УДК 004.4'422
На правах рукописи

КРОТОВ Александр Николаевич

Принципы реализации семантики языка Си++ в системе 3С++

05.13.11 - Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Автореферат

диссертации на соискание ученой степени
кандидата технических наук

Москва – 2002 г.

Работа выполнена в Научно-исследовательском вычислительном центре Московского государственного университета им. М.В. Ломоносова

Научный руководитель доктор технических наук, профессор Сухомлин В. А.

Официальные оппоненты: доктор физико-математических наук, профессор Серебряков В.А.

кандидат технических наук Волконский В.Ю.

Ведущая организация Институт системного программирования РАН

Защита состоится «4» декабря 2002 г. в 13 часов на заседании диссертационного совета Д 002.043.01 при Институте микропроцессорных вычислительных систем РАН по адресу: 119435, г. Москва, Б. Саввиновский переулок, дом 14.

С диссертацией можно ознакомиться в библиотеке Института микропроцессорных вычислительных систем РАН.

Автореферат разослан «__» _____ 2002 г.

Ученый секретарь
диссертационного совета
д.ф-м.н., профессор Михайлюк М.В.

Общая характеристика работы

Актуальность работы

В настоящее время одним из перспективных и экономически оправданных подходов к развитию информационной индустрии является создание информационных технологий (ИТ) и реализующих их систем (ИТ-систем) на принципах открытости. Основными свойствам открытых систем являются переносимость (программ, данных, пользовательских окружений), интероперабельность (сетевая взаимосвязь и совместное использование ресурсов и данных компонентами распределенных систем), масштабируемость (эффективность функционирования в широких диапазонах характеристик производительности и ресурсов). Достижимость этих качеств возможна лишь на основе стандартизации интерфейсов ИТ-систем и поддерживающих их платформ.

Одним из ключевых направлений в процессе стандартизации информационных технологий является стандартизация языков программирования и их библиотечных окружений.

В настоящее время язык Си++ занимает важное место среди индустриальных языков программирования. Он применяется для реализации широкого спектра системных и прикладных задач. Важной вехой в развитии этого языка является принятие в 1998г. Международного стандарта ISO для этого языка и его основных библиотек.

Диссертационная работа составила часть проекта по созданию системы программирования 3С++ («проект тройного стандарта») для языка Си++. В 1996-98гг. этот проект поддерживался Грантом РФФИ 96-01-01449 «Разработка методологии, методов и алгоритмов построения переносимой и адаптируемой к машинным архитектурам системы программирования тройного стандарта для языка С++». Основными задачами проекта являлись:

- разработка концепции создания и сопровождения систем программирования на основе языка программирования Си++, обеспечивающих реализацию стандарта входного языка и стандарта его библиотечного окружения, а также содержащих аппарат аттестации систем программирования на соответствие их языковому стандарту;
- исследование и разработка алгоритмов и структур данных для реализации на их основе переносимого компилятора объектно-ориентированного языка Си++ с адаптируемой генерацией кода, соответствующего полному стандарту языка.
- создание базового компилятора языка Си++, соответствующего стандарту языка, удовлетворяющего требованию параметризации

кодогенерации для настройки его на широкий спектр машинных архитектур, а также требованию переносимости компилятора (на уровне исходного текста) на различные платформы.

Цель исследования

Цель диссертационной работы состояла в исследовании, разработке и программном воплощении в производственном компиляторе методов и алгоритмов реализации важнейших механизмов языка C++ таких, как проверка типов, механизмы классов, наследования, виртуальных функций, определения и идентификации имен.
В соответствии с этой целью были определены следующие задачи:

- Исследование семантических свойств механизма классов языка Си++, построение формальной модели, семантически адекватной этому механизму**,** и разработка на основе данной модели эффективных методов и алгоритмов поддержки в компиляторе семантики механизма классов, включая механизмы наследования и виртуальных функций языка.
- Исследование семантических свойств механизма поиска имен в языке Си++, разработка формализованной модели механизма поиска имен, а также разработка основанных на ее свойствах эффективных методов и алгоритмов реализации поиска имен в компиляторе.
- Исследование семантических свойств системы типов языка Си++, разработка методов и алгоритмов поддержки в компиляторе семантики системы типов языка Си++.
- Разработка алгоритмов генерации кода, реализующих семантику периода выполнения механизмов классов, наследования и виртуальных функций языка Си++.
- Программная реализация разработанных в диссертационной работе методов и алгоритмов в производственном компиляторе языка Си++.

Результаты работы

В процессе проведения исследований и разработок автором получены следующие научные результаты:

- Введены формальные определения графов наследования и подобъектов, исследованы их свойства. На основании изученных свойств графов предложены методы реализации процедур выявления доминирования имен, однозначности и доступности. При этом улучшены опубликованные другими авторами алгоритмы выявления доминирования имен, определенных в базовых классах.
- Исследованы семантические свойства механизма поиска имен в языке Си++, сформулированы требования к реализации механизма поиска имен в компиляторе и предложен эффективный метод реализации данного механизма, обладающий лучшими производственными характеристиками по сравнению с традиционно применяемыми методами.

- Исследованы семантические свойства системы типов языка Си++, выделены требования к реализации системы типов в компиляторе. Предложен эффективный метод реализации семантики системы типов в компиляторе и в промежуточном представлении высокого уровня. Приведено описание наиболее сложных частей механизма проверки типов.
- Разработаны алгоритмы генерации кода и структур данных (таблиц) периода выполнения, реализующие семантику механизмов классов, множественного и виртуального наследования, виртуальных функций, идентификации типов на этапе выполнения. Для алгоритмов генерации таблиц приведены оценки сложности, а для таблиц приведены оценки их возможного роста.
- Предложенные методы реализации были использованы в первом отечественном промышленном компиляторе языка Си++, являющемся одним из первых компиляторов, реализующих полный стандарт Си++.

Апробация работы

Результаты, полученные в работе, изложены в ряде печатных публикаций, докладывались на научных конференциях и семинарах, в частности:

- на Ломоносовских чтениях в НИВЦ МГУ в 1994, 1995, 1998гг.
- на Первой российской конференции "Индустрия программирования'96", Москва, 3-4 октября 1996 г.
- на IV Международной конференции "Развитие и применение открытых систем", Н.Новгород, 27-31 октября 1997 г.
- на конференции “Теоретические и прикладные проблемы информационных технологий”, ВМиК МГУ, Москва, 19-21 июня 2001г.

Научная новизна работы

Научная новизна представляемой работы может быть охарактеризована следующими тезисами:

1. Изучены свойства графов наследования и подобъектов и предложены эффективные алгоритмы выявления наиболее существенных свойств этих графов. Предложенные алгоритмы имеют лучшую оценку сложности по сравнению с опубликованными другими авторами алгоритмами.
2. На основе предложенного графа подобъектов описаны методы реализации механизмов классов и виртуальных функций. Получены оценки сложности для реализации механизмов классов и виртуальных функций, основанной на генерации таблиц. Показано, что для такой реализации не может быть предложено полиномиальных алгоритмов. Тем не менее эти алгоритмы вполне

применимы в промышленном компиляторе, так как в наиболее часто встречающихся на практике случаях экспоненциального роста размера таблиц не происходит.

3. Предложен метод реализации поиска имен в компиляторе, более адекватно отражающий семантику языка Си++, чем традиционные методы реализации поиска имен в компиляторах.
4. Предложен метод реализации типов, использующий технику хеширования.

Практическая ценность

На основе исследований, выполненных по теме диссертации, разработаны следующие программные системы:

- Компилятор переднего плана, соответствующий предварительному стандарту Си++ от декабря 1996г., является ядром кросс-системы программирования для цифрового нейронного процессора NM6403, разработанного НТЦ "Модуль" (Москва).
- Компилятор переднего плана, соответствующий окончательной версии международного стандарта Си++ ISO/IEC 14882:1998, передан компании «Интерстрон» (Москва) для включения в создаваемую систему программирования Си++, ориентированную на цели обучения.

Структура и объем работы

Диссертация состоит из введения, четырех глав, заключения и списка литературы из 90 наименований. Объем диссертации составляет 154 страницы текста. Диссертация содержит 34 рисунка и 3 таблицы.

Содержание работы

Введение

Введение содержит обзор процесса стандартизации языка программирования Си++ в контексте общей тенденции стандартизации информационных технологий и языков программирования. Также приводится описание проекта 3С++, в рамках которого выполнялась работа и указывается место использования проведенных в работе исследований в этом проекте.

Семантическая сложность современных языков программирования и необходимость для разработчиков языков и компиляторов строгого и точного описания семантики языков повышает роль применения

формальных методов как при описании языков, так и в процессе их реализации. Именно поэтому в данной работе, посвященной созданию компилятора языка Си++, основное внимание уделяется разработке формальных моделей для важнейших механизмов этого языка. Использование формальных моделей позволило тщательно проанализировать семантические свойства языка и разработать хорошо обоснованные и эффективные алгоритмы реализации. Такой подход полностью оправдал себя при создании системы программирования 3С++, ядром который является компилятор стандарта языка Си++.

Основными компонентами компилятора являются:

1. Препроцессор.
2. Компилятор переднего плана (front-end compiler).
3. Генератор кода.

Во введении представлена модель компилятора, изображенная на рисунке 1.

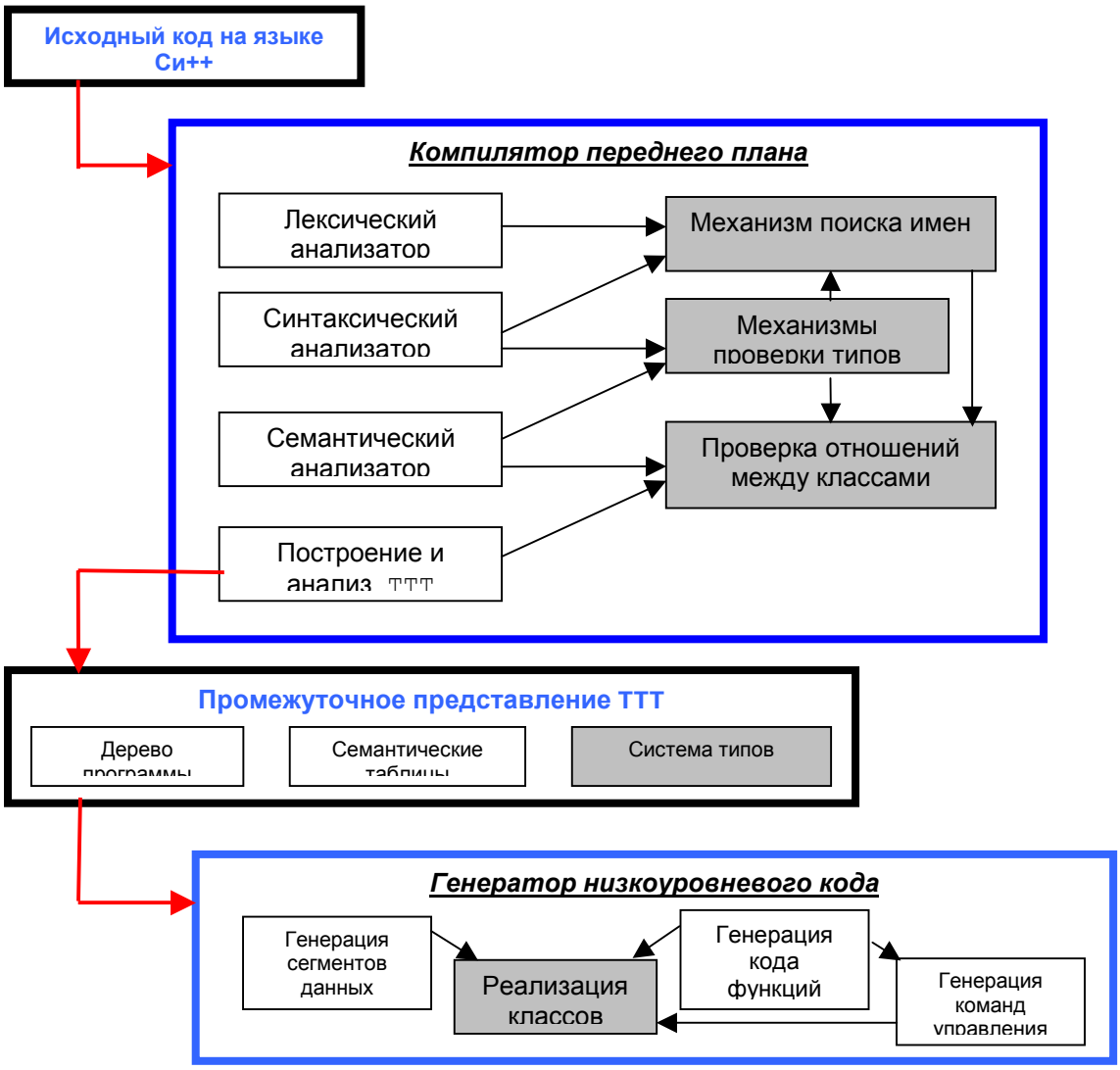

Рис. 1.

В представленной на рисунке модели цветом фона выделены части компилятора, реализованные на основе разработанных в диссертации алгоритмов и методов.

Также во введении приводится краткое содержание диссертации по главам. В частности: алгоритмы выявления отношений между классами рассматриваются в первой главе, во второй главе описаны методы реализации классов, в третьей главе исследуются процедуры поиска имен, методы проверки типов и реализации типов в промежуточном представлении составляют основное содержание четвертой главы.

Глава 1

В первой главе исследуются отношения между классами в языке программирования Си++. Наиболее важными с точки зрения семантики языка и реализации компилятора являются следующие отношения между классами:

- наследование;
- однозначность наследования;
- доминирование имен, определенных в базовых классах;
- доступность членов и подобъектов базовых классов.

Для отношения наследования строится формальная модель – граф наследования. На основании графа наследования строятся графы подобъектов для отдельных классов. Эти графы используются как формальная модель для изучения свойств подобъектов базовых классов. На основании этого строятся две функции - *sub** и *sub*, определенные на парах вершин графа наследования. Затем эти функции используются для определения однозначности наследования и доминирования имен. Отношение доступности также исследуется на основании графа подобъектов.

Для отношения доминирования имен, определенных в базовых классах, другими авторами были предложены формальные модели, также использующие графы подобъектов и наследования. Основное отличие предложенного алгоритма реализации заключается в том, что предложенный алгоритм позволяет не строить графы во внутренних структурах компилятора, так как не нуждается в приписывании атрибутов ребрам графа. Кроме этого, предложенный алгоритм определения доминирования имен имеет полиномиальную сложность $O(n^2)$, что лучше других опубликованных алгоритмов.

Затем рассматривается метод реализации отношений между классами в компиляторе, основанный на использовании хеш-таблицы. Для каждой пары классов, между которыми есть отношение наследования, в хеш-таблице делается запись, хранящая небольшой набор атрибутов,

который позволяет рекурсивно достраивать записи для других классов на основании относительно простых правил.

В **разделе 1.1** содержится введение в первую главу. В нем рассматриваются основы подхода к реализации поддержки объектно-ориентированного программирования в языке Си++, а также особенности отношений между классами по сравнению с другими языками программирования.

Раздел 1.2 посвящен определению и исследованию свойств графа наследования. Граф наследования можно построить на основании информации, содержащейся в семантических таблицах компилятора. Хотя в компиляторе этот граф в явном виде не стоится, он является удобной моделью для дальнейших рассуждений.

Для классов определяются три отношения: лексического следования, непосредственного наследования, обозначаемое как *A>B* и общего наследования (которое определяется как транзитивное и рефлексивное замыкание отношения непосредственного наследования), обозначаемое как $A \Rightarrow B$. Наиболее интересным для изучения является отношение непосредственного наследования, именно на основании него строится граф наследования *Inh(C,>)*. Вершинами этого графа являются классы (множество *C*); ребра соответствуют непосредственному наследованию и направлены от производных классов к базовым.

Затем вводятся понятия пути для этого графа и атрибут виртуальности наследования для ребер. Для ребер графа определяется предикат *virt*. Для ребра (*A*,*B*) предикат является истинным, если ребро является ребром виртуального наследования.

Участок пути $p = \{A=X_1, X_2,..X_k=B\}$ из вершины *A* в вершину *B* называется *фиксированным,* если для любого ребра $(X_j, X_{j+1}), 1<=j<k$ не верно $virt(X_j, X_{j+1})$ и либо $j=1$, либо $virt(X_{j-1}, X_j)$. Фиксированный участок пути *p* обозначается как *fix(p)*.

Для путей определяется отношение эквивалентности как равенство их фиксированных частей. Класс эквивалентности, содержащий путь *p*, обозначается как *[p]*, а начальная вершина его фиксированной части как *[p]**.

Одним из основных вопросов, который исследуется в этой главе является поиск способа вычисления числа классов эквивалентности путей из одной вершины графа в другую. Для ответа на этот вопрос вводятся вспомогательные множества *V* и *W*, содержательно соответствующие множествам виртуальных базовых классов. Множество *V(s,k)*, зависящее от пары вершин *s* и *k,* определяется как множество вершин *v* графа *Inh* таких, что *v* достижима из *s* и для некоторого пути *p* из *s* в *k* верно *[p]*=v.* Множество *W(s)*, зависящее от одной вершины *s*, определяется как объединение по всем достижимым из *s* вершинам *k* множеств *V(s,k)*.

Доказывается, что число классов эквивалентности путей из вершины *s* в *k* равно сумме по всем вершинам объединения множества *W(s)* и *{s}*

числа путей из вершин множества в вершину *k*, не содержащих ребер виртуального наследования.

Раздел 1.3 содержит определение графа подобъектов *Sub(s)* для конкретной вершины *s* графа наследования. Определение дается в виде рекурсивной процедуры построения. Также определяется понятие фиксированной части *Sub*(s)* этого графа. Для вершин графов определяется их отображение на вершины графа наследования.

Содержательно граф *Sub(s)* является моделью строения объекта класса *s*. Вершинами графа являются подобъекты базовых классов. Каждой вершине графа ставится в соответствие некоторый класс (вершина графа *Inh*). Граф *Sub*(s)* являются подграфом *Sub(s)*. Для его построения достаточно из графа *Sub(s)* удалить все ребра, соответствующие виртуальному наследованию, а затем все вершины и ребра, которые в результате удаления стали недоступны из начальной вершины.

Для графов подобъектов доказывается, что если класс *s* является производным от класса *k*, то в графе *Sub(s)* найдется подграф, изоморфный графу *Sub(k)*, а в графе *Sub*(s)* подграф, изоморфный *Sub*(k)*.

Также доказывается, что любому пути в графе *Sub(s)* соответствует некоторый путь в графе *Inh*.

Основным утверждением о графах подобъектов является построение взаимно-однозначного соответствия между вершинами графа *Sub(s)* и классами эквивалентности путей, начинающихся в вершине *s* графа *Inh*.

Затем определяются функции *sub* и *sub** как функции, определяемые с помощью графов подобъектов. Значениями функций *sub(s,k)* и *sub*(s,k)*, зависящих от вершин *s* и *k* графа *Inh*, являются число вершин, соответствующих классу *k* в графах *Sub(s)* и *Sub*(k)* соответственно. Содержательно значением функции *sub(s,k)* является число подобъектов класса *k* в объекте класса *s*.

Благодаря соответствию между вершинами графа *Sub(s)* и классами эквивалентности путей в графе *Inh* можно дать эквивалентное определение этих функций без апелляции к графам *Sub(s)*, в терминах классов эквивалентности путей.

Для вычисления этих функций предлагается алгоритм, основанный на рекурсивном вычислении значения функций на основе того, что известны их значения для базовых классов класса *s*. Алгоритм сводится к следующим формулам:

$$sub(A,B) = \sum_{I \in W(A) \cup \{A\}} sub^*(I,B)$$

$$sub^*(A,B) = \sum_{I:A>I \;\&\; \neg virt(A,I)} sub^*(I,B)$$

$$\forall A \; sub^*(A,A) = sub(A,A) = 1$$

Такой алгоритм не нуждается в построении графов *Sub(s)* в явном виде во внутренних структурах компилятора. Это свойство является существенным с точки зрения реализации, так как число вершин графа *Sub(s)* может расти экспоненциально по отношению к числу вершин графа *Inh* и графы *Sub(s)* зависят от конкретных классов, в то время как граф *Inh* строится для всей программы целиком.

С помощью функции *sub* дается формальное определение отношения однозначности наследования. Класс *k* является *однозначным базовым классом* для класса *s*, если *sub(s,k)=1*.

В **Разделе 1.4** рассматривается отношение доминирования для имен, определенных в базовых классах. Сначала дается неформальное определение для доминирования имен, затем показывается, что это отношение индуцирует отношение доминирования в графе подобъектов. (Такое отношение доминирования отличается от известного отношения доминирования в теории графов. Ниже без дополнительных оговорок именно отношение доминирования имен будет называться отношением доминирования.) Вопрос об определении доминирования имен сводится к определению доминирования для базовых классов в графе подобъектов.

В терминах предложенных формальных моделей можно дать следующее определение доминирования:

Пусть $A \Rightarrow B \Rightarrow D$ - три класса. В графе *Sub(A)* базовый класс *B доминирует над* классом *D*, если в этом графе для любой вершины *d* класса *D* существует путь *p* в эту вершину из начальной вершины *a*, проходящий через вершину класса *B*.

Как видно из определения, отношение доминирования формулируется для трех классов и основывается на графе подобъектов. В нескольких доказанных в этом разделе утверждениях вопрос об определении доминирования сводится к выявлению свойств графа *Inh*, а затем приводится алгоритм определения доминирования. Алгоритм сводится к проверке равенства *sub(A,D)=(sub(A,D)-1)sub*(B,D)+sub(B,D)*.

Пусть *n* – мощность множества *C* (число классов, определенных в программе). Предложенный алгоритм выявления доминирования имеет сложность $O(n^2)$, учитывая вычисления функций *sub* и *sub**, если значения функций известны для всех базовых классов класса *A*. Выявление доминирования и вычисление значений функций *sub(s,k)* для всех возможных пар вершин графа *Inh* потребует $O(n^3)$ операций.

Раздел 1.5 посвящен исследованию отношения доступности для базовых классов и имен, определенных в базовых классах. Это отношение, также как и все предыдущие, можно определить в терминах графов наследования и подобъектов. Для выявления доступности предложен рекурсивный алгоритм, позволяющий легко определять доступность по путям в этих графах. Этот алгоритм обладает тем же свойством, что и алгоритм вычисления функций *sub* и *sub**, а именно, зная все базовые классы и значения функции доступности для них, легко определяется доступность для производного класса.

В случае непосредственного наследования доступность является одним из атрибутов, доступных в семантических таблицах компилятора. Для произвольного пути в графе Inh доступность определяется как минимум из атрибутов доступности ребер пути. Доступность базового класса определяется как максимум по всем путям из вершины производного класса в вершину базового класса.

В завершающем содержательную часть первой главы **разделе 1.7** рассматриваются вопросы реализации отношений между классами в компиляторе. В этом разделе рассматриваются структуры данных, используемые для хранения отношений между классами. Основной структурой данных является хеш-таблица, содержащая записи для всех пар классов, между которыми установлено отношение наследования. Элементами таблицы являются записи, описывающие свойства отношений между двумя классами, а именно значения функций *sub, sub** и функции доступности.

Предложенный способ вычисления функций *sub, sub** позволяет избегать построения графов *Sub(s)* во внутренних структурах компилятора, что существенно с точки зрения реализации.

Благодаря найденному алгоритму выявления доминирования для имен, определенных в базовых классах, компилятор не нуждается в хранении промежуточной информации о доминировании (что могло бы потребовать построения трехмерной таблицы).

Кроме этого, описываются структуры данных, используемые для хранения информации о непосредственных базовых классах. В дальнейшем эти же структуры используются в главе 2 при описании механизмов времени компиляции.

Глава 2

Во второй главе обсуждаются принципы и методы реализации механизма классов и тесно связанного с ним механизма виртуальных функций с точки зрения генерации низкоуровневого (близкого к машинному) промежуточного кода. Рассматриваемые методы являются развитием известного метода генерации таблиц виртуальных функций.

Также в этой главе обсуждаются вопросы генерации кода для конструкторов и деструкторов, вызовов функций-членов классов, указателей на члены классов, а также преобразований указателей и указателей на члены классов.

Предложенные механизмы являются по сути машинно-независимыми и позволяют генерировать эффективный код для машин с традиционной архитектурой.

В этой главе для обоснования корректности алгоритмов активно используются графы подобъектов, определенные в первой главе. Для каждой предложенной части алгоритма генерации таблиц приводятся оценки сложности и размера генерируемых таблиц. Для данного метода

реализации не может существовать полиномиального алгоритма, так как размер таблиц может расти экспоненциально по отношению к числу классов в программе, но на практике подобные иерархии встречаются очень редко. Такой подход к реализации был выбран сознательно, он является наиболее практичным с точки зрения эффективности и простоты генерируемого кода, а также применимости его для компиляции реальных программ на языке Си++. Следует также отметить, что этот подход не является единственным возможным. На основании рассмотренных выше моделей и с использованием приведенных полиномиальных алгоритмов возможны альтернативные реализации, например, создание виртуальной машины, не нуждающейся в генерации таблиц. Описываемые в последующих главах методы реализации не основываются на экспоненциально сложных алгоритмах, предложенных в этой главе.

При описании алгоритмов приводятся примеры возможных оптимизаций при построении таблиц. Такие оптимизации не улучшают оценку сложности алгоритмов в общем случае, но существенно сокращают размер генерируемых таблиц в наиболее часто встречающихся при программировании на Си++ случаях.

Раздел 2.1 содержит обсуждение общих принципов реализации рассматриваемых механизмов. Отмечается, что при кажущейся простоте выбранного метода реализации приходится учитывать большое число тонкостей, которые при их неправильном понимании могут привести к ошибкам в генерации кода.

При выборе метода реализации отдельных частей механизма классов предпочтение отдавалось методам, позволяющим генерировать максимально простой и эффективный код, даже если это приводит к усложнению отдельных частей компилятора.

Раздел 2.2 посвящен обсуждению реализации объектов классов. В нем излагаются основы реализации объектов классов как областей в памяти, устанавливается связь между строением объекта класса *s* и графом подобъектов *Sub(s)*.

Также в этом разделе вводятся основные понятия и обозначения, используемые во второй главе, такие как размер, смещения членов и подобъектов. В графе подобъектов *Sub(s)* смещения подобъектов представляются как целочисленные атрибуты ребер. При этом для любых двух вершин графа сумма атрибутов, приписанных ребрам пути из одной вершину в другую, не зависит от выбора пути.

Раздел 2.3 также является вводным. В нем вводятся используемые в дальнейшем обозначения, определяются понятия сигнатуры функции, равенства сигнатур и определяется отношение доминирования для виртуальных функций.

Сигнатура функции определяется как ее имя и типы ее явных параметров. Соответственно, равенство сигнатур определяется как совпадение имен и типов всех явных параметров функции.

Отношение доминирования для виртуальных функций сводится к рассмотренному в первой главе отношению доминирования для базовых классов и может проверяться с использованием предложенного в первой главе алгоритма.

В **разделе 2.4** обсуждаются механизмы времени компиляции, реализующие механизм классов. В этом разделе, содержащем несколько подразделов, описываются алгоритмы обработки класса, которые применяются к каждому классу, определенному в программе, и методы генерации таблиц, используемых для реализации механизмов виртуального наследования и виртуальных функций.

Подразделы 2.4.2 и 2.4.3 содержат описание метода построения списка виртуальных функций и принципа его сортировки. Предложенный метод позволяет легко выявлять неявную виртуальность функций-членов, а также абстрактность классов. В этом разделе приводятся примеры неоднозначности виртуальных функций и показывается, что эти неоднозначности нельзя свести к изученным в первой главе неоднозначностям имен или базовых классов.

Построение списка виртуальных функций для класса состоит в слиянии таких списков для всех непосредственных базовых классов и добавлении в него виртуальных функций, описанных в самом классе.

Затем список сортируется. Критерием сортировки является сравнение сигнатур функций. Таким образом, функции с одинаковой сигнатурой, определенные в различных классах, в списке группируются. После сортировки из списка удаляются все элементы, доминируемые другими функциями из списка, имеющими совпадающую сигнатуру. В результате, в сокращенном списке остаются только элементы, соответствующие доминирующим функциям для всех встретившихся сигнатур. После удаления доминируемых элементов однозначность виртуальной функции определяется как наличие в списке единственной функции с данной сигнатурой.

Алгоритм проверки абстрактности класса сводится к проходу по списку и проверке всех оставшихся в нем функций на наличие атрибута чистой виртуальности. Если хотя бы одна функция из списка таким атрибутом обладает, класс считается абстрактным.

Сложность этой части алгоритма обработки класса оценивается как $O(nN \ln nN)$, где n – число классов в программе и N – число виртуальных функций.

Подраздел 2.4.5 посвящен описанию процесса «раскладки» класса – приписывания смещений его членам и базовым классам. Этот же алгоритм вычисляет размер памяти, необходимый для хранения объекта класса, и учитывает требования по выравниванию членов и подобъектов в памяти.

Алгоритм сводится к последовательному выделению памяти для всех подобъектов графа *Sub*(s)*, членов класса *s* и, затем, вершин графа *Sub(s),* не являющихся вершинами подграфа *Sub*(s)*. Так как такое распределение памяти производится только на основании списка

непосредственных базовых классов и множества *W(s),* этот алгоритм не требует обхода графа *Sub(s),* и его сложность можно оценить как *O(|k:s>k|+|W(s)|+|attr(s)|)*, где *attr(s)* – множество членов класса *s.*

В подразделе 2.4.6 обсуждается алгоритм генерации таблиц смещений виртуальных базовых классов – основы метода реализации механизма виртуального наследования. Этот алгоритм является достаточно простыми, в следующем подразделе он обобщается для подобъектов базовых классов.

Алгоритм состоит из приписывания элементам множества *W(s)* последовательных индексов и генерации таблицы, которая состоит из смещений подобъектов базовых классов для всех элементов множества *W(s)* в порядке приписанных им индексов. В данном случае индексы виртуальным базовым классам могут приписываться достаточно произвольно, относительный порядок классов не имеет существенного значения.

Подраздел 2.4.7 посвящен описанию алгоритма генерации смещений виртуальных базовых классов для подобъектов базовых классов. Этот алгоритм несколько более сложен по сравнению с алгоритмом, рассмотренным в предыдущем подразделе, так как такая генерация является рекурсивной и, фактически, реализует механизм аналогичный обходу графа *Sub(s)*. По этой причине алгоритм имеет в худшем случае экспоненциальную сложность по отношению к числу классов. Также экспоненциально может расти размер генерируемых таблиц.

Для каждой вершины *k* графа *Sub(s)* строится таблица, состоящая из относительных смещений достижимых из нее вершин, соответствующих множеству *W(s)*. Таблица строится в порядке возрастания индексов, приписанных классам как виртуальным базовым классам класса *k*. Функция, реализующая эту генерацию, вызывает себя рекурсивно для всех базовых классов. В качестве дополнительного параметра в функцию передается сумма смещений, приписанных ребрам пройденного участка пути (пути из начальной вершины в вершину *k*) в графе *Sub(s)*.

В этом же разделе рассматриваются такие вопросы, как необходимость генерации таблиц в частных случаях и метод оптимизации, позволяющий существенно сокращать число генерируемых таблиц для иерархий классов, которые часто встречаются на практике.

Такие оптимизации возможны, так как таблицы смещений, построенные для конкретного класса *k*, как вершины различных графов *Sub(s)*, во многих случаях совпадают.

В подразделах 2.4.8, 2.4.9 и 2.4.10 рассматривается генерация таблиц виртуальных функций. Как и в случае таблиц смещений виртуальных базовых классов сначала описывается генерация таблицы для класса, затем рассматривается более общий случай – генерация таблиц для подобъектов базовых классов. По сравнению с таблицами смещений виртуальных базовых классов этот процесс более сложен, так как в некоторых случаях необходимо генерировать отдельные таблицы

для использования в конструкторах и деструкторах подобъектов. В самом общем случае генерация таблиц требует выполнения двойного рекурсивного обхода графа *Sub(s)*. По этой причине алгоритм также является экспоненциальным, но в большинстве встречающихся на практике случаев генерация всех возможных таблиц не требуется.

Таблица виртуальных функций является таблицей, состоящей из пар (*адрес кода функции, смещение подобъекта*). В качестве адреса кода функции используется адрес доминирующей функции из списка, построенного для класса, как описано в разделе 2.4.3. В качестве смещения используется относительное смещение подобъекта, в котором определена доминирующая функция в обрабатываемом объекте – вершине графа *Sub(s)*.

В конце подраздела 2.4.10 приводится подробный пример, иллюстрирующий работу алгоритмов генерации таблиц. Этот пример интересен тем, что многие реализации Си++ при генерации таблиц виртуальных функций не выполняют двойной обход графа *Sub(s)* и в результате строят некорректные таблицы.

В подразделе 2.4.13 описываются структуры данных, используемые в компиляторе, позволяющие эффективно реализовать генерацию таблиц без явного построения графов *Sub(s)*. Часть структур данных является расширением элементов хеш-таблицы, предложенной в разделе 1.7.

Остальные структуры данных используются для хранения списка доминирующих виртуальных функций и информации о смещениях базовых классов.

Раздел 2.5 содержит обсуждение методов реализации механизмов классов и виртуальных функций с точки зрения генерации кода, использующего таблицы.

В подразделе 2.5.1 рассматривается метод генерации кода для создания и уничтожения объектов классов. Этот код может включать инициализацию указателей на таблицы и вызовы конструкторов для подобъектов базовых классов и членов классов в случае создания объекта класса и вызовы деструкторов при уничтожении объекта.

Основой для реализации является обход части графа *Sub(s)* в фиксированном порядке – порядке вызова конструкторов или деструкторов для подобъектов базовых классов. Этот порядок строго определен в стандарте языка.

Подраздел 2.5.2 посвящен генерации кода, реализующего преобразования указателей и ссылок на объекты классов. Генерируемый код зависит от свойств графа подобъектов преобразуемого класса и положения в нем вершины класса, к которому производится преобразование.

В наиболее простых случаях, когда преобразование можно выполнить используя только ребра подграфа *Sub*(s),* код сводится к прибавлению к указателю фиксированного смещения – суммы смещений

приписанных ребрам пути в подграфе *Sub*(s)*; в наиболее сложных случаях могут использоваться таблицы смещений виртуальных базовых классов, указатель на которые доступен как член преобразуемого класса. Доступ к таким таблицам производится с использованием индекса виртуального базового класса, присвоение которого описано в разделе 2.4.6.

В подразделе 2.5.3 описывается реализация доступа к статическим членам класса. Фактически статические члены класса реализуются как глобальные объекты, неявно имеющие внешнее связывание.

Подраздел 2.5.4 посвящен обсуждению методов генерации кода для доступа к нестатическим членам классов. Генерируемый для операторов доступа код сводится к генерации кода для преобразования указателей на классы и кода доступа к члену по фиксированному смещению. Выполнение преобразования указателей необходимо в тех случаях, когда производится доступ к члену базового класса через объект производного класса.

В подразделе 2.5.5 описаны методы генерации кода для вызова функций-членов классов. Рассматривается три основных случая: вызов статической функции-члена, вызов виртуальной функции и вызов обычной функции. Наиболее сложным с точки зрения генерации кода является случай вызова виртуальной функции-члена. Генерируемый для этого случая код может содержать преобразования указателей на класс и доступ к таблице виртуальных функций. При этом доступ к таблице может использоваться дважды: для нахождения адреса функции и для определения смещения скрытого параметра `this`. В остальных случаях генерация кода для вызова функции-члена мало отличается от генерации кода для вызова обычной функции.

В завершающих разделах второй главы рассматриваются методы реализации операций `typeid`, `dynamic_cast` и определения типов на этапе выполнения при обработке исключительных ситуаций.

Глава 3

В третьей главе обсуждается поиск имен в языке Си++ и реализация механизма поиска имен в компиляторе. Правила поиска имен в этом языке обладают рядом особенностей, что делает неэффективным применение в компиляторе традиционных способов реализации. В этой главе предлагается нестандартный метод, который адекватен семантике языка, и при этом является достаточно практичным для реализации. Он может быть использован для реализации механизмов поиска имен в компиляторах других языков программирования.

В **разделе 3.1** рассматривается место механизма поиска имен в структуре компилятора и формулируются основные требования к его реализации.

Требования предъявляемые к реализации этого механизма: полная поддержка правил поиска имен в Си++, эффективность и простота в программной реализации.

Раздел 3.2 посвящен обсуждению особенностей поиска имен в языке Си++. Рассматриваются общие для многих языков программирования правила поиска имен, затем рассматриваются особенности правил поиска имен в Си++, в котором есть большое число исключений из общих правил. Для многих таких особенностей приводятся иллюстрирующие их примеры. В результате обсуждения особенностей становится очевидно, что задача поиска имен в этом языке не является тривиальной. Некоторое внимание уделяется особенностям, связанным со сложившимся стилем программирования на этом языке, которые необходимо учитывать при реализации промышленного компилятора.

Раздел 3.3 посвящен описанию принципов реализации поиска имен в компиляторе.

В подразделе 3.3.1 рассматривается традиционный метод реализации поиска имен с использованием отдельных хеш-таблиц для каждой области действия и обсуждаются его недостатки при использовании в компиляторе Си++.

Этот метод был использован в очень ранней версии компилятора. Основные недостатки применения этого метода обусловлены большим числом областей действия имен в программах на Си++ и неравномерностью распределения числа именованных сущностей по областям действия.

Подраздел 3.3.2 содержит обсуждение более развитого метода реализации поиска имен, основанного на использовании единой хеш-таблицы, который также часто используется при реализации компиляторов. Метод основан на разрешении коллизий методом цепочек, при этом в хеш-цепочке имя может встречаться более одного раза. При поиске имени в такой хеш-таблице всегда выбирается совпадающее имя, находящееся в цепочке первым.

Показывается, что этот метод также плохо отражает особенности поиска имен в языке Си++. Недостатки применения этого метода обусловлены тем, что в программах на Си++ очень часты случаи использования имен в областях действия, синтаксически не вложенных в области действия, в которых они определены. Наиболее характерным примером такого использования имен является использование имен членов классов.

В подразделе 3.3.3 предлагается метод реализации поиска имен, также основанный на использовании единой хеш-таблицы, но обладающий рядом преимуществ по сравнению с рассмотренными ранее методами. Такой метод позволяет быстро определять наличие определения идентификатора в нескольких областях действия. Описываются структуры данных, используемые для этого метода.

Основной структурой данных для такой реализации является хеш-таблица, в которой каждый используемый в программе идентификатор встречается только один раз. С каждым именем в хеш-таблице ассоциируется множество (реализованное как дерево поиска) ссылок на области действия, в которых имя определено. Такая реализация позволяет на самых ранних этапах компиляции (во время лексического разбора) избавиться от необходимости хранить копии идентификатора в памяти, а во время поиска имен не требует выполнения большого количества сравнения символьных строк, вместо этого производится поиск элемента в множестве ссылок.

Такой метод можно считать «инверсией» первого рассмотренного метода (раздел 3.3.2). Если при реализации первого метода первичными являются области действия, а имена считаются принадлежащими областям действия, то в предложенный метод реализации полагает первичным литеральное представление имени.

В **разделе 3.4** описываются и обсуждаются области действия имен в Си++. Выделяются их свойства, существенные для реализации, такие, как лексическая замкнутость и время доступности. Последнее свойство изучается дополнительно. Доказывается 4 утверждения о свойствах его семантики. На основании этих утверждений формализуются условия, при выполнении которых из таблицы поиска имен можно удалять элементы. Это делается в тех случаях, когда известно, что в дальнейшем поиск имени в удовлетворяющих сформулированным условиям областях действия производиться не будет. Такое удаление возможно, так как в языке нет ни одной конструкции, позволяющей сослаться на эти области действия. Примером может служить область действия имен параметров функции – после завершения ее определения невозможно сослаться на имена определенных в ней локальных переменных и типов.

Раздел 3.5 посвящен описанию полного механизма поиска имен для случаев квалифицированных и неквалифицированных имен.

В подразделе 3.5.1 описывается метод поиска неквалифицированных имен - наиболее часто используемого в языке случая поиска имен. В этом случае алгоритм состоит из поиска имени в хеш-таблице и проверке наличия в множестве ассоциированных с имением областей действия наличия определенных элементов. В случае поиска имен в области действия класса выполняется рекурсивный обход части графа наследования *Inh* и проверяется наличие определения имени в базовых классах, а затем из найденного множества при помощи алгоритма проверки доминирования (раздел 1.4) выделяется единственный доминирующий элемент. Если в множестве не нашлось единственного доминирующего элемента, использование имени считается семантически некорректным.

Подраздел 3.5.2 посвящен рассмотрению более сложного случая – поиску квалифицированных имен. В этом случае алгоритм последовательно выполняет поиск первого имени в квалификаторе, используя алгоритм из предыдущего раздела, а затем поиск всех

последующих имен квалификатора выполняется в областях действия классов или namespace, полученных на предыдущем шаге.

В конце этого подраздела приводится диаграмма взаимодействия различных частей механизма, реализующего поиск имен в компиляторе.

Глава 4

Четвертая глава посвящена обсуждению вопросов реализации типов и проверки типов в компиляторе. Язык Си++ обладает гибкой и сложной системой типов, поддерживает полиморфизм как встроенных, так и пользовательских операторов и функций. Стиль программирования на этом языке предполагает активное использование возможностей по построению новых типов на основании уже определенных и частое применение неявных преобразований типов.

Реализация проверки типов опирается на механизмы проверки отношений между классами и поиска имен. Таким образом, она является, кроме прочего, иллюстрацией взаимодействия различных частей компилятора.

Раздел 4.1 является вводным. В нем обсуждаются общие свойства понятий типов и проверки типов в различных языках программирования.

В **разделе 4.2** формулируются основные требования к реализации типов в компиляторе. Также в нем рассматриваются различные допустимые в языке способы образования новых типов на основе уже определенных.

Основным требованием к реализации типов в компиляторе является возможно простая и эффективная реализация проверки типов на совпадение и возможность преобразования значений одного типа к значениям другого типа.

В **разделе 4.3** на основе требований, определенных в предыдущем разделе, обсуждается два способа реализации типов.

Подраздел 4.3.1 посвящен обсуждению традиционного способа реализации типов на основ типовых цепочек. Этот способ был опробован в первой версии компилятора, но в процессе его развития выявилось большое число недостатков, которые также перечисляются в этом подразделе.

В подразделе 4.3.2 описывается подход, примененный при разработке второй версии компилятора. Этот метод основан на использовании для представления типов таблицы.

В этом разделе приводится формальное рекурсивное определение типа, на котором основываются все дальнейшие рассуждения. Даются понятия семантически корректного типа, эквивалентности типов, групп типов и приводятся примеры представления типов в виде ациклических графов.

Формальное определение типа дается рекурсивно. В качестве базы используются фундаментальные, классовые и перечислимые типы. Все остальные типы строятся на их основе с помощью применения правил образования квалифицированного, указательного, ссылочного, массивного или функционального типа. Язык накладывает ограничения на возможные комбинации применения этих правил.

Типы разбиваются на следующие непересекающиеся группы: фундаментальные, классовые, перечислимые, указательные, ссылочные, массивные, функциональные, указатели на члены классов.

В следующем подразделе 4.3.3 на основании предложенного концептуального представления типов определятся какими свойствами должна обладать таблица для того, чтобы эффективно представлять типы в компиляторе и промежуточном представлении. Основными требованиями являются: возможность быстро определять принадлежность типа к определенной группе, возможность построения нового типа по одному из правил построения новых типов и выполнение обратного преобразования – нахождение базового типа на основе производного, построенного по одному из перечисленных правил.

Затем предлагается два способа реализации поиска в таблице, удовлетворяющих сформулированным требованиям. В компиляторе используется один из предложенных способов, основанный на рекурсивном построении хеш-функций для типов и реализации таблицы типов как хеш-таблицы.

Основным принципом такой реализации является обеспечение «уникальности» типов во внутреннем представлении компилятора. Каждый конкретный тип должен быть представлен в таблице только один раз, что существенно упрощает проверку типов на совпадение – в этом случае такая проверка сводится к проверке на совпадение двух указателей в таблицу типов. Уникальность обеспечивается использованием хеш-таблицы. Хеш-значения типов вычисляются рекурсивно на основании хеш-значений типов, от которых он порожден. В случае необходимости создания нового типа из уже существующего сначала проверяется наличие нового типа в таблице, и если он не найден создается новая запись, в противном случае используется ссылка на уже существующую запись.

В качестве базы для рекурсивного определения хеш-значений типов используется множество «терминальных» типов, к которым отнесены все фундаментальные (встроенные), классовые и перечислимые типы. Для каждого правила образования нового типа определяется правило преобразования хеш-значения. Например если хеш-значение типа T есть h, то хеш-значение типа «указатель на T» есть `5h+21`.

Для выполнения поиска типа, на основании которого создан данный тип, используется переход по ссылке: в записях в таблице типов для этого используется специальное поле.

Раздел 4.3.4 посвящен подробному рассмотрению одного из способов образования новых типов – добавлению квалификаторов `const` и `volatile`. Этот способ построения новых типов обладает рядом особенностей, позволяющих реализовать его в компиляторе более эффективно, чем остальные правила порождения типов. Метод основан на том, что на инструментальных платформах динамически выделяемые блоки памяти выравниваются по адресам, кратным четырем, и в качестве представления для квалификаторов можно использовать свободные младшие биты указателя.

В **разделе 4.5** описывается структура классов, использованная для реализации элементов таблицы типов. Рассматриваются члены классов, представляющих типы в компиляторе и промежуточном представлении TTT.

Каждая запись в таблице реализована как класс, общими полями для всех типов являются: хеш-значение, целое число, используемое как признак принадлежности типа к определенной группе, размер памяти и выравнивание, необходимые для хранения значения данного типа, целое число, используемое как индекс типа при генерации отладочной информации. Использование других полей в таблице зависит от принадлежности типа к определенной группе: элементы, соответствующие классовым и перечислимым типам содержат поля–ссылки на дескрипторы соответствующих сущностей; указательные, ссылочные и массивные типы содержат ссылки на типы, от которых они порождены; массивные типы содержат целочисленное поле – число элементов массива и т.д.

Вторая часть данной главы (разделы 4.6 - 4.7) посвящена рассмотрению вопросов реализации проверки типов в компиляторе. Проверка типов в языке Си++ нуждается в отдельном рассмотрении потому, что по сравнению с другими языками программирования правила проверки типов в этом языке достаточно сложны. При программировании на Си++ активно используются неявные преобразования типов, встроенные операторы являются полиморфными, кроме этого поддерживается две формы полиморфизма пользовательских операторов и функций.

В **Разделе 4.6** обсуждается подход к реализации проверки типов в компиляторе. Одним из ключевых моментов в реализации является проверка неявных преобразований типов. Такие преобразования могут быть весьма нетривиальными, использующими пользовательские функции преобразования и конструкторы. В процессе выполнения преобразования могут возникать временные объекты с нетривиальными деструкторами. В этом разделе рассматриваются особенности преобразования типов в Си++, обсуждаются различные виды преобразований, в качестве иллюстрации строится достаточно простая формальная модель.

В качестве формальной модели используется граф, вершинами которого являются пары (тип, lvalue), где lvalue – булево значение. Для каждого типа, явно или неявно используемого в программе, создаются две вершины с различными значениями lvalue. Две вершины графа

соединяются ребром в том случае, когда существует семантически корректное неявное преобразование от одного типа к другому. При таком определении в графе допустимы петли. По семантике языка отношение инцидентности в таком графе не является транзитивным, но в нем существуют отдельные подграфы, ограничение отношения инцидентности на которые транзитивно замкнуто (например, множество пар (фундаментальный тип, false)).

При таком определении графа наличие ребра между двумя вершинами не означает того, что преобразование значения одного типа к другому возможно в произвольном месте программы. Примером могут служить ребра, соответствующие преобразованиям указателей на классы. По правилам языка такие преобразования должны проверяться дополнительно на однозначность и доступность.

Раздел 4.7 продолжает начатое выше обсуждение проверки типов в компиляторе. В нем обсуждается часть процедур, реализующих проверку типов. Основное внимание уделяется взаимодействию процедур проверки между собой и их взаимодействию с механизмами проверки отношений между классами и поиска имен. Рассматриваемый набор процедур реализует самую сложную часть всего механизма, а именно проверку неявных преобразований, выбор наилучшей из множества перегруженных функций, учитывающий полиморфизм и существование шаблонов функций.

Выбор наилучшей из множества совместно используемых функций основан на сравнении типов формальных и фактических параметров. В терминах предложенной формальной модели на множестве дуг графа, исходящих из одной вершины, можно определить частичный порядок. Для каждого фактического параметра каждой из множества перегруженных функций выполняется выбор «наименьших» ребер. Для однозначного выбора одной из перегруженных функций необходимо, чтобы для одного из параметров соответствующее ребро, ведущее от вершины типа фактического параметра к вершине формального параметра, было минимальным, а ребра для всех остальных параметров были наименьшими. Таким образом, отношение частичного порядка на ребрах графа при заданных типах фактических параметров индуцирует отношение частичного порядка на множестве совместно используемых функций. Функция f считается меньшей, по сравнению с функцией g, если для одного из фактических параметров соответствующее ему ребро преобразования для функции f меньше ребра для функции g, а для остальных параметров ребра для функции g не меньше, чем для функции f (при этом допускается, что порядка для ребер не определено).

При таком определении выбор функции из множества совместно используемых сводится к выбору минимального элемента. Невозможность такого выбора означает семантическую ошибку.

То, что по приведенному выше определению выбора функции из множества совместно используемых возможно выбрать наименьшую, не всегда означает семантическую корректность вызова функции. Как

отмечалось ранее, даже при наличии в графе ребра преобразования само преобразование может быть семантически некорректно. Корректность использования преобразования по правилам языка должна проверяться для каждого параметра по общим правилам уже после выбора минимального элемента.

Заключение

В работе получены следующие результаты:

1. Введено формальное определение графа наследования, на основе которого проведено систематическое исследование семантических свойств механизма наследования классов в языке Си++ и сформулированы требования к реализации механизма наследования классов в компиляторе. Построена и изучена формальная модель (граф подобъектов) строения объектов классов языка Си++, на основе которой разработан эффективный способ вычисления числа подобъектов базового класса и дано формальное определение однозначности базового класса. Дано определение доминирующего класса, также основанное на данной формальной модели, и предложен способ эффективного определения доминирующего класса. Предложен способ определения доступности базового класса и члена базового класса на основании графа наследования.
2. Разработаны метод и структуры данных компилятора, эффективно реализующие семантические проверки, основанные на отношениях между классами.
3. Разработаны алгоритмы генерации машинно-независимого низкоуровневого промежуточного кода и структур данных (таблиц) периода выполнения, реализующие семантику механизмов классов, множественного и виртуального наследования, виртуальных функций. В частности, предложены следующие решения:
   - метод реализации виртуального наследования, основанный на генерации таблиц смещений виртуальных базовых классов, и алгоритмы генерации таких таблиц;
   - метод генерации кода для доступа к членам классов и членам базовых классов;
   - метод реализации преобразований указателей и ссылок на классы с учетом особенностей множественного и виртуального наследования;
   - метод реализации виртуальных функций, основанный на генерации таблиц специального вида;
   - дано формальное определение доминирования для виртуальных функций и описан способ его выявления;
   - метод определения абстрактности классов, неявной виртуальности функций;
   - реализация и метод генерации кода для преобразования указателей на члены классов;

- реализация операторов `typeid` и `dynamic_cast` с использованием объектов расширенного класса `type_info`;
- реализация динамического определения типов для исключительных ситуаций.

4. Исследована сложность алгоритмов генерации таблиц, используемых для реализации механизма классов, и даны оценки размера требуемых таблиц.
5. Разработаны структуры данных компилятора, используемые механизмами времени компиляции при генерации таблиц, поддерживающих реализацию семантики механизма виртуальных функций и виртуального наследования.
6. Исследованы семантические свойства правил поиска имен в языке Си++, определены требования к реализации механизма поиска имен в компиляторе и предложен метод реализации данного механизма, обладающий лучшими производственными характеристиками по сравнению с традиционно применяемыми методами.
7. Предложен способ проверки наличия определения имени в различных областях действия с использованием хеш-таблицы. Разработан метод реализации механизма поиска имен, на основе использования графа областей действия и дисплея областей действия. Формализованы условия, при которых объявленные в области действия имена становятся недоступными и могут быть удалены из хеш-таблицы. Разработаны решения по реализации конструкций using namespace и using. Описано взаимодействие механизма поиска имен с другими частями компилятора. Описано взаимодействие частей компилятора, реализующих различные варианты поиска имен.
8. Исследованы особенности проверки типов в языке Си++ и определены требования к реализации проверки типов в компиляторе.
9. Проанализированы недостатки традиционного способа реализации типов в виде цепочек при использовании в компиляторе Си++. Предложен метод реализации системы типов в компиляторе на основе хеш-таблицы, удовлетворяющий сформулированным требованиям. Разработано отображение системы типов в промежуточное представление высокого уровня.
10. Разработан метод проверки типов для операторов языка, учитывающий возможность перегрузки операторов и наличие операторов преобразования типов, определяемых пользователем, а также метод проверки типов при вызове перегруженных функций.

Публикации

Основные результаты диссертации опубликованы в следующих работах:

1. Зуев Е.А., Кротов А.Н. *Новые возможности Си++* // PC Magazine/ Russian Edition, 1994, №7. - С. 176-182.
2. Кротов А.Н., Зуев Е.А. *Подход к реализации исключительных ситуаций в компиляторе языка Си++*. “Компьютерные аспекты в научных исследованиях и учебном процессе”. Сборник по материалам научной конференции МГУ “Ломоносовские чтения - 96”. М. 1996. С. 43-51
3. Зуев Е.А., Кротов А.Н. *Компилятор полного стандарта Си++* // Известия ВУЗов. Приборостроение, том 40, 1997, №3, С. 17-21.
4. Кротов А.Н. *Реализация механизмов времени выполнения языка С++ в компиляторе переднего плана* // Известия ВУЗов. Приборостроение, том 40, 1997, №3, С. 22-28.
5. Кротов А.Н., Миронов А.Г. *Генерация результирующего кода в компиляторе Си++* // Известия ВУЗов. Приборостроение, том 40, 1997, №3, С. 28-32.
6. Кротов А.Н. *Особенности правил поиска имен в языке программирования Си++*. Теоретические и прикладные проблемы информационных технологий: сборник трудов. М., 2001. С. 301-313.
7. Кротов А.Н. *Реализация поиска имен в компиляторе Си++*. Теоретические и прикладные проблемы информационных технологий: сборник трудов. М., 2001. С. 314-329.