ДАННЫХ ЯЗЫКА EL И ОПЕРАЦИИ НАД НИМИ

В языке El runtime-библиотека состоит из набора условно независимых модулей кода. В каждом модуле содержится код, предназначенный для работы с каким-либо типом данных языка El. Общие для всех типов данных опе-

рации определены в модуле el_types (например, операция «+»). В функциях, определенных в этом модуле, производится обработка ошибок, подготовка к вызову, выбор и вызов реализации операции для соответствующего переданным аргументам типа данных. В программе на языке LLVM IR в основном присутствуют объявления обобщенных операций из модуля el_types.

Доработка runtime-библиотеки описывает следующие высокоуровневые структуры данных [12] и операции над ними.

1. Список

Списком традиционно называют структуру данных, содержащую ссылки на предыдущий и/или следующий элемент [13]. В языке El списки двусвязные, т. е. допускают доступ к элементам как с головы, так и с хвоста. Этот тип данных может содержать одновременно несколько элементов любых типов (списки, кортежи, векторы и т. д.). Над списками можно выполнять такие операции, как слияние и удаление всех элементов.

2. Вектор

Вектор – это структура данных с фиксированным числом элементов одного и того же типа [14]. Эту структуру данных часто называют одномерным массивом. В языке El к ней могут применяться все знаки операций, также можно получить индексы элементов и размерность вектора.

3. Кортеж

Кортежем называют неизменяемый аналог списка [15]. Он защищает данные от непреднамеренных изменений. В языке El кортеж, как правило, представляет собой совокупность разнородных элементов данных. Они записываются в программе в виде последовательности элементов, заключенных в круглые скобки. Кортежи в некотором смысле аналогичны структурам в C/C++ [16] с тем отличием, что их элементы не имеют имен. Однако использование записей и соответствующих объявлений переменных позволяет обращаться к элементам кортежей по именам. Для кортежей существует ряд методов, позволяющих узнавать количество элементов, извлекать и заменять элементы данных по номеру.

ЗАКЛЮЧЕНИЕ

В работе описан подход к доработке и усовершенствованию компилятора для языка El, сочетающего в себе особенности двух парадигм программирования – функциональной и императивной. Кратко описана необходимая последовательность преобразований внутреннего представления транслируемой программы и установлены требования к операциям над высокоуровневыми структурами данных.

СПИСОК ЛИТЕРАТУРЫ

1. *Себеста Р.В.* Основные концепции языков программирования: пер. с англ. – 5-е изд. – М.: Вильямс, 2001. – 672 с.

2. *Малявко А.А.* Функционально-императивный язык программирования El // Научный вестник НГТУ. – 2018. – № 1 (70). – С. 117–136. – DOI: 10.17212/1814-1196-2018-1-117-136.

3. *Чезарини Ф., Томпсон С.* Программирование в Erlang. – М.: ДМК Пресс, 2012. – 488 с.

4. Компиляторы: принципы, технологии и инструментарий: пер. с англ. / А.В. Ахо, М.С. Лам, Р. Сети, Д.Д. Ульман. – 2-е изд. – М.: Вильямс, 2008. – 1184 с.

5. C++ Programming Language: website. – URL: https://devdocs.io/cpp/ (accessed: 28.05.2021).

6. *Малявко А.А.* Использование веб-приложений и веб-технологий при разработке учебного программного обеспечения для изучения методов трансляции // Современное образование: технические университеты в модернизации экономики России: материалы Международной научно-методической конференции, 27–28 января 2011 года. – Томск: ТУСУР, 2011. – С. 45–46.

7. CLion: web-сайт. – URL: https://www.jetbrains.com/ru-ru/clion/ (дата обращения: 28.05.2021).

8. *Лопес Б.К., Аулер Р.* LLVM: инфраструктура для разработки компиляторов. – М.: ДМК Пресс, 2015. – 342 с.

9. GCC online documentation: web-сайт. – URL: https://gcc.gnu.org/onlinedocs/ (accessed: 28.05.2021).

10. *Хантер Р.* Проектирование и конструирование компиляторов: пер. с англ. / предисл. В.М. Савинкова. – М.: Финансы и статистика, 1984. – 232 с.

11. LLVM Language Reference Manual: web-сайт. – URL: https://llvm.org/docs/LangRef.html (дата обращения 05.03.2021).

12. *Ахо А.В., Хопкрофт Д., Ульман Д.Д.* Структуры данных и алгоритмы. – М.: Вильямс, 2000. – 384 с.

13. *Володин А.М., Дрождин В.В.* Разнообразие структур данных // Известия ПГПУ им. В.Г. Белинского. – 2009. – № 13 (17). – С. 79–83.

14. Структура данных – вектор: web-сайт. – URL: https://russianblogs.com/article/11141566726/ (дата обращения: 28.05.2021).

15. Кортежи. – URL: https://lawbooks.news/programmirovanie_964/kortezhi-68448.html (дата обращения: 28.05.2021).

16. Структуры данных. – URL: http://www.realcoding.net/articles/struktury-dannykh.html (дата обращения: 28.05.2021).

Зеленчук Никита Андреевич, лаборант кафедры вычислительной техники Новосибирского государственного технического университета. Основное направление научных исследований – языки программирования и теория трансляции. E-mail: nikitazelenchuk@yandex.ru

Приставка Екатерина Дмитриевна, лаборант кафедры вычислительной техники Новосибирского государственного технического университета. Основное направление научных исследований – языки программирования и теория трансляции. E-mail: pristavka_katya@mail.ru

Малявко Александр Антонович, кандидат технических наук, доцент кафедры вычислительной техники Новосибирского государственного технического университета. Основное направление научных исследований – языки программирования и теория трансляции, нейронные сети, параллельные вычисления. Имеет более 60 публикаций. E-mail: a.maliavko@corp.nstu.ru

DOI: 10.17212/2782-2230-2021-2-9-19

Compiler from El programing language: improvement and development*

N.A. Zelenchuk[1], E.D. Pristavka[2], A.A. Maliavko[3]

[1] *Novosibirsk State Technical University, 20 Karl Marx Prospekt, Novosibirsk, 630073, Russian Federation, laboratory assistant of Faculty of Automation and Computer Science, of the department of computer engineering. E-mail: nikitazelenchuk@yandex.ru*

[2] *Novosibirsk State Technical University, 20 Karl Marx Prospekt, Novosibirsk, 630073, Russian Federation, laboratory assistant of Faculty of Automation and Computer Science, of the department of computer engineering. E-mail: pristavka_katya@mail.ru*

[3] *Novosibirsk State Technical University, 20 Karl Marx Prospekt, Novosibirsk, 630073, Russian Federation, candidate of technical sciences, associate professor of the computer engineering department. E-mail: a.Maliavko@corp.nstu.ru*

The implementation of the new multi-paradigm (functionally- imperative) programming language El, developed at the Department of Computer Science of the Novosibirsk State Technical University, in the form of a compiler is associated with the need to find ways to solve a number of complex problems. The current version of the compiler does implement only partially functionality of the language and generates far from optimal executable codes. In this paper, we consider the problem of an efficient compilation of an El-program, taking into account the need to implement new high-level data structures (two-sided lists, vectors with special forms of access, and a number of others) and control structures of the language, which make it possible to uniformly define cyclic and branching computational processes, as well as those laid down in the language a mechanism for explicitly controlling the mutability of variables. The tasks of improving and developing a compiler organized according to the classical multi-platform scheme are briefly considered, in which the front-end (lexical, syntactic, and

* *Received 14 April 2021.*

semantic analyzers) converts the program to be translated into pseudocode of a single format, and used efficient infrastructure for building LLVM compilers as a back-end that turns pseudocode into executable code for different platforms. Execution of all possible operations on elements of high-level data structures (lists, tuples, vectors), as well as on arbitrary-precision numbers, has been moved to the runtime support library and, accordingly, can be deeply optimized. For this structure, the outlined ways of solving the problem of developing and improving the compiler by deep reforming and optimization of the chain of transformations of the translated program implemented by the front-end are formulated. At the initial stage, it is planned to implement a new compiler for two platforms: Linux and Windows.

Keywords: programming language, program, compiler, standard library, high-level data structures, vector, list, tuple, chain of conversions, postfix notation, pseudocode, LLVM

REFERENCES

1. Sebesta R.W. *Concepts of programming languages*. Boston, Addison Wesley, 2002 (Russ. ed.: Sebesta R.V. *Osnovnye kontseptsii yazykov programmirovaniya*. Moscow, Vil'yams Publ., 2001. 672 p.).

2. Malyavko A.A. Funktsional'no-imperativnyi yazyk programmirovaniya El [The El functional-imperative programming language]. *Nauchnyi vestnik Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta = Science bulletin of the Novosibirsk state technical university*, 2018, no. 1 (70), pp. 117–136. DOI: 10.17212/1814-1196-2018-1-117-136.

3. Cesarini F., Thompson S. *Erlang programming*. Beijing, Cambridge, O'Reilly, 2009 (Russ. ed.: Chezarini F., Tompson S. *Programmirovanie v Erlang*. Moscow, DMK Press, 2012. 488 p.).

4. Aho A.V., Lam M.S., Sethi R., Ullman J.D. *Compilers: principles, techniques & tools*. 2nd ed. Boston, Pearson Addison Wesley, 2007 (Russ. ed.: Akho A.V., Lam M.S., Seti R., Ul'man D.D. *Kompilyatory: printsipy, tekhnologii i instrumentarii*. Moscow, Vil'yams, 2008. 1184 p.).

5. *C++ Programming Language:* website. Available at: https://devdocs.io/cpp/ (accessed 28.05.2021).

6. Malyavko A.A. [Using web applications and web technologies in the development of educational software for studying methods of translation]. *Sovremennoe obrazovanie: tekhnicheskie universitety v modernizatsii ekonomiki Rossii*: materialy Mezhdunarodnoi nauchno-metodicheskoi konferentsii [Materials of the scientific and methodological conference "Modern education: technical universities in the modernization of the Russian economy"]. Tomsk, TUSUR Publ., 2011, pp. 45–46. (In Russian).

7. *CLion*: website. Available at: https://www.jetbrains.com/ru-ru/clion/ (accessed 28.05.2021).

8. Lopes B., Auler R. *Getting started with LLVM core libraries*. Birmingham, Packt Publishing, 2014 (Russ. ed.: Lopes B, Auler R. *LLVM: infrastruktura dlya razrabotki kompilyatorov*. Moscow, DMK Press, 2015. 342 p.).

9. *GCC online documentatio*n: website. Available at: https://gcc.gnu.org/onlinedocs/ (accessed 28.05.2021).

10. Hanter R. *The design and construction of compilers*. Chichester, New York, Wiley, 1981 (Russ. ed.: Khanter R. *Proektirovanie i konstruirovanie kompilyatorov*. Moscow, Finansy i statistika Publ., 1984. 232 p.).

11. *LLVM Language Reference Manual*: website. Available at: https://llvm.org/docs/LangRef.html (accessed 28.05.2021).

12. Aho A.V., Hopcroft J.E., Ullman J.D. *Data structures and algorithms*. Reading, Addison-Wesley, 1983 (Russ. ed.: Akho A.V., Khopkroft D., Ul'man D.D. *Struktury dannykh i algoritmy*. Moscow, Vil'yams Publ., 2000. 384 p.).

13. Volodin A.M., Drozhdin V.V. Raznoobrazie struktur dannykh [The variety of data structure]. *Izvestiya Penzenskogo gosudarstvennogo pedagogicheskogo universiteta im. V. G. Belinskogo*, 2009, no. 13 (17), pp. 79–83. (In Russian).

14. *Struktura dannykh – vektor* [Data structure – vector]. Available at: https://russianblogs.com/article/11141566726/ (accessed 28.05.2021).

15. *Kortezhi* [Tuples]. (In Russian). Available at: https://lawbooks.news/programmirovanie_964/korteji-68448.html (accessed 28.05.2021).

16. *Struktury dannykh* [Data structures]. (In Russian). Available at: http://www.realcoding.net/articles/struktury-dannykh.html (accessed 28.05.2021).

Для цитирования:

Зеленчук Н.А., Приставка Е.Д., Малявко А.А. Компилятор с языка программирования EL: усовершенствование и развитие // Безопасность цифровых технологий. – 2021. – № 2 (101). – С. 9–19. – DOI: 10.17212/2782-2230-2021-2-9-19.

For citation:

Zelenchuk N.A., Pristavka E.D., Maliavko A.A. Kompilyator s yazyka programmirovaniya EL: usovershenstvovanie i razvitie [Compiler from El programing language: improvement and development]. *Bezopasnost' tsifrovykh tekhnologii = Digital technology security*, 2021, no. 2 (101), pp. 9–19. DOI: 10.17212/2782-2230-2021-2-9-19.