

А. В. Девятков, магистрант

И. О. Архипов, кандидат технических наук, доцент

кафедра «Программное обеспечение»

Ижевский государственный технический университет имени

М. Т. Калашникова

Система автоматического оценивания сложности программ на C++.

Вычисляется сложность программного кода, написанного на языке программирования C++. В качестве образцов программного кода выступают решения задач базового курса по программированию, который проводится на системе BACS. Вычисление сложности выполняется на основе лексического анализа кода. Практическое применение метода позволит оценивать эффективность учебных курсов по программированию.

Ключевые слова: методика, метрика программного обеспечения, статический анализ кода, динамический анализ кода.

Введение

Развитие методологии программированного обучения и компьютеризации всех сфер человеческой деятельности привели к широкому применению ЭВМ в учебном процессе. Компьютерное обучение предполагает использование специализированного программного обеспечения — обучающие программы, автоматизированные обучающие системы (АОС)[1].

Одним из важных элементов обучения является контроль усвоения знаний, умений и навыков. Контролирующие программы и модули обучающих систем предназначены для текущего и итогового контроля. При этом АОС позволяют накапливать статистическую информацию по

нескольким параметрам и проследить успеваемость каждого обучаемого в динамике, определить эффективность обучения в зависимости от начального уровня знаний обучаемого, сложности и объема материала, времени, затраченного на проработку темы и т. д.

Контроль знаний является неотъемлемой частью на каждом этапе обучения, поэтому АОС в области программирования должна включать улучшенную подсистему тестирования знаний, умений и навыков обучаемых. Эта подсистема призвана обеспечивать оценку исходного уровня подготовки обучаемого, текущий и итоговый контроль усвоения материала. Основным видом заданий для такого контроля являются задачи на составление программ[1].

Для постоянного контроля качества знаний учащихся необходимо разработать множество задач. Выделяются различные тематики, каждой из которых требуется посвятить отдельный учебный курс. Задачи требуется также относить к определенному этапу обучения, различая их по сложности и по охватываемому материалу.

Для начального обучения необходимо выбрать наиболее простые задачи. С ростом уровня подготовки необходимо выбирать более сложные задачи, требующие использования неочевидных рекурсивных преобразований, оценки сложности алгоритмов, сведения задачи к динамическому программированию. Возникает необходимость автоматизации выбора задач для построения курса обучения.

Процесс анализа

Процесс статического анализа состоит из двух основных шагов: создания дерева кода (также называемого абстрактным деревом синтаксиса) и анализа этого дерева[2].

Для того чтобы проанализировать исходный код, анализатор должен сначала "понять" этот код, т.е. разобрать его по составу и создать структуру, описывающую анализируемый код в удобной форме. Эта форма и называется

деревом кода. Чтобы проверить, соответствует ли код стандарту кодирования, необходимо построить такое дерево.

В общем случае дерево строится только для анализируемого фрагмента кода (например, для какой-то конкретной функции). Для того чтобы создать дерево код обрабатывается сначала лексером, а затем синтаксическим анализатором.

Лексер отвечает за разбиение входных данных на отдельные лексемы, а также за определение типа этих лексем и их последовательную передачу синтаксическому анализатору. Лексер считывает текст исходного кода строку за строкой, а затем разбивает полученные строки на зарезервированные слова, идентификаторы и константы, называемые лексемами. После получения лексемы лексер определяет ее тип.

Рассмотрим примерный алгоритм определения типа лексемы.

Если первый символ лексемы является цифрой, лексема считается числом, если этот символ является знаком "минус", то это - отрицательное число. Если лексема является числом, она может быть числом целым или дробным. Если в числе содержится буква E, определяющая экспоненциальное представление, или десятичная точка, число считается дробным, в противном случае - целым. Заметим, что при этом может возникнуть лексическая ошибка - если в анализируемом исходном коде содержится лексема "4xyz", лексер сочтет ее целым числом 4. Это породит синтаксическую ошибку, которую сможет выявить синтаксический анализатор. Однако подобные ошибки могут обнаруживаться и лексером.

Если лексема не является числом, она может быть строкой. Строковые константы могут распознаваться по одинарным кавычкам, двойным кавычкам, или каким-либо другим символам, в зависимости от синтаксиса анализируемого языка.

Наконец, если лексема не является строкой, она должна быть идентификатором, зарезервированным словом, или зарезервированным символом. Если лексема не подходит и под эти категории, возникает

лексическая ошибка. Лексер не будет обрабатывать эту ошибку самостоятельно - он только сообщит синтаксическому анализатору, что обнаружена лексема неизвестного типа. Обработкой этой ошибки займется синтаксический анализатор.

Синтаксический анализатор понимает грамматику языка. Он отвечает за обнаружение синтаксических ошибок и за преобразование программы, в которой такие ошибки отсутствуют, в структуры данных, называемые деревьями кода. Эти структуры в свою очередь поступают на вход статического анализатора и обрабатываются им.

Дерево кода

Дерево кода представляет самую суть поданных на вход данных в форме дерева, опуская несущественные детали синтаксиса[3]. Такие деревья отличаются от конкретных деревьев синтаксиса тем, что в них нет вершин, представляющих знаки препинания вроде точки с запятой, завершающей строку, или запятой, которая ставится между аргументами функции.

Синтаксические анализаторы, используемые для создания деревьев кода, могут быть написаны вручную, а могут и создаваться генераторами синтаксических анализаторов. Деревья кода обычно создаются снизу вверх.

При разработке вершин дерева в первую очередь обычно определяется уровень модульности. Иными словами, определяется, будут ли все конструкции языка представлены вершинами одного типа, различаемыми по значениям. В качестве примера рассмотрим представление бинарных арифметических операций. Один вариант - использовать для всех бинарных операций одинаковые вершины, одним из атрибутов которых будет тип операции, например, "+". Другой вариант - использовать для разных операций вершины различного типа.

Математическая модель

Так как о сложности программы приходится судить по ряду

показателей - метрик, то для получения интегральной оценки воспользуемся комплексной мерой. Для расчета комплексной оценки используем формулу:

$$H = R_1 \times M_1 + \dots + R_n \times M_n,$$

где M_i – значения выбранных метрик, R_i - значения весовых коэффициентов.

Расчет метрики суммарной сложности условий производится следующим образом. Для каждого составного условия подсчитывается количество логических операторов $cntLO$. Итоговое значение метрики получается по формуле :

$$C_{SLO} = \sum_i^n cntLO_i$$

где C_{slo} - итоговое значение метрики, $cntLO_i$ – значение количество логических операторов i – го составного условия.

Расчет метрики вложенности программы осуществляется по формуле

$$C_{Sld} = \sum_i^n maxLD_i ,$$

где C_{sld} - суммарный уровень вложенности программы, $maxLD_i$ – максимальный уровень вложенности i -той подпрограммы.

Предлагается оценивать факторы по некоторой балльной шкале, например от 1 до 10. Тогда получаем выражения

$$Y_{11}, Y_{21}, \dots, Y_{n1};$$

$$Y_{12}, Y_{22}, \dots, Y_{n2};$$

$$\dots\dots\dots;$$

$$Y_{1m}, Y_{2m}, \dots, Y_{nm}$$

где Y_{ij} - балльная оценка фактора, полученная от j -го эксперта, n - количество факторов, m — число экспертов.

Сводные оценки весовых коэффициентов обычно находят путем подбора соответствующей регрессионной модели. Среднюю оценку w_i весовых коэффициентов факторов можно получить по простым формулам

$$w_i = \frac{\sum_{j=1}^m w_{ij}}{\sum_{j=1}^m \sum_{i=1}^n w_{ij}},$$

где w_{ij} - вес i -го объекта, рассчитанный по оценкам всех экспертов;

$$w_{ij} = \frac{x_{ij}}{\sum_{i=1}^n x_{ij}},$$

где x_{ij} - оценка фактора i , данная экспертом j ; n - число факторов, m - число экспертов.

Заключение

Результаты экспериментов подтверждают, что предложенный в работе набор метрик может быть использован для вычисления сложности программного кода. В ходе исследования было разработано приложение на языке C++.

Список литературы

1. Агальцов В. П. Контроль знаний — доминирующая составляющая образовательного процесса // Информатика и образование вып. 2: Образование и Информатика, 1986. – С.94–96.
2. Dirk Giesen. Philosophy and practical implementation of static analyzer tools [Электронный ресурс] // URL: http://www.dse.nl/~thelosen/artikelen/static_analysis.pdf (дата обращения 15.01.2013).
3. Joel Jones. Abstract syntax tree implementation idioms [Электронный ресурс] // Proceedings of the 10th Conference on Pattern Languages of Programs 2003. URL: <http://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf> (дата обращения: 10.02.2013).
4. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика. – М.: «Мир», 1992. – 184 с.
5. Кнут Д. Искусство программирования, том 1. Основные алгоритмы. – 3-е изд. – М.: «Вильямс», 2006. – 720 с.

