Федеральное государственное бюджетное образовательное учреждение
высшего образования «Саратовский государственный технический
университет имени Гагарина Ю.А.»

Институт прикладных информационных технологий и коммуникаций
Кафедра «Прикладные информационные технологии»

КУРСОВАЯ РАБОТА
по дисциплине «Методы и средства проектирования информационных систем и технологий»
на тему «Проектирование структуры информационной системы»

Выполнил студент группы
б1 ПИНФ-41
Нефедов Данил Вадимович
Проверил д.ф.-м.н., доцент, профессор кафедры ПИТ,
Кондратов Д.В.
Комиссия по защите:
д.ф.-м.н., доцент, профессор каф. ПИТ, Кондратов Д.В.
ассистент каф. ПИТ, Кулакова Е.М.

Курсовая работа защищена на оценку «_____»

_____
(дата, подпись члена комиссии)
_____
(дата, подпись члена комиссии)

Саратов 2021

Замечания

_____ / Кондратов Д. В.

(дата, подпись члена комиссии)

_____ / Кулакова Е. М.

(дата, подпись члена комиссии)

Федеральное государственное бюджетное образовательное учреждение высшего образования «Саратовский государственный технический университет имени Гагарина Ю.А.»

Кафедра «Прикладные информационные технологии»

ЗАДАНИЕ
на выполнение курсовой работы
по дисциплине «Проектирование структуры информационной системы»

студенту ИнПИТ группы б1 ПИНФ-41

В курсовой работе необходимо:

1. Создать пояснительную записку к информационной системе.

Пояснительная записка должна содержать:

1. Словесное описание предметной области.
2. Описание предметной области с помощью методологий IDEF0, IDEF3, DFD и языка UML.
3. Выбор средств разработки.
4. Оценивание стоимости создания информационной системы.
5. Техническое задание.

Дата выдачи: 7 сентября 2021 г.

Срок выполнения: 14 декабря 2021 г.

Руководитель: _____ д.ф.-м.н., доцент,

профессор каф. ПИТ, Кондратов Д.В.

Студент: _____ Нефедов Д.В.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компиляторы — это критические важные программные системы. Без компиляторов было бы невозможно представить себе сегодняшний мир, ведь написание любой сколько-нибудь сложной компьютерной программы требует использования языков программирования высокого уровня.

Такие языки избавляют программистов от трудоемкости и сложности написания программного кода на языках ассемблера, которые требуют от программиста детального знания устройства ISA конкретного процессора [1]. Языки программирования высокого уровня позволяют абстрагироваться от деталей конкретной архитектуры и сфокусироваться на решение задачи, которая стоит перед разрабатываемой компьютерной программой.

Кроме того, использование ЯП высокого уровня позволяет допускать гораздо меньше ошибок по сравнению с языками ассемблера. В предотвращении ошибок помогает встроенные в компилятор проверки синтаксиса и семантики. Это особенно актуально для строго типизированных языков программирования.

Зачастую компиляторы генерируют гораздо более эффективный ассемблерный код, чем смог бы написать программист [2]. Это не удивительно, ведь написание эффективного ассемблерного кода требует от программиста глубочайшего понимания архитектуры процессора, под который пишется программа, внимание к мельчайшим деталям, стоимость выполнения каждой инструкций в микрооперациях и многое другое. Хорошо известного, что человеку тяжело работать с большим объемом чисел и просчитывать мельчайшие детали. Для такой работы и были придуманы компьютеры, а значит именно у них получится генерировать более эффективный низкоуровневый код, чем мог бы написать программист.

К сожалению, компиляторы сами по себе являются сложнейшими компьютерными программами, которые состоят из десятков модулей, требуют

знания в самых различных областях информатики [3]. Оптимизирующие компиляторы требуют особых навыков, так как разработка алгоритмов оптимизации находится на рубежах науки и людей, которые могли бы разработать и реализовать определенный алгоритм оптимизации в компиляторе не так уж и много.

Более того, так как компиляторы — это огромные компьютерные программы, состоящие из десятков миллионов строк кода, то столь же мало людей, которые разбираются в кодовой базе, причем в основном каждый разработчик разбирается только в своем модуле, не понимая ничего, либо крайне мало о других модулях компилятора.

Для примера, в компиляторе GCC по состоянию на 5 января 2015 года было чуть более 14,5 миллионов строк кода [4]. Что интересно, код компилятора GCC был настолько сложен, что в нем существовал модуль под названием «reload», функционал которого до сих пор не до конца понятен самим разработчикам GCC [5].

Чрезвычайная сложность компиляторов представляет большую проблему для студентов: нет простых и понятных примеров, на которые можно было бы взглянуть. Учебники по разработке компиляторов порой слишком обстоятельны и формальны, что усложняет усвоение информации — достаточно вспомнить классическую «Книгу дракона».

Одним из возможных решений данной проблемы является создание простого и понятного компилятора, в исходном коде и архитектуре которого легко разобраться. Наша АИС направлена на решение этой проблемы.

1 Описание предметной области

1.1 Цель курсовой работы

Целью курсовой работы являются закрепление теоретических знаний, получение практических навыков и новых знаний в области проектирования информационных систем.

Цель работы достигается в результате участия в решении практических задач и проблем, возникающих при разработке информационной системы компилятора языка C.

1.2 Словесное описание предметной области

Информационная система компилятора языка C предназначена для образовательных целей и предназначена для обучения студентов и прочих желающих внутреннему устройству и построению в частности компиляторов, а в общем случае — программ-трансляторов с одного языка программирования в другой.

Система ориентирована в первую очередь на студентов технических направлений. Техническое задание по разработанной системе находится в Приложении А.

2 Уточнение описания предметной области

2.1 IDEF0

IDEF0 является методологией функционального моделирования. Она используется для создания функциональной модели, которая отображает структуру и функции системы, а также потоки информации и материальных объектов, связывающих эти функции. Компонентами синтаксиса IDEF0 являются блоки, стрелки, диаграммы и правила [6].

Диаграмма IDEF0 состоит из *блоков* и *стрелок*, направление которых заданы правила. Блоки изображаются в виде прямоугольников и содержат в себе описание той функции, которую они представляют, и номер. Номер блока должен быть расположен в правом нижнем углу.

К каждому блоку должно вести несколько стрелок, такие как:

— *левые стрелки* (направлены в блок) обозначают информацию или продукты, которую функция получает на входе;

— *правые стрелки* (направлены из блока) обозначают информацию или продукты, которую функция дает на выходе;

— *верхние стрелки* (направлены в блок) обозначают документы, которые регламентируют работы системы;

— *нижние левые стрелки* (направлены в блок) обозначают механизмы, которые влияют на работу нашей системы;

— *нижние правые стрелки* (направлены из блока) обозначают обращение к дополнительной информационной системе, которая существует совершенно отдельно от нашей и которая необходима для осуществления процесса или функции. Может как присутствовать на схеме, так и нет.

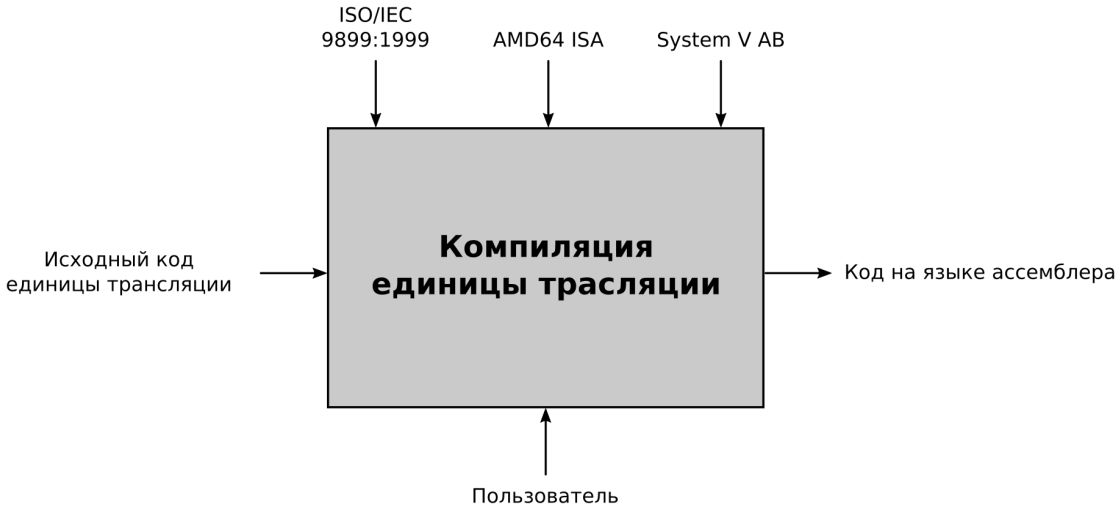

Рисунок 2.1 – Комплексная диаграмма IDEF0

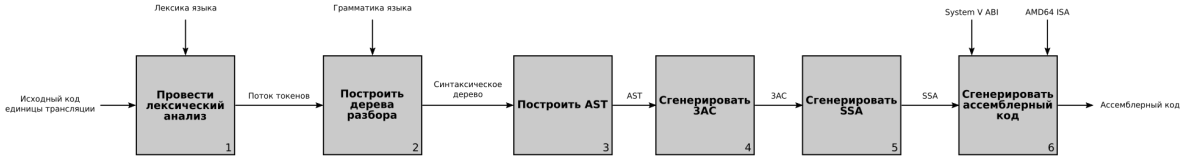

Рисунок 2.2 – Декомпозиция комплексной диаграммы IDEF0

2.2 IDEF3

Методология IDEF3 —- это методология описания процессов. В отличие от IDEF0 она рассматривает последовательность выполнения задач, а также причинно-следственные связи между ними [7]. Диаграммы IDEF3 состоят из блоков, описывающих функции, стрелок-связей и перекрёстков, которые показывают, как именно выполняются процессы.

Блок (функциональный элемент) представляет собой прямоугольник, разделенный на три другие прямоугольника: один большой наверху и два маленьких друг рядом с другом внизу. В верхнем прямоугольнике содержится

имя функции, в нижнем левом номер его выполнения, в нижнем правом, при необходимости, находится ссылка на другую функцию. Связи бывают простыми, относительными и связями с условием.

Перекрёстки подразделяются на следующие типы: *И*, *ИЛИ*, *синхронное И*, *синхронное ИЛИ*, а также *исключающее ИЛИ*.

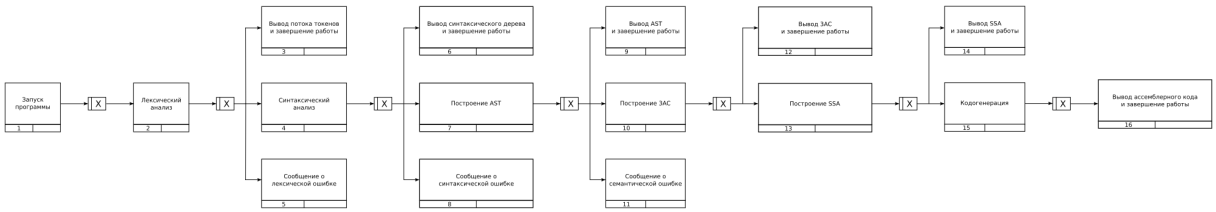

Рисунок 2.3 – IDEF3 диаграмма

2.3 DFD

Диаграммы потоков данных (англ. *DFD*) —- это способ представления процессов обработки информации. Они показывают, как информация перемещается из одной функции к другой. Подобное представление потока данных отражает движение объектов, их хранение и распространение.

DFD состоит из следующих компонентов: внешняя сущность, процесс, поток данных и хранилище данных [8]. Внешняя сущность представляет собой источник или приёмник информации и изображается прямоугольником с прямыми углами. Процессы в DFD – это функции системы, преобразующие входы и выходы, которые изображаются как прямоугольники со скругленными углами. Потоки данных изображаются стрелками. Если стрелка соединяет какую-либо функцию с хранилищем данных, то на ней должно быть отображено имя, отражающее содержание данного потока. Хранилище данных является прообразом базы данных и изображается как прямоугольник без правой стороны.

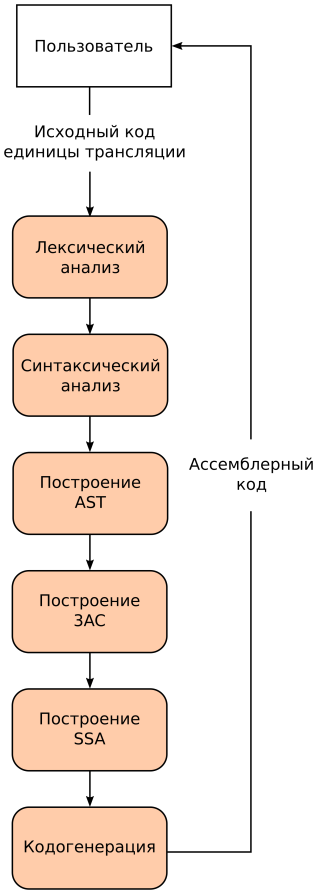

Рисунок 2.4 – DFD диаграмма

2.4 UML

UML является унифицированным языком моделирования, объекты в котором это упрощенное представление предметов окружающего нас мира [9]. Его используют при разработке ПО для моделирования бизнес-процессов, системного проектирования и отображения организационных структур [10]. UML диаграммы подразделяются на два типа: структурные и поведенческие.

2.4.1 Диаграмма пакетов

Диаграмма пакетов — это структурная диаграмма UML, которая показывает структуру проектируемой системы на уровне пакетов [11]. Обычно на диаграмме изображаются следующие элементы: пакет, пакетированный элемент, зависимость, импортируемый элемент, импорт пакета, объединение пакета.

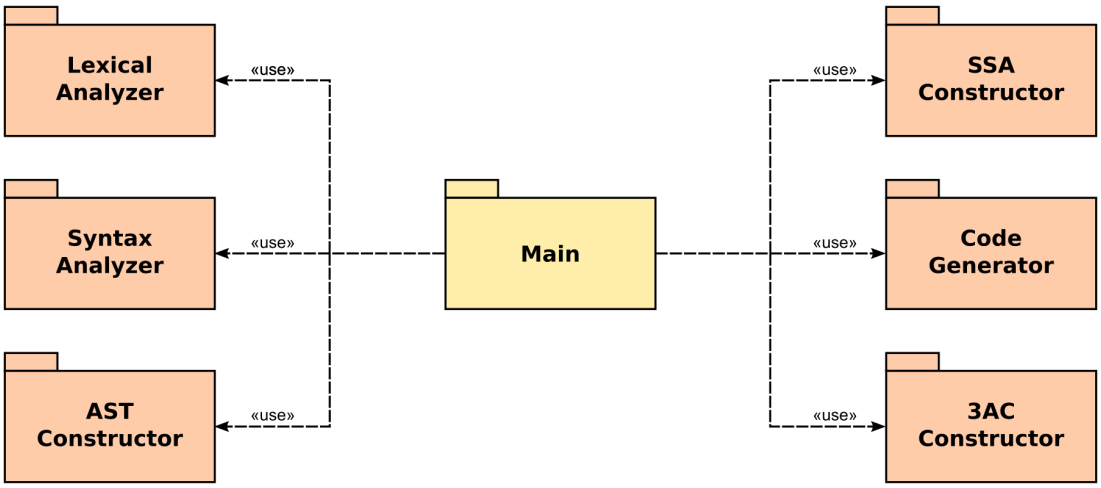

Рисунок 2.5 – Диаграмма пакетов UML

2.4.2 Диаграмма компонентов

Диаграмма компонентов показывает компоненты, предоставляемые и требуемые интерфейсы, порты и отношения между ними [12]. Этот тип диаграмм используется в компонентно-ориентированной разработке для описания систем с сервисно-ориентированной архитектурой [13].

Компонентно-ориентированная разработка основана на предположения, что ранее сконструированные компоненты могут быть переиспользованы или, при необходимости, заменены на некие “эквивалентные” или “совместимые” компоненты.

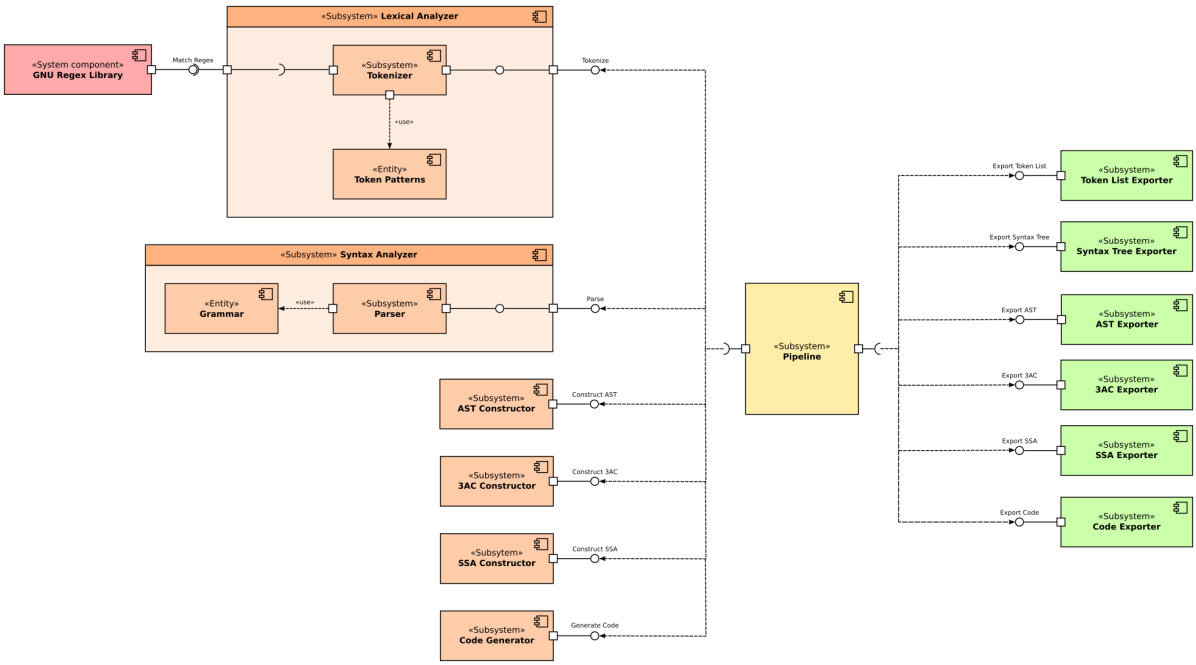

Рисунок 2.6 – Диаграмма компонентов UML

2.4.3 Диаграмма потока информации

Диаграмма потока информации — это поведенческая диаграмма UML, которая показывает обмен информацией между сущностями системы на высоком уровне абстракции. Потоки информации могут быть полезны для описания циркуляции информации через систему через представление аспектов модели, которые еще не полностью специфицированы или недостаточно детализированы.

Потоки информации не показывают природу информации, механизмы передачи, порядок обмена или какие-либо контрольные условия [14]. Единицы информации могут быть использованы для представления информации, которая протекает через систему вместе с информационными потоками еще прежде, чем их реализация будет проработана.

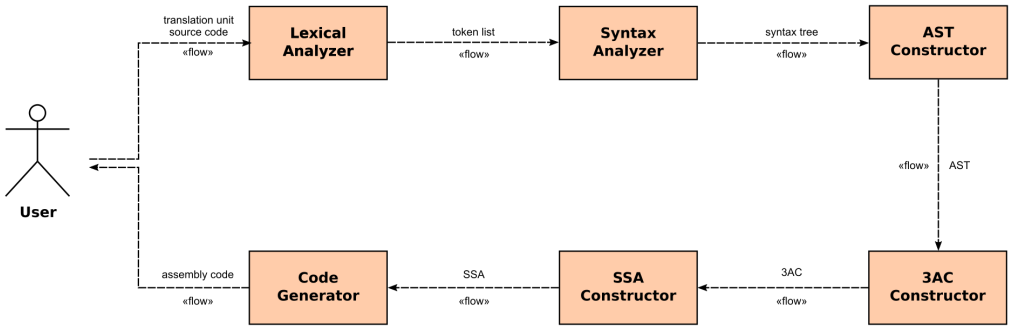

Рисунок 2.7 – Диаграмма потока информации UML

3 Выбор средств разработки

Так как для компилятора требуется возможность скомпилировать самого себя, то выбор реализации языка становится очевиден — это язык C стандарта ISO/IEC 9899:1999.

Кроме того, из соображений минимизации зависимостей для более полного понимания всего процесса компиляции единственной зависимостью компилятора будет использование функций GNU C Library, реализующих POSIX-совместимые регулярные выражения, которые не входят в используемый стандарт языка.

В рамках реализации компилятора не ставится задача реализации собственного движка регулярных выражений, так как это довольно простая математическая модель (регулярные языки), разработка которых лишь усложнит код компилятора, который мы стараемся сделать максимально простым.

Для облегчения процесса сборки приложения в качестве средств систем сборки будет использован такой инструмент как GNU Make.

В качестве компилятора может быть использован любой компилятор, который поддерживает стандарт ISO/IEC 9899:1999. Более того, использование разных компиляторов позволит убедится, что исходный код компилятора соответствует используемому стандарту.

4 Оценка стоимости работ

Варианты использования описывают основные процедуры работы с системой, поэтому путём их анализа можно определить сложность системы, а, значит, и затраты, необходимые для её разработки. Во многих случаях исследование вариантов использования проще, чем поиск функциональных точек, необходимых для работы других методов оценивания затрат, таких как COCOMO II и Functional Points.

Сложность системы при этом определяется следующими факторами:

— количество и сложность действующих лиц, участвующих в варианте использования;

— дополнительные требования, такие как многозадачность, безопасность и производительность;

— внешние факторы, такие как опыт членов команды разработчиков.

В качестве единицы сложности системы используется прецедентная точка (Use-Case Point, UCP), количество таких точек для каждого варианта использования определяется по формуле:

$$\mathrm{UCP} = \mathrm{TCP} \times \mathrm{ECF} \times \mathrm{UUCP} \times \mathrm{PF} \tag{1}$$

— UCP – количество прецедентных точек;

— TCP – коэффициент технической сложности;

— ECF – коэффициент сложности взаимодействия с окружающей средой;

— PF – эффективность работы разработчиков;

— UUCP – количество нескорректированных прецедентных точек.

Значение UUCP определяется числом шагов, необходимых для выполнения варианта использования, действующими лицами каждого варианта и общим количеством, и сложностью вариантов использования рассматриваемом варианте использования 3 шага. Сложность варианта использования определяется в соответствие со следующими правилами:

— *простой* (5 баллов) – предполагается простой пользовательский интерфейс и работа не более, чем с одной сущностью базы данных; количество шагов не превышает 3, а в реализации задействовано не более 5 классов;

— *средней сложности* (10 баллов) – более сложный пользовательский интерфейс, работа с двумя или более сущностями базы данных; количество шагов составляет от 4 до 7, в реализации задействовано 5-10 классов;

— *сложный* (15) – сложная обработка запросов пользователя, работа с тремя и более сущностями базы данных; количество шагов превышает 7, количество классов в реализации более 10.

Варианты использования:

— *простой*: выполнение лексического анализа единицы трансляции; выполнение синтаксического анализа, вывод диагностических сообщений о лексических ошибках в программе;

— *средней*: построение AST по дереву разбора, вывод диагностических сообщений о синтаксических и лексических ошибках;

— *сложный*: построение трехадресного кода, построение SSA, кодогенерация кода под целевую платформу.

$$\mathrm{UUCW} = 3 \times 5 + 2 \times 10 + 3 \times 15 = 80 \tag{2}$$

Действующие лица также разделяются на три класса по следующим правилам:

— *простое действующее лицо* (1 балл) – сторонняя система, обращающаяся к нашей системе через программный интерфейс;

— *действующее лицо средней сложности* (2 балла) – сторонняя система, обращающаяся к нашей по сети;

— *сложное действующее лицо* (3 балла) – пользователь системы.

После этого подсчитывается количество действующих лиц в каждом классе, количество умножается на соответствующие баллы. В результате получается нескорректированный вес действующих лиц (UAW). В нашем варианте одно сложное действующее лицо — пользователь.

$$\mathrm{UAW} = 1 \times 3 = 3. \tag{3}$$

Значение UUCP вычисляется по формуле:

$$\mathrm{UUCP} = \mathrm{UUCW} + \mathrm{UAW} = 80 + 3 = 83. \tag{4}$$

Для вычисления TCF сначала каждому фактору присваивается вес в зависимости от того, насколько сильно его влияние на систему (от 0 до 5). После этого оценивается сложность, связанная с данным фактором (также от 0 до 5). Полученные значения перемножаются, результаты складываются и получают значение PreTCF (см. таблицу 1).

Таблица 1 – Получение значения PreTCF

Описание	Вес	Сложность	Вес × сложность
Распределённая система	0	5	0
Производительность	0	4	0
Эффективность взаимодействия с пользователем	5	3	15
Сложные алгоритмы обработки данных	5	5	25
Возможность повторного использования	5	5	25
Простота установки	5	2	10
Простота использования	5	3	15
Переносимость	0	5	0
Простота модификации	5	5	25
Многозадачность	0	5	0
Безопасность	0	4	0
Открытость для сторонних приложений	5	4	20
Необходимость специальных навыков пользователей	5	1	5
PreTCF			144

Значение TCF вычисляется по формуле:

$$\mathrm{TCF} = 0.6 + 0.01 \times \mathrm{PreTCF} = 0.6 + 1.44 = 2.04. \quad (5)$$

Аналогичным образом вычисляется значение PreECF (см. таблицу 2).

Таблица 2 – Получение значения PreECF

Описание	Вес	Сложность	Вес × сложность
Знакомство с UML	1	1	1
Знакомство с предметной областью	4	5	20
Знакомство с объектно-ориентированными технологиями	3	2	6
Опыт ведущего аналитика	2	3	6
Мотивация	2	3	6
Постоянство требований	1	1	1
Наличие совместителей	0	3	0
Трудный язык программирования	2	3	6
PreECF			46

Значение ECF вычисляется по формуле:

$$\mathrm{ECF} = 1.4 + (-0.03 \times \mathrm{PreECF}) = 1.4 - 1.38 = 0.02. \tag{6}$$

Значение PF определяется как количество человеко-часов, необходимых для реализации одного варианта использования. Данное значение определяется из опыта выполнения предыдущих проектов. Если такой опыт отсутствует, то берётся число из интервала от 15 до 30, обычно 20. Тогда:

$$\mathrm{UCP} = 83 \times 2.04 \times 0.02 \times 20 \approx 67.60. \tag{7}$$

Получили, что для реализации системы потребуется 67.60 человеко-часов.

ЗАКЛЮЧЕНИЕ

Таким образом, по итогу проделанной работы сделано следующее:

— Изучена предметная область, проведен её анализ, определены цели и задачи системы.

— Сформулировано словесное описание разрабатываемой информационной системы.

— Изучены графические нотации методологий IDEF0, IDEF3, DFD и UML, их принципы построения, а также построены соответствующие модели, описывающие работу данной системы.

— Произведено оценивание стоимости создания информационной системы и выявлено, сколько потребуется человеко-часов для создания данной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *London K.* Introduction to Computers. — 24 Russell Square London WC1 : Faber, Faber Limited, 1968. — C. 186. — ISBN 0571085938.
2. *Hyde R.* The art of assembly language. — 2-е изд. — San Francisco : No Starch Press, 2010. — ISBN 9781593273019.
3. *Aho A. V.*, *Sethi R.*, *Ullman J. D.* Compilers: Principles, Techniques, and Tools. — Reading, Massachusetts : Addison-Wesley, 1986. — C. 585. — ISBN 0-201-10088-6.
4. *Larabel M.* GCC Soars Past 14.5 Million Lines Of Code & I'm Real Excited For GCC 5 [Электронный ресурс]. — 01.2015. — URL: `https://www.phoronix.com/scan.php?page=news_item%5C&px=MTg3OTQ` (дата обр. 14.12.2021).
5. reload [Электронный ресурс] / J. Wakely [и др.]. — 08.2017. — URL: `https://gcc.gnu.org/wiki/reload` (дата обр. 14.12.2021).
6. *U.S. Department of Defense*. Systems Engineering Fundamentals. — Fort Belvoir, Virginia : Defense Acquisition University Press, 01.2001.
7. *Harzallah M.* Incorporating IDEF3 into the Unified Enterprise Modelling Language // Proceedings of the 2007 Eleventh International IEEE EDOC Conference Workshop. — 2007. — C. 133—140.
8. *Gane C.*, *Sarson T.* Structured Systems Analysis: Tools and Techniques. — New York : Improved Systems Technologies, 1977. — C. 373. — ISBN 978-0930196004.
9. *Hunt J.* The Unified Process for Practitioners: Object-oriented Design, UML and Java. — Springer, 2000. — C. 25. — ISBN 1-85233-275-1.
10. *Fowler M.* UML Distilled: A Brief Guide to the Standard Object Modeling Language. — 3-е изд. — Addison-Wesley, 2004. — ISBN 0-321-19368-7.

11. *Martin R. C.* UML for Java Programmers. — Prentice Hall, 2003. — ISBN 0-13-142848-9.

12. *Буч Г.*, *Рамбо Д.*, *Якобсон И.* Краткая история UML. — 2-е изд. — Москва : ДМК Пресс, 2006. — С. 14. 496 с. — ISBN 5-94074-334-X.

13. *Douglass B.* Real-Time UML. — 3-е изд. — Newnes, 2004. — ISBN 9780321160768.

14. *Penker M.*, *Eriksson H.-E.* Business Modeling with UML. — John Wiley & Sons, 2000. — ISBN 0-471-29551-5.

ПРИЛОЖЕНИЕ А

САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ ГАГАРИНА Ю.А.

УТВЕРЖДАЮ	УТВЕРЖДАЮ
Руководитель (должность, наименование предприятия — заказчика АС)	Руководитель (должность, наименование предприятия — заказчика АС)
Личная подпись Расшифровка подписи	Личная подпись Расшифровка подписи
Печать	Печать
Дата	Дата

Программного обеспечения «Компилятор языка С»

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

На 21 листе

Действует с «___» _____ 2021 г.

СОГЛАСОВАНО

Руководитель (должность, наименование согласующей организации)

Личная подпись Расшифровка подписи

Печать

Дата

Саратов 2021

1 ОБЩИЕ ПОЛОЖЕНИЯ

1.1 Полное наименование системы и ее условное обозначение

Полное наименование системы: «Автоматизированная информационная система “Компилятор языка С”».

Краткое наименование системы: АИС «Компилятор».

1.2 Номер договора (контракта)

Шифр темы: АИС-КА-ФА-07.

Номер контракта: №1/11-11-11-001 от 29.09.2021.

1.3 Наименования организации-заказчика и организаций-участников работ

Заказчиком системы является Саратовский Государственный технический университет имени Гагарина Ю.А. Адрес заказчика: 410054, г. Саратов, ул. Политехническая, д. 77.

Разработчиком системы является Нефедов Данил Вадимович. Адрес разработчика: 415011, г. Саратов, ул. Московская, д. 27.

1.4 Перечень документов, на основании которых создается система

Основанием для разработки компилятора «*CC99*» являются следующие документы и нормативные акты:

— ISO/IEC 9989:1999. Programming languages — C.

— Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1: Basic Architecture.

— The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D: Instruction Set Reference.

— The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C & 3D: System Programming Guide.

— The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers.

— System V Application Binary Interface. AMD64 Architecture Processor Supplement (With LP64 and ILP32 Programming Models).

— IEEE Std 1003.1-2017.

— ISO/IEC JTC 1 Working Group. Rationale for International Standard — Programming Languages — C.

— Documentation for GNU binutils 2.37.

1.5 Плановые сроки начала и окончания работы по созданию системы

Плановый срок начала работ по созданию АИС «Компилятор» — 29 сентября 2021 года.

Плановый срок окончания работ по созданию АИС «Компилятор» — 28 февраля 2022 года.

1.6 Порядок оформления и предъявления заказчику результатов работ по созданию системы

Исходный код разработанного программного обеспечения, список технических средств к закупке, а также инструкция по установке, настройке,

запуске рабочего окружения и его сопровождение должна быть передана Заказчику в установленные им сроки. Приёмка и проверка работоспособности системы будет осуществлена комиссией, сформированной заказчиком.

1.7 Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ

При создании и разработки программного обеспечения и проектно-эксплуатационной документации необходимо руководствоваться требованиями следующий документов:

— ГОСТ 34.602-89.

1.8 Определения, обозначения и сокращения

Сокращение	Расшифровка
ТЗ	Техническое задание
АИС	Автоматизированная информационная система
ЯП	Язык программирования
ABI	Application Binary Interface
AST	Abstract Syntax Tree, абстрактное синтаксическое дерево
3AC	Three Address Code
SSA	Static Single Assignment form

2 НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ

2.1 Назначение системы

АИС «Компилятор» создается для образовательных целей и предназначена для обучения студентов и прочих желающих внутреннему устройству и построению в частности компиляторов, а в общем случае — программ-трансляторов с одного языка программирования в другой.

2.2 Цели создания системы

Основными целями создания АИС «Компилятор» являются:

— Создание компилятора языка C, который полностью или частично соответствует международному стандарту ISO/IEC 9899:1999.

— Создание компилятора, который поможет студентам и прочим желающим разобраться в работе программ-компиляторов и программ-трансляторов с одного ЯП в другой.

— Создание компилятора, который сможет скомпилировать сам себя (так называемая раскрутка, англ. *bootstrapping*).

Для реализации поставленных целей система должна решать следующие задачи:

— Переводить программу, написанную на ЯП C версии определенной и описанной в международном стандарте ISO/IEC 9899:1999 в язык ассемблера GNU Assembler с синтаксисом AT&T для архитектуры Intel® 64 в соответствии System V ABI.

— Иметь простой и понятный исходный код.

— Быть простой в использовании.

— Иметь возможность скомпилировать саму себя.

3 ТРЕБОВАНИЯ К СИСТЕМЕ

3.1 Требования к системе в целом

3.1.1 Требования к структуре и функционированию системы

3.1.1.1 Перечень подсистем, их назначение и основные характеристики

В состав АИС «Компилятор» должны входить следующие подсистемы:

— Подсистема лексического анализа (лексический анализатор).

— Подсистема синтаксического анализа (синтаксический анализатор).

— Подсистема построения AST.

— Подсистема перевода AST в 3AC.

— Подсистема перевода 3AC в SSA.

— Подсистема перевода SSA в код ассемблера GNU Assembler с синтаксисом AT&T для архитектуры Intel® 64 в соответствии с System V ABI.

Подсистема лексического анализа предназначена для а) разбиения исходного кода входной программа, представленного в качестве массива байт в определенной кодировке, на отдельные лексические единицы ЯП C; б) выявления и диагностики лексических ошибок во входной программе.

Подсистема синтаксического анализа предназначена для а) перевода потока токенов, полученных от лексического анализатора в синтаксическое дерево по грамматике, соответствующей грамматике языка C, описанного международным стандартом ISO/IEC 9899:1999; б) выявления и диагностики синтаксических ошибок во входной программе.

Подсистема построения AST предназначена для а) перевода синтаксического дерева, полученного от синтаксического анализатора в AST; б) выявления и диагностики семантических ошибок во входной программе.

Подсистема перевода AST в 3AC предназначена для перевода абстрактного синтаксического дерева, полученного из подсистемы построения AST, в 3AC.

Подсистема перевода 3AC в SSA предназначена для перевод 3AC, полученного из предыдущей подсистемы в SSA.

Предназначение подсистемы перевода SSA в код ассемблера GNU Assembler с синтаксисом AT&T для архитектуры Intel® 64 в соответствии с System V ABI очевидно из ее названия.

3.1.1.2 Требования к способам и средствам связи для информационного обмена между компонентами системы

Требования не предъявляются.

3.1.1.3 Требования к характеристикам взаимосвязей создаваемой системы со смежными системами

АИС «Компилятор» должна взаимодействовать со следующими смежными системами:

— Операционная система.

3.1.1.4 Требования к режимам функционирования системы

Для АИС «Компилятор» определены следующие режимы функционирования:

— Нормальный режим функционирования.

В нормальном режиме функционирования системы АИС «Компилятор» программное обеспечение обеспечивает возможность функционирования по запросу пользователя.

3.1.1.5 Требования по диагностированию системы

Требования не предъявляются.

3.1.1.6 Перспективы развития, модернизации системы

АИС должна иметь возможность дальнейшей модернизации как программного обеспечения, так комплекса технических средств.

Необходимо предусмотреть возможность добавления одного или нескольких проходов оптимизации SSA; возможность добавления других бэкендов компилятора — то есть генерация другого ассемблерного кода, кода для другой архитектуры и/или ABI.

3.1.2 Требования к численности и квалификации персонала системы

Для эксплуатации АИС «Компилятор» определены следующие роли:

— Пользователь.

Основными возможностями пользователя являются:

— Компиляция исходной программы.

— Просмотр промежуточных стадий трансляции.

Пользователи системы должны иметь опыт работы с любой командной оболочкой совместимой со стандартом IEEE Std 1003.1-2017.

3.1.3 Показатели назначения

Требования на время отклика системы не налагаются и зависят от входных данных, переданных системе на обработку.

Система должна предусматривать возможность масштабирования по производительности и объему обрабатываемой информации без модификации ее программного обеспечения путем модернизации используемого комплекса технических средств.

3.1.4 Требования к надежности

Требования к надежности не предъявляются, так как это позволит сильно упростить исходный код АИС для понимания.

Система не обязана сохранять работоспособность и обеспечивать восстановление своих функций при возникновении каких-либо внештатных ситуаций.

3.1.5 Требования к эргономике и технической эстетике

АИС не должна иметь визуальный графический интерфейс. Все взаимодействие с АИС должно производится из командной оболочки соответствующей стандарту IEEE Std 1003.1-2017.

3.1.6 Требования к транспортабельности для подвижных АС

Требования не предъявляются.

3.1.7 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы

Система должна быть рассчитана на эксплуатацию на персональных компьютерах пользователей.

3.1.8 Требования к защите информации от несанкционированного доступа

Требования не предъявляются.

3.1.9 Требования по сохранности информации при авариях

Требования не предъявляются.

3.1.10 Требования к защите от влияния внешних воздействий

Требования не предъявляются.

3.1.11 Требования к патентной чистоте

Разработка системы должна осуществляться в рамках рекомендаций по стандартизации Р50.1.028-2001 «Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования».

3.1.12 Требования по стандартизации и унификации

— ГОСТ 18421-93.

— ГОСТ 19.001-77.

— ГОСТ 19.005-85.

— ГОСТ 19.402-78.

— ГОСТ Р 51904-2002.

3.1.13 Дополнительные требования

Дополнительные требования не предъявляются.

3.2 Требования к функциям (задачам), выполняемым системой

3.2.1 Подсистема лексического анализа

Подсистема лексического анализа должна разбивать исходный код входной программы, представленной в качестве массива байт в определенной кодировке, на отдельные лексические единицы ЯП C.

Подсистема лексического анализа должна предоставлять возможность экспортировать результат своей работы (полученный поток токенов) в произвольном формате.

Также для подсистемы допускается выявление и диагностика лексических ошибок во входной программе с выводом диагностических сообщений пользователю, что соответствует требованиям ISO/IEC 9899:1999.

3.2.2 Подсистема синтаксического анализа

Подсистема синтаксического анализа должна переводить потока токенов, полученных от лексического анализатора в синтаксическое дерево по грамматике, соответствующей грамматике языка C, описанного международным стандартом ISO/IEC 9899:1999.

Подсистема лексического анализа должна предоставлять возможность экспортировать результат своей работы (полученное синтаксическое дерево) в произвольном формате.

Также для подсистемы допускается выявление и диагностика синтаксических ошибок во входной программе с выводом диагностических сообщений пользователю, что соответствует требованиям ISO/IEC 9899:1999.

3.2.3 Подсистема построения AST

Данная подсистема должна переводит синтаксическое дерево, полученное от синтаксического анализатора в AST.

Также подсистема должна предоставлять возможность экспортировать результат своей работы (полученное абстрактное синтаксическое дерево) в произвольном формате.

Также для подсистемы допускается выявление и диагностика семантических ошибок во входной программе с выводом диагностических сообщений пользователю, что соответствует требованиям ISO/IEC 9899:1999.

3.2.4 Подсистема перевода AST в 3AC

Подсистема перевода AST в 3AC должна переводить абстрактное синтаксическое дерево в 3AC. Также подсистема должна предоставлять возможность экспортировать результат своей работы (полученный 3AC) в произвольном формате.

3.2.5 Подсистема перевода 3AC в SSA

Подсистема перевода 3AC в SSA должна переводить 3AC в SSA. Также подсистема должна предоставлять возможность экспортировать результат своей работы (полученный SSA) в произвольном формате.

3.2.6 Подсистема перевода SSA в код ассемблера

Подсистема должна переводить SSA в код ассемблера GNU Assembler с синтаксисом AT&T для архитектуры Intel® 64 в соответствии с System V ABI. Полученный результат должен быть выведен пользователю любым способом. Например, с помощью вывода в стандартный поток вывода или файл, указанный пользователем.

3.3 Требования к видам обеспечения

3.3.1 Требования к математическому обеспечению системы

Для лексического анализа допустимо и желательно применение готовых библиотек программных кодов, реализующих функции работы с регулярными выражениями.

Для синтаксического анализа требуется использовать алгоритм рекурсивного спуска как наиболее простого. Так как грамматика языка C является контекстно-зависимой, потребуется применение постпроцессинга полученного синтаксического дерева.

3.3.2 Требования к лингвистическому обеспечению системы

Исходя из целей, АИС должна быть разработка с применением языка программирования C версии, соответствующей международному стандарту ISO/IEC 9899:1999. Также допустимо применение такого языка как GNU Make для создания системы сборки АИС.

Языком взаимодействия с пользователем является английский язык (американский).

Окончательное требования к кодировке входных данных уточняется в процессе создания программного обеспечения АИС и не обязано быть согласовано с Заказчиком.

3.3.3 Требования к программному обеспечению системы

При проектировании и разработке системы необходимо использовать только библиотеки программных кодов с открытым исходным кодом.

Базовой программной платформой должна являться POSIX-совместимая операционная система, в которой используется GNU C Library.

3.3.4 Требования к техническому обеспечению

Требования к техническому обеспечению не предъявляются.

3.3.5 Требования к метрологическому обеспечению

Требования к метрологическому обеспечению не предъявляются.

4 СОСТАВ И СОДЕРЖАНИЕ РАБОТ ПО СОЗДАНИЮ (РАЗВИТИЮ) СИСТЕМЫ

Этап	Содержание работ	Результаты работ
1	Разработка документов технического проекта АИС «Компилятор»	Документы технического проекта АИС «Компилятор»
2	Проектирование, создание и тестирование программного обеспечения АИС «Компилятор»	Программное обеспечение АИС «Компилятор»
3	Приемка АИС «Компилятор»	Акт о приемке АИС «Компилятор»

5 ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ СИСТЕМЫ

5.1 Виды, состав, объем и методы испытаний системы

Каждая подсистема АИС должна пройти комплексное тестирование для проверки корректности работы. Должно быть также проведено интеграционное тестирование каждой подсистемы по отдельности и всех вместе для выявления проблем взаимодействия как внутри подсистем, так и подсистем между собой.

Кроме того АИС должна пройти инспекцию кода комиссией, в состав которой входят представители Заказчика и Исполнителя. Инспекция кода должна подтвердить, что исходный код АИС соответствует предъявляемым к нему требованиям качества.

5.2 Общие сведения к приемке работ по стадиям

Сдача-приемка осуществляется комиссией, в состав которой входят представители Заказчика и Исполнителя. По результатам приемки подписывается акт приемочной комиссии.

5.3 Статус приемочной комиссии

Статус приемочной комиссии определяется Заказчиком до проведения испытаний.

6 ТРЕБОВАНИЯ К СОСТАВУ И СОДЕРЖАНИЮ РАБОТ ПО ПОДГОТОВКЕ ОБЪЕКТА АВТОМАТИЗАЦИИ К ВВОДУ СИСТЕМЫ В ДЕЙСТВИЕ

Требования не предъявляются.

7 ТРЕБОВАНИЯ К ДОКУМЕНТИРОВАНИЮ

Требования не предъявляются.

8 ИСТОЧНИКИ РАЗРАБОТКИ

Учебники, учебные пособия, научные работы и другие материалы по а) построению трансляторов кода; б) формальным языкам; в) системному и низкоуровневому программированию. Документы, приведенные в пункте 1.4.