Учебник RATFIV

Ratfiv - это препроцессор Fortran. Его основная цель - поощрять читаемый и хорошо структурированный код, используя преимущества универсальности, переносимости и эффективности Fortran. Приложив скромные усилия, программисты на основе Fortran могут повысить свою продуктивность, используя язык, который предоставляет им управляющие структуры и косметические функции, необходимые для проектирования структурного программирования. Время отладки и последующей ревизии намного быстрее, чем аналогичные усилия в Fortran, главным образом потому, что код легко читается. Таким образом, становится проще писать код, который будет читабельным, надежным и даже эстетически приятным, а также переносимым в другие среды.

Ratfiv поддерживает все функции обычного Fortran, а также предоставляет следующие управляющие структуры:

* условные выражения "if" - "else"
* конструкции цикла конструкции цикла "while", "for", "do" и "repeat" - "until"
* оператор переключения "case"
* "break" и "next" для выхода из цикла
* группировка операторов с помощью фигурных скобок

Косметические аспекты Ratfiv были разработаны таким образом, чтобы сделать его лаконичным, удобным для ухода и приятным для глаз:

* инструкция "include" для включения исходных файлов
* оператор "return value" в функциях
* форматы, указанные в операторах read, write, encode и decode
* оператор "string" для инициализации символьных массивов
* указание чисел в основаниях 2-36
* ввод в свободной форме
* условность ненавязчивых комментариев
* перевод >, <=, и т. д. и .GT., .LE., и т. д.
* строки символов в кавычках
* условная компиляция

Кроме того, Ratfiv предоставляет сложный макропроцессор для определения символьных констант, а также макросов с аргументами.

Программа Ratfiv реализована как препроцессор, который переводит вышеуказанные функции в Fortran (опционально VAX Fortran 77), который затем может быть загружен практически в любой компилятор Fortran. Программы Ratfiv могут быть написаны как в верхнем, так и в нижнем регистре, однако ключевые слова Ratfiv (такие как "if", "else" и т. д.) должны появляться полностью в верхнем или нижнем регистре, например, "If" не распознается как оператор "if". В разделе этого документа под названием ["Использование Ratfiv"](#_Использование_RATFIV_1) рассказывается, как вызвать Ratfiv в вашей системе.

Теперь мы обсудим каждую из функций Ratfiv более подробно. Далее "оператор" - это любой допустимый оператор Fortran: присваивание, объявление, вызов подпрограммы, ввод-вывод и т. д. Или любой из операторов Ratfiv. Любой оператор Fortran или Ratfiv или их группу можно заключить в фигурные скобки - { } - или квадратные скобки - [ ] -, чтобы сделать его составным оператором, который затем эквивалентен одному оператору и может использоваться везде, где может использоваться один оператор.

# IF-ELSE

Компилятор Ratfiv предоставляет операторы "if" и "else" для обработки конструкции "если условие истинно, сделайте это, в противном случае сделайте то". Их синтаксис:

**if (допустимое условие Fortran)**

**оператор(ы)**

**else**

**оператор(ы)**

где часть else является необязательной. Поле "допустимое условие Fortran" - это все, что может легально перейти в логический IF на Fortran. Операторы Ratfiv могут быть одним или несколькими действительными операторами Ratfiv или Fortran любого рода. Если требуется более одного оператора, они должны быть заключены в фигурные скобки. Например,

**if (a > b)**

**{**

**k = 1**

**call remark (...)**

**}**

**else if (a < b)**

**{**

**k = 2**

**call remark (...)**

**}**

**else**

**return**

# WHILE

Ratfiv предоставляет оператор while, который представляет собой просто цикл: "пока выполняется какое-то условие, повторите эту группу операторов". Его синтаксис:

**while(допустимое условие Fortran)**

**оператор(ы)**

Как и в случае с if, "допустимое условие Fortran" может быть включено в логический IF Fortran. Условие проверяется перед выполнением любого из операторов Ratfiv, поэтому, если условие не выполняется, цикл будет выполняться ноль раз. Также, как и в случае с IF, операторы Ratfiv могут быть любыми допустимыми конструкциями Ratfiv или Fortran. Если требуется более одного оператора, они должны быть заключены в фигурные скобки. Например,

**WHILE (GETC(C) != EOF) [**

**C = CNVT(C)**

**CALL PUTC(C)**

**]**

Обратите внимание, что для Ratfiv вполне приемлемы прописные буквы, и что вместо фигурных скобок ( { } ) можно использовать скобки ( [ ] ).

# FOR

Оператор "for" аналогичен оператору "while" за исключением того, что он допускает явную инициализацию и шаги приращения как часть оператора. Он имеет синтаксис:

**for (предложение инициализации; условие; предложение приращения)**

**оператор(ы)**

где "предложение" означает одиночный оператор Fortran или группу операторов Fortran, заключенных в скобки и разделенных точками с запятой. "Предложение инициализации" выполняется один раз перед началом цикла. "Предложение приращения" выполняется в конце каждого прохода цикла перед тестом. "Условие" - это снова все, что разрешено в логическом IF. Как и в случае с оператором while, условие проверяется перед выполнением любого из операторов Ratfiv, поэтому, если условие не выполняется, цикл будет выполняться ноль раз. Любые из параметров инициализации, условия и приращения могут быть опущены, хотя точки с запятой должны оставаться. Несуществующее условие рассматривается как всегда истинное, поэтому "for(;;)" - это неопределенное повторение. Оператор "for" особенно полезен для объединения списков, циклы, которые можно выполнить ноль раз, и тому подобные вещи, которые трудно выразить с помощью оператора DO. Вот два примера циклов for:

**for ({ nch=0; i=1 }; getarg(i, file, MAXLINE) != EOF; i=i+1)**

**{**

**int = open (file, READ)**

**while (getlin (line, int) != EOF)**

**for (j = length(line); j > 0; { j=j-1; nch=nch+1 })**

**call putc (line(j))**

**call close (int)**

**}**

Приведенный выше код читает из списка файлов и записывает каждую строку из каждого файла в обратном порядке. Переменная "i" отслеживает количество прочитанных файлов, а переменная "nch" отслеживает количество записанных символов. Знак "!=" Означает .NE.

# DO

Оператор "do" похож на цикл do-loop в Fortran. Синтаксис:

**do Fortran часть-do**

**оператор(ы)**

Поле Fortran часть-do - это то, что Fortran примет после выполнения, за исключением метки оператора. Если требуется более одного оператора, их необходимо заключить в квадратные скобки. Например:

**do i = 1, 4**

**{**

**do j = 1, 4**

**x(i,j) = 0.0**

**x(i,i) = 1.0**

**}**

В этом примере матрица 4 на 4 инициализируется единичной матрицей.

# REPEAT-UNTIL

Операторы "repeat" и "until" позволяют повторять группу операторов до тех пор, пока не будет выполнено указанное условие. Синтаксис:

**repeat**

**оператор(ы)**

**until (условие)**

Оператор "until (условие)" не является обязательным. Поле "условие" - это снова все, что разрешено в логическом IF. «**условие**» проверяется в конце цикла, так что будет выполнен хотя бы один проход через цикл. Если часть "until (условие)" опущена, результатом будет бесконечный цикл, который необходимо прервать с помощью оператора "break" или "next" ([см. Ниже](#_BREAK_и_NEXT)). Еще раз, если требуется более одного оператора, они должны быть заключены в квадратные скобки. Пример цикла repeat-until:

**repeat**

**{**

**call putc (BLANK)**

**col = col + 1**

**}**

**until (col >= tabpos)**

В примере всегда помещается хотя бы один пробел.

# SWITCH

Оператор switch - это многосторонний механизм принятия решений, который позволяет выбрать один путь выполнения из множества на основе значения выражения. Его синтаксис:

**switch (допустимое выражение fortran)**

**{**

**case выражение(я):**

**оператор(ы)**

**.....**

**default:**

**оператор(ы)**

**}**

Допустимое выражение fortran в круглых скобках оценивается и затем проверяется, чтобы определить, какому из константных выражений в каждом операторе case оно соответствует. Когда найдено подходящее значение, в этом case запускается выполнение. После того, как операторы в соответствующем case выполнены, выполнение продолжается после оператора "switch". case с пометкой "default" выполняется, если ни один из других case не соответствует. "default" не является обязательным. Если он опущен и ни один из вариантов не совпадает, никаких действий не предпринимается и выполнение прерывается до конца оператора case. Поле "выражение(я)" может быть списком констант, разделенных запятыми, единственной константой или диапазоном значений констант, разделенных знаком "-" (минус). Константы должны быть целыми числами, символьными константами в форме 'c' или "c", или экранированная символьная константа. (См. Раздел, посвященный оператору ["string"](#_STRING_1), для описания экранированных символьных констант.) Кроме того, """" обозначает буквальную двойную кавычку, а '''' обозначает буквальную одинарную кавычку. case и default могут указываться в любом порядке и должны отличаться. Операторы case и default могут указываться в любом порядке и должны отличаться. Обратите внимание, что заключительные скобки необходимы после "switch (допустимое выражение fortran)" и в конце оператора "switch". Скобки **не** нужны после "case" или "default". Ниже приведен пример оператора switch:

**switch (lin(i))**

**{**

**case 0: return # конец строки**

**case "0"-"9":**

**j = ctoi(lin, i) # преобразовать в число**

**k = max(j,k)**

**case "A"-"Z", "a"-"z":**

**.... # сделать что-нибудь**

**.... #**

**default:**

**write(5, (' error at lin(',i3,')')) i**

**}**

Ratfiv может определить, лучше ли реализовать данный оператор "switch" как Fortran вычисляемый в goto или как последовательные логические тесты if. Есть одна возможная проблема с оператором switch, он генерирует переменную, начинающуюся с буквы "i". Если переменные, начинающиеся с "i", объявлены неявно, кроме целочисленных (например, с помощью оператора "implicit real (a - z)", тогда в некоторых случаях оператор "switch" не будет работать должным образом.

# BREAK и NEXT

Компилятор Ratfiv предоставляет инструкции для досрочного выхода из цикла и для начала следующей итерации.

Оператор "break" вызывает немедленный выход из любого цикла (который может быть "while", "for", "do" или "repeat"), в котором он содержится. Управление возобновляется со следующего оператора после цикла. Если за словом "break" следует число, то это множество включающих циклов завершается, в противном случае цикл, в котором появляется "break", завершается. Например:

**repeat**

**{**

**if (getc(c) == EOF) break**

**....**

**}**

Оператор "next" - это переход к концу цикла, поэтому он вызывает выполнение следующей итерации. Оператор "next" переходит к условной части "while" или "until", к началу бесконечного цикла "repeat", к части приращения "for" и к следующей итерации "do". Если число указано после «следующего», то управление передается циклу, который представляет собой множество вложенных циклов, выходящих из "next", в противном случае управление передается охватывающему циклу.

Для примера в следующей программе:

**for (i = 1; i < 10; i = i+1)**

**{**

**if (array(i) == BLANK) next**

**....**

**}**

"next" вызывает переход к части приращения цикла "for", которая добавляет 1 к "i".

**RETURN**

Оператор "return" может использоваться как обычно. Однако в подпрограмме функции значение функции может быть неявно присвоено перед возвратом с использованием следующего синтаксиса:

**return значение**

Здесь "значение" - это значение подпрограммы функции . Например:

**integer function index(char,strng)**

**byte strng(80), char**

**for (i = 1; i <= 80; i = i+1)**

**if (strng(i) == char)**

**return i**

**return 0**

**end**

# Операторы FORMAT

Спецификации format могут быть включены в операторы read, write, encode и decode путем включения спецификации, заключенной в круглые скобки, в ту же позицию в операторе, где обычно появляется номер оператора format. Например:

**write(5, ('$FILE? '))**

**read(5, (80a1), end=99) file**

Форматы могут быть указаны обычным способом, указав обычный номер оператора вместо спецификации формата в скобках. Например:

**read(5, 10, end=99) file**

**10 format(80a1)**

# STRING

Оператор "string" определяет размер и содержимое символьного массива. Синтаксис следующий:

**string name "character string"**

или

**string name(size) "character string"**

Первая форма оператора string определяет "name" как массив символов (байт по умолчанию в системах DEC), достаточно большой, чтобы вместить все символы в строке символов плюс завершающий символ конец строки (по умолчанию 0 (nul) в системах DEC).

Вторая форма выделяет символы "size" для "name" и инициализирует его указанной символьной строкой с символом EOS. "size" должен быть достаточно большим, чтобы вместить всю строку плюс терминатор, и может быть больше.

Одинарные кавычки могут использоваться вместо двойных кавычек для разграничения символьной строки. Как и в случае с литералами в кавычках, две соседние двойные кавычки, встречающиеся в строке в двойных кавычках, интерпретируются как литеральная двойная кавычка, а две соседние одинарные кавычки в строке в одинарных кавычках интерпретируются как литеральная одинарная кавычка.

Символ "@" в строке имеет особое значение. Он и следующий за ним символ заменяются одним из специальных символов, показанных ниже, в зависимости от символа, следующего за "@":

@@ заменяется на @

@B или @b заменяется пробелом

@E или @e заменяется на конец строки (по умолчанию 0 в системах DEC)

@F или @f заменяется переводом страницы

@G или @g заменяется звонком

@L или @l заменяется на перевод строки

@N или @n заменяется NEWLINE (перевод строки по умолчанию в системах DEC)

@R или @r заменяется возвратом каретки

@T или @t заменяется табуляцией

@V или @v заменяется вертикальной табуляцией

@$ заменяется escape

@цифры заменяется восьмеричным значением "цифры"

Символ "@", за которым следует любой символ, не указанный выше, заменяется символом, следующим за "@". Исключение составляют случаи, когда "@" стоит перед символом завершающей кавычки; в этом случае "@" интерпретируется буквально как "@" и не имеет особого значения.

Все операторы string должны появляться вместе после всех обычных объявлений Fortran и перед любыми операторами DATA. В операторе string может быть объявлено более одной строки, если объявления разделены запятыми.

Примеры:

**string error '@gError reading file. Execution terminated.'**

**string infile(FILENAMESIZE) "TI:", outfil(FILENAMESIZE) "TO:"**

**string del "@177" # восьмеричное число 177 - символ ASCII "delete"**

*Примечания для опытных пользователей:*

Значения по умолчанию, используемые для замены "@e" и "@n" в строках (см. выше), равны 0 и 10 в системах DEC. Эти значения по умолчанию можно переопределить, определив символы EOS и NEWLINE соответственно как целое число или символ в кавычках, который затем будет использоваться вместо значения по умолчанию.

Аналогично, тип данных по умолчанию для строк (byte в системах DEC) и завершающий символ по умолчанию (0 в системах DEC) могут быть изменены путем переопределения символов EOS и character соответственно.

Примеры:

**sbcne(character,integer)**

**define(EOS,-1)**

**....**

**string comnds "READ@eWRITE"**

определяет comnds как целочисленный массив длины 11 и создает операторы данных, которые инициализируют его. Элемент comnds(5) инициализируется в -1, как и comnds(11).

Вполне законно давать значение для строки, которая не заключена в кавычки. В этом случае в качестве строкового значения используется первый непустой токен после объявления имени (и необязательного размера). Это полезно при определении строки, которая должна принимать значение макроса, например:

**define(character,byte)**

**....**

**string chrdef character**

инициализирует строку chrdef как "byte".

# INCLUDE

Файлы могут быть вставлены во входной поток с помощью оператора "include". Оператор

**include имя\_файла**

или

**include "имя\_файла"**

или

**include 'имя\_файла'**

вставляет файл, найденный во входном файле **"**имя\_файла**"**, во вход Ratfiv вместо оператора include. Это особенно полезно при вставке common блоков. Например,

**function exampl (x)**

**include "comblk.cmn"**

**exampl = x + z**

**return**

**end**

может быть переведено на

**function exampl (x)**

**common /comblk/ q, r, z**

**real q, r, z**

**exampl = x + z**

**return**

**end**

Форма оператора include, в которой используются одинарные или двойные кавычки вокруг имени файла, является предпочтительной и необходима, если в имени файла есть забавные символы, такие как запятая.

# Объединение операторов и пустой оператор

Ratfiv позволяет рассматривать группу операторов как единое целое, заключая их в фигурные скобки - { и } или [ и ]. Это верно для всего языка: везде, где можно использовать один оператор, несколько операторов могут быть заключены в фигурные скобки. Например:

**if (x > 100)**

**{**

**call error (...)**

**err = 1**

**return**

**}**

Если фигурные скобки не являются допустимыми символами в локальной операционной системе или если вы хотите использовать только верхний регистр, символы **"**[**"** и **"**]**"** могут использоваться вместо **"**{**"** и **"**}**"** соответственно.

Компилятор Ratfiv также позволяет использовать пустые операторы, наиболее полезные после операторов "for" и "while". Только точка с запятой указывает на пустой оператор. Например,

**while (getlin(line, int) != EOF) ;**

будет читать строки из файла до тех пор, пока не будет достигнут конец файла и

**for (i=1; line(i) == BLANK; i=i+1) ;**

позиция "i" после ведущих пробелов в строке.

# Ввод в свободной форме

Операторы могут быть размещены в любом месте строки, и несколько могут появиться в одной строке, если они разделены точкой с запятой, однако нет необходимости разделять операторы, начинающиеся с ключевых слов Ratfiv, точкой с запятой. Точка с запятой в конце каждой строки не требуется, потому что Ratfiv предполагает, что в каждой строке есть один оператор, если не указано иное. Однако Ratfiv будет продолжать строки, когда кажется очевидным, что они еще не закончены, или если строка явно заканчивается подчеркиванием ("\_"). Обратите внимание, что подчеркивание не включено в Вывод Fortran Ratfiv. Не пытайтесь использовать символ продолжения в столбце 6 для продолжения строки.

Любой оператор, начинающийся с полностью числового поля, считается меткой Fortran и помещается в столбцы 1-5 при выводе. Ratfiv генерирует метки, начинающиеся с 2000 и увеличивающиеся с интервалом в 10, поэтому старайтесь использовать метки ниже 2000 для собственных целей.

Операторы могут передаваться через компилятор Ratfiv неизмененными двумя способами; если символ переключения "%" появляется в строке сам по себе до и после строк, которые должны быть литеральными, строки будут передаваться через неизмененный Fortran. Однако если строка, в которой появляется символ "%", содержит другие непустые символы после символа "%", эти символы будут переданы в Fortran, и соответствующий символ "%" не нужен. Это удобный способ передачи обычного кода Fortran через компилятор Ratfiv. Обратите внимание, что "%" в начале строки не считается столбцом Fortran, поэтому код должен иметь отступ в 6 пробелов после "%", или символ табуляции может быть использован, если он допустим в вашем Fortran.

# Комментарии

Символ решетки "#" в строке обозначает начало комментария, и остальная часть строки считается этим комментарием. Комментарии и код могут сосуществовать в одной строке. Например,

**function dummy (x)**

**# Я создал эту функцию, чтобы показать некоторые комментарии**

**dummy = x # Я просто возвращаю параметр**

**return**

**end**

# Перевод символов

Иногда символы >, <= и т. д. легче читать в выражениях условий Fortran, чем стандартные символы Fortran .EQ., .LT. И т. д. Ratfiv допускает любое соглашение. Если используются специальные символы, они переводятся следующим образом:

== .EQ.

!= или ^= или ~= .NE.

< .LT.

> .GT.

<= .LE.

>= .GE.

| или .OR.

& .AND.

! или ^ или .NOT.

Например,

**for (i = 1; i <= 5; i = i+1)**

**...**

**if (j != 100)**

**...**

# Указание чисел в основаниях, отличных от основания десять

Числа могут быть указаны в любом из оснований 2-36. По умолчанию используется основание десять. Числа в других основаниях задаются как n%dddd..., где "n" - десятичное число, указывающее основание, а "dddd..." - цифры в этом основании. Цифры больше 9 обозначаются буквами a-z(A-Z), где, например,

16%ff означает 255 по основанию 10

8%100000 означает -32768 по основанию 10 на PDP11

8%100000 означает +32768 по основанию 10 на VAX

# Строки символов в кавычках

Строки символов могут быть заключены в одинарные или двойные кавычки. Чтобы указать одинарную кавычку внутри строки с одиночными кавычками, повторите одинарную кавычку дважды, например:

**write(5, (' can''t open file'))**

Точно так же, чтобы включить символ двойной кавычки в строку с двойными кавычками, поместите две двойные кавычки в строку внутри строки.

Восьмеричные константы, которые определены предшествующими двойными кавычками, как в DEC Fortran, будут успешно переданы через Ratfiv, если в каждой строке появляется только одна восьмеричная константа; в противном случае Ratfiv будет считать, что нужна строка в кавычках. Однако рекомендуется использовать метод Ratfiv для определения оснований, отличных от десятичного ([см. Выше](#_Указание_чисел_в)).

Точно так же операторы чтения и записи прямого доступа, которые имеют одинарную кавычку перед номером записи, также будут успешно переданы через Ratfiv, если в строке появляется только одна одинарная кавычка. Например,

**write(5'n) x, y, z**

будет передана в Fortran как

**WRITE(5'N) X, Y, Z**

однако

**write(5'n, ('ATOM '3f10.5)) x, y, z**

будут не правильно интерпретированы из-за несоответствия кавычек.

# Определение макросов

Оператор "define" позволяет расширить синтаксис Ratfiv определениями макросов. Макрос - это буквенно-цифровой символ, с которым связано определение; всякий раз, когда этот символ появляется во входных данных, он заменяется его определением. Процесс замены называется «макрорасширением». Простейшее использование «определения» - определение символьной константы; после этого всякий раз, когда эта символическая константа встречается во входных данных, она заменяется определением этой константы. Оператор "define" может использоваться для создания этих символических констант:

**define(ROW,10)**

**define(COLUMN,25)**

**dimension array (ROW, COLUMN)**

и

**define(EOF,-10)**

**if (getlin(line, int) == EOF)**

**....**

Такой макрос, как ROW, полезен, потому что его имя более значимо, чем число "10". Кроме того, гораздо проще модифицировать программы, использующие символические константы. Если бы ROW использовался последовательно во всей программе, тогда нужно было бы изменить только определение ROW, когда требуется больше (или меньше) строк в "array".

Определения могут быть включены в любое место кода, если они появляются до ссылки на макрос. Определенные имена могут содержать буквы, цифры и символы подчеркивания (\_) и доллара ($). Регистр имеет значение, поэтому имена в верхнем регистре отличаются от имен в нижнем регистре.

# Макросы с аргументами

Также можно определять макросы с аргументами. Например:

**define(bump,$1 = $1+1)**

определяет макрос, который можно использовать для увеличения переменной на 1. Когда на входе появляется

**bump(i)**

он расширяется до:

**i = i + 1**

Символ "$1" в определении макроса является заполнителем для первого аргумента макроса. Когда макрос раскрывается, все вхождения "$1" заменяются фактическим первым аргументом, в данном случае "i". Допускается до девяти аргументов макроса, не считая аргумента 0, который относится к имени макроса. Вот еще один пример:

**define(write\_buf,if ($3 > 0)**

**write($1,(1x,<$3>a1)) ($2(ii),ii=1,$3))**

определяет макрос, который при ссылке:

**write\_buf(5, buf, n)**

расширяется до:

**if (n > 0)**

**write(5, (1x,<n>a1)) (buf(ii), ii=1,n)**

"write\_buf" имеет три аргумента; первый аргумент - это номер блока, второй аргумент - это записываемый буфер, а третий аргумент - количество записываемых элементов в буфере. При ссылке на макрос аргументы разделяются запятыми.

Следующий макрос выводит запрос на терминал, а затем осуществляет чтение:

**define(read\_prompt,{ write(5,($1)); read(5,($2)) $3 })**

Когда на read\_prompt ссылается:

**read\_prompt('$Enter X, Y, and Z: ', 3F10.5, (x, y, z))**

он расширяется до:

**{ write(5,('$Enter X, Y, and Z: ')); read(5,(3F10.5)) (x, y, z) }**

Скобки вокруг "x, y, z" необходимы для группировки "x, y, z" в качестве аргумента 3. В противном случае он будет передан как аргументы 3, 4 и 5, поскольку запятые будут действовать как разделители аргументов. Фигурные скобки ("{" и "}") в определении не нужны в большинстве случаев, но позволяют использовать read\_prompt после структурированных операторов, как если бы это был один оператор. Например:

**if (unit == 5)**

**read\_prompt('$Output file? ', 80a1, buf)**

**Предопределенные макросы**

В дополнение к "define" существует ряд других встроенных макросов:

|  |  |
| --- | --- |
| **\_macro(x,y)** | эквивалентно "define(x,y)". |
| **\_undef(x)** | отменяет самое последнее определение "x". Если бы «x» был определен дважды, "\_undef(x)" вернул бы "x" обратно в его первое определение. |
| **\_repdef(x,y)** | заменяет текущее определение "x" (если есть) на "y". Эквивалент последовательности "\_undef(x) \_macro(x,y)". |
| **\_incr(x)** | преобразует "x" в число и добавляет к нему единицу. |
| **\_arith(x1,op1,x2[,op2,x3,...])** | выполняет целочисленную арифметику (-, +, /, \*), заданную "op1", "op2" и т. д. над "x1", "x2" и т. д. Оценка выполняется слева направо. В "\_arith" можно передать до 9 аргументов (5 операндов и 4 оператора). |
| **\_len(x)** | возвращает длину строки "x". В строке могут быть запятые. |
| **\_substr(s,m,n)** | возвращает подстроку "s", которая начинается в позиции "m" и имеет длину "n". Если "n" не указано или слишком велико, "\_substr" возвращает оставшуюся часть строки, начиная с "m". Если "m" не указано, возвращается вся строка "s". |
| **\_index(s,c)** | возвращает индекс в строке «s» символа "c". Если \_len(c) больше 1, возвращается индекс первого символа в "c". Если "c" не встречается в "s", возвращается 0. |
| **\_ifelse(a,b,c,d)** | расширяет "a" и "b"; если "a" равно "b" в виде символьной строки, "c" раскрывается и возвращается, иначе "d" раскрывается и возвращается. Обратите внимание, что "d" не раскрывается, если "a" равно "b", и "c" не раскрывается, если "a" не равно "b". "\_ifelse" может использоваться для проверки необязательного аргумента макроса; "\_ifelse($2,,A,B)" проверяет, является ли аргумент 2 нулевым; если да, то он расширяется до "А", иначе до "B". |
| **\_include(file)** | эквивалентно описанной выше функции include. |

# Аргументы макросов

Есть два способа передать аргумент макросу, когда он раскрывается, с макрорасширением аргумента и без него. Например, аргументы встроенного макроса "define" не раскрываются до того, как будут переданы ему. Это позволяет вам переопределить макрос, который был ранее определен. Например, если "X" был определен как "Y" (посредством "define(X, Y)"), то результатом

**define(X,Z)**

было бы переопределение "X" на "Z". Если аргументы для "define" были раскрыты перед передачей ему, то результатом "define(X,Z)" было бы определение "Y" как "Z", поскольку расширение "X" равно "Y".

Когда вы пишете макрос, вы можете указать, раскрывается ли каждый аргумент перед его передачей вашему макросу. Аргументы, указанные как "$n", раскрываются перед передачей в макрос (они называются аргументами "eval"), в то время как аргументы, указанные как "%n", не раскрываются перед передачей в макрос (они называются аргументами "noeval"). Например,

**define(set,define(%1,$2))**

определяет макрос "set", который похож на встроенный макрос "define", за исключением того, что второй аргумент, $2, раскрывается перед тем, как стать определением первого аргумента, который **не** раскрывается. В большинстве случаев расширение второго аргумента во время определения не имеет значения. Однако макрос "set" позволяет вам определять макросы, которые могут использоваться в качестве "переменных" макросов, которые могут передавать информацию о состоянии между макросами. Пример связи между макросами через макропеременные приведен ниже в разделе ["Примеры макросов"](#_Примеры_макросов).

Для большинства целей предпочтительна форма определения аргументов макроса eval ($). Аргументы noeval особенно полезны в макросах, которые определяют или отменяют определение других макросов или проверяют, определен ли другой макрос.

Данный аргумент (например, аргумент 1) не может быть одновременно типом eval и noeval, поэтому "$1" и "%1" не могут одновременно присутствовать в определении макроса.

# Специальные аргументы "$&" и "%&"

Когда "$&" или "%&" появляются в определении макроса, они являются заполнителями для всех аргументов, которые передаются в макрос, включая запятые, разделяющие их. "$&" - это оценочная форма аргументов, а "%&" - это новая форма. "$&" и "%&" не могут одновременно появляться в одном определении макроса, а макросы с "$&" или "%&" в их определениях не могут также иметь числовые аргументы, такие как "$1", "%2" и т. д. .

Пример:

**define(macro,define(%&))**

определяет "macro" как эквивалент "define", поскольку "define" не расширяет свои аргументы и позволяет использовать запятые в определениях.

# Примеры макросов

Ниже приведен пример макросов, реализующих операторы "while" и "endwhile". Хотя это не очень полезно, учитывая, что у Ratfiv есть оператор "while", он иллюстрирует некоторые продвинутые методы написания макросов. Первый макрос - "push":

**define(push,define(\_stack,$1)\_stack)**

"push" помещает новое определение поверх старого определения макроса с именем "\_stack", затем возвращает новое значение "\_stack". "\_stack" используется здесь как макропеременная, т.е. его функция - сохранять информацию между вызовами макросов. Указание "$1" вместо "%1" гарантирует, что определение, помещаемое в "\_stack", будет полностью развернуто до того, как оно будет использовано push.

**define(pop,\_stack \_undef(\_stack))**

"pop" возвращает текущее значение "\_stack", а затем отменяет его обратно к предыдущему значению. Теперь мы можем определить "while" и "endwhile":

**define(while,**

**push(\_incr(\_stack)) if (.not.($1)) goto push(\_incr(\_stack)))**

**define(endwhile, goto \_arith(pop,-,1)**

**\_incr(pop) continue)**

С определением "while" и "endwhile" последовательность:

**while (x < y)**

**y = y/2**

**endwhile**

расширяется до:

**1 if (.not. (x < y)) goto 2**

**y = y/2**

**goto 1**

**2 continue**

Конечно, выход Ratfiv будет выглядеть по-другому. Приведенный выше вывод - это расширение макроса до того, как Ratfiv преобразует его в Fortran.

# Передача литеральных строк через RATFIV

Строку можно передать буквально через Ratfiv без расширения или интерпретации как ключевого слова Ratfiv, заключив ее в символы акцент (`). Таким образом, строка

**`define(X,Y)`**

передается на вывод как

**define(X,Y)**

и не расширяется. Акцентированные литеральные строки также могут появляться в макроопределениях, однако вхождения строк замены аргументов ($n или %n) в пределах акцентов все равно заменяются соответствующими аргументами.

Чтобы указать литеральный символ акцент, поместите два из них в ряд внутри заключающих символов акцент.

# Условное расширение кода

Код может быть условно обойден в зависимости от того, определен ли макрос. оператор

**\_ifdef(x)**

проверяет, определен ли x как макрос. Если это так, то обработка продолжается нормально до появления макроса \_elsedef или \_enddef. Если x не определен, то весь код, следующий за \_ifdef, полностью обходится до появления макроса \_elsedef или \_enddef.

Например:

**define(PDP11)**

**....**

**\_ifdef(PDP11)**

**define(INTEGER\_SIZE,2)**

**define(MAXINT,32767)**

**\_elsedef**

**define(INTEGER\_SIZE,4)**

**define(MAXINT,16%7FFFFFFF)**

**\_enddef**

Оператор:

**\_ifnotdef(x)**

и

**\_ifndef(x)**

они похожи на \_ifdef, за исключением того, что следующий код расширяется, если x не определен. "\_ifnotdef" - это предпочтительная форма.

Допускается вложение \_ifdefs, \_ifnotdefs, \_elsedefs и \_enddefs.

# Использование RATFIV

Программа Ratfiv ожидает один или несколько входных файлов в командной строке и один выходной файл, которому предшествует ">". Например:

**RAT FILE1.RAT FILE2.RAT >FILE.FTN**

Вышеупомянутая команда приведет к компиляции файлов FILE1.RAT и FILE2.RAT в FILE.FTN с помощью Ratfiv. Если компилятор Ratfiv обнаруживает ошибку в коде Ratfiv, сообщение об ошибке будет напечатано на терминал и в выходной файл Fortran в том месте, где произошла ошибка. После предварительной обработки Ratfiv должен быть вызван компилятор Fortran, производящий FILE.OBJ. Многие ошибки Fortran не могут быть обнаружены Ratfiv. Когда это происходит, проверьте файл списка Fortran на предмет сообщения об ошибке. Обычно не так сложно выяснить, где именно в источнике Ratfiv находится ошибка, если она была обнаружена в источнике Fortran.

Программа Ratfiv имеет несколько опций. Опции должны быть размещены только после команды RAT и не могут быть помещены в имена файлов. Все опции могут быть сокращены до одной или нескольких букв.

Опция **/SYMBOLS** заставляет Ratfiv искать в вашем каталоге файл с именем "SYMBOLS."; этот файл открывается и читается как файл префикса для других файлов в вашей командной строке. Если "SYMBOLS." отсутствует в вашем каталоге, Ratfiv ищет файл "SYMBOLS." в системном каталоге, который обычно имеет некоторые стандартные определения, такие как EOS, EOF и т. д.

Опция **/F77** заставляет Ratfiv выдавать выходные данные VAX Fortran 77 вместо "стандартного" Fortran. Вывод Fortran 77 совместим с Digital Equipment Corporation VAX Fortran 77. Это не ANSII Fortran 77, поскольку он использует операторы DO WHILE и END DO.

Опция **/HOLLERITH** заставляет Ratfiv выдавать строки Холлерита вместо строк в кавычках. Этот ключ включен в основном для совместимости с версиями Ratfor, которые выпускают строки Холлерита. До сих пор этот переключатель требовался только при компиляции подпрограмм, которые вызывают макропрограммы DEC VAX, которые ожидают в качестве аргументов строки Холлерита вместо дескрипторов символов. Поскольку Ratfiv обычно создает строки в кавычках, в программах Ratfiv могут использоваться операторы Fortran "open" и другие операторы, требующие строки в кавычках.

**Пример**

Ниже приведен пример программы Ratfiv, предназначенной для демонстрации некоторых часто используемых операторов Ratfiv. Программа читает список файлов, считая строки по ходу. Подпрограммы и функции, которые используются, но не определены здесь, доступны в библиотеке Ratfiv. Такие символы, как EOF, ERR, MAXLINE и READ, определяются в файле "SYMBOLS.", который входит в состав Ratfiv. Пример не будет компилироваться правильно, если файл "SYMBOLS." читается. (См. раздел ["Использование RATFIV"](#_Использование_RATFIV) выше.)

# main вызывает подпрограмму count

call initr4

call count

call endr4

end

# count – считает строки в файлах

subroutine count

include comblk # этот файл включает a common блок который

# содержит integer переменную "linect"

character file(FILENAMESIZE), line(MAXLINE)

integer i, f, getarg, open, getlin, itoc

# цикл по списку файлов

linect = 0

for (i = 1; getarg(i, file, FILENAMESIZE) != EOF; i = i+1) {

f = open (file, READ) # открытие файла

if (f == ERR) { # файл не может быть найден

call putlin(file, ERROUT)

call remark (": can't open file.")

next

}

else # чтение строк из файла

while (getlin(line, f) != EOF)

linect = linect + 1

call close(f) # закрытие файла

}

i = itoc(linect, line, MAXLINE) # преобразует linect в character

call putlin(line, STDOUT) # печатает значение linect

call putch(NEWLINE, STDOUT) # очистка вызодного буфера

return

end

# BOX

Команда BOX - это очень удобный способ понять структуру исходной программы.

**BOX EXAMPLE.RAT**

произведет:

# main вызывает подпрограмму count

call initr4

call count

call endr4

end

# count – считает строки в файлах

subroutine count

include comblk # этот файл включает a common блок который

# содержит integer переменную "linect"

character file(FILENAMESIZE), line(MAXLINE)

integer i, f, getarg, open, getlin, itoc

# цикл по списку файлов

linect = 0

for (i = 1; getarg(i, file, FILENAMESIZE) != EOF; i = i+1)

+------------------------------------------------------------+

| f = open (file, READ) # открытие файла |

| if (f == ERR) |

| +--------------------------------------------------------+ |

| | call putlin(file, ERROUT) | |

| | call remark (": can't open file.") | |

| | next | |

| +--------------------------------------------------------+ |

| else # чтение строк из файла |

| while (getlin(line, f) != EOF) |

| linect = linect + 1 |

| call close(f) # закрытие файла |

+------------------------------------------------------------+

i = itoc(linect, line, MAXLINE) # преобразует linect в character

call putlin(line, STDOUT) # печатает значение linect

call putch(NEWLINE, STDOUT) # очистка вызодного буфера

return

end

# Пример вывода FORTRAN

Ниже приведены четыре идентичные подпрограммы, первая-оригинальная, непонятная, вторая-структурированная программа Ratfiv, третья-вывод Fortran /77 (ключ Ratfiv), четвертая-стандартный Fortran Холлерита, пятая-блочная структура, которая может быть опубликована, а не первая, которая была опубликована.

C/////////////////////////////// S P A D VERSION DU 01.04.83

C$OPTIONS NOLIST

C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ FSPAD1 ++

C FORTRAN PRINCIPAL POUR APPEL DES ETAPES DE S.P.A.D. +

C SUBROUTINE SPAD1, LISTP. +

C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

SUBROUTINE SPAD1 ( Q , MOTS )

C==01.04.83

C \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

C PROGRAMME SELECTIONNANT LES ETAPES DE \* SPAD \* A EXECUTER \*

C DEFINITION DES NUMEROS DE FICHIERS EN DATA, \*

C ET DES UNITES DE LECTURE (LECA) ET D-IMPRESSION (IMP). \*

C CREATION DU FICHIER (LEC) DES PARAMETRES DE COMMANDE. \*

C L-ARGUMENT MEMOT SERT DE TEST POUR LA POURSUITE DU PROGRAMME \*

C \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

DIMENSION Q(MOTS),LETAP(25) ,KART(20)

COMMON / VAL / TEST

COMMON /ENSOR/ LEC, IMP

DATA NDICA/1/, NDONA/2/,

1 NDIC /8/, NDON /9/, NLEG/10/, NGUS/11/, NGRI/12/, NGRO/13/,

2 NSAV/14/, NBAND/15/,NBFOR/16/

DATA LETAP /4HDONN,4HDPLU,4HLILE,4HCOMP,4HCORB,

1 4HMULT,4HMULD,4HAPLU,4HGRAP,4HECLA,4HSEMI,4HTAMI,

2 4HGRAF,4HCLAI,4HMCRE,4HTABU,4HTRIH,4HAGRA,4HARCH,

3 4HECRI,4HCODA,4HRECI,4HTRAN,4HLIST,4HSTOP/

C........... LECA= LECTEUR DE CARTES, IMP= IMPRIMANTE

C LEC = FICHIER AUXILIAIRE DE COPIE DES PARAMETRES

LECA = 5

LEC = 19

IMP = 6

C............. NOMBRE D-ETAPES ACTIVES DANS SPAD (AJOUTER LISTP,STOP)

NETAP = 23 + 2

N1 = NETAP - 1

C........... CREATION DU FICHIER DES PARAMETRES SUR LEC

REWIND LEC

KLIST = 0

1111 READ (LECA,5000) (KART(J),J=1,20)

WRITE (LEC,5000) (KART(J),J=1,20)

IF (KART(1) .EQ. LETAP(N1)) KLIST = 1

IF (KART(1) .EQ. LETAP(NETAP)) GO TO 1112

GO TO 1111

1112 REWIND LEC

IF (KLIST .EQ. 1) CALL LISTP (BID)

C .......... APPEL DES ETAPES SUCCESSIVES

REWIND LEC

MEMOT = 0

1113 IF ( MEMOT .EQ. 0 ) GO TO 1114

WRITE (IMP,4000)

RETURN

1114 READ(LEC,1000) METAP , M1

KETAP = 0

DO 1115 I = 1,NETAP

IF(METAP .EQ. LETAP(I) ) KETAP = I

1115 CONTINUE

IF (KETAP .EQ. N1) GO TO 1114

IF (KETAP .EQ. 0) GO TO 100

IF (KETAP .EQ. NETAP) GO TO 200

C ......... CHOIX DE L ETAPE DEMANDEE

C

GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,

1 19,20,21,22,23,200) , KETAP

C

C.... LECTURE DES DONNEES ET CREATION DU FICHIER-ARCHIVE NDONA

1 CALL DONNE ( Q, MOTS, MEMOT, NDONA )

C

GO TO 1113

C.... CREATION DE NLEG (TABLEAU DE CORRESP.LEGERE)

2 CALL DPLUM ( Q, MOTS, MEMOT, NLEG, NBAND )

C

GO TO 1113

C.... LECTURE DU DICO-ARCHIVE NDICA, CREATION DE NDIC ET NDON

3 CALL LILEX ( Q, MOTS , MEMOT, NDICA , NDONA , NDIC , NDON ,

1 NBAND, NBFOR)

GO TO 1113

C.... COMPOSANTES PRINCIPALES. CREATION DE NGUS

4 CALL COMPL ( Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND , NSAV )

C

GO TO 1113

C.... CORRESPONDANCES SIMPLES. CREATION DE NGUS

5 CALL CORBI ( Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV )

C

GO TO 1113

C.... CORRESP.MULTIPLES (MEMOIRE CENTRALE). CREATION NGUS

6 CALL MULTC ( Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV )

C

GO TO 1113

C.... CORRESP.MULTIPLES (LECTURE DIRECTE). CREATION NGUS

7 CALL MULDI ( Q, MOTS , MEMOT, NDIC , NDON , NGUS , NBAND , NSAV )

C

GO TO 1113

C.... CORRESP.LEGERE SUR NLEG. CREATION DE NGUS

8 CALL APLUM ( Q, MOTS, MEMOT, NLEG, NGUS, NBAND, NSAV )

C

GO TO 1113

C.... APPELS DES GRAPHIQUES (VARIABLES ET INDIVIDUS)

9 CALL GRAPH ( Q, MOTS , MEMOT, NGUS , NBAND)

C

GO TO 1113

C.... CLASSIFICATION (CENTRES MOBILES) PUIS ARBRE HIERARCHIQUE

10 CALL ECLAT ( Q, MOTS, MEMOT, NDIC, NDON, NGRI, NBAND, NBFOR )

C

GO TO 1113

C.... CLASSIFICATION SUR FACTEURS. CREATION DE NGRI

11 CALL SEMIS ( Q, MOTS , MEMOT, NGUS , NGRI , NBFOR )

C

GO TO 1113

C.... COUPURE DE L-ARBRE ET DESCRIPTION DES CLASSES

12 CALL TAMIS ( Q, MOTS , MEMOT, NDIC , NDON , NGRI , NGRO ,

1 NLEG , NBFOR )

GO TO 1113

C.... GRAPHIQUES POUR LA CLASSIFICATION (CENTRES ET DENSITE)

13 CALL GRAFK ( Q, MOTS , MEMOT, NGUS , NGRO , NDON , NBAND , NBFOR)

C

GO TO 1113

C.... EDITION EN CLAIR DES FACTEURS (POUR COMPL ET CORBI)

14 CALL CLAIR ( Q, MOTS, MEMOT, NDIC, NGUS )

C

GO TO 1113

C.... REGRESSION, ANAVAR ET ANACOV

15 CALL MCREG ( Q, MOTS, MEMOT, NDIC, NDON )

C

GO TO 1113

C.... TABULATIONS

16 CALL TABUL ( Q, MOTS, MEMOT,

1 NDICA, NDONA, NDIC, NDON, NBAND, NBFOR )

GO TO 1113

C.... TRIS-A-PLAT, HISTOGRAMMES

17 CALL TRIHI ( Q, MOTS, MEMOT, NDIC, NDON )

GO TO 1113

C.... CLASSIFICATION SUR UN GRAPHE

18 CALL AGRAF ( Q, MOTS, MEMOT, NGUS,NGRO,NBAND,NSAV,NBFOR )

C

GO TO 1113

C.... ARCHIVAGE DE COORDONNEES ET/OU CLASSIFICATIONS

19 CALL ARCHI ( Q, MOTS, MEMOT,NDICA,NDONA,NGUS,NGRO,NBAND,NSAV )

C

GO TO 1113

C.... GESTION DE DICTIONNAIRE

20 CALL ECRIT ( Q, MOTS, MEMOT, NDICA )

C

GO TO 1113

C.... APUREMENT DES VALEURS HORS-PLAGE

21 CALL CODAJ ( Q, MOTS, MEMOT, NDICA,NDONA,NBFOR)

C

GO TO 1113

C .... CLASSIFICATION ASCENDENTE HIERARCHIQUE

22 CALL RECIP ( Q, MOTS, MEMOT, NGUS, NGRI, NBAND, NBFOR)

C

GO TO 1113

C..... CREATION DES FICHIERS NDIC ET NDON A PARTIR DE NLEG

23 CALL TRANS ( Q, MOTS, MEMOT, NLEG, NDIC, NDON )

C

GO TO 1113

100 WRITE (IMP,2000) METAP,M1

RETURN

200 WRITE (IMP,3000) METAP,M1

1000 FORMAT (A4,A1)

2000 FORMAT (1H ,//,35H ERREUR SUR LE NOM D ETAPE ,A4,A1,/)

3000 FORMAT (1H0,65X,3H\*\* ,A4,A1,3H \*\*/1H0,62X,

1 16HFIN DE L-ANALYSE /1H0,130(1H-) )

4000 FORMAT (1H1,131(1H-)//,30X,33HERREUR FATALE : DEFAUT DE MEMOIRE)

5000 FORMAT (20A4)

RETURN

END

#/////////////////////////////// s p a d version du 01.04.83

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ fspad1 ++

# fortran principal pour appel des etapes de s.p.a.d. +

# subroutine spad1, listp. +

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

subroutine spad1 ( q , mots )

#==13.02.84 Version ecrite en RATFOR (Rationnal Fortran

# de Software-Tools via Ratfiv de Institute of Cancer Research)

# par N.Brouard (Institut National d'Etudes Demographiques)

# \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

# programme selectionnant les etapes de \* spad \* a executer \*

# definition des numeros de fichiers en data, \*

# et des unites de lecture (leca) et d-impression (imp). \*

# creation du fichier (lec) des parametres de commande. \*

# l-argument memot sert de test pour la poursuite du programme \*

# \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

define (YES,1)

define (NO,0)

define (MAXA4READER,33)

define (ASSEZ\_MEMOIRE,0)

define (ETAPE\_INCONNUE,0)

dimension q(mots),letap(25) ,kart(MAXA4READER)

common / val / test

common /ensor/ lec, imp

data ndica/1/, ndona/2/,

ndic /8/, ndon /9/, nleg/10/, ngus/11/, ngri/12/, ngro/13/,

nsav/14/, nband/15/,nbfor/16/

data letap /4hdonn,4hdplu,4hlile,4hcomp,4hcorb,

4hmult,4hmuld,4haplu,4hgrap,4hecla,4hsemi,4htami,

4hgraf,4hclai,4hmcre,4htabu,4htrih,4hagra,4harch,

4hecri,4hcoda,4hreci,4htran,4hlist,4hstop/

leca = 5 #........... leca= lecteur de cartes

lec = 19 # lec = fichier auxiliaire de copie des parametres

imp = 6 # imp = imprimante

netap = 23 + 2 #..... nombre d-etapes actives dans spad (ajouter listp,stop)

n1 = netap - 1

rewind lec #........... creation du fichier des parametres sur lec

klist = NO

repeat

{

read (leca,500) (kart(j),j=1,MAXA4READER)

write (lec,500) (kart(j),j=1,MAXA4READER)

if (kart(1) == letap(n1)) klist = YES

}

until (kart(1) == letap(netap))

rewind lec

if (klist == YES)

{

call listp (bid)

rewind lec

}

memot = ASSEZ\_MEMOIRE

list = netap - 1

ifin = netap

# .......... appel des etapes successives

for( ; ; )

{

read(lec,100) metap , m1

ketap = 0

do i = 1,netap

{

if(metap .eq. letap(i) ) ketap = i

}

# Interruption de la boucle des etapes

if(ketap == ETAPE\_INCONNUE | memot != ASSEZ\_MEMOIRE | ketap == ifin) break

if(ketap == list ) next

switch ( ketap) # ......... choix de l etape demandee

{

#.... lecture des donnees et creation du fichier-archive ndon

case 1: call donne ( q, mots, memot, ndona )

#.... creation de nleg (tableau de corresp.legere)

case 2: call dplum ( q, mots, memot, nleg, nband )

#.... lecture du dico-archive ndica, creation de ndic et ndon

case 3: call lilex ( q, mots , memot, ndica , ndona , ndic , ndon ,

nband, nbfor )

#.... composantes principales. creation de ngus

case 4: call compl ( q, mots, memot, ndic, ndon, ngus, nband , nsav)

#.... correspondances simples. creation de ngus

case 5: call corbi ( q, mots, memot, ndic, ndon, ngus, nband, nsav )

#.... corresp.multiples (memoire centrale). creation ngus

case 6: call multc ( q, mots, memot, ndic, ndon, ngus, nband, nsav )

#.... corresp.multiples (lecture directe). creation ngus

case 7: call muldi ( q, mots , memot, ndic , ndon , ngus , nband , nsav)

#.... corresp.legere sur nleg. creation de ngus

case 8: call aplum ( q, mots, memot, nleg, ngus, nband, nsav )

#.... appels des graphiques (variables et individus)

case 9: call graph ( q, mots , memot, ngus , nband)

#.... classification (centres mobiles) puis arbre hierarchique

case 10: call eclat ( q, mots, memot, ndic, ndon, ngri, nband, nbfor )

#.... classification sur facteurs. creation de ngri

case 11: call semis ( q, mots , memot, ngus , ngri , nbfor )

#.... coupure de l-arbre et description des classes

case 12: call tamis ( q, mots , memot, ndic , ndon , ngri , ngro ,

nleg , nbfor )

#.... graphiques pour la classification (centres et densite)

case 13: call grafk ( q, mots , memot, ngus , ngro , ndon , nband ,nbfor)

#.... edition en clair des facteurs (pour compl et corbi)

case 14: call clair ( q, mots, memot, ndic, ngus )

#.... regression, anavar et anacov

case 15: call mcreg ( q, mots, memot, ndic, ndon )

#.... tabulations

case 16: call tabul ( q, mots, memot,

ndica, ndona, ndic, ndon, nband, nbfor )

#.... tris-a-plat, histogrammes

case 17: call trihi ( q, mots, memot, ndic, ndon )

#.... classification sur un graphe

case 18: call agraf ( q, mots, memot, ngus,ngro,nband,nsav,nbfor )

#.... archivage de coordonnees et/ou classifications

case 19: call archi ( q, mots, memot,ndica,ndona,ngus,ngro,nband,nsav)

#.... gestion de dictionnaire

case 20: call ecrit ( q, mots, memot, ndica )

#.... apurement des valeurs hors-plage

case 21: call codaj ( q, mots, memot, ndica,ndona,nbfor)

#.... classification ascendente hierarchique

case 22: call recip ( q, mots, memot, ngus, ngri, nband, nbfor)

#..... creation des fichiers ndic et ndon a partir de nleg

case 23: call trans ( q, mots, memot, nleg, ndic, ndon )

}

}

if( ketap == ifin ) write (imp,(1h0,65x,'\*\* ',a4,a1,'\*\* '/1h0,62x,

'fin de l-analyse '/1h0,130('-') ))

else if( memot != ASSEZ\_MEMOIRE)

write(imp,(1h1,131('-')//,30x,'erreur fatale : defaut de memoire')) metap, m1

else

write(imp,(1h ,//," erreur sur le nom d etape ",a4,a1,/) )metap, m1

100 format (a4,a1)

500 format (MAXA4READER a4)

return

end

C/////////////////////////////// s p a d version du 01.04.83

C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ fspad1 ++

C fortran principal pour appel des etapes de s.p.a.d. +

C subroutine spad1, listp. +

C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

SUBROUTINE SPAD1(Q, MOTS)

C==13.02.84 Version ecrite en RATFOR (Rationnal Fortran

C de Software-Tools via Ratfiv de Institute of Cancer Research)

C par N.Brouard (Institut National d'Etudes Demographiques)

C \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

C programme selectionnant les etapes de \* spad \* a executer \*

C definition des numeros de fichiers en data, \*

C et des unites de lecture (leca) et d-impression (imp). \*

C creation du fichier (lec) des parametres de commande. \*

C l-argument memot sert de test pour la poursuite du programme \*

C \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

DIMENSION Q(MOTS), LETAP(25), KART(33)

COMMON/VAL/TEST

COMMON/ENSOR/LEC, IMP

DATA NDICA/1/, NDONA/2/, NDIC/8/, NDON/9/, NLEG/10/, NGUS/11/, NGR

\*I/12/, NGRO/13/, NSAV/14/, NBAND/15/, NBFOR/16/

DATA LETAP/4HDONN, 4HDPLU, 4HLILE, 4HCOMP, 4HCORB, 4HMULT, 4HMULD,

\* 4HAPLU, 4HGRAP, 4HECLA, 4HSEMI, 4HTAMI, 4HGRAF, 4HCLAI, 4HMCRE, 4

\*HTABU, 4HTRIH, 4HAGRA, 4HARCH, 4HECRI, 4HCODA, 4HRECI, 4HTRAN, 4HL

\*IST, 4HSTOP/

C........... leca= lecteur de cartes

LECA = 5

C lec = fichier auxiliaire de copie des parametres

LEC = 19

C imp = imprimante

IMP = 6

C..... nombre d-etapes actives dans spad (ajouter listp,stop)

NETAP = 23 + 2

N1 = NETAP - 1

C........... creation du fichier des parametres sur lec

REWIND LEC

KLIST = 0

2000 CONTINUE

READ(LECA, 500) (KART(J), J = 1, 33)

WRITE(LEC, 500) (KART(J), J = 1, 33)

IF (.NOT.(KART(1) .EQ. LETAP(N1))) GOTO 2030

KLIST = 1

2030 CONTINUE

2010 IF (.NOT.(KART(1) .EQ. LETAP(NETAP))) GOTO 2000

REWIND LEC

IF (.NOT.(KLIST .EQ. 1)) GOTO 2050

CALL LISTP(BID)

REWIND LEC

2050 CONTINUE

MEMOT = 0

LIST = NETAP - 1

IFIN = NETAP

C .......... appel des etapes successives

2070 CONTINUE

READ(LEC, 100) METAP, M1

KETAP = 0

DO 2100 I = 1, NETAP

IF (.NOT.(METAP.EQ.LETAP(I))) GOTO 2120

KETAP = I

2120 CONTINUE

C Interruption de la boucle des etapes

2100 CONTINUE

IF (.NOT.(KETAP .EQ. 0 .OR. MEMOT .NE. 0 .OR. KETAP .EQ. IFIN))

\*GOTO 2140

GOTO 2090

2140 CONTINUE

IF (.NOT.(KETAP .EQ. LIST)) GOTO 2160

GOTO 2080

2160 CONTINUE

I2180 = (KETAP)

GOTO 2180

C ......... choix de l etape demandee

C.... lecture des donnees et creation du fichier-archive ndon

2200 CONTINUE

CALL DONNE(Q, MOTS, MEMOT, NDONA)

C.... creation de nleg (tableau de corresp.legere)

GOTO 2190

2210 CONTINUE

CALL DPLUM(Q, MOTS, MEMOT, NLEG, NBAND)

C.... lecture du dico-archive ndica, creation de ndic et ndon

GOTO 2190

2220 CONTINUE

CALL LILEX(Q, MOTS, MEMOT, NDICA, NDONA, NDIC, NDON, NBAND, NB

\*FOR)

C.... composantes principales. creation de ngus

GOTO 2190

2230 CONTINUE

CALL COMPL(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)

C.... correspondances simples. creation de ngus

GOTO 2190

2240 CONTINUE

CALL CORBI(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)

C.... corresp.multiples (memoire centrale). creation ngus

GOTO 2190

2250 CONTINUE

CALL MULTC(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)

C.... corresp.multiples (lecture directe). creation ngus

GOTO 2190

2260 CONTINUE

CALL MULDI(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)

C.... corresp.legere sur nleg. creation de ngus

GOTO 2190

2270 CONTINUE

CALL APLUM(Q, MOTS, MEMOT, NLEG, NGUS, NBAND, NSAV)

C.... appels des graphiques (variables et individus)

GOTO 2190

2280 CONTINUE

CALL GRAPH(Q, MOTS, MEMOT, NGUS, NBAND)

C.... classification (centres mobiles) puis arbre hierarchique

GOTO 2190

2290 CONTINUE

CALL ECLAT(Q, MOTS, MEMOT, NDIC, NDON, NGRI, NBAND, NBFOR)

C.... classification sur facteurs. creation de ngri

GOTO 2190

2300 CONTINUE

CALL SEMIS(Q, MOTS, MEMOT, NGUS, NGRI, NBFOR)

C.... coupure de l-arbre et description des classes

GOTO 2190

2310 CONTINUE

CALL TAMIS(Q, MOTS, MEMOT, NDIC, NDON, NGRI, NGRO, NLEG, NBFOR

\*)

C.... graphiques pour la classification (centres et densite)

GOTO 2190

2320 CONTINUE

CALL GRAFK(Q, MOTS, MEMOT, NGUS, NGRO, NDON, NBAND, NBFOR)

C.... edition en clair des facteurs (pour compl et corbi)

GOTO 2190

2330 CONTINUE

CALL CLAIR(Q, MOTS, MEMOT, NDIC, NGUS)

C.... regression, anavar et anacov

GOTO 2190

2340 CONTINUE

CALL MCREG(Q, MOTS, MEMOT, NDIC, NDON)

C.... tabulations

GOTO 2190

2350 CONTINUE

CALL TABUL(Q, MOTS, MEMOT, NDICA, NDONA, NDIC, NDON, NBAND, NB

\*FOR)

C.... tris-a-plat, histogrammes

GOTO 2190

2360 CONTINUE

CALL TRIHI(Q, MOTS, MEMOT, NDIC, NDON)

C.... classification sur un graphe

GOTO 2190

2370 CONTINUE

CALL AGRAF(Q, MOTS, MEMOT, NGUS, NGRO, NBAND, NSAV, NBFOR)

C.... archivage de coordonnees et/ou classifications

GOTO 2190

2380 CONTINUE

CALL ARCHI(Q, MOTS, MEMOT, NDICA, NDONA, NGUS, NGRO, NBAND, NS

\*AV)

C.... gestion de dictionnaire

GOTO 2190

2390 CONTINUE

CALL ECRIT(Q, MOTS, MEMOT, NDICA)

C.... apurement des valeurs hors-plage

GOTO 2190

2400 CONTINUE

CALL CODAJ(Q, MOTS, MEMOT, NDICA, NDONA, NBFOR)

C.... classification ascendente hierarchique

GOTO 2190

2410 CONTINUE

CALL RECIP(Q, MOTS, MEMOT, NGUS, NGRI, NBAND, NBFOR)

C..... creation des fichiers ndic et ndon a partir de nleg

GOTO 2190

2420 CONTINUE

CALL TRANS(Q, MOTS, MEMOT, NLEG, NDIC, NDON)

GOTO 2190

2180 CONTINUE

IF (I2180 .LT. 1 .OR. I2180 .GT. 23) GOTO 2190

GOTO (2200,2210,2220,2230,2240,2250,2260,2270,2280,2290,2300,231

\*0,2320,2330,2340,2350,2360,2370,2380,2390,2400,2410,2420), I2180

2190 CONTINUE

2080 GOTO 2070

2090 CONTINUE

IF (.NOT.(KETAP .EQ. IFIN)) GOTO 2430

WRITE(IMP, 2450)

2450 FORMAT(1H0,65X,3H\*\* ,A4,A1,3H\*\* /1H0,62X,17Hfin de l-analyse /

\*1H0,130(1H-))

GOTO 2440

2430 CONTINUE

IF (.NOT.(MEMOT .NE. 0)) GOTO 2460

WRITE(IMP, 2480) METAP, M1

2480 FORMAT(1H1,131(1H-)//,30X,33Herreur fatale : defaut de memoire

\*)

GOTO 2440

2460 CONTINUE

WRITE(IMP, 2490) METAP, M1

2490 FORMAT(1H,//,35H erreur sur le nom d etape ,A4,A1,/)

2440 CONTINUE

100 FORMAT(A4, A1)

500 FORMAT(33A4)

RETURN

END

C/////////////////////////////// s p a d version du 01.04.83

C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ fspad1 ++

C fortran principal pour appel des etapes de s.p.a.d. +

C subroutine spad1, listp. +

C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

SUBROUTINE SPAD1(Q, MOTS)

C==13.02.84 Version ecrite en RATFOR (Rationnal Fortran

C de Software-Tools via Ratfiv de Institute of Cancer Research)

C par N.Brouard (Institut National d'Etudes Demographiques)

C \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

C programme selectionnant les etapes de \* spad \* a executer \*

C definition des numeros de fichiers en data, \*

C et des unites de lecture (leca) et d-impression (imp). \*

C creation du fichier (lec) des parametres de commande. \*

C l-argument memot sert de test pour la poursuite du programme \*

C \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

DIMENSION Q(MOTS), LETAP(25), KART(33)

COMMON/VAL/TEST

COMMON/ENSOR/LEC, IMP

DATA NDICA/1/, NDONA/2/, NDIC/8/, NDON/9/, NLEG/10/, NGUS/11/, NGR

\*I/12/, NGRO/13/, NSAV/14/, NBAND/15/, NBFOR/16/

DATA LETAP/4HDONN, 4HDPLU, 4HLILE, 4HCOMP, 4HCORB, 4HMULT, 4HMULD,

\* 4HAPLU, 4HGRAP, 4HECLA, 4HSEMI, 4HTAMI, 4HGRAF, 4HCLAI, 4HMCRE, 4

\*HTABU, 4HTRIH, 4HAGRA, 4HARCH, 4HECRI, 4HCODA, 4HRECI, 4HTRAN, 4HL

\*IST, 4HSTOP/

C........... leca= lecteur de cartes

LECA = 5

C lec = fichier auxiliaire de copie des parametres

LEC = 19

C imp = imprimante

IMP = 6

C..... nombre d-etapes actives dans spad (ajouter listp,stop)

NETAP = 23 + 2

N1 = NETAP - 1

C........... creation du fichier des parametres sur lec

REWIND LEC

KLIST = 0

2000 CONTINUE

READ(LECA, 500) (KART(J), J = 1, 33)

WRITE(LEC, 500) (KART(J), J = 1, 33)

IF (KART(1) .EQ. LETAP(N1)) THEN

KLIST = 1

END IF

2010 IF (.NOT.(KART(1) .EQ. LETAP(NETAP))) GOTO 2000

REWIND LEC

IF (KLIST .EQ. 1) THEN

CALL LISTP(BID)

REWIND LEC

END IF

MEMOT = 0

LIST = NETAP - 1

IFIN = NETAP

C .......... appel des etapes successives

DO WHILE (.TRUE.)

READ(LEC, 100) METAP, M1

KETAP = 0

DO 2060 I = 1, NETAP

IF (METAP.EQ.LETAP(I)) THEN

KETAP = I

END IF

C Interruption de la boucle des etapes

2060 CONTINUE

IF (KETAP .EQ. 0 .OR. MEMOT .NE. 0 .OR. KETAP .EQ. IFIN) THEN

GOTO 2050

END IF

IF (KETAP .EQ. LIST) THEN

GOTO 2040

END IF

I2080 = (KETAP)

GOTO 2080

C ......... choix de l etape demandee

C.... lecture des donnees et creation du fichier-archive ndon

2100 CONTINUE

CALL DONNE(Q, MOTS, MEMOT, NDONA)

C.... creation de nleg (tableau de corresp.legere)

GOTO 2090

2110 CONTINUE

CALL DPLUM(Q, MOTS, MEMOT, NLEG, NBAND)

C.... lecture du dico-archive ndica, creation de ndic et ndon

GOTO 2090

2120 CONTINUE

CALL LILEX(Q, MOTS, MEMOT, NDICA, NDONA, NDIC, NDON, NBAND, NB

\*FOR)

C.... composantes principales. creation de ngus

GOTO 2090

2130 CONTINUE

CALL COMPL(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)

C.... correspondances simples. creation de ngus

GOTO 2090

2140 CONTINUE

CALL CORBI(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)

C.... corresp.multiples (memoire centrale). creation ngus

GOTO 2090

2150 CONTINUE

CALL MULTC(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)

C.... corresp.multiples (lecture directe). creation ngus

GOTO 2090

2160 CONTINUE

CALL MULDI(Q, MOTS, MEMOT, NDIC, NDON, NGUS, NBAND, NSAV)

C.... corresp.legere sur nleg. creation de ngus

GOTO 2090

2170 CONTINUE

CALL APLUM(Q, MOTS, MEMOT, NLEG, NGUS, NBAND, NSAV)

C.... appels des graphiques (variables et individus)

GOTO 2090

2180 CONTINUE

CALL GRAPH(Q, MOTS, MEMOT, NGUS, NBAND)

C.... classification (centres mobiles) puis arbre hierarchique

GOTO 2090

2190 CONTINUE

CALL ECLAT(Q, MOTS, MEMOT, NDIC, NDON, NGRI, NBAND, NBFOR)

C.... classification sur facteurs. creation de ngri

GOTO 2090

2200 CONTINUE

CALL SEMIS(Q, MOTS, MEMOT, NGUS, NGRI, NBFOR)

C.... coupure de l-arbre et description des classes

GOTO 2090

2210 CONTINUE

CALL TAMIS(Q, MOTS, MEMOT, NDIC, NDON, NGRI, NGRO, NLEG, NBFOR

\*)

C.... graphiques pour la classification (centres et densite)

GOTO 2090

2220 CONTINUE

CALL GRAFK(Q, MOTS, MEMOT, NGUS, NGRO, NDON, NBAND, NBFOR)

C.... edition en clair des facteurs (pour compl et corbi)

GOTO 2090

2230 CONTINUE

CALL CLAIR(Q, MOTS, MEMOT, NDIC, NGUS)

C.... regression, anavar et anacov

GOTO 2090

2240 CONTINUE

CALL MCREG(Q, MOTS, MEMOT, NDIC, NDON)

C.... tabulations

GOTO 2090

2250 CONTINUE

CALL TABUL(Q, MOTS, MEMOT, NDICA, NDONA, NDIC, NDON, NBAND, NB

\*FOR)

C.... tris-a-plat, histogrammes

GOTO 2090

2260 CONTINUE

CALL TRIHI(Q, MOTS, MEMOT, NDIC, NDON)

C.... classification sur un graphe

GOTO 2090

2270 CONTINUE

CALL AGRAF(Q, MOTS, MEMOT, NGUS, NGRO, NBAND, NSAV, NBFOR)

C.... archivage de coordonnees et/ou classifications

GOTO 2090

2280 CONTINUE

CALL ARCHI(Q, MOTS, MEMOT, NDICA, NDONA, NGUS, NGRO, NBAND, NS

\*AV)

C.... gestion de dictionnaire

GOTO 2090

2290 CONTINUE

CALL ECRIT(Q, MOTS, MEMOT, NDICA)

C.... apurement des valeurs hors-plage

GOTO 2090

2300 CONTINUE

CALL CODAJ(Q, MOTS, MEMOT, NDICA, NDONA, NBFOR)

C.... classification ascendente hierarchique

GOTO 2090

2310 CONTINUE

CALL RECIP(Q, MOTS, MEMOT, NGUS, NGRI, NBAND, NBFOR)

C..... creation des fichiers ndic et ndon a partir de nleg

GOTO 2090

2320 CONTINUE

CALL TRANS(Q, MOTS, MEMOT, NLEG, NDIC, NDON)

GOTO 2090

2080 CONTINUE

GOTO (2100,2110,2120,2130,2140,2150,2160,2170,2180,2190,2200,221

\*0,2220,2230,2240,2250,2260,2270,2280,2290,2300,2310,2320), I2080

2090 CONTINUE

2040 CONTINUE

END DO

2050 CONTINUE

IF (KETAP .EQ. IFIN) THEN

WRITE(IMP, 2330)

2330 FORMAT(1H0,65X,'\*\* ',A4,A1,'\*\* '/1H0,62X,'fin de l-analyse '/1

\*H0,130('-'))

ELSE IF (MEMOT .NE. 0) THEN

WRITE(IMP, 2340) METAP, M1

2340 FORMAT(1H1,131('-')//,30X,'erreur fatale : defaut de memoire')

ELSE

WRITE(IMP, 2350) METAP, M1

2350 FORMAT(1H,//,' erreur sur le nom d etape ',A4,A1,/)

END IF

100 FORMAT(A4, A1)

500 FORMAT(33A4)

RETURN

END

#/////////////////////////////// s p a d version du 01.04.83

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ fspad1 ++

# fortran principal pour appel des etapes de s.p.a.d. +

# subroutine spad1, listp. +

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

subroutine spad1 ( q , mots )

#==13.02.84 Version ecrite en RATFOR (Rationnal Fortran

# de Software-Tools via Ratfiv de Institute of Cancer Research)

# par N.Brouard (Institut National d'Etudes Demographiques)

# \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

# programme selectionnant les etapes de \* spad \* a executer \*

# definition des numeros de fichiers en data, \*

# et des unites de lecture (leca) et d-impression (imp). \*

# creation du fichier (lec) des parametres de commande. \*

# l-argument memot sert de test pour la poursuite du programme \*

# \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

define (YES,1)

define (NO,0)

define (MAXA4READER,33)

define (ASSEZ\_MEMOIRE,0)

define (ETAPE\_INCONNUE,0)

dimension q(mots),letap(25) ,kart(MAXA4READER)

common / val / test

common /ensor/ lec, imp

data ndica/1/, ndona/2/,

ndic /8/, ndon /9/, nleg/10/, ngus/11/, ngri/12/, ngro/13/,

nsav/14/, nband/15/,nbfor/16/

data letap /4hdonn,4hdplu,4hlile,4hcomp,4hcorb,

4hmult,4hmuld,4haplu,4hgrap,4hecla,4hsemi,4htami,

4hgraf,4hclai,4hmcre,4htabu,4htrih,4hagra,4harch,

4hecri,4hcoda,4hreci,4htran,4hlist,4hstop/

leca = 5 #........... leca= lecteur de cartes

lec = 19 # lec = fichier auxiliaire de copie des parametres

imp = 6 # imp = imprimante

netap = 23 + 2 #..... nombre d-etapes actives dans spad (ajouter listp,stop)

n1 = netap - 1

rewind lec #........... creation du fichier des parametres sur lec

klist = NO

repeat

+-----------------------------------------------------------------------+

| read (leca,500) (kart(j),j=1,MAXA4READER) |

| write (lec,500) (kart(j),j=1,MAXA4READER) |

| if (kart(1) == letap(n1)) klist = YES |

| |

+-----------------------------------------------------------------------+

until (kart(1) == letap(netap))

rewind lec

if (klist == YES)

+--------------------------------------------------------------------+

| call listp (bid) |

| rewind lec |

+--------------------------------------------------------------------+

memot = ASSEZ\_MEMOIRE

list = netap - 1

ifin = netap

# .......... appel des etapes successives

for( ; ; )

+-------------------------------------------------------------------------------+

| read(lec,100) metap , m1 |

| ketap = 0 |

| do i = 1,netap |

| +---------------------------------------------------------------------------+ |

| | if(metap .eq. letap(i) ) ketap = i | |

| +---------------------------------------------------------------------------+ |

| |

| # Interruption de la boucle des etapes |

| if(ketap == ETAPE\_INCONNUE | memot != ASSEZ\_MEMOIRE | ketap == ifin) break |

| if(ketap == list ) next |

| |

| switch ( ketap) # ......... choix de l etape demandee |

| +---------------------------------------------------------------------------+ |

| | #.... lecture des donnees et creation du fichier-archive ndon | |

| | case 1: call donne ( q, mots, memot, ndona ) | |

| | #.... creation de nleg (tableau de corresp.legere) | |

| | case 2: call dplum ( q, mots, memot, nleg, nband ) | |

| | #.... lecture du dico-archive ndica, creation de ndic et ndon | |

| | case 3: call lilex ( q, mots , memot, ndica , ndona , ndic , ndon , | |

| | nband, nbfor ) | |

| | #.... composantes principales. creation de ngus | |

| | case 4: call compl ( q, mots, memot, ndic, ndon, ngus, nband , nsav) | |

| | #.... correspondances simples. creation de ngus | |

| | case 5: call corbi ( q, mots, memot, ndic, ndon, ngus, nband, nsav ) | |

| | #.... corresp.multiples (memoire centrale). creation ngus | |

| | case 6: call multc ( q, mots, memot, ndic, ndon, ngus, nband, nsav ) | |

| | #.... corresp.multiples (lecture directe). creation ngus | |

| | case 7: call muldi ( q, mots , memot, ndic , ndon , ngus , nband , nsav) | |

| | #.... corresp.legere sur nleg. creation de ngus | |

| | case 8: call aplum ( q, mots, memot, nleg, ngus, nband, nsav ) | |

| | #.... appels des graphiques (variables et individus) | |

| | case 9: call graph ( q, mots , memot, ngus , nband) | |

| | #.... classification (centres mobiles) puis arbre hierarchique | |

| | case 10: call eclat ( q, mots, memot, ndic, ndon, ngri, nband, nbfor ) | |

| | #.... classification sur facteurs. creation de ngri | |

| | case 11: call semis ( q, mots , memot, ngus , ngri , nbfor ) | |

| | #.... coupure de l-arbre et description des classes | |

| | case 12: call tamis ( q, mots , memot, ndic , ndon , ngri , ngro , | |

| | nleg , nbfor ) | |

| | #.... graphiques pour la classification (centres et densite) | |

| | case 13: call grafk ( q, mots , memot, ngus , ngro , ndon , nband ,nbfor) | |

| | #.... edition en clair des facteurs (pour compl et corbi) | |

| | case 14: call clair ( q, mots, memot, ndic, ngus ) | |

| | #.... regression, anavar et anacov | |

| | case 15: call mcreg ( q, mots, memot, ndic, ndon ) | |

| | #.... tabulations | |

| | case 16: call tabul ( q, mots, memot, | |

| | ndica, ndona, ndic, ndon, nband, nbfor ) | |

| | #.... tris-a-plat, histogrammes | |

| | case 17: call trihi ( q, mots, memot, ndic, ndon ) | |

| | #.... classification sur un graphe | |

| | case 18: call agraf ( q, mots, memot, ngus,ngro,nband,nsav,nbfor ) | |

| | #.... archivage de coordonnees et/ou classifications | |

| | case 19: call archi ( q, mots, memot,ndica,ndona,ngus,ngro,nband,nsav) | |

| | #.... gestion de dictionnaire | |

| | case 20: call ecrit ( q, mots, memot, ndica ) | |

| | #.... apurement des valeurs hors-plage | |

| | case 21: call codaj ( q, mots, memot, ndica,ndona,nbfor) | |

| | #.... classification ascendente hierarchique | |

| | case 22: call recip ( q, mots, memot, ngus, ngri, nband, nbfor) | |

| | #..... creation des fichiers ndic et ndon a partir de nleg | |

| | case 23: call trans ( q, mots, memot, nleg, ndic, ndon ) | |

| +---------------------------------------------------------------------------+ |

+-------------------------------------------------------------------------------+

if( ketap == ifin ) write (imp,(1h0,65x,'\*\* ',a4,a1,'\*\* '/1h0,62x,

'fin de l-analyse '/1h0,130('-') ))

else if( memot != ASSEZ\_MEMOIRE)

write(imp,(1h1,131('-')//,30x,'erreur fatale : defaut de memoire')) metap, m1

else

write(imp,(1h ,//," erreur sur le nom d etape ",a4,a1,/) )metap, m1

100 format (a4,a1)

500 format (MAXA4READER a4)

return

end

# Диагностика

К сожалению, некоторые сообщения об ошибках здесь не обсуждаются.

**can't open symbols file**

Файл специальных символов, содержащий определения Ratfiv общего назначения, открыть не удалось. Возможно, у пользователя не было доступа к конкретной библиотеке, которую должен был прочитать препроцессор.

**can't open included file**

Файл для включения не найден.

**definition too long**

Количество символов в имени, которое будет определено, превышает размер внутреннего массива Ratfiv (текущий максимум - 500 символов на определение)

**for clause too long**

Предложение приращения в предложении for было слишком длинным. Это фатальная ошибка.

**format too long**

Спецификация формата в операторе read, write, encode и decode была слишком длинной (текущий максимум - 600 символов)

**illegal break**

"Break" не происходит внутри допустимого цикла "for", "while", "do" или "repeat"

**illegal case or default**

оператор case или default обнаружен не внутри оператора switch

**illegal else**

Предложение else, вероятно, не следовало за предложением "if"

**illegal next**

"Next" не встречается внутри допустимого цикла "for", "while", "do" или "repeat"

**illegal right brace**

Обнаружена правая скобка без соответствующей левой скобки

**includes nested too deeply**

В настоящее время include могут быть вложены только в 4 файла глубиной, считая текущий входной файл

**invalid for clause**

Предложение "for" не содержало допустимого раздела инициализации, условия и/или приращения.

**missing left paren**

Скобка ожидалась, вероятно, в выражении "if", но не была найдена

**missing parenthesis in condition**

Правая скобка ожидалась, вероятно, в выражении "if", но не была найдена

**missing quote**

ожидаемая кавычка не найдена

**missing right paren**

Правая скобка ожидалась в операторе Fortran (в отличие от Ratfiv), но не была найдена

**illegal macro name**

Имена макросов не должны начинаться с цифры и должны содержать только буквенно-цифровые символы или символы подчеркивания ( \_ ) и доллара ( $ ).

**stack overflow in parser**

Операторы были вложены на слишком глубоком уровне. Текущий максимум - 100 вложенных операторов одновременно. Это фатальная ошибка.

**token too long**

Токен (слово) в исходном коде был слишком длинным, чтобы поместиться в один из внутренних массивов Ratfiv. (Текущий максимальный размер слова составляет 200 символов.)

**too many characters pushed back**

Исходный код незаконно указал команду Ratfiv или использовал ключевое слово или макрос незаконным образом, и лексический анализатор попытался, но не смог извлечь из этого смысла. Это фатальная ошибка.

**too many definitions**

Внутренние массивы Ratfiv не могли вместить все определения.

**unbalanced parentheses**

Несбалансированные круглые скобки, обнаруженные в операторе Fortran (в отличие от Ratfiv)

**unexpected EOF**

Конец файла был достигнут неожиданно. Часто это вызвано несопоставимыми фигурными скобками где-то глубоко в исходном коде.

**warning possible label conflict**

Это сообщение печатается, когда пользователь пометил оператор меткой в диапазоне 2000 и более, если метка делится на 10. Операторы Ratfiv назначаются в этом диапазоне, и определенный пользователем оператор может конфликтовать с сгенерированным Ratfiv.

**file: can't open**

Компилятор Ratfiv не смог открыть входной файл, указанный пользователем.

# Реализация

Компилятор Ratfiv генерирует код, считывая входные файлы и переводя любые ключевые слова Ratfiv в стандартный Fortran. Компилятор Ratfiv не знает никакого Fortran и, следовательно, не обрабатывает ошибки Fortran. Ошибки в синтаксисе ключевого слова Ratfiv отмечаются сообщением на терминал пользователя и в выходной файл Fortran вместе с указанием номера исходной строки, вызвавшей проблему.

Этот компилятор был первоначально написан Б. Керниганом {B. Kernighan} и П. Дж. Плаугером {P. J. Plauger}, с изменениями и улучшениями Дэвида Хэнсона {David Hanson} и его друзей (Университет Аризоны), Джо Свентекома {Joe Sventek} и Дебби Шеррера {Debbie Scherrer} (Лаборатория Лоуренса Беркли) и Уильяма Вуда {William Wood} (Институт исследований рака).

Для получения информации, комментариев или отчетов об ошибках обращайтесь:

William Wood

The Institute For Cancer Research

7701 Burholme Ave.

Philadelphia, Pa. 19111

(215) 728 2760

# Смотрите также

1. Kernighan, Brian W., "Ratfor--a Preprocessor for a Rational Fortran". Bell Laboratories publication. Также доступна в библиотеке компьютерных наук Калифорнийского университета в Беркли.
2. Kernighan, Brian W. and P. J. Plauger, "Software Tools". Addison-Wesley Publishing Company, Reading, Mass., 1976.
3. The rat4 user and design documents; the rc user document.
4. The Unix command "rc" in the Unix Manual (RC(1))

**Содержание**

IF-ELSE 2

WHILE 2

FOR 3

DO 3

REPEAT-UNTIL 4

SWITCH 4

BREAK и NEXT 5

Операторы FORMAT 6

STRING 6

INCLUDE 8

Объединение операторов и пустой оператор 9

Ввод в свободной форме 9

Комментарии 10

Перевод символов 10

Указание чисел в основаниях, отличных от основания десять 10

Строки символов в кавычках 11

Определение макросов 11

Макросы с аргументами 12

Аргументы макросов 14

Специальные аргументы "$&" и "%&" 14

Примеры макросов 15

Передача литеральных строк через RATFIV 15

Условное расширение кода 16

Использование RATFIV 16

BOX 18

Пример вывода FORTRAN 19

Диагностика 31

Реализация 33

Смотрите также 33