ВИЗУАЛЬНО-ГРАФИЧЕСКАЯ СИСТЕМА ПРОГРАММИРОВАНИЯ НА ОСНОВЕ РАЗРАБОТКИ БЛОК-СХЕМ АЛГОРИТМОВ. КОНВЕРТИРОВАНИЕ ТЕКСТОВ ПРОГРАММ НА РАЗЛИЧНЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ В БЛОК-СХЕМЫ И ОБРАТНО

А.А. Вохмин, О.А. Евдокимова, А.А. Малявко

Новосибирский государственный технический университет, г. Новосибирск

В работе представлены результаты исследований и разработки средств визуально-графического проектирования сложных алгоритмов в виде блок-схем в целом и, в частности, интерфейсной компоненты таких средств – конвертора текстов программ на различных языках программирования во внутреннее структурно-вложенное представление и обратно в тексты.

Визуально-графическое представление алгоритмов лучше воспринимается человеком-разработчиком, чем традиционное текстовое представление, поэтому развитию подобных систем программирования в последнее время уделяется определенное внимание. Однако реализация максимально удобных для человека технологий создания и редактирования блок-схем сопряжена с необходимостью решения ряда сложных задач формирования и адекватного отображения управляющих структур, используемых в популярных языках программирования. Проведен анализ таких структур для наиболее популярных по разным метрикам языков программирования, представлены его результаты и предложен способ выявления и преобразования управляющих структур во внутреннее представление визуально-графического редактора. Описаны основные алгоритмы работы конвертора текстов программ как в прямом, так и в обратном направлениях.

Ключевые слова: конвертер, синтаксический анализ, программный код, языки программирования, блок-схема, алгоритм.

ВВЕДЕНИЕ

Наблюдающийся в настоящее время почти взрывной рост сложности программных средств порождает потребность в создании адекватных этой сложности инструментов разработки программного обеспечения. В этом направлении сделано очень многое: созданы высокоэффективные трансляторы современных языков программирования, отладчики и профайлеры, предоставляющие возможности работы с оптимизированными машинными кодами в терминах исходных текстов программ, многопользовательские системы контроля версий и ряд других технологических средств, обычно объединяемых в единое целое с использованием интегрированных оболочек.

Однако поддержка собственно процесса разработки по-настоящему сложных алгоритмов, составляющих суть современных программных систем, работающих с большими данными или обрабатывающих множественные потоки запросов и т.д., остается на привычном уровне. Разработчик преобразует идею алгоритма в реализующую его программу с использованием обычных текстовых редакторов. Необходимость оформлять тексты программ на одном или нескольких языках приводит к «загромождению» алгоритма множеством требуемых стандартами языков деталей, таких как

определение используемых переменных и задание их типов, выбор наиболее подходящих управляющих конструкций (условных операторов или переключателей, циклов различных видов и т.д.), оформлением большого количества методов и интерфейсов.

Одним из потенциально возможных путей усовершенствования системы инструментов для разработки сложных программных систем является визуально-графическое проектирование блок-схем алгоритмов. Этот подход ранее почти не развивался вследствие недостаточной производительности аппаратных средств для поддержки всех необходимых манипуляций с визуальными примитивами в процессе проектирования алгоритма. Современное состояние оборудования для работы с изображениями позволяет рассчитывать на то, что мощности обычного десктопного компьютера или ноутбука будет достаточно для реализации всех необходимых функций визуально-графического редактора блок-схем в составе новой интегрированной оболочки. Создание такой оболочки требует решения ряда непростых задач, к числу которых относится реализация интерфейсных компонент для обеспечения возможностей импорта и экспорта обычных текстов программ.

ПОСТАНОВКА ЗАДАЧИ

Целью работы является разработка приложения конвертирования текстов программ в/из структурированного внутреннего представления как выполнение отдельной части общего проекта по разработке визуально-графической системы программирования на основе разработки блок-схем алгоритмов

Для достижения этой цели необходимо выполнить ряд задач:

1. Провести обзор существующих методов синтаксического анализа программного кода с целью обнаружения наиболее эффективного в рамках поставленной задачи;

2. Проведение анализа существующих программ-аналогов визуальных редакторов блок-схем, предоставляющих функциональную возможность импорта/экспорта текста программ;

3. Выявление перечня типовых управляющих конструкций в распространенных языках программирования;

3. Разработка обобщенной структуры внутреннего представления управляющего оператора для использования в визуальном редакторе;

4. Разработка конвертера прямого преобразования текстов программ на наиболее употребляемых языках программирования во внутреннее представление визуального редактора блок-схем алгоритмов;

5. Разработка конвертера обратного преобразования для формирования текста программы на основе построенной блок-схемы алгоритма в визуально-графическом редакторе.

МЕТОДЫ СИНТАКСИЧЕСКОГО АНАЛИЗА ТЕКСТА

Разрабатываемое приложение конвертирования текстов программ представляет собой синтаксический анализатор, т.е. является частью программы, преобразующей входные данные, как правило, являющиеся текстом, в некий структурированный формат, с целью их дальнейшей обработки [1].

Любой язык программирования (ЯП) можно описать набором правил, которые выделяют некоторое подмножество известных символов из множества слов конечного алфавита языка [2]. Данный набор правил называется грамматикой языка, и для возможности ее описания необходимо иметь список языковых конструкций, используемых в рассматриваемом языке.

Как правило, при обработке программного кода в компиляторах применяется метод, основанный либо на LL (k)-грамматиках, либо на LR (k)-грамматиках.

В большинстве компиляторов современных ЯП используется LR-анализ, как наиболее эффективный с точки зрения анализа текста и выявления в нем ошибок. Однако LL-анализ позволяет проводить обработку текста намного быстрее. Дж. Козн и М.С.

Рот в своей работе [3] сравнили максимальное, минимальное и среднее время разбора текста с помощью LL- и LR-анализаторов. Результаты показывают, что LL-анализ в среднем работает на 50% быстрее.

При конвертировании программного кода во внутренний формат визуального редактора не ставится задача полного анализа кода с нахождением всех конструкций, формирующих грамматику языка, и их посимвольного разбора, поэтому в качестве метода синтаксического анализа текстов программ будет взят за основу LL-метод, а именно его реализация в виде предсказывающего парсера, являющимся простым в программной реализации, а также демонстрирующим высокую скорость выполняемого анализа, относительно других методов.

ПОДХОД К ОПРЕДЕЛЕНИЮ ПОПУЛЯРНОСТИ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Популярность того или иного языка программирования – понятие субъективное. Для создания объективного рейтинга языков программирования недостаточно данных. А именно, невозможно отследить какое количество программистов начало изучать конкретный язык в недавнее время, как невозможно отследить и количество существующих проектов, в которых он используется [4]. В связи с этим были разработаны различные системы рейтингов ЯП, использующие в своей основе различные подходы. Наиболее популярные из них:

- индекс TIOBE;
- рейтинг PYPL;
- проведение опроса.

Согласно индексу TIOBE, используемому в качестве анализируемых данных запросы пользователей в поисковых системах [5] рейтинг популярности ЯП соответствует приведенным в таблице 1 данным.

Табл. 1. Рейтинг популярности ЯП согласно индексу TIOBE

№ п/п	ЯП	Рейтинг
1	C	16.48%
2	Java	12.53%
3	Python	12.21%
4	C++	6.91%
5	C#	4.20%
6	Visual Basic	3.92%
7	JavaScript	2.35%
8	PHP	2.12%
9	R	1.60%

Рейтинг PYPL (popularity of programming language) использует для анализа систему Google-тренды. В качестве анализируемых данных выступают обучающие статьи и видео по какому-либо ЯП. Их популярность в данный момент времени и говорит о востребованности того или иного ЯП [6]. В таблице 2 представлен рейтинг десяти популярных ЯП согласно

PYPL актуальный на момент написания данной работы.

Одним из существенных недостатков вышеописанных систем является анализ исключительно зарубежных ресурсов.

Табл. 2. Рейтинг популярности ЯП согласно рейтингу PYPL

№ п/п	ЯП	Рейтинг
1	Python	30.34%
2	Java	17.23%
3	JavaScript	8.65%
4	C#	6.44%
5	C/C++	6.11%
6	PHP	5.88%
7	R	3.84%
8	Objective-C	3.75%
9	Swift	2.17%
10	TypeScript	1.89%

В начале 2020 года на популярном веб-ресурсе, посвященном тематике программирования, был проведен опрос на предмет практического применения языков программирования на территории стран СНГ. На основе данного подхода был графически сформирован рейтинг ЯП, представленный на рисунке 1.

Рис. 1. Результат проведения опроса

Результаты опроса базируются на выборке из 10000 анкет, что не сравнится с количеством обрабатываемых данных в вышеописанных системах, однако результаты данного опроса базируются на не столь косвенных данных, поэтому имеет смысл учитывать их при определении списка ЯП, которые необходимо поддерживать в разрабатываемой визуально-графической системе программирования.

В результате проведения исследования, целью которого является определение наиболее употребляемых ЯП, было установлено, что в разрабатываемой системе необходимо обеспечить поддержку следующих языков:

- Python;
- Java;
- JavaScript;
- C++;
- C#;
- C.

В процессе реализации проекта для вышеперечисленных ЯП были сформированы списки грамматик, описывающих используемые в них языковые конструкции (ЯК) (операторы циклов, ветвлений, объявление переменных и т.д.).

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРИЛОЖЕНИЙ, РЕШАЮЩИХ БЛИЗКИЕ ЗАДАЧИ

В результате сравнительного анализа существующих программ, предоставляющих функциональную возможность построения блок-схем (БС) алгоритмов с поддержкой импорта и экспорта данных в виде кода программ (КП), была сформирована таблица, отображающая соответствие рассматриваемых программ предъявляемым разрабатываемой системе критериям (табл. 3).

Табл. 3. Сравнительный анализ программ аналогов

	Поддерживаемые ЯП	Загрузка КП	Формирование КП	Обработка БС сложных алгоритмов	Обработка нескольких файлов
Code to FlowChart	-	+	-	±	-
Flowchart to Code	±	-	+	-	-
Code Visual to Flow Chart	±	+	-	-	±
Visustin	+	+	-	-	±
Flowgorithm	+	-	±	-	+

На основе представленных в таблице данных можно сделать вывод о том, что ни один из рассматриваемых программных продуктов не удовлетворяет в полной мере сформированному списку критериев. А именно, ни одна из программ не позволяет пользователю одновременно формировать блок-схему на основе нескольких файлов и в тоже же время формировать файлы с кодами программ на основе построенных БС. Также рассмотренные аналоги не предоставляют в полной мере функциональной возможности упрощения визуального представления БС сложных алгоритмов внутри редактора.

Проведенный сравнительный анализ подчеркивает необходимость создания приложения прямого и обратного конвертирования программного кода в виде группы файлов на разных ЯП с возможностью нахождения уровня вложенности языковых конструкций для их графического представления в

свернутом/развернутом виде в визуальном редакторе блок-схем.

ПРОЕКТИРОВАНИЕ ОБОБЩЕННОЙ СТРУКТУРЫ ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ УПРАВЛЯЮЩЕГО ОПЕРАТОРА ДЛЯ ИСПОЛЬЗОВАНИЯ В ВИЗУАЛЬНОМ РЕДАКТОРЕ

В соответствии с требованиями, предъявляемыми к разрабатываемому проекту, необходимо спроектировать такой обобщенный формат хранения всех языковых конструкции различных ЯП, который позволял бы:

1)формировать визуально-графический аналог каждой конструкции в двух вариантах – свернутом и развернутом;

2)корректно отображать связь между блоками в соответствии с алгоритмом, описанным в программе;

3)содержать комментарии, оставленные пользователем, для конкретной языковой конструкции.

С учетом вышеперечисленных критериев было принято решение в качестве выходных данных работы программы формировать массив объектов с определенным набором полей, где каждый объект соответствует одной отдельно взятой языковой конструкции в тексте программы, а также соответствует одному блоку в визуальном представлении в виде блок-схемы. Список полей объектов с их кратким описанием представлен в таблице 4.

Табл. 4. Поля объекта, описывающего ЯК

Имя поля	Краткое описание	Тип данных
id	уникальный идентификатор	целое число
parent_id	id блока, являющегося родителем текущего блока	целое число
neighbor_id	id блока, являющегося соседним для текущего блока (применяется при ветвлении)	целое число
type	тип блока	строка
inner_lvl	уровень вложенности блока	целое число
content	содержимое блока	строка
parameter	параметр рассматриваемого блока	строка
comment	комментарий, соответствующий данному блоку	строка

Для визуального представления общей структуры иерархии классов разрабатываемой системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними [7] с учетом ранее описанной обобщенной структуры управляющего оператора была спроектирована диаграмма классов (рис. 2). На данной диаграмме также отражено взаимодействие приложений-конвертеров с разрабатываемым визуальным редактором блок-схем.

ОПИСАНИЕ АЛГОРИТМА КОНВЕРТИРОВАНИЯ ТЕКСТА ПРОГРАММЫ ВО ВНУТРЕННИЙ ФОРМАТ ВИЗУАЛЬНОГО РЕДАКТОРА

При импорте данных выбранные пользователем файлы отправляются из пакета «DataLoader» в пакет «Converter_in». Дальнейший алгоритм работы программы конвертирования можно отобразить в виде блок-схемы, представленной на рисунке 3.

Рис. 3. Алгоритм работы программы прямого конвертирования

Как следует из алгоритма в цикле обрабатываются все выбранные пользователем файлы. Для каждого файла автоматически определяется используемый в нем ЯП на основе его расширения. После чего из базы данных подгружается набор языковых грамматик, описывающих правила формирования конструкций и параметров для определенного ранее ЯП. Затем следует обработка файла, отмеченного на блок-схеме подпрограммой «нахождение ЯК». Алгоритм данной подпрограммы, в свою очередь представлен в виде блок-схемы на рисунке 4.

Рис. 4. Алгоритм нахождения языковых конструкций в тексте

Приведенный выше алгоритм описывает нахождение всех языковых конструкций в тексте программы. В случае обнаружения конструкции, которая может содержать в себе вложенные структуры, происходит рекурсивный вызов данной подпрограммы, при этом параметр, отражающий уровень вложенности структуры, увеличивается на единицу при каждом ее вызове, что позволяет при

обработке БС в визуальном редакторе отображать конструкции в свернутом/развернутом виде.

ОПИСАНИЕ АЛГОРИТМА КОНВЕРТИРОВАНИЯ БЛОК-СХЕМЫ В ТЕКСТ ПРОГРАММЫ

После построения БС алгоритма в визуальном редакторе пользователь должен иметь возможность выгрузить разработанный им алгоритм в виде программного кода, на выбранном им ЯП.

При экспорте данных редактор отправляет массив, содержащий сведения обо всех построенных блоках, а также о выбранном пользователем для конкретного алгоритма ЯП. После чего запускается приложение-конвертер, обрабатывающее входной массив данных и отправляющее в качестве выходного значения массив файлов, содержащих программный код. Алгоритм, описывающий обработку массива данных с целью формирования единого файла проекта, представлен на рисунке 5. В качестве метода обработки данных используется обход дерева.

Как видно из графического представления данного алгоритма в процессе обработки данных первоначально находится родительский блок, после чего находятся и поочередно обрабатываются блоки-потомки и блоки-соседи, формируя при этом очередь добавления в текстовый файл конструкций ветвления. При нахождении блока, уровень вложенности которого больше, чем уровень вложенности предыдущего блока, происходит обращение к списку, хранящему уникальные идентификаторы и уровень вложенности блоков, которые относятся к ответвлениям алгоритма. Если данный список не пуст, а уровень вложенности данного блока соответствует уровню, обработанному ранее, то происходит построение строки текста в файле на его основе. Также к данному списку происходит обращение, если следующего за рассмотренным блоком не существует. Выход из цикла (прекращение формирования файла) происходит при условии пустого списка ожидания и при отсутствующем блоке-потомке.

Внутри подпрограммы «createLine(CurrentBlock)» при помощи оператора switch (на основе типа блока) выбирается набор инструкций по формированию строки текстового файла в зависимости от указанного при передаче данных ЯП.

Внутри этой подпрограммы также обрабатывается условие, при котором указанная в БС конструкция не существует в выбранном ЯП. Так, например, при формировании файла на языке Python конструкция «switch», в силу своего отсутствия в данном языке, будет приведена к формату «if/elif/else», а константы, также отсутствующие в Python, будут обозначены в соответствии с установленными правилами [8] – заглавными буквами.

РАЗРАБОТАННАЯ ПРОГРАММА ПРЯМОГО И ОБРАТНОГО КОНВЕРТИРОВАНИЯ ТЕКСТОВ

На основе описанных в данной работе обобщенной структуры внутреннего представления управляющего оператора, диаграммы классов и алгоритмов конвертирования на языке JavaScript было разработано приложение прямого и обратного преобразования текстов программ на наиболее употребляемых ЯП.

В качестве примера работы программы загрузим в визуально-графическую систему программирования файл, содержащий код программы на языке JavaScript (рис. 6), вычисляющий факториал введенного пользователем числа.

```
let some_value;
some_value = prompt( message: "Введите число");
let fact = 1;
for (let i = 1; i <= some_value; i++)
    fact *= i;
alert(some_value + "! = " + fact);
```

Рис. 6. Код программы на ЯП JavaScript

В результате обработки данного файла был получен массив объектов, представленный на рисунке 7. На основе полученных данных представляется возможным автоматически построить БС, представленную на рисунке 8.

Рис. 8. Блок-схема алгоритма загруженного пользователем файла

Стоит отметить, что блок цикла на данной БС отображен в свернутом виде и при необходимости может быть развернут пользователем (рис. 9).

Рис. 9. Представление блока цикла в развернутом виде

Как можно заметить из построенной БС цикл «for» был приведен к группе блоков – инициализации итерационной переменной, цикл с предусловием и переопределение значения итерационной переменной (увеличение на единицу).

В конечном итоге пользователь может выгрузить из редактора текст программы на любом из ранее описанных ЯП. Так, при выборе языка Python будет получен файл, содержащий код, представленный на рисунке 10.

```
print("Введите число")
some_value = input()
fact = 1
i = 1
while i <= int(some_value):
    fact *= i
    i += 1
print(str(some_value) + "! = " + str(fact))
```

Рис. 10. Результат конвертирования блок-схемы в код программы на ЯП Python

Из представленного выше рисунка видно, что цикл «for», использовавшийся изначально в программе (рис. 6), был приведен к циклу «while», что не является оптимальным решением в данном случае, однако данное представление цикла не нарушает алгоритма работы программы. Анализ блоков, описывающих цикл, с целью формирования в текстовом виде операторов цикла «for», «for-each» и «while» является одной из задач будущего развития проекта.

ЗАКЛЮЧЕНИЕ

В этой работе представлены результаты исследований и разработки средств визуально-графического проектирования сложных алгоритмов в виде блок-схем в целом и, в частности, интерфейсной компоненты таких средств – конвертора текстов программ на различных языках программирования во внутреннее структурно-вложенное представление и обратно в тексты. Визуально-графическое представление алгоритмов лучше воспринимается

человеком-разработчиком, чем традиционное текстовое представление, поэтому развитию подобных систем программирования в последнее время уделяется определенное внимание. Однако реализация максимально удобных для человека технологий создания и редактирования блок-схем сопряжена с необходимостью решения ряда сложных задач формирования и адекватного отображения управляющих структур, используемых в популярных языках программирования. Поэтому в данной работе был проведен анализ таких структур для наиболее популярных по разным метрикам языков программирования и предложен способ выявления и преобразования управляющих структур во внутреннее представление визуально-графического редактора. Описаны основные алгоритмы работы конвертора текстов программ как в прямом, так и в обратном направлениях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ахо, А.В. Теория синтаксического анализа, перевода и компиляции [Текст] / А.В. Ахо, Д.Д. Ульман. – США: Изд-во Prentice Hall, 1972. – Т. 1. – 613 с.
2. Context-free grammar [Электронный ресурс] // Wikipedia. – Режим доступа: https://en.wikipedia.org/wiki/Context-free_grammar

3. Cohen, J. and Roth, M.S., “Analyses of Deterministic Parsing Algorithms”, CACM 21, vol. 6, pp. 448 – 458. 1978

4. Самые популярные языки программирования [Электронный ресурс] // Хабр. – Режим доступа: <https://habr.com/ru/company/skillfactory/blog/531360>.

5. TIOBE Index for December 2020 [Электронный ресурс] // TIOBE. – Режим доступа: <https://www.tiobe.com/tiobe-index/>.

6. PYPL PopularitY of Programming Language [Электронный ресурс] // PYPL Index. – Режим доступа: <https://pypl.github.io/PYPL.html>.

7. Проектирование диаграммы классов UML [Электронный ресурс] // WordSkills Russia. – Режим доступа: <https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-klassov-uml-class-diagram/>.

8. Константы в Python [Электронный ресурс] // Pythonz. – Режим доступа: <https://pythonz.net/references/named/constants/>

Вохмин Александр Андреевич – студент, магистрант, Новосибирский государственный технический университет, тел. (383)3460492, e-mail: alexandervokhmin@gmail.com.

Евдокимова Ольга Алексеевна – студент, магистрант, Новосибирский государственный технический университет, тел. (383)3460492, e-mail: evdokimova_olga_15@mail.ru.

Малявко Александр Антонович – кандидат технических наук, доцент кафедры ВТ, Новосибирский государственный технический университет, тел. (383)3460492, e-mail: a.malyavko@corp.nstu.ru.

Рис. 2. Диаграмма классов разрабатываемого проекта

Рис. 5. Алгоритм обработки массива данных при формировании программного кода на основе построенной блок-схемы алгоритма

id	parent_id	neighbour_id	type	inner_lvl	content	parameter	comment
0	-1	-1	"declaring"	1	"some_value"	" "	" "
1	-1	-1	"output"	1	"Введите число "	" "	" "
2	-1	1	"input"	1	" some_value"	" "	" "
3	-1	2	"initializing"	1	"sum = 0"	" "	" "
4	-1	3	"initializing"	1	"i = 1"	" "	" "
5	-1	3	"for"	1	" "	" i < some_value"	" "
6	5	5	"change_value"	2	"sum += i;"	" "	" "
7	5	5	"change_value"	2	"i = i + 1;"	" "	" "
8	-1	5	"output"	1	" some_value "	" "	" "
9	-1	8	"output"	1	" i = " "	" "	" "
10	-1	9	"output"	1	" sum"	" "	" "

Рис. 7. Результат конвертирования файла программы на ЯП JavaScript во внутренний формат редактора блок-схем

VISUAL-GRAPHIC PROGRAMMING SYSTEM BASED ON DEVELOPMENT OF BLOCK DIAGRAMS OF ALGORITHMS. CONVERTING PROGRAMS TEXTS IN DIFFERENT PROGRAMMING LANGUAGES INTO BLOCK DIAGRAMS AND BACK

A.A. Vokhmin, O.A. Evdokimova, A.A. Malyavko

Novosibirsk State Technical University, Novosibirsk

Abstract – The paper presents the results of research and development of tools for visual and graphic design of complex algorithms in the form of block diagrams in general and, in particular, the interface component of such tools - a converter of programs texts in various programming languages into an internal structurally nested representation and back into texts.

The visual-graphical representation of algorithms is better perceived by a human developer than the traditional textual representation, therefore, some attention has been paid to the development of such programming systems in recent years. However, the implementation of the most human-friendly technologies for creating and editing block diagrams is associated with the need to solve a number of complex problems of forming and adequately displaying control structures used in popular programming languages. The analysis of such structures for the most popular programming languages in terms of various metrics is carried out, its results are presented, and a method for identifying and transforming control structures into an internal representation of a visual-graphic editor is proposed. The main algorithms for the operation of the program text converter both in forward and backward directions are described.

Index terms: converter, parsing, program code, programming languages, block diagram, algorithm.

REFERENCES

1. Aho, A.V and J.D. Ullman. The theory of parsing translation and compiling, vol. 1, USA: Prentice Hall, 1972.
2. Context-free grammar. Wikipedia, https://en.wikipedia.org/wiki/Context-free_grammar
3. Cohen, J. and Roth, M.S., “Analyses of Deterministic Parsing Algorithms”, CACM 21, vol. 6, pp. 448 – 458. 1978
4. The most popular programming languages. Habr, <https://habr.com/ru/company/skillfactory/blog/531360>.
5. TIOBE Index for December 2020, <https://www.tiobe.com/tiobe-index/>.
6. PYPL Popularity of Programming Language. PYPL Index, <https://pypl.github.io/PYPL.html>.
7. Designing a UML Class Diagram. WordSkills Russia, <https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-klassov-uml-class-diagram/>.
8. Constants in Python. Pythonz, <https://pythonz.net/references/named/constants/>

Vokhmin Alexander Andreevich – student, Novosibirsk State Technical University, (383)3460492, e-mail: alexandervokhmin@gmail.com.

Evdokimova Olga Alekseevna – student, Novosibirsk State Technical University, (383)3460492, e-mail: evdokimova_olga_15@mail.ru.

Malyavko Alexander Antonovich – Candidate of Technical Sciences, Associate Professor of the Department of Computing Technologies, Novosibirsk State Technical University, (383)3460492, e-mail: a.malyavko@corp.nstu.ru.