УДК 004, 004.4

Специальность: 05.13.11

**Проблемы и пути их решения при осуществлении миграции программных продуктов на современные версии языка C**

**стр.** ХХ – ХХ

**А.А. Бирюкова1,2, С.А. Басыров2**

1 Федеральное государственное автономное научное учреждение «Центр информационных технологий и систем органов исполнительной власти» (ФГАНУ ЦИТиС), 123557, Москва, Пресненский Вал, 19, стр. 1

2 Федеральное государственное бюджетное образовательное учреждение высшего образования "МИРЭА - Российский технологический университет" (РТУ МИРЭА), 119454 г. Москва, проспект Вернадского, дом 78

|  |  |
| --- | --- |
| *Контактные данные:* | *Бирюкова А.А., e-mail:* [*birukova@inevm.ru*](mailto:birukova@inevm.ru) |
|  | *Басыров С.А., e-mail:* [*pmswga@gmail.com*](mailto:pmswga@gmail.com) |
|  |  |

***Резюме:*** *Миграция является актуальной задачей в разработке программного обеспечения. Необходимость миграции возникает с выходом обновлений языка, библиотек, фреймворков или более совершенных инструментальных средств. Более узкой задачей является миграция кода, то есть переход с одного языка программирования на другой. Миграция кода на актуальную версию языка позволяет избежать уязвимостей старых версий, избежать ошибок (исправленных в новых версиях языка), повысить быстродействие и эффективность работы кода. Однако, данная задача является сложной и до сих пор наблюдается дефицит соответствующих инструментальных средств, позволяющих мигрировать с одного языка на другой в автоматическом режиме или хотя бы позволяющих облегчить этот процесс[1][2].*

*Статья посвящена миграции кода, а именно переходу со старых версий языка С89/С99 на новые версии C11-C17/C23 и прототипированию транскомпилятора для автоматической миграции.*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***Ключевые слова:*** *портирование программного обеспечения, переносимость программ, миграция кода, транскомпиляторы.*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problems and ways of their solving while migration to the new C programming language vertion**

**pages** XX – XX

A.A. Biryukova1,2, S. A. Basyrov2

1 Federal State Autonomous Research Institution “Centre of Information Technologies and Systems for Executive Power Authorities”, 19, Presnensky Val, Moscow, 123557, Russia

2 Federal State Budget Educational Institution of Higher Education «MIREA - Russian Technological University»,78, Vernadsky Avenue, Moscow, 119454, Russia

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Summary:*** *Summary: Migration is an actual issue in software engineering. Need of migration appears when updates of languages, libraries, frameworks and more perfect tools come out. The more narrow task of migration is code migration, which means migration from the current programming language to another programming language. Code migration on actual programming language allows to avoid vulnerabilities of earlier vertions, errors (fixed in new programming language version), increase the speed and efficiency of work of the code. However, this task is hard and nowadays there are not enough tools, that allow to migrate from one programming language to another in automatic mode or even facilitate this process.*

*This article describes code migration from the old C programming language version C89/C99 to the new C programming language version C11-C17/C23 and prototyping the transcompiler.*

***Keywords:*** *porting, program portability, code migration, transcompiler*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**doi:** (это будет присваивать редакция)

# Введение

Миграция (англ. Migration) — процесс перехода от одних инструментальных средств к другим. Сам процесс миграция встречается в базах данных, прикладных и веб-приложениях, при переходе с одной версии языка на другую. С миграцией связаны следующие проблемы[3]:

1) Неорганизованность — перед началом миграции необходимо составить соответствующий план;

2) Потеря данных — при переносе данных из одного места в другое, они могут потеряться. Поэтому необходимо использовать резервные;

3) Вопросы совместимости — при переходе на новую версию инструментального средства (язык, библиотека, фреймворк и т.д.) может вызывать проблемы с обратной совместимостью, так как в новых версиях может отсутствовать ранее присутствующая функциональность;

4) Проблемы с оборудованием — при переходе на новую аппаратную или программную среду (операционную систему) необходимо учитывать их возможности и совместимость.

Другие источники также подчёркивают эти проблемы миграции[4][5][6] и важность обновления программного обеспечения[7].

Перейдём к рассмотрению вопроса миграции кода, а именно перехода с одной версии языка на новую версию.

## *Миграция кода*

Миграция кода — процесс непосредственного перехода от одной версии языка на другую версию языка. Например, с C99 на C11.

Вот несколько причин, по которым миграция кода необходима:

— Исправление ошибок в новых версиях языка;

— Уточнение поведения языка;

— Добавление новых возможностей;

— Повышение производительности за счёт улучшения работы компилятора;

— Повышение надёжности и безопасности кода;

— Исправление ошибок в работе компилятора;

— Избавление от устаревших возможностей.

Примеры миграций приведены в источниках [8][9][10][11], которые подтверждают актуальность данного вопроса.

В примере [8] миграция осуществлялась за счёт скриптов, которые заменяли в исходных файлах имена типов, классов, пространств имён и имена подключаемых файлов. Переход на стандарт C++14 занял 6 месяцев.

В примере [9] разъясняется зачем необходима миграция баз данных. Основные утверждения в пользу миграции: снижение расходов на поддержку старой системы и поддержку современных средств, минимизация затрат на оборудование, улучшение процессов хранения и сохранности данных. Основные факторы, от которых зависит миграция: объём кода на стороне БД, объём БД, различия в языках исходной и целевой БД.

В примере [10] приведён транскомпилятор, используемый компанией Facebook для трансляции кода из PHP в C++. В последствии транскомпилятор был заменён виртуальной машиной.

В примере [11] описана модель миграции из C/C++ в Haxe на мобильные платформы.

## *Актуальность языка C*

Перед тем как рассматривать необходимость миграции, стоит выяснить, а является язык C актуальным? По существующим рейтингам языков программирования TIOBE и PYPL, на август 2021 года он занимает 1-е[12] и 5-е места[13], актуальность языков по регионам приведена в [14]. Однако, более убедительным аргументом в пользу актуальности будет перечисление проектов, которые пишутся на языке C сегодня[15] (Таблица 1).

Таблица 1 — Проекты написанные на языке C

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Проект** | **Описание** | **Актуальная версия** |
| 1 | Linux kernel[16] | Ядро операционной системы | 30 августа 2021, 5.14 |
| 2 | Git[17] | Система контроля версий | 16 августа 2021, 2.33.0 |
| 3 | Redis[18] | NoSQL база данных | 21 июля 2021, 6.2.5 |
| 4 | PHP src[19] | Интерпретатор языка PHP | 24 августа 2021, 8.0.10 |
| 5 | SQLite[20] | Встраиваемая СУБД | 18 июня 2021, 3.36.0 |

Таким образом, актуальность языка подтверждается не только отраслевой необходимостью, а именно в области системного программирования, но непосредственно развитием самого языка.

## *Необходимость миграции*

Развитие языка C связано с постепенным заимствованием существующих средств из C++, исправления ошибок в поведении языка и добавление новых возможностей. Однако, ввиду того что это всё же разные языки, то соответственно и развиваются они по-разному. Вот основные направления в развитии языка C[21]:

— поддержка и вычислений чисел с плавающей запятой;

— добавление новых математических функций и атрибутов;

— поддержка параллелизма[22];

— поддержка работы с транзакционной памятью[23].

Миграцию стоит рассматривать в трёх направлениях:

1) С C89 на C99;

2) С C99 на C11/C17;

3) С C11/C17 на C23.

Схематично это представлено на рисунке 1.

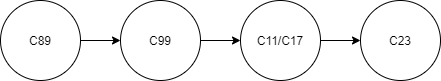


Рисунок 1 — Направления миграции

Необходимость миграции с C89 на C99 связана с тем, что в C99 добавлены новые ключевые слова, изменены сигнатуры некоторых функций, добавлены новые типы данных. Более подробное сравнение приведено в [24].

Миграция же с C99 к C11-C17 в основном связана с тем, что в C11 добавлено атомарные типы данных, поддержка параллелизма, шаблонов. C17 является работой над ошибками и их исправление [25]. Переход позволит писать многопоточные программы, обобщить работу с функциями (за счёт использования шаблонов), избежать возникновения ошибок и неопределённого поведения.

Миграция с C11-C17 на C23 также в основном связана с тем, что в C23 будет добавлена работа с транзакционной памятью, улучшенная поддержки параллелизма и исправлены ошибки [26]. Переход позволит использовать новые возможности для улучшения многопоточных программ написанных на C11-C17.

Все эти различия требуют переписывание кода, с учётом нового синтаксиса и семантики поведения. Где-то в большей степени (с C89 на C99), где-то в меньшей степени (с C99 на C11-17). Переход с C99 на C11-C17 менее затруднителен, так как он успешно компилируется и с использованием C11. Далее предпримем попытку автоматизации этого процесса.

## *Способы миграции кода*

Способы миграции кода условно можно разделить на ручной и автоматический.

Ручной — разработчики самостоятельно изменяют или переписывают код. Данный способ менее предпочтителен, так как может повлечь за собой появление ошибок. Помимо этого данный способ требует очень много времени на осуществление миграции.

Автоматический — использование инструментальных средств позволяющие облегчающих миграцию. Данный же способ наиболее предпочтителен, так как минимизирует участие человека в процессе миграции кода. Что приводит к сокращению количества потенциальных ошибок, повышению эффективности и скорости миграции[27].

Однако, сложность состоит в том, что практически нет средств, осуществляющие полноценную миграцию без участия разработчика. Сложность, в разработке таких средств, состоит в том, что, не смотря на строгую формальную систему языка, необходимо учитывать и понимать семантику.

## *Инструментальные средства*

Ниже приведены примеры инструментальных средств, которые помогают облегчить миграцию кода:

— Port Assist[28]. Плагин для Kdevelop позволяющий оценить количество изменяемого кода с возможностью преобразования кода. Пример приведён на рисунке 2;

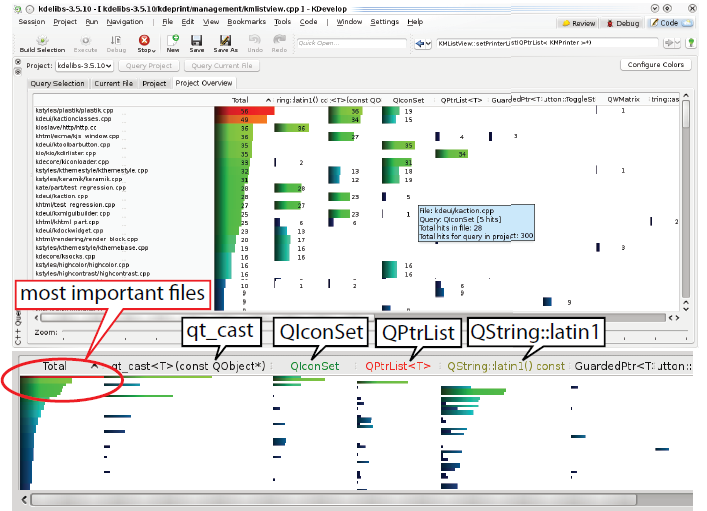


Рисунок 2 — Port Assist

— CodeCheck[29]. Средство позволяющее проверить соответствие кода определённому стандарту, собрать метрики кода (сложность, ООП метрики). Но при этом оно не позволяет каким-либо образом изменять код;

— DMS Software Reengineering Toolkit[30]. Набор инструментальных средств, включающий в себя анализатор, трансформатор и генератор кода. Схема работы представлена на рисунке 3.

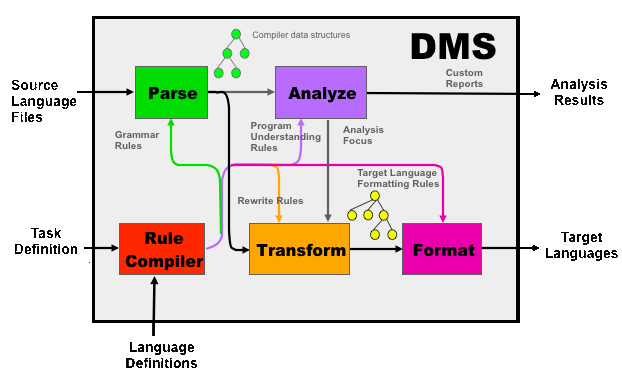


Рисунок 3 — Схема работы DMS

# Прототипирование транскомпилятора

Задачу трансляции старых исходных кодов из С89/С99 в C11/C17-C23 можно решить следующими способами:

1) Использование регулярных выражений;

2) Использование абстрактного синтаксического дерева.

В случае использования регулярных выражений, миграция ограничится лишь изменением синтаксиса кода. Например, можно применить такое правило (Таблица 2).

Таблица 2 — Пример правила

|  |
| --- |
| В строке **«\w+ \w+\(\)»** заменить () на (void) |

где **«\w+ \w+\(\)»** означает любую функцию без аргументов (в том числе и некорректную функцию).

Данный способ не эффективен, так как зависит от того насколько правильно составлены регулярные выражения. При этом никак не учитывается семантика кода.

В случае использования абстрактного синтаксического дерева уже становится возможным анализировать семантику выражения. Например, имея функцию без аргументов, дерево может быть представлено так (Рисунок 4).

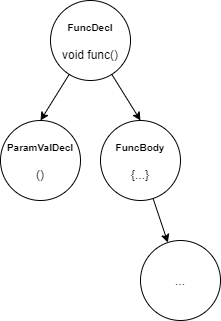


Рисунок 4 — Старая функция без аргументов

Для того чтобы добавить void вместо пустых скобок необходимо изменить соответствующий узел ParamValDecl (Рисунок 5).

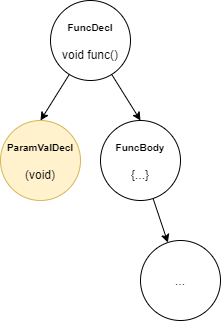


Рисунок 5 — Новая функция без аргументов

В результате было применено такое правило: если в объявлении функции нет аргументов, то добавить ключевое слово void.

Для автоматизации решения задачи миграции кода предлагается спроектировать транскомпилятор[31], который преобразовывает исходное абстрактное синтаксическое дерево, по ранее сформированным правилам, в целевое абстрактное синтаксическое дерево соответствующее указанному стандарту.

## *Формирование правила перехода*

Для формирования правил необходимо проанализировать различия между стандартами. В пункте 2.3 были кратко описаны различия между стандартами и выделены основные направления миграции, теперь же следует изложить их более конкретно и точно (Таблица 3-5).

Таблица 3 — Направление миграции с C89 в C99

|  |  |
| --- | --- |
| **Сравнение C89 и C99** | |
| **С89** | **С99** |
| — | Добавлено ключевое слово inline указывающее на оптимизацию, в месте вызова функции будет подставляться её тело |
| — | Добавлено ключевое слово restrict указывающее компилятору, что на объект ссылается не более одного указателя |
| — | Добавлено ключевое слово \_Bool представляющее новый тип данных — булево значение |
| — | Добавлено ключевое слово \_Complex представляющее новый тип данных — комплексное число |
| — | Добавлено ключевое слово \_Imaginary представляющее собой мнимую часть комплексного числа |
| Результат / и % от отрицательных чисел округляется либо в меньшую, либо в большую сторону | Результат / и % всегда усекается до нуля |
| Знак результата i % j зависит от реализации | Знак результата i % j зависит от знака i |
| В цикле for нельзя объявлять управляющие переменные | В цикле for можно объявлять управляющие переменные |
| — | При инициализации массива можно указывать значения элементов массива |
| — | В макросах можно указываться неопределённое число аргументов |
| — | Добавлен заголовочный файл <stdbool.h> содержащий макросы true и false |
| — | Добавлен заголовочный файл <stdint.h> с объявлением типов фиксированной длины |
| — | Добавлен заголовочный файл <inttypes.h> с макросами размеров типов |
| — | Добавлен заголовочный файл <complex.h> содержащий объявление \_Complex и функции для работы с комплексными числами |
| — | Добавлен заголовочный файл <tgmath.h> содержащий шаблонные функции, упрощающие вызов функций из <math.h> и <complex.h> |
| — | Добавлен заголовочный файл <fenv.h> содержащий функции для работы с окружением вещественных чисел |

Таблица 4 — Направление миграции с C99 в C11-C17

|  |  |
| --- | --- |
| **Сравнение C99 и C11-C17** | |
| **C99** | **C11-C17** |
| — | Добавлены ключевые слова \_Alignas, \_Alignof указывающие на способ выравнивая в памяти. Содержится в заголовочном файле <stdalign.h> |
| — | Добавлено ключевое слово \_Generic позволяющее объединять семантически одинаковые функции под псевдонимом одной. |
| — | Добавлено ключевое слово \_Noreturn указывающее, что функция не будет возвращаться в точку вызова. Содержится в заголовочном файле <stdnoreturn.h> |
| — | Добавлен заголовочный файл <threads.h> для поддержки параллелизма |
| — | Добавлено ключевое слово \_Atomic представляющее атомарный тип данных. Содержится в заголовочном файле <stdatomic.h> |
| — | Добавлена возможность объявлять анонимные структуры и объединения |
| — | Добавлена поддержка Unicode. Соответствующие объявления содержатся в заголовочном файле <uchar.h> |
| — | Добавлены проверки границ строк при использовании функций stract\_s() and strncpy\_s() |
| Функция gets() не проверяет границы | Функция gets() заменена на get\_s() |
| — | Добавлена функция fopen\_s() |

Таблица 5 — Направление миграции с C11-C17 в C23

|  |  |
| --- | --- |
| **Сравнение C11-C17 и C23** | |
| **C11-17** | **C23** |
| — | На данный момент стандарт находится в разработке и имеет статус черновика |

На основании составленной таблицы сформируем правила в формате «условие-действие» (Таблица 6) для различных направлений миграции. В дальнейшем это позволит более точно представить их, например, в виде алгоритмов или блок-схем.

Таблица 6 — Правила преобразований

|  |  |  |
| --- | --- | --- |
| **№** | **Условие** | **Действие** |
| **Направление миграции** | | |
| **С C89 в C99** | | |
| 1 | Функция не имеет аргументов | Добавить void в качестве аргументов |
| 2 | Функция содержит одно выражение return | Добавить inline в объявление функции |
| 3 | Если указатель на объект далее не копируется в блочном выражении | Добавить в объявление указателя restrict |
| 4 | Если значение целочисленной переменной используется как флаг | Заменить тип соответствующей переменной на \_Bool и заменить значения 1 и 0 на true и false соответственно |
| 5 | Если переменные не объявлены в цикле for, а только инициализируются | Добавить инициализацию переменных к инициализации и удалить внешнее объявление |
| 6 | Если после объявления массива, ему задаются начальные значения в формате arr[1] = 1; arr[2] = 2 и т.д. | Добавить при объявлении инициализацию в виде {val\_1, val\_2, ...} |
| **С C99 и C11-C17** | | |
| 7 | В функции нет выражения return и имеет тип void | Заменить тип функции void на \_Noreturn |
| 8 | В файле описано несколько функции с одинаковым и префиксами, но разными типами | Добавить макрос с \_Generic, в котором перечислить соответствующие функции |
| 9 | В выражении используется функция strcat() или strncpy() | Заменить на strcat\_s() и strncpy\_s() соответственно. В качестве размера указать SET\_SIZE, чтобы в дальнейшем программист сам указал размер |
| 10 | В выражении используется функция gets() | Заменить на get\_s(). В качестве размера указать SET\_SIZE, чтобы в дальнейшем программист сам указал размер |
| 11 | В выражении используется fopen() | Заменить на fopen\_s(). Перед вызовом добавить errno\_t err = |

Приведём пример как будет применяться правило 1 для миграции с C89 на C99. Пусть функция не имеет аргументов (Таблица 7). Однако, при её вызове можно указать фактические аргументы в виде литер или переменных.

Таблица 7 — Исходный код функции

|  |  |
| --- | --- |
| **№** | **Код** |
| 1  2  3  4  5 | void func()  {  // code  }  // func(1,2,3,4) <- compilation ok |

На уровне ассемблерного кода перед вызовом функции будут созданы соответствующие аргументы, что в свою очередь представляет лишние операции (Таблица 8). См. строки кода под номерами 10-13.

Таблица 8 — Ассемблерный код программы

|  |  |
| --- | --- |
| **№** | **Код** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | ; Attributes: bp-based frame  ; int \_\_cdecl main(int argc, const char \*\*argv, const char \*\*envp)  public main  main proc near  push rbp  mov rbp, rsp  sub rsp, 20h  call \_\_main  mov r9d, 4  mov r8d, 3  mov edx, 2  mov ecx, 1  call func  nop  add rsp, 20h  pop rbp  retn  main endp |

Для избежания подобного кода и появления лишних операций в ассемблерном коде, необходимо указывать void в списке объявлений аргументов (Таблица 9).

Таблица 9 — Итоговый результат преобразования

|  |  |
| --- | --- |
| **№** | **Код** |
| 1  2  3  4  5 | void func(void)  {  // code  }  // func(1,2,3,4) <- compilation error |

Приведённые примеры являются достаточно искусственными, однако они отражают общий подход к решению задачи. Сформированные правила также требуют более точного и формального определения, так же как и поведения. Например, если применять 11-е правило к коду, в котором последовательно вызывается функция fopen(), то мы получим два одинаковых объявления переменной errno\_t err, что приведёт к ошибке компиляции.

## *Характеристики транскомпилятора*

Задачи стоящие перед транскомпилятором, перед тем как он выполнит преобразование:

1) определение стандарта, которому соответствует исходный код;

2) построение абстрактного синтаксического дерева по исходному коду;

3) обход полученного дерева и применение правил преобразования;

4) генерация кода из полученного целевого дерева.

Для решения первой задачи, на этапе прототипа, будет чётко указываться стандарт.

Для решения второй задачи следует использовать front-end транслятор clang, так как он позволяет получить наиболее полное абстрактное синтаксическое дерево.

Для решения третьей задачи необходимо сформировать правила, которые выполнялись бы в зависимости от условия соответствия (например, если функция не содержит аргументов, то добавить узел с ключевым словом void).

Для решения четвёртой задачи следует также использовать clang, который позволяет сгенерировать код из абстрактного синтаксического дерева.

В прототипе правила преобразований будут чётко прописаны в логике самого транскомпилятора.

Разрабатываемый транскомпилятор должен обладать следующими характеристиками:

— кроссплатформенность. То есть работать в операционных системах Windows, Linux и MacOS X;

— представлять собой консольное приложение, принимающие аргументы командной строки;

— совместимость с front-end транслятором clang. То есть транскомпилятор должен использовать получаемое AST при помощи clang и далее работать с ним, выполняя преобразования.

Аргументами командной строки транскомпилятора должны быть следующие параметры (Таблица 10).

Таблица 10 — Аргументы транскомпилятора

|  |  |
| --- | --- |
| **Аргумент** | **Назначение** |
| -stdIn | Исходный стандарт |
| -stdOut | Целевой стандарт |
| -srcIn | Папка исходных кодов |
| -srcOut | Папка, в которой будет сгенерирован новый код |
| -help | Получение справки по командам |

# Заключение

Миграция кода является важным процессом при работе с унаследованным кодом или большой базой кода, которую необходимо модернизировать. Использование инструментальных средств, позволяющих автоматизировать и облегчить данный процесс дают следующие преимущества:

1) сокращение затрат времени на миграцию кода;

2) сокращение количества потенциальных ошибок;

3) повышение безопасности и надёжности кода;

4) повышения производительности;

5) облегчение дальнейшей поддержки кодовой базы.

Ожидаемые эффекты от использования транскомпилятора, осуществляющего миграцию кода:

1) Исправление проблем связанных с неопределённым поведением;

2) Исправление ошибок, которые были в ранних версиях языка;

2) Улучшение совместимости с C++;

3) Появление новых возможностей.

Конфликт интересов отсутствует.

There is no conflict of interest.

# Список используемых источников

1) Migrate your legacy system to low code (Перенос устаревшей системы на низкий код) [Электронный ресурс]. — URL: <https://bit.ly/3mpFTgF> (Дата обращения: 24.08.2021);

2) Legacy Software Migrations (Миграция устаревшего программного обеспечения) [Электронный ресурс]. — URL: <https://bit.ly/3zer1VX> (Дата обращения: 24.08.2021);

3) Common Data Migration Problems (Общие проблемы миграции данных) [Электронный ресурс]. — URL: <https://bit.ly/3yi70MF> (Дата обращения: 24.08.2021);

4) Five biggest challenges of software migration projects (Пять самых больших проблем проектов миграции программного обеспечения) [Электронный ресурс]. — URL: <https://bit.ly/3sGaffY> (Дата обращения: 24.08.2021);

5) Data Migration Challenges (And The Techniques To Solve Them) (Проблемы миграции данных (и методы их решения) [Электронный ресурс]. — URL: <https://bit.ly/3B7H3kZ> (Дата обращения: 24.08.2021);

6) Seven reasons why system migrations fail — and how you can avoid them (Семь причин, по которым системные миграции завершаются неудачно, и как их можно избежать) [Электронный ресурс]. — URL: <https://bit.ly/3gv0xYN> (Дата обращения: 24.08.2021);

7) Reasons why you should update your software (Причины, по которым вам следует обновить свое программное обеспечение) [Электронный ресурс]. — URL: <https://bit.ly/3878K0T> (Дата обращения: 24.08.2021);

8) Как мы перевели 10 миллионов строк кода C++ на стандарт C++14 (а потом и на C++17) [Электронный ресурс]. — URL: <https://bit.ly/3mHD3Uv> (Дата обращения: 24.08.2021);

9) Миграция баз данных: зачем и почему [Электронный ресурс]. — URL: <https://bit.ly/3B9dGig> (Дата обращения: 24.08.2021);

10) HipHop for PHP (Хип-хоп для PHP) [Электронный ресурс]. — URL: <https://bit.ly/2XHggNR> (Дата обращения: 24.08.2021);

11) Liliana Martinez, Claudia Pereira, Liliana Favre Migration C/C++ Software to Mobile Platforms in the ADM Context (Лилиана Мартинес, Клаудия Перейра, Лилиана Фавре Перенос программного обеспечения C/C++ на мобильные платформы в ADMContext) // International Journal of Interactive Multimedia and Artificial Intelligence Vol. 4. — 2017. — №3. — P. 33 — 44;

12) Index | TIOBE — The Software Quality Company (Индекс | TIOBE — Компания по Качеству Программного Обеспечения) [Электронный ресурс]. — URL: <https://bit.ly/3mLBapt> (Дата обращения: 25.08.2021);

13) PYPL PopularitY of Programming Languages Index (Индекс популярности языков программирования PYPL) [Электронный ресурс]. — URL: <https://bit.ly/38mf5FI> (Дата обращения: 25.08.2021);

14) Top Computer Languages 2021 (Лучшие компьютерные языки 2021 года) [Электронный ресурс]. — URL: <https://bit.ly/3mLBh4n> (Дата обращения: 25.08.2021);

15) The Top C Open Source Projects (Лучшие проекты с открытым исходным кодом) [Электронный ресурс]. — URL: <https://bit.ly/3zubDF0> (Дата обращения: 25.08.2021);

16) Linux kernel source tree (Дерево исходных текстов ядра Linux) [Электронный ресурс]. — URL: <https://bit.ly/38pvDww> (Дата обращения: 25.08.2021);

17) Git Source Code Mirror (Зеркало исходного кода Git) [Электронный ресурс]. — URL: <https://bit.ly/3mHKNWr> (Дата обращения: 25.08.2021);

18) Redis (Редис) [Электронный ресурс]. — URL: <https://bit.ly/38r6q54> (Дата обращения: 25.08.2021);

19) The PHP Interpreter (Интерпретатор PHP) [Электронный ресурс]. — URL:<https://bit.ly/2YaW3QT> (Дата обращения: 25.08.2021);

20) Official Git mirror of the SQLite source tree (Официальное зеркало Git дерева исходных текстов SQLite) [Электронный ресурс]. — URL: <https://bit.ly/3h3izBZ> (Дата обращения: 25.08.2021);

21) History of C — cppreference.com (История С — cppreference.com) [Электронный ресурс]. URL: <https://bit.ly/3yjFXRs> (Дата обращения: 25.08.2021);

22) Extensions for parallel programming (Расширения для параллельного программирования) [Электронный ресурс]. — URL: <https://bit.ly/3kmKFsy> (Дата обращения: 25.08.2021);

23) Transactional Memory Support of C (Поддержка транзакционной памяти C) [Электронный ресурс]. — URL: <https://bit.ly/2Wl68tz> (Дата обращения: 24.08.2021);

24) C89 vs C99 (C89 против C99) [Электронный ресурс]. — URL: <https://cw.fel.cvut.cz/old/_media/courses/be5b99cpl/lectures/be5b99cpl-lec10-handout-3x3.pdf> (Дата обращения: 24.08.2021);

25) Clarification Request Summary for C11 (Резюме запроса на разъяснение для C11) [Электронный ресурс]. — URL: <https://bit.ly/3DojOow> (Дата обращения: 24.08.2021);

26) Clarification Request Summary for C2x (Резюме запроса на разъяснение для C2x) [Электронный ресурс]. — URL: <https://bit.ly/3zrCWQs> (Дата обращения: 24.08.2021);

27) How migration legacy systems improves performance and adaptability (Как миграция устаревших систем повышает производительность и адаптивность) [Электронный ресурс]. URL: <https://bit.ly/2WdGhEo> (Дата обращения: 24.08.2021);

28) PortAssist: Visual Analysis for Porting Large Code Bases (Помощь в переносе: Визуальный анализ для переноса больших баз кода) [Электронный ресурс]. — URL: <https://bit.ly/3mqY31x> (Дата обращения: 17.08.2021);

29) CodeCheck Program Overview (Обзор программы проверки кода) [Электронный ресрус]. — URL: <https://bit.ly/38cYvbr> (Дата обращения: 17.08.2021);

30) DMS Software Reengineering Toolkit (Инструментарий для реинжиниринга программного обеспечения DMS) [Электронный ресрус]. — URL: <https://bit.ly/3znSbtr> (Дата обращения: 24.08.2021);

31) Source-to-source compiler — Wikipedia (Компилятор от источника к источнику — Wikipedia) [Электронный ресурс]. — URL: <https://bit.ly/3zk5T0A> (Дата обращения: 24.08.2021).

**Поступила \_\_\_\_\_\_**

**УДК 004.4**

**REFERENCES**

1) Migrate your legacy system to low code [Electronic resource]. — URL: <https://bit.ly/3mpFTgF> (Date of request: 24.08.2021);

2) Legacy Software Migrations [Electronic resource]. — URL: <https://bit.ly/3zer1VX> (Дата обращения: 24.08.2021);

3) Common Data Migration Problems [Electronic resource]. — URL: <https://bit.ly/3yi70MF> (Date of request: 24.08.2021);

4) Five biggest challenges of software migration projects [Electronic resource]. — URL: <https://bit.ly/3sGaffY> (Date of request: 24.08.2021);

5) Data Migration Challenges (And The Techniques To Solve Them) [Electronic resource]. — URL: <https://bit.ly/3B7H3kZ> (Date of request: 24.08.2021);

6) Seven reasons why system migrations fail — and how you can avoid them [Electronic resource]. — URL: <https://bit.ly/3gv0xYN> (Date of request: 24.08.2021);

7) Reasons why you should update your software [Electronic resource]. — URL: <https://bit.ly/3878K0T> (Date of request: 24.08.2021);

8) How we translated 10 million lines of C++ code to the C++14 standard (and then to C++17) [Electronic resource]. — URL: <https://bit.ly/3mHD3Uv> (Date of request: 24.08.2021) (In Russian)

9) Database migration: why and why [Electronic resource]. — URL: <https://bit.ly/3B9dGig> (Date of request: 24.08.2021) (In Russian)

10) HipHop for PHP [Electronic resource]. — URL: <https://bit.ly/2XHggNR> (Date of request: 24.08.2021);

11) Liliana Martinez, Claudia Pereira, Liliana Favre Migration C/C++ Software to Mobile Platforms in the ADM Context // International Journal of Interactive Multimedia and Artificial Intelligence Vol. 4. — 2017. — №3. — P. 33 — 44;

12) Index | TIOBE — The Software Quality Company [Electronic resource]. — URL: <https://bit.ly/3mLBapt> (Date of request: 25.08.2021);

13) PYPL PopularitY of Programming Languages Index [Electronic resource]. — URL: <https://bit.ly/38mf5FI> (Date of request: 25.08.2021);

14) Top Computer Languages 2021 [Electronic resource]. — URL: <https://bit.ly/3mLBh4n> (Date of request: 25.08.2021);

15) The Top C Open Source Projects [Electronic resource]. — URL: <https://bit.ly/3zubDF0> (Date of request: 25.08.2021);

16) Linux kernel source tree [Electronic resource]. — URL: <https://bit.ly/38pvDww> (Date of request: 25.08.2021);

17) Git Source Code Mirror [Electronic resource]. — URL: <https://bit.ly/3mHKNWr> (Date of request: 25.08.2021);

18) Redis [Electronic resource]. — URL: <https://bit.ly/38r6q54> (Date of request: 25.08.2021);

19) The PHP Interpreter [Electronic resource]. — URL:<https://bit.ly/2YaW3QT> (Date of request: 25.08.2021);

20) Official Git mirror of the SQLite source tree [Electronic resource]. — URL: <https://bit.ly/3h3izBZ> (Date of request: 25.08.2021);

21) History of C — cppreference.com [Electronic resource]. URL: <https://bit.ly/3yjFXRs> (Date of request: 25.08.2021);

22) Extensions for parallel programming [Electronic resource]. — URL: <https://bit.ly/3kmKFsy> (Date of request: 25.08.2021);

23) Transactional Memory Support of C [Electronic resource]. — URL: <https://bit.ly/2Wl68tz> (Date of request: 24.08.2021);

24) C89 vs C99 [Electronic resource]. — URL: <https://cw.fel.cvut.cz/old/_media/courses/be5b99cpl/lectures/be5b99cpl-lec10-handout-3x3.pdf> (Date of request: 24.08.2021);

25) Clarification Request Summary for C11 [Electronic resource]. — URL: <https://bit.ly/3DojOow> (Date of request: 24.08.2021);

26) Clarification Request Summary for C2x [Electronic resource]. — URL: <https://bit.ly/3zrCWQs> (Date of request: 24.08.2021);

27) How migration legacy systems improves performance and adaptability [Electronic resource]. URL: <https://bit.ly/2WdGhEo> (Date of request: 24.08.2021);

28) PortAssist: Visual Analysis for Porting Large Code Bases [Electronic resource]. — URL: <https://bit.ly/3mqY31x> (Date of request: 17.08.2021);

29) CodeCheck Program Overview [Electronic resource] — URL: <https://bit.ly/38cYvbr> (Date of request: 17.08.2021);

30) DMS Software Reengineering Toolkit [Electronic resource]. — URL: <https://bit.ly/3znSbtr> (Date of request: 24.08.2021);

31) Source-to-source compiler — Wikipedia [Electronic resource]. — URL: <https://bit.ly/3zk5T0A> (Date of request: 24.08.2021).

**Received\_\_\_\_**

**UDC 004.942**

**Авторы:**

**Бирюкова Анна Алексеевна** – кандидат технических наук, доцент, инженер-исследователь 2-й категории, отдел тестирования информационного обеспечения Управления информационного взаимодействия Научно-технического центра перспективных технологий информационных процессов Федерального государственного автономного научного учреждения «Центр информационных технологий и систем органов исполнительной власти» (ФГАНУ «ЦИТиС»), доцент кафедры Математического обеспечения и стандартизации информационных технологий, Федерального государственного бюджетного образовательного учреждения высшего образования «МИРЭА ‑ Российский технологический университет» (РТУ МИРЭА).

**Biryukova Anna Alekseevna** – candidate of Engineering Sciences, docent, research engineer of 2nd category, information support testing department of the information interaction directorate of the scientific and technical center for advanced technologies of information processes, Federal Autonomous State Research Institution «Centre of Information Technologies and Systems for Executive Power Authorities», associate professor at the Department of Mathematics and ensure standardization of information technology, Federal State Budget Educational Institution of Higher Education «MIREA ‑ Russian Technological University» (RTU MIREA), e-mail: [biryukova@mirea.ru](mailto:biryukova@mirea.ru)

**Басыров Сергей Амирович** – студент Федерального государственного бюджетного образовательного учреждения высшего образования «МИРЭА ‑ Российский технологический университет» (РТУ МИРЭА).

**Basyrov Sergey Amirovich** –student Federal State Budget Educational Institution of Higher Education «MIREA ‑ Russian Technological University» (RTU MIREA), e-mail: [pmswga@gmail.com](mailto:pmswga@gmail.com)