Хэш-функция - функция, осуществляющая хэширование массива данных посредством отображения значений из (очень) большого множества-значений в (существенно) меньшее множество-значений.

ЛАБОРАТОРНАЯ РАБОТА № 1

Цель работы: изучить основные методы организации таблиц идентификаторов, получить представление о преимуществах и недостатках, присущих различным методам организации таблиц символов (идентификаторов).

Для выполнения лабораторной работы требуется написать программу, которая получает на входе набор идентификаторов, организует таблицу по заданному методу и позволяет осуществить **многократный** поиск идентификатора в этой таблице. Список идентификаторов считать заданным в виде текстового файла. Длина идентификаторов ограничена 32 символами.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Простейшие способы организации таблицы идентификаторов

Проверка правильности семантики и генерация кода требуют знания характеристик идентификаторов, используемых в программе на исходном языке. Эти характеристики выясняются из описаний и из того, как идентификаторы используются в программе и накапливаются в ***таблице символов*** или ***таблице идентификаторов***. Любая таблица символов состоит из набора полей, количество которых равно числу идентификаторов программы. Каждое поле содержит в себе полную информацию о данном элементе таблицы. Под идентификаторами подразумеваются константы, переменные, имена процедур и функций, формальные и фактические параметры.

Простейший способ организации таблицы состоит в том, чтобы добавлять элементы в порядке их поступления. Поиск в этом случае требует сравнения с каждым элементом таблицы, пока не будет найден подходящий. Для таблицы, содержащей n элементов, в среднем будет выполнено n/2 сравнений. Если n велико, то способ не является эффективным.

Поиск может быть выполнен более эффективно, если элементы таблицы упорядочены (отсортированы) согласно некоторому естественному порядку. В нашем случае, где поиск будет осуществляться по имени идентификатора, наиболее естественным будет расположить элементы таблицы в алфавитном порядке. Эффективным методом поиска в упорядоченном списке из n элементов является бинарный или логарифмический поиск. Символ S, который следует найти, сравнивается с элементом (n + 1)/2 в середине таблицы. Если этот элемент не является требуемым, мы должны просмотреть только блок элементов, пронумерованных от 1 до (n + 1)/2 - 1, или блок элементов от (n + 1)/2 + 1 до n в зависимости от того, меньше искомый элемент S или больше того, с которым его сравнили. Затем мы повторяем процесс над блоком меньшего размера. Так как на каждом шаге число элементов, которые могут содержать S, сокращается наполовину, то максимальное число сравнений равно $1 + \log_2 n$. Для сравнения: при для n = 128 бинарный поиск требует самое большее 8 сравнений, поиск в неупорядоченной таблице - в среднем 64 сравнения.

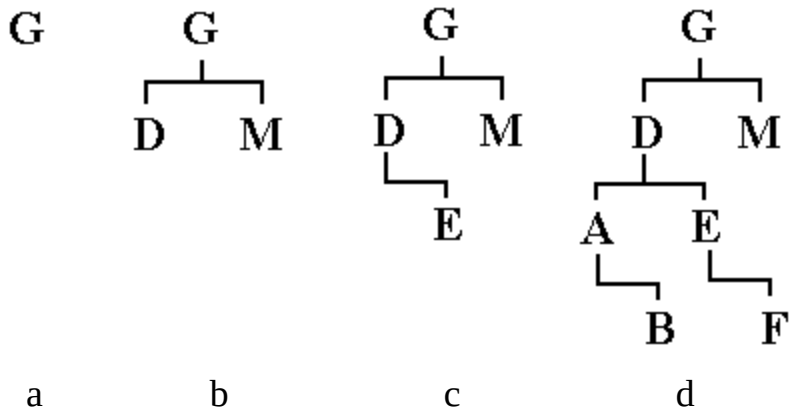
Существует часто используемый метод упорядочивания элементов в таблице, называемый хеш-адресация. Этот метод преобразует символ в индекс элемента в таблице. Индекс получается "хешированием" символа - выполнением над символом некоторых простых арифметических и логических операций. Простой хеш-функцией является внутреннее представление первой литеры символа. Так, если двоичное ASCII представление символа A есть 00100001, то результатом хеширования идентификатора ATable будет код 00100001.

Пока для двух различных элементов результаты хеширования различны, время поиска совпадает с временем, затраченным на хеширование. Однако возникает затруднение, когда результаты хеширования двух разных элементов совпадают. Это называется коллизией. В одну позицию таблицы может быть помещен только один из конфликтующих элементов. Хорошая хеш-функция распределяет вычисляемые адреса равномерно на все имеющиеся в распоряжении адреса, так что коллизии возникают не столь часто. Хеш-функция, предложенная выше, очевидно плоха, так как все идентификаторы, начинающиеся с одной буквы, ссылаются на один и тот же адрес. Существует большое множество хеш-функций. Каждая из них стремится распределить адреса под идентификаторы по своему алгоритму, но идеального хеширования достичь не удается.

Для решения задачи коллизии можно использовать много способов. Одним из них является метод цепочек. Метод цепочек использует хеш-таблицу, где помимо основных полей для каждого элемента добавлено еще одно поле, в котором может содержаться ссылка на любой элемент таблицы. Метод работает следующим образом: если по какой-либо причине при формировании таблицы у вносимого элемента N адрес совпадает с адресом уже существующего элемента E, то элемент N заносится в таблицу по некоторому другому адресу, а в поле ссылки элемента E заносится указатель на элемент N (это может быть, для примера, его адрес или индекс в таблице). В свою очередь данное поле ссылки у элемента N также в силу коллизии может оказаться заполненным, то есть возникают цепочки из элементов таблицы.

При поиске необходимого элемента цепочка просматривается до тех пор, пока не будет встречен нужный элемент или цепочка не закончится (тогда элемент не найден).

Существует метод построения таблиц, при котором таблица имеет форму бинарного дерева. Каждый узел дерева представляет собой элемент таблицы, причем корневой узел является первым элементом. При построении дерева элементы сравниваются между собой, и в зависимости от результатов, выбирается путь в дереве. На рис.1,a показана таблица с одним элементом для идентификатора G. Предположим, что надо записать идентификатор D. Для него выбирается левое поддерево, так как D<G (рис.1,b). Теперь запишем идентификатор M. Так как G<M, для M выбирается правое поддерево G (рис.1.,b). Наконец, запишем идентификатор E. Так как E<G, то идем по левой дуге от G и попадаем в D (рис.1,c). На рис.1,d изображено дерево после того, как в него были добавлены идентификаторы A,B и F.



a b c d

Рис. 1. Пример бинарного дерева

Для данного метода число требуемых сравнений и форма получившегося дерева во многом зависят от того порядка, в котором поступают идентификаторы.

Хеш-функции и хеш-адресация

Принципы работы хеш-функций

Логарифмическая зависимость времени поиска и времени заполнения таблицы идентификаторов — это самый хороший результат, которого можно достичь за счет применения различных методов организации таблиц. Однако в реальных исходных программах количество идентификаторов столь велико, что даже логарифмическую зависимость времени поиска от их числа нельзя признать удовлетворительной. Необходимы более эффективные методы поиска информации в таблице идентификаторов.

Лучших результатов можно достичь, если применить методы, связанные с использованием хеш-функций и хеш-адресации.

Хеш-функцией F называется некоторое отображение множества входных элементов **R** на множество целых неотрицательных чисел **Z**: $F(r) = n$, $r \in \mathbf{R}$, $n \in \mathbf{Z}$. Сам термин «хеш-функция» происходит от английского термина «hash function» (hash — «мешать», «смешивать», «путать»). Вместо термина «хеширование» иногда используются термины «рандомизация», «переупорядочивание».

Множество допустимых входных элементов **R** называется областью определения хеш-функции. Множеством значений хеш-функции F называется подмножество **M** из множества целых неотрицательных чисел **Z**: $\mathbf{M} \subseteq \mathbf{Z}$, содержащее все возможные значения, возвращаемые функцией F: $\forall r \in \mathbf{R}$: $F(r) \in \mathbf{M}$ и $\forall m \in \mathbf{M}$: $\exists r \in \mathbf{R}$: $F(r)=m$. Процесс отображения области определения хеш-функции на множество значений называется «хешированием».

При работе с таблицей идентификаторов хеш-функция должна выполнять отображение имен идентификаторов на множество целых неотрицательных чисел. Областью определения хеш-функции будет множество всех возможных имен идентификаторов.

Хеш-адресация заключается в использовании значения, возвращаемого хеш-функцией, в качестве адреса ячейки из некоторого массива данных. Тогда размер массива данных должен соответствовать области значений используемой хеш-функции. Следовательно, в реальном компиляторе область значений хеш-функции никак не должна превышать размер доступного адресного пространства компьютера.

Метод организации таблиц идентификаторов, основанный на использовании хеш-адресации, заключается в размещении каждого элемента таблицы в ячейке, адрес которой возвращает хеш-функция, вычисленная для этого элемента. Тогда в идеальном случае для размещения любого элемента в таблице идентификаторов достаточно только вычислить его хеш-функцию и обратиться к нужной ячейке массива данных. Для поиска элемента в таблице необходимо вычислить хеш-функцию для искомого элемента и проверить, не является ли заданная ею ячейка массива пустой (если она не пуста — элемент найден, если пуста — не найден). Первоначально таблица идентификаторов должна быть заполнена информацией, которая позволила бы говорить о том, что все ее ячейки являются пустыми.

Этот метод весьма эффективен поскольку как время размещения элемента в таблице, так и время его поиска определяются только временем, затрачиваемым на вычисление хеш-функции, которое в общем случае несопоставимо меньше времени, необходимого на многократные сравнения элементов таблицы.

Метод имеет два очевидных недостатка. Первый из них — неэффективное использование объема памяти под таблицу идентификаторов: размер массива для ее хранения должен соответствовать области значений хеш-функции, в то время как реально хранимых в таблице идентификаторов может быть существенно меньше. Второй недостаток — необходимость соответствующего разумного выбора хеш-функции. Этому существенному вопросу посвящены следующие два подпункта.

Построение таблиц идентификаторов на основе хеш-функций

Существуют различные варианты хеш-функций. Получение результата хеш-функции — «хеширование» — обычно достигается за счет выполнения над цепочкой символов некоторых простых арифметических и логических операций. Самой простой хеш-функцией для символа является код внутреннего представления в компьютере литеры символа. Эту хеш-функцию можно использовать и для цепочки символов, выбирая первый символ в цепочке. Так, если двоичное ASCII представление символа `A` есть двоичный код 00100001_2, то результатом хеширования идентификатора `ATable` будет код 00100001_2.

Хеш-функция, предложенная выше, очевидно не удовлетворительна: при использовании такой хеш-функции возникнет проблема — двум различным идентификаторам, начинающимся с одной и той же буквы, будет соответствовать одно и то же значение хеш-функции. Тогда при хеш-адресации в одну ячейку таблицы идентификаторов по одному и тому же адресу должны быть помещены два различных идентификатора, что явно невозможно. Такая ситуация, когда двум или более идентификаторам соответствует одно и то же значение функции, называется *коллизией*.

Естественно, что хеш-функция, допускающая коллизии, не может быть напрямую использована для хеш-адресации в таблице идентификаторов. Причем достаточно получить хотя бы один случай коллизии на всем множестве идентификаторов, чтобы такой хеш-функцией нельзя было пользоваться непосредственно. Но в примере взята самая элементарная хеш-функция. А возможно ли построить хеш-функцию, которая бы полностью исключала возникновение коллизий?

Для полного исключения коллизий хеш-функция должна быть взаимно однозначной: каждому элементу из области определения хеш-функции должно соответствовать одно значение из ее множества значений, и каждому значению из множества значений этой функции должен соответствовать только один элемент из области ее определения. Тогда любым двум произвольным элементам из области определения хеш-функции будут всегда соответствовать два различных ее значения. Теоретически для идентификаторов такую хеш-функцию построить можно, так как и область определения хеш-функции (все возможные имена идентификаторов), и область ее значений (целые неотрицательные числа) являются бесконечными счетными множествами. Теоретически можно организовать взаимно однозначное отображение одного счетного множества на другое, но практически это сделать исключительно сложно.

Практически существует ограничение, делающее создание взаимно однозначной хеш-функции для идентификаторов невозможным. Дело в том, что в реальности область значений любой хеш-функции ограничена размером доступного адресного пространства в данной архитектуре компьютера. При организации хеш-адресации значение, используемое в качестве адреса таблицы идентификаторов, не может выходить за пределы, заданные разрядностью адреса компьютера. Множество адресов любого компьютера с традиционной архитектурой может быть велико, но всегда конечно, то есть ограничено. Организовать взаимно однозначное отображение бесконечного множества на конечное даже теоретически невозможно. Можно учесть, что длина принимаемой во внимание части имени идентификатора в реальных компиляторах также практически ограничена — обычно она лежит в пределах от 32 до 128 символов (то есть и область определения хеш-функции конечна). Но и тогда количество элементов в конечном множестве, составляющем область определения функции, будет превышать их количество в конечном множестве области значений функции (количество всех возможных идентификаторов все равно больше количества допустимых адресов в современных компьютерах). Таким образом, создать взаимно однозначную хеш-функцию практически ни в каком варианте невозможно. Следовательно, невозможно избежать возникновения коллизий.

Для решения проблемы коллизии можно использовать много способов. Одним из них является метод *рехеширования* (или расстановка). Согласно этому методу, если для элемента A адрес h(A), вычисленный с помощью хеш-функции h, указывает на уже занятую ячейку, то необходимо вычислить значение функции $n_1=h_1(A)$ и проверить занятость ячейки по адресу n_1. Если и она занята, то вычисляется значение $h_2(A)$ и так до тех пор, пока либо не будет найдена свободная ячейка, либо очередное значение $h_i(A)$ совпадет с h(A). В последнем случае считается, что таблица идентификаторов заполнена, и места в ней больше нет — выдается информация об ошибке размещения идентификатора в таблице.

Такую таблицу идентификаторов можно организовать по следующему алгоритму размещения элемента:

Шаг 1: Вычислить значение хеш-функции n = h(A) для нового элемента A.

Шаг 2: Если ячейка по адресу n пустая, то поместить в нее элемент A и завершить алгоритм, иначе i:=1 и перейти к шагу 3.

Шаг 3: Вычислить $n_i = h_i(A)$. Если ячейка по адресу n_i пустая, то поместить в нее элемент A и завершить алгоритм, иначе перейти к шагу 4.

Шаг 4: Если $n = n_i$, то сообщить об ошибке и завершить алгоритм, иначе i:=i+1 и вернуться к шагу 3.

Тогда поиск элемента A в таблице идентификаторов, организованной таким образом, будет выполняться по следующему алгоритму:

Шаг 1: Вычислить значение хеш-функции n = h(A) для искомого элемента A.

Шаг 2: Если ячейка по адресу n пустая, то элемент не найден, алгоритм завершен, иначе сравнить имя элемента в ячейке n с именем искомого элемента A. Если они совпадают, то элемент найден и алгоритм завершен, иначе i:=1 и перейти к шагу 3.

Шаг 3: Вычислить $n_i = h_i(A)$. Если ячейка по адресу n_i пустая или $n = n_i$, то элемент не найден и алгоритм завершен, иначе сравнить имя элемента в ячейке n_i с именем искомого элемента A. Если они совпадают, то элемент найден и алгоритм завершен, иначе i:=i+1 и повторить шаг 3.

Алгоритмы размещения и поиска элемента схожи по выполняемым операциям. Поэтому они будут иметь одинаковые оценки времени, необходимого для их выполнения.

При такой организации таблиц идентификаторов в случае возникновения коллизии алгоритм размещает элементы в пустых ячейках таблицы, выбирая их определенным образом. При этом элементы могут попадать в ячейки с адресами, которые потом будут совпадать со значениями хеш-функции, что приведет к возникновению новых, дополнительных коллизий. Таким образом, количество операций, необходимых для поиска или размещения в таблице элемента, зависит от заполненности таблицы.

Для организации таблицы идентификаторов по методу рехеширования необходимо определить все хеш-функции h_i для всех i. Чаще всего функции h_i определяют как некоторую модификации хеш-функции h. Например, самым простым методом вычисления функции $h_i(A)$ является ее организация в виде $h_i(A) = (h(A) + p_i) \bmod N_m$, где p_i — некоторое вычисляемое целое число, а N_m — максимальное значение из области значений хеш-функции h. В свою очередь, самым простым подходом здесь будет положить $p_i = i$. Тогда получаем формулу $h_i(A) = (h(A)+i) \bmod N_m$. В этом случае при совпадении значений хеш-функции для каких-либо элементов поиск свободной ячейки в таблице начинается последовательно от текущей позиции, заданной хеш-функцией h(A).

Этот способ нельзя признать особенно удачным — при совпадении хеш-адресов элементы в таблице начинают группироваться вокруг них, что увеличивает число необходимых сравнений при поиске и размещении. Но даже такой примитивный метод рехеширования является достаточно эффективным средством организации таблиц идентификаторов при неполном заполнении таблицы.

Среднее время на помещение одного элемента в таблицу и на поиск элемента в таблице можно снизить, если применить более совершенный метод рехеширования.

Одним из таких методов является использование в качестве p_i для функции $h_i(A)=(h(A)+p_i) \bmod N_m$ последовательности псевдослучайных целых чисел $p_1, p_2, \ldots, p_k$. При хорошем выборе генератора псевдослучайных чисел длина последовательности k будет $k=N_m$.

Существуют и другие методы организации функций рехеширования $h_i(A)$, основанные на квадратичных вычислениях или, например, на вычислении по формуле: $h_i(A) = (h(A)*i) \bmod N_m$, если N_m — простое число. В целом рехеширование позволяет добиться неплохих результатов для эффективного поиска элемента в таблице (лучших, чем бинарный поиск и бинарное дерево), но эффективность метода сильно зависит от заполненности таблицы идентификаторов и качества используемой хеш-функции — чем реже возникают коллизии, тем выше эффективность метода. Требование неполного заполнения таблицы ведет к неэффективному использованию объема доступной памяти.

Построение таблиц идентификаторов по методу цепочек

Неполное заполнение таблицы идентификаторов при применении хеш-функций ведет к неэффективному использованию всего объема памяти, доступного компилятору. Причем объем неиспользуемой памяти будет тем выше, чем больше информации хранится для каждого идентификатора. Этого недостатка можно избежать, если дополнить таблицу идентификаторов некоторой промежуточной хеш-таблицей.

В ячейках хеш-таблицы может храниться либо пустое значение, либо значение указателя на некоторую область памяти из основной таблицы идентификаторов. Тогда хеш-функция вычисляет адрес, по которому происходит обращение сначала к хеш-таблице, а потом уже через нее по найденному адресу — к самой таблице идентификаторов. Если соответствующая ячейка таблицы идентификаторов пуста, то ячейка хеш-таблицы будет содержать пустое значение. Тогда вовсе не обязательно иметь в самой таблице идентификаторов ячейку для каждого возможного значения хеш-функции — таблицу можно сделать динамической так, чтобы ее объем рос по мере заполнения (первоначально таблица идентификаторов не содержит ни одной ячейки, а все ячейки хеш-таблицы имеют пустое значение).

Такой подход позволяет добиться двух положительных результатов: во-первых, нет необходимости заполнять пустыми значениями таблицу идентификаторов — это можно сделать только для хеш-таблицы; во-вторых, каждому идентификатору будет соответствовать строго одна ячейка в таблице идентификаторов (в ней не будет пустых неиспользуемых ячеек). Пустые ячейки в таком случае будут только в хеш-таблице, и объем неиспользуемой памяти не будет зависеть от объема информации, хранимой для каждого идентификатора — для каждого значения хеш-функции будет расходоваться только память, необходимая для хранения одного указателя на основную таблицу идентификаторов.

На основе этой схемы можно реализовать еще один способ организации таблиц идентификаторов с помощью хеш-функций, называемый «метод цепочек». Для метода цепочек в таблицу идентификаторов для каждого элемента добавляется еще одно поле, в котором может содержаться ссылка на любой элемент таблицы. Первоначально это поле всегда пустое (никуда не указывает). Также для этого метода необходимо иметь одну специальную переменную, которая всегда указывает на первую свободную ячейку основной таблицы идентификаторов (первоначально — указывает на начало таблицы).

Метод цепочек работает следующим образом по следующему алгоритму:

Шаг 1: Во все ячейки хеш-таблицы поместить пустое значение, таблица идентификаторов пуста, переменная *FreePtr* (указатель первой свободной ячейки) указывает на начало таблицы идентификаторов; i:=1.

Шаг 2: Вычислить значение хеш-функции n_i для нового элемента A_i. Если ячейка хеш-таблицы по адресу n_i пустая, то поместить в нее значение переменной *FreePtr* и перейти к шагу 5; иначе перейти к шагу 3.

Шаг 3: Положить j:=1, выбрать из хеш-таблицы адрес ячейки таблицы идентификаторов m_j и перейти к шагу 4.

Шаг 4: Для ячейки таблицы идентификаторов по адресу m_j проверить значение поля ссылки. Если оно пустое, то записать в него адрес из переменной *FreePtr* и перейти к шагу 5; иначе j:=j+1, выбрать из поля ссылки адрес m_j и повторить шаг 4.

Шаг 5: Добавить в таблицу идентификаторов новую ячейку, записать в нее информацию для элемента A_i (поле ссылки должно быть пустым), в переменную *FreePtr* поместить адрес за концом добавленной ячейки. Если больше нет идентификаторов, которые надо разместить в таблице, то выполнение алгоритма закончено, иначе i:=i+1 и перейти к шагу 2.

Поиск элемента в таблице идентификаторов, организованной таким образом, будет выполняться по следующему алгоритму:

Шаг 1: Вычислить значение хеш-функции n для искомого элемента A. Если ячейка хеш-таблицы по адресу n пустая, то элемент не найден и алгоритм завершен, иначе положить j:=1, выбрать из хеш-таблицы адрес ячейки таблицы идентификаторов m_j.

Шаг 2: Сравнить имя элемента в ячейке таблицы идентификаторов по адресу m_j с именем искомого элемента A. Если они совпадают, то искомый элемент найден и алгоритм завершен, иначе перейти к шагу 3.

Шаг 3: Проверить значение поля ссылки в ячейке таблицы идентификаторов по адресу m_j. Если оно пустое, то искомый элемент не найден и алгоритм завершен; иначе j:=j+1, выбрать из поля ссылки адрес m_j и перейти к шагу 2.

При такой организации таблиц идентификаторов в случае возникновения коллизии алгоритм размещает элементы в ячейках таблицы, связывая их друг с другом последовательно через поле ссылки. При этом элементы не могут попадать в ячейки с адресами, которые потом будут совпадать со значениями хеш-функции. Таким образом, дополнительные коллизии не возникают. В итоге в таблице возникают своеобразные цепочки связанных элементов, откуда происходит и название данного метода — «метод цепочек».

Метод цепочек является очень эффективным средством организации таблиц идентификаторов. Среднее время на размещение одного элемента и на поиск элемента в таблице для него зависит только от среднего числа коллизий, возникающих при вычислении хеш-функции. Накладные расходы памяти, связанные с необходимостью иметь одно дополнительное поле указателя в таблице идентификаторов на каждый ее элемент, можно признать вполне оправданными. Этот метод позволяет более экономно использовать память, но требует организации работы с динамическими массивами данных.

Комбинированные способы построения таблиц идентификаторов

Выше в примере была рассмотрена весьма примитивная хеш-функция, которую никак нельзя назвать удовлетворительной. Хорошая хеш-функция распределяет поступающие на ее вход идентификаторы равномерно на все имеющиеся в распоряжении адреса, так что коллизии возникают не столь часто. Существует большое множество хеш-функций. Каждая из них стремится распределить адреса под идентификаторы по своему алгоритму, но, как было показано выше, идеального хеширования достичь невозможно.

То, какой конкретно метод применяется в компиляторе для организации таблиц идентификаторов, зависит от реализации компилятора. Один и тот же компилятор может иметь даже несколько разных таблиц идентификаторов, организованных на основе различных методов.

Как правило, применяются комбинированные методы. В этом случае, как и для метода цепочек, в таблице идентификаторов организуется специальное дополнительное поле ссылки. Но в отличие от метода цепочек оно имеет несколько иное значение. При отсутствии коллизий для выборки информации из таблицы используется хеш-функция,

поле ссылки остается пустым. Если же возникает коллизия, то через поле ссылки организуется поиск идентификаторов, для которых значения хеш-функции совпадают по одному из рассмотренных выше методов: неупорядоченный список, упорядоченный список или же бинарное дерево. При хорошо построенной хеш-функции коллизии будут возникать редко, поэтому количество идентификаторов, для которых значения хеш-функции совпали, будет не столь велико. Тогда и время поиска одного среди них будет незначительным (в принципе, при высоком качестве хеш-функции подойдет даже перебор по неупорядоченному списку).

Такой подход имеет преимущество по сравнению с методом цепочек, поскольку не требует использования промежуточной хеш-таблицы. Недостатком метода является необходимость работы с динамически распределяемыми областями памяти. Эффективность такого метода, очевидно, в первую очередь зависит от качества применяемой хеш-функции, а во вторую — от метода организации дополнительных хранилищ данных.

Хеш-адресация — это метод, который применяется не только для организации таблиц идентификаторов в компиляторах. Данный метод нашел свое применение и в операционных системах, и в системах управления базами данных.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Получить вариант задания у преподавателя.
2. Написать и отладить программу на ЭВМ.
3. Подготовить и защитить отчет.
4. Сдать работающую программу преподавателю.

ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ОТЧЕТА

Отчет по лабораторной работе должен содержать следующие разделы:

- Задание по лабораторной работе.
- Схему организации хеш-таблицы (в соответствии с вариантом задания).
- Описание алгоритма поиска в хеш-таблице (в соответствии с вариантом задания).
- Выводы по проделанной работе.

ВАРИАНТЫ ЗАДАНИЙ

Во всех вариантах требуется разработать программу, реализующую комбинированный способ организации таблицы идентификаторов. Для организации таблицы используется простейшая хэш-функция, указанная в варианте задания, а при возникновении коллизий используется дополнительный метод размещения идентификаторов в памяти. Если в качестве этого метода используется дерево или список, то они должны быть связаны с элементом главной хэш-таблицы.

В каждом варианте требуется, чтобы программа сообщала среднее число коллизий и среднее количество сравнений, выполненных для поиска идентификатора.

№	Тип хеш-функции (таблицы)	Способ разрешения коллизий
1.	Сумма кодов первых трех букв	Бинарное дерево
2.	Сумма кодов первых трех букв	Список с простым перебором
3.	Сумма кодов первых трех букв	Метод цепочек
4.	Сумма кодов первых трех букв	Простое рехеширование

№	Тип хеш-функции (таблицы)	Способ разрешения коллизий
5.	Сумма кодов первых трех букв	Рехеширование с использованием случайных чисел
6.	Сумма кодов первой, второй и последней букв	Бинарное дерево
7.	Сумма кодов первой, второй и последней букв	Список с простым перебором
8.	Сумма кодов первой, второй и последней букв	Метод цепочек
9.	Сумма кодов первой, второй и последней букв	Простое рехеширование
10.	Сумма кодов первой, второй и последней букв	Рехеширование с использованием случайных чисел
11.	Сумма кодов первой, предпоследней и последней букв	Бинарное дерево
12.	Сумма кодов первой, предпоследней и последней букв	Список с простым перебором
13.	Сумма кодов первой, предпоследней и последней букв	Метод цепочек
14.	Сумма кодов первой, предпоследней и последней букв	Простое рехеширование
15.	Сумма кодов первой, предпоследней и последней букв	Рехеширование с использованием случайных чисел
16.	Сумма кодов последних трех букв	Бинарное дерево
17.	Сумма кодов последних трех букв	Список с простым перебором
18.	Сумма кодов последних трех букв	Метод цепочек
19.	Сумма кодов последних трех букв	Простое рехеширование
20.	Сумма кодов последних трех букв	Рехеширование с использованием случайных чисел