**1) Основні способи організації пошуку в таблицях**

Один из наиболее распространенных линейный по ключу в неупорядоченных структурах данных. При реализации такого алгоритма ключевой аргумент поиска должен поочередно сравниваться со значениями ключей в элементах или рядах таблиц. В компиляторах, как правило, используется поиск по однозначным ключам, поэтому результат поиска можно получить при первом совпадении аргумента с ключом. Для ускорения поиска выполняется два дополнительных подхода – упорядочивание и поиск по прямому адресу, который может быть выполнен в условиях небольшого диапазона значений аргументов поиска. Например, для классификатора букв в лексическом анализаторе, если буквы заданны в Аски-коде достаточно значений 0..255. Чтоб засечь символ в AL необходимо

Mov al, tab[eax]

Xlat

Алгоритм линейного поиска в неупорядоченной таблице

1. Установка индекса первого элемента таблицы.
2. Сравнение ключа искомого элемента с ключом элемента таблицы.
3. Если ключи равны,перейти на обработку ситуации "поиск удачный",иначе на 4.
4. Инкремент индекса элемента таблицы.
5. Проверка конца таблицы.

BINSRCHARG STRUC ; структура для передачі параметрів

dd ? ; проміжок для BP та адреси повернення

ProtPtr dd ? ; вказівник елементу, що перевіряється

ArgLn dw ? ; довжина елементу, що перевіряється

TabPtr dd ? ; вказівник початку таблиці

TabLn dw ? ; кількість елементів таблиці

ElmLn dw ? ; довжина елементу таблиці

BINSRCHARG ENDS

public BinSrch

BinSrch PROC

push BP ; збереження старого BP

mov BP, SP ; Визначення бази зони параметрів

push DS ; збереження сегменту даних

les DI, ProtPtr[BP] ; підготовка адреси аргументу

LD6 SI, TabPtr[BP] ; підготовка адреси таблиці

xor BX, BX ; встановлення початкової нижньої межі

lb:

add AX, TabLn[BP] ; Обчислення

shr AX, 1 ; номеру середнього аргументу

cmp AX, BX ; Контроль границі розбивання

jz NotFound ; Перехід, якщо таблиця переглянута

push AX

mul ElmLn[BP] ; Визначення зміщення елемента

push SI ; Збереження адреси наступного елемента

push DI ; Збереження адреси аргументу пошуку

add SI, AX ; Обчислення адреси середнього елемента

mov CX, ArgLn[BP] ; Підготовка довжини аргумента

repz CMPSB ; Використання групового ресурсу порівняння

pop DI ; Відновлення адреси аргументу пошуку

jz Found

pop SI ; Відновлення адреси наступного елементу

ja LCorrUp

pop BX ; корекція нижньої межі індексу

jmp lb

LCorrUp:

pop TabLn[BP] ; корекція верхньої межі індексу

jmp lb

NotFound:

mul ElmLn[BP] ; визначення зміщення елементу

add AX, SI ; визначення зміщення в сегменті

xor DI, DI ; формування признаку відсутності

jmp SHORT Lret

Found:

add SP, 4 ; компенсація записів у стек

Lret:

mov DX, DS ; формування сегментної частини вказівника

pop DS ; відновлення сегменту даних

pop BP

RET

BinSrch ENDP

Приклад реалізації [двійкового пошуку](http://uk.wikipedia.org/wiki/Двійковий_пошук) на [Java:](http://uk.wikipedia.org/wiki/Java)

public static int binarySearch(int[] a, int key) {

int low = 0;

int high = a.length - 1;

while (low <= high) {

int mid = (low + high) >>> 1;

int midVal = a[mid];

if (midVal < key)

low = mid + 1;

else if (midVal > key)

high = mid - 1;

else

return mid; // key found

}

return -(low + 1); // key not found.

}

***Вопрядкування таблиць і методи двійного пошуку***

1. Загрузка нач. (Ан) и кон. (Ак) адресов таблицы
2. Определение адреса ср. элемента таблицы (Аср)
3. Сравнение искомого ключа с ключевой частью ср. элемента таблицы
4. При равенстве ключей поиск удачный. Если искомый ключ меньше ключа среднего элемента, то Ак=Аср, переход на 5.Если искомый ключ больше ключа среднего элемента,то Ан=Аср+длина элемента,переход на 5
5. Сравнение Ан и Ак. Если Ан=Ак, поиск неудачный, иначе переход на 2

**При определении адреса среднего элемента следует выполнить следующие действия:**

* вычислить длину таблицы Ак-Ан
* определить число элементов таблицы (Ак-Ан)/Lэ, где Lэ- длина элемента
* определить длину половины таблицы ((Ак-Ан) /Lэ)/2\*Lэ
* определить адрес среднего элемента таблицы Аср= Ан + ((Ак-Ан)/Lэ)/2\*Lэ

***Пошук за прямою адресою.***

Поиск в древовидных структурах осуществляется рекурсивно. Необходимо составить процедуру, в которую, как параметры передаются ссылка на элемент дерева и ключ поиска. Процедура при начальной инициализации получает ссылку на вершину дерева. В теле процедуры проверяется равенство ключа поиска с ключом элемента дерева, если равенство выполняется, то считываются значения всех полей данного элемента дерева. За тем процедура вызывает сама себя с указанием ссылок из текущего элемента. Таким образом осуществляется столько вызовов процедуры, сколько ветвей у данного элемента дерева. Такой алгоритм позволяет пройти по всем элементам неоднородного дерева.

***Хеш пошуку.***

Алгоритм:

1. Формирование хеш-таблицы
2. Выбор искомого ключа
3. Вычисление хеш-функции ключа Н(Кi)
4. Вычисление хеш-адреса ключа
5. Сравнение ключевой части элемента таблицы по вычисленному хеш-адресу с искомым ключом. При равенстве обработка ситуации "поиск удачный "и переход на 6; при неравенстве - обработка ситуации "поиск неудачный" и переход на 6.
6. Проверка все ли ключи выбраны; если да, то конец, а если нет ,то переход на 2.

**2) Метод синтаксичних графів для синтаксичного аналізу**

**В цьому методі послідовність правил синтаксичного розбору організована у формі графа, кожен вузол якого має 4 складових :**

1. **мітка, що призначена для зв’язування вузлів графа**
2. **розпізнавач, який може розпізнати термінальні або не термінальні елементи синтаксичної конструкції для синтаксичного аналізу в компіляторах. За термінальні позначення обирають імена, константи, ключові слова, тощо. За нетермнальні – вирази, списки, оператори, тощо.**
3. **покажчик на наступний вузол графу, до якого відбувається перехід у випадку успішної роботи розпізновача**
4. **показник на альтернативне відгалуження синтаксичного графу, коли робота розпізнувача не буде успішною. По формі структур даних синтаксичний граф може бути представлений як сукупність вузлів з двійковим розг. У випадку коли синтаксична конструкція виявиться неправильною необхідно передбачити нейтралізацію помилок, тобто видачу діагностики з пошуком конструкції після якої можна продовжити синтаксичний аналіз.**

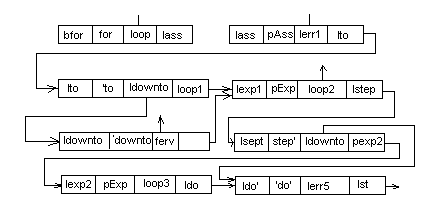
Результати синтаксичного розбору подаються у вигляді зв’язаних вузлів графа розбору, в якому кожний вузол відповідає окремій лексемі, а зв’язки визначають підлеглість операндів до операцій. Вузол об’єднує 4 поля:

* Ідентифікація або мітка вузла
* Прототип співставлення ( термінал або нетермінал )
* Мітка продовження обробки ( при успішному результаті синтаксичного аналізу )
* Вказівник на альтернативну вітку, яку можна перевірити ( якщо аналіз був невдачним)

Синтаксично буває зручним використовувати змішані алгоритми аналізу. Як раз для обробки операторів мов програмування краще використовувати синтаксичний граф, а для обернених виразів – висхідний розбір.

Також існують зв’язки передачі управління в операторах розгалуження та блоках циклів та варіантному блоці. Таким чином будь-який закінчений фрагмент програми має головний вузол, в якому як підлеглі вузли різних рівнів зберігаються лексеми та синтаксичні структури, які входять до цього блоку. Такий граф розбору є основою для всіх видів подальшої семантичної обробки.

Методи низхідного розбору ефективніші при обробці структурованих операторів(for)



**3) Організація генерації кодів**

Для генерації машинних кодів слід використовувати машинні команди або підпрограми з відповідними аргументами. В результаті генерації кодів формуються машинні команди або послідовності виклику підпрограм або функцій, які повертають потрібний результат. Більшість компіляторів системних програм включає такі коди в об’єктні файли з розширенням OBJ. Такі файли включають 4 групи записів:

* Записи двійкового коду – двійкові коди констант, машинні коди, відносні адреси відносно початку відповідного сегменту
* Записи ( елементи ) переміщуванності – спосіб настроювання відповідної відносної адреси
* Елементи словника зовнішніх посилань – фіксуються імена, які заявлені як зовнішні або доступні для зовнішніх посилань
* Кінцевий запис модуля – для розділення модулю

Генерація кодів в цілому являє собою формування машинних команд і збірку об’єктного і загрузочного модулів. При генерації об’єктних кодів виконується направлений прохід по графу. Для реалізації такого генератора потрібно визначити повні таблиці співставлення усіх можливих вузлів графа з потрібним набором машинних команд.

Більш ранні компілятори одразу формувалимашинні коди. Компілятори з мови Паскаль формували свої результати у так званих Р-кодах. На етапі виконання цей код оброблювався середовищем Паскаль, яке включало підпрограми для обробки всіх операндів та операторів. Ця схема є проміжною між копіюючою та інтерпретуючою програмами. Тобто за такою схемою для даних виділяють як правило фіксовану пам’ять, а до кодів звертається виконуюча програма, яка дозволяє формувати результат виконання порграми. Але щоб раціонально організовувати модульне програмування необхідно, щоб поєднувані модулі подавалися в однаковому форматі, тому в традиційн. Реалізаціях Паскаля вони добре поєднувалися з модулями на цій же мові, але погано з іншими мовами. Результати трансляції формуються у вигляді obj файлів. В них використ. 4 загальні типи записів або елементів. Генератор кодів формує команди з дрібних фрагментів команд, операцій, імен або адрес даних, індексних і базових регістрів. У формі напрямленого ациклічного графу для семантично коректної роботи програми є достатньо інф. Для її інтерпретації(ген. кодів). Генерація кодів складається з формування окремих машинних команд і зборки з них об’єктного і завантажувального модулів. Для виклику стандартних підпрограм, що використовуються у мові використовують бібліотеки періоду виконання, яка може співпадати з бібліотекою інтерпретатора. При генерації кодів виконується напрямлений перегляд ациклічного графу, де генератор породжує команди обробки цих даних. Для реалізації генераторів необхідно визначити повні табл. Відповідності усіх можливих вузлів напрямленого ациклічного графу у необх. Набор машинних команд