МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
Санкт-Петербургский политехнический университет Петра Великого

Веренинов И. А., Фёдоров С. А.

Программирование
Упражнения и лабораторные работы

Редакция от 05.03.2021

Санкт-Петербург

Содержание

Глава 1. Дисциплина программирования

§ 1.1 Организация проектов

- Каждое задание выполняется в виде отдельного проекта программного обеспечения;
- проект содержит директории:
  src — исходные файлы;
  bin — исполняемые файлы и выходные файлы;
  data — входные файлы
  obj — объектные файлы;
- сборочный файл (Makefile) размещается в корне директории проекта:
  - цель all содержит компиляцию каждого исходного файла по отдельности и их компоновку в исполняемый файл;
  - компилятор и его опции задаются через переменные;
  - обязательные опции при компиляции и компоновке:
    - `-Wall` — вывод всех прелупреждений;
    - `-std=f2008ts` — проверка на соответствие исходного кода стандарту ISO/IEC 1539-1:2010 Information technology — Programming languages — Fortran - Part 1: Base language (Fortran 2008) и техническому описанию ISO/IEC TS 29113:2012 Information technology — Further interoperability of Fortran with C;
    - `-Wno-maybe-uninitialized` — отключение предупреждений о *возможно* не инициализированных переменных;
    - `-fimplicit-none` — запрет неявного задания типов;
    - `-static-libgfortran` — статическая компоновка библиотеки Fortran для переносимости исполняемого кода;
    - `-flto` — оптимизация времени компоновки (link time optimization);
  - цель clean должна удалять все файлы в директориях bin и obj:

```
rm -f obj/*.*
rm -f bin/*.*
```

- проект не должен содержать *ошибок и предупреждений* на этапе сборки;
- каждый проект использует модуль Environment;
- при разработке использовать текстовый редактор Vim и компилятор gfortran.

§ 1.2 Организация исходного кода

Исходный код пишется кратко и лаконично для возможности его повторного использования как литературный текст «без запутанного сюжета и с наименьшим числом героев». При этом:

- головная программа содержит только вызовы процедур — подпрограмм и функций, — каждая решающая свою задачу (кроме первой части лабораторной работы № 1):

```
program reference_lab_1_5
   use Environment

   implicit none
   type (student), pointer  :: GroupList => Null()

   call Read_class_list(INPUT_FILE, GroupList)
   call Output_class_list(OUTPUT_FILE, GroupList)
   call Average_marks(GroupList)
   call Sort_class_list(GroupList, STUD_AMOUNT)
   call Output_sorted_class_list(OUTPUT_FILE, GroupList)
```

```
end program reference_lab_1_6
```

- головная программа и модули обязательно содержат оператор:

```
implicit none
```

- в исполняемых операторах используются **только** именованные константы, а не безымянные:

```
integer, parameter :: M = 24
allocate (A(M))
allocate (A(10))                    ! Не допустимо.
```

- описания всех процедур, а также содержательные участки года снабжаются комментариями;
- номера устройств файлов присваиваются **только** автоматически:

```
open (file=InputFile, encoding=E_, newunit=In)
```

- **везде**, где это возможно, используется средства регулярного программирования. Например удаление нулевых строк и столбцов из матрицы проводится за **3 оператора**:

```
NotNullRows(:,1) = Any(A /= 0, dim=2)  ! Позиции ненулевых строк (N, 1).
NotNullCols(1,:) = Any(A /= 0, dim=1)  ! Позиции ненулевых столбцов (1, N).
A = Reshape( Pack(A, MatMul(NotNullRows, NotNullCols)), &
             [Count(NotNullRows), Count(NotNullCols)])
```

- все данные задаются в отдельных входных файлах;
- формат ввода/вывода программируется, а сам ввод/вывод описывается с использованием регулярного программирования и списков:

```
stud_form = '(3(a, 1x), ' // MARKS_AMOUNT // 'i1)'
read (In, stud_form, iostat=IO) (Surnames(i), Initials(i), Sex(i), Marks(i, :),
i = 1, STUD_AMOUNT)
```

- процедуры:
  - процедуры ввода/вывода при чтении одинаковых структур данных используются повторно:

```
call Output_list(OutputFile, List, "Исходный список: ", "rewind")
call Output_list(OutputFile, SortedList, "Отсортированный список: ", "append")
```

здесь в процедуре Output_list используется последний входной параметр position:

```
open (file=InputFile, encoding=E_, newunit=In, position=position)
```

  - **все** процедуры кроме процедур ввода/вывода описываются чистыми (с квалификатором pure);
  - вид связи (intent) указывается для всех параметров;
  - процедуры используются с соблюдением требования наибольшей независимости по данным;
  - глобальные переменные внутри процедур не используются;
  - массивы и строки передаются как перенимающие форму:

```
subroutine ReadClassList(Input_File, Surnames, Initials)
   character(*)         Input_File
   character(kind=CH_)  Surnames(:, :), Initials(:, :)
   intent (in)          Input_File
   intent (out)         Surnames, Initials
```

§ 1.3 Основы профессионального стиля оформления кода

Стараться придерживаться стилю по именованию объектов:

- **только** имена констант пишутся строчными буквами с нижним подчёркиванием:

```
character(*) :: STUD_AMOUNT = 12;
```

- подряд идущие операторы объявления выравниваются по символу разделения типа и оператору присваивания:

```
character(SURNAME_LEN, kind=CH_)  :: Surnames(STUD_AMOUNT)             = CH__""
character(INITIALS_LEN, kind=CH_) :: Initials(STUD_AMOUNT)             = CH__""
character(kind=CH_)               :: Sex(STUD_AMOUNT)                  = CH__""
integer                           :: Marks(STUD_AMOUNT, MARKS_AMOUNT)  = 0
real(R_)                          :: AverMarks(STUD_AMOUNT)            = 0
```

- имена объектов должны быть лаконичным и отражать их назначение:

```
ClassList
```

- имена процедур пишутся с большой буквы с нижним подчёркиванием, а их параметры задаются через запятую:

```
subroutine Read_file(file_name, list)
```

- индексы и триплеты массивов задаются через запятую и пробел:

```
A(1:K, 1:L) = 0
```

- имена переменных и типов пишутся с большой буквы без нижнего подчёркивания:

```
type(Student), pointer :: ClassList
```

- все ключевые слова языка пишутся с маленькой буквы:

```
select case(IO)
```

Строго придерживаться стилю оформления кода:

- операторы описания отделяются от исполняемых операторов пустой строкой;
- описание входных параметров процедур отделяются от локальных переменных пустой строкой;
- операторы внутри каждой программной единицы и конструкции отделяются слева 3 пробелами:

```
if (Associated(InitialCurrent)) &
   allocate (DiffCode)
```

- после имени каждого оператора ставится пробел:

```
open (file=InputFile, encoding=E_, newunit=In)
```

- символы операций и присваивания отделяются пробелами, а имена дополнительных параметров в операторах и процедурах передаются без пробелов:

```
Aver_marks = Real(Sum(marks, dim=2), R_) / MARKS_AMOUNT
```

Глава 2. Эффективная работа разработчика в редакторе Vim

§ 2.1 Основные команды по работе с исходным кодом

:help windows — справка по эффективной работе с окнами

:help quickfix — справка по эффективной работе с исходным кодом

:make [arg] — вызов программы сборки

:copen — список быстрых исправлений

:cclose — закрытие списка быстрых исправлений

:clist — список ошибок

:cc [nr] — переход к месту текущей ошибки или к ошибке номер nr

:cn[ext] — переход к следующему месту ошибки

:cp[revious] — переход к предыдущему месту ошибки

§ 2.2 Порядок работы в Vim с иерархией файловой системы

Иерархию файлов вы можете просматривать, используя, например

```
open .
```

2.2.1 Установка NERDTree

Рекомендуется установить модуль NERDTree, позволяющий эффективнее работать с исходными файлами. Рекомендуется:

- установить этот модуль как пакет (Vim 8.+): https://github.com/scrooloose/nerdtree Например:

```
git clone https://github.com/preservim/nerdtree.git
~/.vim/pack/vendor/start/nerdtree
vim -u NONE -c "helptags ~/.vim/pack/vendor/start/nerdtree/doc" -c q
```

- запускать панель по горячей клавише (добавить в ~/.vimrc):

```
map <C-n> :NERDTreeToggle<CR>
```

- закрывать NERDTree, если открыт только он (добавить в ~/.vimrc):

```
autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTree") &&
b:NERDTree.isTabTree()) | q | endif
```

2.2.2 Порядок работы с NERDTree

Ctrl+n – вызов панели NERDTree

Ctrl+w+w – переключение между окном NERDTree и редактируемым файлом

Shitf+i – отображение скрытых файлов

m – меню NERDTree

m a – создание файла (при создании директории в конце указать «/»)

§ 2.3 Предлагаемый настойки для файла конфигурации vimrc

- Создайте файл ~/.vimrc **(начинается с «.»)** на основе /etc/vim/vimrc:

```
cp /etc/vim/vimrc ~/.vimrc
```

- Добавьте в него следующие конфигурации:

```
filetype plugin indent on
let fortran_do_enddo=1
set smartindent
set autoindent
set expandtab
set tabstop=3
set shiftwidth=3
set nu

"" Считать текущей директорией директорию открытого файла.
autocmd BufEnter * silent! lcd %:p:h/../

" Перейти к первой ошибке.
map <F4> :1000cp<Cr>zvzz:cc<Cr>
" Перейти к предыдущей ошибке.
map <F5> :cp<Cr>zvzz:cc<Cr>
" Перейти к следующей ошибке.
map <F6> :cn<Cr>zvzz:cc<Cr>
" Сохранить все файлы и собрать проект (цель all в Makefile).
map <F7> :wall \| make all<Cr>
" Запустить проект (цель run в Makefile).
map <F8> :make run<Cr>

" запускать панель NERDTree по горячей клавише
map <C-n> :NERDTreeToggle<CR>
" закрывать NERDTree, если открыт только он (добавить в ~/.vimrc):
autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTree") &&
b:NERDTree.isTabTree()) | q | endif
```

§ 2.4 Порядок работы в Vim с проектом

- Создайте дерево проекта. Например, директорию ex_1_4/ с поддиректориями:
  - ex_1_4/bin
  - ex_1_4/src
  - ex_1_4/bin
  - ex_1_4/data
  - ex_1_4/obj
- Перейдите в корневую директорию проекта ex_1_4/.
- Создайте Makefile. Не забывайте, что команды цели имеют в начале табуляцию.
- Создавайте или открывайте исходные файлы, находясь в любой директории. Например, vim src/ex_1_4.f90
- При открытии файла откройте окно сообщений через :copen.

- Переключение между окнами Ctrl+w+стрелка. См. Руководства по Vim.
- Используйте:
  - F4 для перехода к первой ошибке;
  - F5 для перехода к предыдущей ошибке,
  - F6 — к следующей;
  - F7 для сохранения и сборки проекта (достигается цель all),
  - F8 для запуска (цель run).

Глава 3. Упражнения

При выполнении упражнений, создавая своё произведение, необходимо следовать дисциплине программирования, описанной в главе 1. При сдаче принимается исходный код ПО и результаты его работы.

§ 3.1 Общие требования к упражнениям

1. Все исходные данные вводятся из входного форматированного файла.
2. Текст во входных и выходных файлах написан на русском языке (если присутствует).
3. Условно исходный код делится на *три* части:
    1. прочитанная информация выводится в выходной форматированный файл;
    2. проводится обработка данных с помощью *чистых* процедур;
    3. полученный результат выводится в выходной форматированный файл.
4. Все входные и выходные форматированные файлы имеют формат UTF-8.
5. Вещественные переменные и константы должны иметь явно заданную разновидность типа (R_).
6. Каждое упражнение выполняется в императивном стиле (необязательно) и регулярном стиле (обязательно) (императивный стиль оформляется в комментариях).
7. Данные в памяти размещать так, чтобы согласно выбранному алгоритму обход данных проводился в порядке их расположения в памяти. Например, если при работе с матрицей A размером *NxM* выбран алгоритм, оперирующий *в основном со строками*, то для представления матрицы в коде разумнее использовать двумерный массив A*(M, N)*. Тогда i-ая строка представляется в виде A(:, i) и в памяти является сплошной (contiguous).
8. Все процедуры *обработки данных* должны обладать следующими характеристиками:
    1. Гарантированно *на этапе компиляции* не имеют побочных эффектов (квалификатор pure в Fortran).
    2. Задействуют векторные инструкции современных процессоров (код должен быть векторизуем компилятором).
    3. Могут эффективно запускаться множество раз – при своей работе не выделяют памяти порядка N (O(N)), где N – одна из размерностей входных массивов-параметров (меньше N допустимо, например, для временных переменных и т. п.).

§ 3.2 Опорные варианты упражнений

Номера опорных упражнений: 1.пример, 1.3, 2.пример, 3.пример, 4.пример, 4.2a, 4.3a, 5.пример, 6.пример, 7.5a, 7.14a, 7.24, 7.30, 7.48, 8.пример1, 8.пример2, 8.22

§ 3.3 Указания и допуски к упражнениям

Не забывайте использовать встроенные математические функции, включая математические Abs, Exp, SqRt и т. д.

3.3.1 Указания по сортировке

Используйте функции Count, MinLoc, MaxLoc, CShift в зависимости от *метода сортировки*, заменяя этими функциями *внутренний цикл* сортировки.

3.3.2 Допуск к упражнениям § 1

1. Собрать и запустить опорные упражнения 1.пример и 1.3.
2. Разобраться в их исходном коде.
3. Ответить на вопросы преподавателя по исходному коду.

3.3.3 Указания к § I

1.1 Знаменатель вычислять один раз.

1.2 Коэффициенты записывать в массив A. Завести массив X со значениями x^n (неявный цикл). Произвести скалярное произведение X на массив коэффициентов A (Dot_product).

1.3 x = 0.3. Слагаемые записывать в массив из 4-ёх элементов. Очередное слагаемое вычислять относительно предыдущего. Разницу между членами нужно вычислить один раз. Частичные суммы не вычислять. Sum.

1.4 Слагаемые записывать в массив из 4-ёх элементов. Очередное слагаемое вычислять относительно предыдущего. Разницу между членами нужно вычислить один раз. Частичные суммы не вычислять. Sum.

1.5 Слагаемые записывать в массив из 4-ёх элементов. Очередное слагаемое вычислять относительно предыдущего. x^2 вычислять один раз. Частичные суммы не вычислять. Sum.

1.6 x^2 вычислять один раз.

1.7a Использовать либо сложениели бо деленеи,ие,ножение для смены значений.

1.8 Переменные записывать в массив. CShift.

3.3.4 Допуск к упражнениям § II

1. Собрать и запустить опорные упражнения 2.пример.
2. Разобраться в их исходном коде.
3. Ответить на вопросы преподавателя по исходному коду.

3.3.5 Указания к § II (не весь)

2.1 Нет указаний.

2.2 Использовать комплексный тип данных.

2.3 Слагаемые записывать в массив из 4-ёх элементов. Count.

2.4 Нет указаний.

2.5 Не использовать встроенные функции.

2.6 Max, Min.

2.7

2.8

2.9

2.10

2.11

2.12 Использовать конструкцию block.

2.13

2.14

2.15

2.16 Значения переменных занести в массив. MinLoc (с параметром dim).

2.17 Max, Min.

2.18 Использовать конструкцию select case.

2.19 $\frac{a+b}{2}$ вычислять один раз.

2.20

2.21

2.22

3.3.6 Допуск к упражнениям § III

1. Собрать и запустить опорные упражнения 3.пример.
2. Разобраться в их исходном коде.
3. Ответить на вопросы преподавателя по исходному коду.

3.3.7 Указания к § III (не весь)

3.1 Sum.

3.2 Product с конструктором массива и неявным циклом внутри последнего.

3.3 do (построить массив факториалов), forall (записать в него же каждый множитель), Product.

3.4 Sum (для сечения).

3.5 сИспользоватьсумму двух сечений.

3.6 Dot_product.

3.7 Norm2.

3.8 Сформировать массив из элементов диагонали, используя do concurrent. Sum.

3.9 Sum с параметром dim.

3.10 Sum с сечением.

3.12-13 Использовать арифметические выражения над массивами.

3.14 MatMul.

3.15 Сформировать массив из элементов побочной диагонали. Sum.

3.16 Суммировать по столбцам.

3.18 MatMul.

3.20 Reshape.

3.3.8 Допуск к упражнениям § IV

1. Собрать и запустить опорные упражнения 4.пример, 4.2a, 4.3a.
2. Разобраться в их исходном коде.
3. Ответить на вопросы преподавателя по исходному коду.

3.3.9 Указания к § IV

4.1 Сформировать вектор X(N), а потом F(N).

4.2 Сформировать вектора X(Nx*Ny) и Y(Nx*Ny) для хранения всех пар: (x_i, y_j), i = 1, …, Nx, j = 1, …, Ny. Вычислить вектор F(Nx*Ny) на этих значениях, используя встроенные элементные функции. См. образцовое упражнение 4.2a.

4.3-6 Сперва вычислить все необходимые точки интерполяции (для гарантированной векторизации). После провести вычисление функции в этих точках. Sum, использовать встроенные элементные функции.

4.5 Вычислять каждый раз дробную часть шага *неэффективно*. Сделайте сперва смещение на h/2, а потом каждый раз только на h.

3.3.10 Допуск к упражнениям § V

1. Собрать и запустить опорные упражнения 5.пример.
2. Разобраться в их исходном коде.
3. Ответить на вопросы преподавателя по исходному коду.

3.3.11 Указания к § V (не весь)

5.1 Pack, Mod, массив индексов.

5.2 Маска для условия, что элемент меньше предыдущего: A(2:N-1) < A(1:N-2). Применять Pack. Маску вычислять один раз. Сперва найти положение экстремумов, а потом сформировать массив их значений, используя *векторный индекс*.

5.3 Отдельно построить маску. Count, Sum.

5.4 FindLoc, Sum.

5.5 Sum, Count.

5.6 Any. Возвращать логическое, а не целое значение.

5.7 Pack, UBound.

5.10 Сечение из четных и нечетных элементов.

5.11-13 Только MinVal, MaxVal (когда не требуется позиция), MinLoc, MaxLoc, All (возможно с маской). Иногда в 5.12 с параметром back = .true..

5.14 Завести массив индексов. Построить массив-маску, удовлетворяющую заданному условию. Pack.

5.15 Постройте маску для удаляемых элементов (MaxVal). Подсчитайте их число M (Count). Используйте функцию Pack, чтобы присвоить к первой части массива уплотнённые элементы.

5.16 Count *(см. 7.25)*.

5.17 MaxLoc, MaxVal.

5.19 Сечения.

5.20 Sum, сечения.

5.21 Сперва найти только индексы трёх максимальных чисел, проводя вычисления в одном цикле. После запомнить их значения и обнулить их позиции в массиве.

5.22 Из-за различия в коэффициентах при функции вычисления лучше проводить с использованием двух массивов. Вычислить и поместить в два отдельных массива все необходимые точки интерполяции – значения x для функции при коэффициенте 2 и значения x для функции при коэффициенте 4. Затем провести суммирование всех значений функции в точках этих массивов с коэффициентом 4 и всех значений функции с коэффициентом 2. Sum.

5.23 Min.

5.24 Последовательность может встречаться, начиная с разных элементов. Вычислить в цикле, сколько раз последовательность встречается (All).

Заводить маску, указывающую с каких позиций появляется последовательность (do concurrent), а потом вычислят число истинных значений может оказаться *неэффективным*.

5.25 См. 5.2. Сформировать массив из индексов повторяющихся значений в массиве экстремумов (см. разбор 7.25).

3.3.12 Допуск к упражнениям § VI

1. Собрать и запустить опорные упражнения 6.пример; 6.1а, 6.1б.
2. Разобраться в их исходном коде.
3. Ответить на вопросы преподавателя по исходному коду.

3.3.13 Указания к § VI

6.1-2 Проводить вычисления, *пока сумма не перестанет меняться* (см. решение примера). Очередной член (или очередной числитель и знаменатель) вычислять относительно предыдущего. Если возможно, то разницу между членами вычислять *один раз до цикла* (например x^2). В некоторых случаях шаг цикла удобно делать равным 2 (например, если требуются только факториалы чётных чисел). Начальные значения переменным давать до цикла. Сравнить результат со встроенной функцией. Меняя разновидность вещественного типа на двойную и четверную точность, посмотреть, сколько членов ряда потребуется для сходимости.

6.1г Разница между членами:

$$q_{n+1}=\frac{s_{n+1}}{s_n}=\frac{(2(n+1))!\,x^{(2(n+1)+1)}}{4^{n+1}((n+1)!)^2(2(n+1)+1)}\,\frac{4^n(n!)^2(2n+1)}{(2n)!\,x^{(2n+1)}}=¿$$

$$¿\,\frac{(2n+2)!\,x^{(2n+3)}}{4^{n+1}((n+1)!)^2(2n+3)}\,\frac{4^n(n!)^2(2n+1)}{(2n)!\,x^{(2n+1)}}=x^2\frac{(2n+1)^2}{2(n+1)(2n+3)}=x^2\frac{(n+0.5)^2}{(n+1)(n+1.5)}$$

Члены нумеруются с нуля: **n = 0, 1, 2**, … Так, например, для **n = 2** множитель составляет (если использовать *предпоследнее* равенство последней формулы):

$q_3=q_{2+1}=x^2\frac{5^2}{6\cdot 7}$, что и видно из ряда.

6.2а Исправленный ряд для функции ошибки:

$$erf(x)=\frac{2x}{\sqrt{\pi}}\left(1-\frac{x^2}{1!3}+\frac{x^4}{2!5}-\frac{x^6}{3!7}+...\right)$$

6.2в В цикле проводить вычисления только суммы членов, заданных в скобках, затем умножить полученную сумму на 2 и прибавить ln*x*, используя встроенную функцию. В первом члене ряда опечатка:

$\frac{a}{2x+a}$. Очередной член ряда лучше не делить каждый раз на целое число в знаменателе (3, 5, и т. д.), чтобы потом не приходилось на него же умножать. Завести отдельные переменные для члена ряда с и без множителя в знаменателе. На каждой итерации цикла рекурентно вычислять очередной член ряда без множителя в знаменателе. Тогда множитель q будет постоянным.

6.3 Проводить вычисления, пока относительная ошибка вычислений не станет меньше epsilon() для используемой разновидности вещественных чисел (см. решение примера).

Пояснение. Для нахождения корня $y=\sqrt{a}$ предлагается найти корень функции $f(y)=y^2-a$. Для нахождения корня функции применяется метод Ньютона:

$$y_{n+1}=y_n-\frac{f(y_n)}{f'(y_n)}\quad .$$

За начальное приближение можно взять $y_0=a$.

6.4 Проводить вычисления, пока относительная ошибка вычислений не станет меньше epsilon() для используемой разновидности вещественных чисел (см. решение примера).

К вычислению числа Пи. Ряд Сриниваса Рамануджана.

Ряд:

$$\pi=\frac{9801}{2\sqrt{2}\sum_{k=0}^{\infty}\frac{(4k)!}{(k!)^4}\cdot\frac{1103+26390k}{(4\cdot 99)^{4k}}}=\frac{9801}{2\sqrt{2}}\cdot\frac{1}{\sum_{k=0}^{\infty}\frac{(4k)!}{(k!)^4(4\cdot 99)^{4k}}\cdot(1103+26390k)}=¿$$

$$¿=\frac{9801}{2\sqrt{2}}\cdot\frac{1}{\sum_{k=0}^{\infty}S_k\cdot P_k}$$

Множители члена ряда:

$$S_k=\frac{(4k)!}{(k!)^4(4\cdot 99)^{4k}}$$

$$q_{k+1}=\frac{S_{k+1}}{S_k}=\frac{(4(k+1))!}{((k+1)!)^4(4\cdot 99)^{4(k+1)}}\cdot\frac{(k!)^4(4\cdot 99)^{4k}}{(4k)!}=\frac{(4k+1)(4k+2)(4k+3)(4k+4)}{(k+1)^4(4\cdot 99)^4}=¿$$

$$¿=\frac{1}{99^4}\cdot\frac{(k+0.25)(k+0.5)(k+0.75)}{(k+1)^3}\quad ;$$

$$S_0=1\quad ;\quad S_{k+1}=q_{k+1}\cdot S_k$$

$$P_k=1103+26390k$$

$$P_0=1103\quad ;\quad P_{k+1}=P_k+26390$$

К вычислению числа Пи. Ряд Чудновских.

Ряд:

$$\pi=\frac{426880\sqrt{10005}}{\sum_{k=0}^{\infty}\frac{(6k)!}{(3k)!(k!)^3}\cdot\frac{13591409+545140134k}{(-640320)^{3k}}}=¿$$

$$¿=\frac{426880\sqrt{10005}}{\sum_{k=0}^{\infty}\frac{(6k)!}{(3k)!(k!)^3(-2^6\cdot 3\cdot 5\cdot 23\cdot 29)^{3k}}\cdot(13591409+545140134k)}=\frac{426880\sqrt{10005}}{\sum_{k=0}^{\infty}S_k\cdot P_k}$$

Множители члена ряда:

$$S_k=\frac{(6k)!}{(3k)!(k!)^3(-2^6\cdot 3\cdot 5\cdot 23\cdot 29)^{3k}}$$

$$q_{k+1}=\frac{S_{k+1}}{S_k}=\frac{(6k+6)!}{(3k+3)!((k+1)!)^3(-2^6\cdot 3\cdot 5\cdot 23\cdot 29)^{(3k+3)}}\cdot\frac{(3k)!(k!)^3(-2^6\cdot 3\cdot 5\cdot 23\cdot 29)^{3k}}{(6k)!}=¿$$

$$¿=\frac{(6k+1)(6k+2)(6k+3)(6k+4)(6k+5)(6k+6)}{(3k+1)(3k+2)(3k+3)(k+1)^3(-2^6\cdot 3\cdot 5\cdot 23\cdot 29)^3}=¿$$

$$¿=\frac{6^6(k+1/6)(k+1/3)(k+0.5)(k+2/3)(k+5/6)(k+1)}{(k+1/3)(k+2/3)(k+1)^4 3^3 2^3 6^3(-2^4\cdot 5\cdot 23\cdot 29)^3}=\frac{(k+1/6)(k+0.5)(k+5/6)}{(k+1)^3(-2^4\cdot 5\cdot 23\cdot 29)^3}$$

$$¿=\frac{1}{(-2^4\cdot 5\cdot 23\cdot 29)^3}\cdot\frac{(k+1/6)(k+0.5)(k+5/6)}{(k+1)^3}\quad ;$$

$$S_0=1\quad ;\quad S_{k+1}=q_{k+1}\cdot S_k$$

$$P_k=13591409+545140134k$$

$$P_0=13591409\quad ;\quad P_{k+1}=P_k+545140134$$

3.3.14 Допуск к упражнениям § VII

1. Собрать и запустить опорные упражнения 7.5a, 7.14a, 7.21, 7.25, 7.30, 7.48.
2. Разобраться в их исходном коде.
3. Ответить на вопросы преподавателя по исходному коду.

3.3.15 Указания к § VII (не весь)

7.1 См. выше указания к сортировке.

7.1в-г Вычислять модуль элементов каждый раз *неэффективно*.

Можно перед сортировкой сформировать: массив из модулей элементов AbsA(:) и массив *номеров элементов* в массиве Indexes(:). Сортируя первый из них, отсортировать и второй. После сортировки присвоить к исходному массиву тот же массив с отсортированным порядком элементов, используя массив индексов как вектор индексов: AbsA = A(Indexes()), A = AbsA.

Такой подход требует использования *нерегулярного* обращения к массиву – A(Indexes()). Для того, чтобы этого избежать, перед сортировкой лучше сформировать *только* массив из модулей элементов AbsA(:). Сортируя первый из них, сортировать и основной.

7.2 Вычисления проводить только в текущем массиве. Построить маску. Использовать Pack внутри конструктора массива A = [Pack(, Mask), Pack(, .not. Mask)]. Count. См. выше указания к сортировке.

7.2а Построить маску для *положительных* элементов. Сформировать массив A = [Pack(, .not. Mask), Pack(, Mask)]. Отсортировать первые отрицательные элементы. Count.

7.2б Построить маску для *положительных* элементов. Сформировать массив A = [Pack(, .not. Mask), Pack(, Mask)]. Отсортировать последние положительные элементы. Count.

7.3 Сформировать вектор C из двух данных, используя конструктор массива. См. указания к сортировке.

7.4 Записать на 5-ый элемент, Count (к обеим частям в отдельности, определив, в какое место вставлять), CShift (циклически сдвинуть соответствующую часть).

7.5б При сортировке использовать шаг равный 2. Можно предварительно занести сортируемые элементы во временный массив длиной N/2 (использовать сечения).

7.9а Norm2.

7.9бв MaxVal, Sum с параметром dim.

7.5а При упаковке и распаковке всё равно потребуется хотя бы N/2 дополнительного места. Лучше завести временный массив с нужными элементами и отсортировать его. Pack, Unpack. См. указания к сортировке.

7.5б Заводить временный массив с нужными элементами для сортировки будет *напрасно.* См. указания к сортировке. При сортировке использовать шаг 2.

7.6 Вычисления провести, используя два подхода: 1) Sum, Product, SqRt; 2) Norm2, Product, SqRt. Сравнить результат двух подходов. Использовать параметр dim в Sum и Norm2.

7.7 Sum с dim, Product с dim.

7.9а Norm2.

7.9б-в MaxVal, Sum с параметром dim, Abs.

7.10 Использование маски, исключающей всего лишь главную диагональ, напрасно при таком *регулярном обращении* к данным. Формировать две новые, но по-разному упакованные матрицы также не оправданно (для p – со сдвигом элементов влево, для q – вправо), т. к. придётся вычислять модуль каждого элемента матрицы 2 раза.

Вместо этого лучше сформировать новую матрицу из модулей элементов данной матри-

цы, а также массив из модулей диагональных элементов сформированной матрицы. Построить вектора P и Q (Sum с параметром dim), используя *всю* матрицу *вместе* с диагональными элементами (при таком регулярном обращении будет возможна векторизация). Вычесть из полученных массивов P и Q сформированный прежде массив из модулей диагональных элементов. Составить логический вектор-условие и проверить условие неособенности матрицы (All).

7.11 Dot_product.

7.12 MinLoc, Sum с параметром dim, All.

7.13 Sum, MinLoc, MaxVal, MinVal. Если таких элементов несколько, то привести ближайшие к началу индексы.

7.14 Использовать Sum для конструктора массива, содержащего суммы элементов: S = Sum((сумма i-ого столбца, I = 1, N). Суммирование проводить по столбцам.

7.15 MinVal, MaxVal, MinLoc, MaxLoc, Sum (с параметром dim).

7.16 MinVal, MaxVal, MinLoc, MaxLoc, Sum.

7.17 Найти нужный элемент. Составить маску. Упаковать массив индексов. MaxVal, MinVal, Reshape, Pack, Spread (см. опорные 7.x).

7.18 Работать непосредственно с диагоналями неэффективно. Тем более неэффективно каждый раз вычислять модули этих элементов. Более того, если число N нечётное, то на побочной диагонали не следует повторно учитывать срединный элемент с индексами (N/2+1, N/2+1).

Вместо этого лучше поместить диагональные элементы в один объединённый массив размером 2N (сразу вычислять модули элементов, если требуется). Найти нужный элемент в объединённом массиве (MaxVal, MinVal.). Составить маску для объединённого массива, указывающую расположение в нём нужных элементов. Если N нечётное, то для того, чтобы заведомо исключить срединный элемент на побочной диагонали, записать в соответствующий ему элемент маски значение .false..

Далее можно было бы составить массив индексов Indexes == [1, …, 2N], упаковать его по маске и продублировать, создав массив индексов для координат строк и столбцов нужных элементов, например: номера индексов строк могли бы быть Ind[:, 1] == [3, 7, N+5, N+9], номера индексов столбцов – Ind[:, 2] == [3, 7, N+5, N+9]. После необходимо будет пересчитать индексы, большие N: Ind[:, 1] == [3, 7, 5, 9], Ind[:, 2] == [3, 7, N-5+1, N-9+1], для этого пришлось бы организовывать отдельный цикл с условием, который бы не векторизовался.

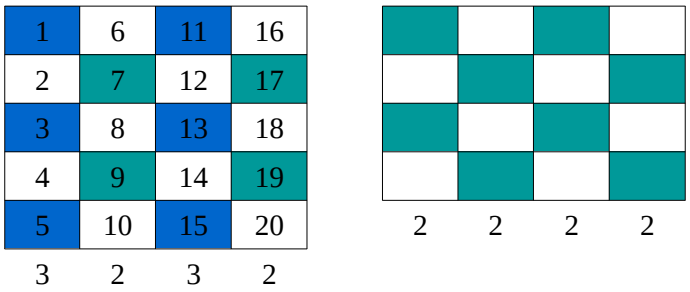
Вместо этого лучше вычислить: число K истинных значений в первой половине маски и и число M истинных значений во второй её половине. Разместить двумерный массив Ind(K+M, 2) необходимого размера для хранения индексов строк и столбцов нужных элементов. Создать с помощью присваивания в цикле (do concurrent) и сечения массив индексов строк для объединённого массива: Indexes == [1, …, N, 1, …, N]. Упаковать его по маске в первый столбец результирующего двумерного массива индексов Ind(:, 1). Присвоить первый столбец (индексы строк) второму столбцу Ind(:, 2) (индексы столбцов(, например: номера индексов строк могли бы быть Ind[:, 1] == [3, 7, 5, 9], номера индексов столбцов – Ind[:, 2] == [3, 7, 5, 9]. Во втором столбце массива индексов пересчитать в векторизуемом цикле только значения столбцов элементов *побочной* диагонали: Ind[**K+1:K+M**, 2] == [**6, 2**] (пример для N==10). В результате массив индексов станет равным Ind[:, 1] == [3, 7, 5, 9], Ind[:, 2] == [3, 7, 6, 2] и будет хранить индексы диагональных элементов (3, 3), (7, 7), (5, 6), (9, 2).

7.19 Count. Составить маску. Упаковать массив индексов (см. раздаточные 7.x).

7.20 Обход массива проводить по столбцам.

Упаковка двумерного массива (Pack) по маске при таком *регулярном* обращении к элементам *неэффективна*. Хотя эту маску и можно получить без проведения вычислений над индексами, т. к. маска — это шахматная доска: Mask(::2, ::2) = .true.; Mask(2::2, 2::2) = .true..

Лучше обойтись *присваиванием* сечений массива, не проводя вычислений и дорогостоящей упаковки по произвольной маске. Число нужных элементов в каждом нечётном столбце и в каждом чётном столбце заранее известно:

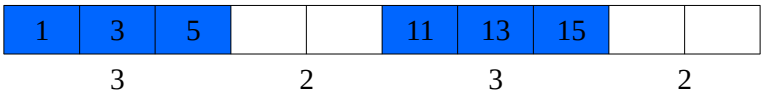


При чётном N число нужных элементов в чётных и нечётных столбцах различается на 1, при нечётном – одинаково. Во втором случае можно вообще присвоить в результирующий массив *каждый второй* элемент исходного массива.

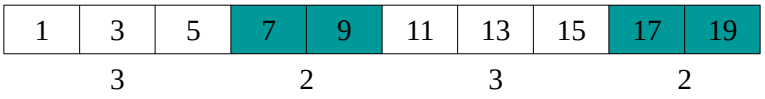
С помощью неявного цикла такое присваивание проводится одним оператором с использованием конструктора массива. На каждой итерации присваивается очередной *чётный* и *нечётный* столбцы:

B = [(A(?:?:?, i), A(?:?:?, i+1), i = 1, N-1, 2)]

Для проведения присваивания с помощью явных циклов необходимо прежде разместить выходной массив с нужным числом элементов. В первом цикле (do concurrent) проводить присваивание *сечения* каждого *нечётного* столбца на нужное место в результирующем массиве.



Во втором цикле (do concurrent) проводить присваивание *сечения* каждого *чётного* столбца на нужное место в результирующем массиве.



7.22 MaxVal.

7.23 Для определения нужных индексов необходимо сперва вычислить массив всех сумм S(N-1,N-1), в котором $s_{ij} = a_{ij}+a_{i(j+1)}+ a_{(i+1)j}+a_{(i+1)(j+1)}$. Сложность прямого алгоритма вычисления массива – число операций сложения – составляет 3x(N-1)x(N-1) ~ $3\mathbf{N}^2$+O(N). При суммировании сечения столбца A(i:i+1, j) с сечением столбца A(i:i+1, j+1) возможна векторизация: за *одну* операцию складываются эти сечения, а за вторую – элементы результата:

$$\begin{bmatrix} a_{ij} \\ a_{(i+1)j} \end{bmatrix} + \begin{bmatrix} a_{i(j+1)} \\ a_{(i+1)(j+1)} \end{bmatrix} = \begin{bmatrix} s'_i \\ ¿+¿ \\ s'_{i+1} \end{bmatrix} = s_{ij} \quad .$$

В итоге для получения элемента потребуется не 3, а 2 операции сложения, поэтому сложность прямого алгоритма составит 2x(N-1)x(N-1) ~ $2\mathbf{N}^2$+O(N).

Важно обратить внимание, что для получения s_{ij} суммируются элементы a_{ij}, $a_{i(j+1)}$, $a_{(i+1)j}$, $a_{(i+1)(j+1)}$, а для получения s_{ij+1} – элементы $a_{i(j+1)}$, $a_{i(j+2)}$, $a_{(i+1)(j+1)}$, $a_{(i+1)(j+2)}$, т. е. пара элементов $a_{i(j+1)}$ и $a_{(i+1)(j+1)}$ суммируются повторно:

a_{ij}	**$a_{i(j+1)}$**	$a_{i(j+2)}$	$a_{i(j+3)}$
$a_{(i+1)j}$	**$a_{(i+1)(j+1)}$**	$a_{(i+1)(j+2)}$	$a_{(i+1)(j+3)}$
$a_{(i+1)j}$	$a_{(i+1)(j+1)}$	$a_{(i+1)(j+2)}$	$a_{(i+1)(j+3)}$

Более того, пары повторяются не только по столбцам, но и по строкам. Например, для элемента $s_{(i+1)j}$ суммируются элементы $a_{(i+1)j}$, $a_{(i+1)(j+1)}$, $a_{(i+2)j}$, $a_{(i+2)(j+1)}$. Здесь повторно используется пара $a_{(i+1)j}$, $a_{(i+1)(j+1)}$ по сравнению с элементом s_{ij}:

a_{ij}	$a_{i(j+1)}$	$a_{i(j+2)}$	$a_{i(j+3)}$
$a_{(i+1)j}$	**$a_{(i+1)(j+1)}$**	$a_{(i+1)(j+2)}$	$a_{(i+1)(j+3)}$
$a_{(i+1)j}$	$a_{(i+1)(j+1)}$	$a_{(i+1)(j+2)}$	$a_{(i+1)(j+3)}$

Для проведения эффективных вычислений элементов массива S *не стоит дважды* проводить суммирование указанных выше пар элементов, повторяющихся по столбцам и по строкам. Для эффективно векторизуемых вычислений массива S можно вначале к каждому столбцу прибавить следующий:

S **сечение A** **сечение A**

$a_{ij}+a_{i(j+1)}$	$a_{i(j+1)}+a_{i(j+2)}$	=	a_{ij}	$a_{i(j+1)}$	$a_{i(j+2)}$	+	a_{ij}	$a_{i(j+1)}$	$a_{i(j+2)}$
$a_{(i+1)j}+a_{(i+1)(j+1)}$	$a_{(i+1)(j+1)}+a_{(i+1)(j+2)}$		$a_{(i+1)j}$	$a_{(i+1)(j+1)}$	$a_{(i+1)(j+2)}$		$a_{(i+1)j}$	$a_{(i+1)(j+1)}$	$a_{(i+1)(j+2)}$
$a_{(i+1)j}+a_{(i+1)(j+1)}$	$a_{(i+1)(j+1)}+a_{(i+1)(j+2)}$		$a_{(i+1)j}$	$a_{(i+1)(j+1)}$	$a_{(i+1)(j+2)}$		$a_{(i+1)j}$	$a_{(i+1)(j+1)}$	$a_{(i+1)(j+2)}$

Массив S будет иметь размерность Nx(N-1). Число операций для сложения двух сечений Nx(N-1). При векторизации сложение будет вестись со столбцам. Для инструкций AVX-512 за одну операцию будет суммироваться не одна, а 16 пар операндов, поэтому сложность этого этапа составит $Nx(N-1)/16 \sim N^2/16+O(N)$.

Для завершения вычисления массива S теперь достаточно к каждой строке прибавить следующую:

S **сечение S** **сечение S**

$a_{ij}+a_{i(j+1)}+$ $+a_{(i+1)j}+a_{(i+1)(j+1)}$	$a_{i(j+1)}+a_{i(j+2)}+$ $+a_{(i+1)(j+1)}+a_{(i+1)(j+2)}$	=	$a_{ij}+a_{i(j+1)}$	$a_{i(j+1)}+a_{i(j+2)}$	+	$a_{ij}+a_{i(j+1)}$	$a_{i(j+1)}+a_{i(j+2)}$
$a_{(i+1)j}+a_{(i+1)(j+1)}+$ $+a_{(i+1)j}+a_{(i+1)(j+1)}$	$a_{(i+1)(j+1)}+a_{(i+1)(j+2)}+$ $+a_{(i+1)(j+1)}+a_{(i+1)(j+2)}$		$a_{(i+1)j}+a_{(i+1)(j+1)}$	$a_{(i+1)(j+1)}+a_{(i+1)(j+2)}$		$a_{(i+1)j}+a_{(i+1)(j+1)}$	$a_{(i+1)(j+1)}+a_{(i+1)(j+2)}$
			$a_{(i+1)j}+a_{(i+1)(j+1)}$	$a_{(i+1)(j+1)}+a_{(i+1)(j+2)}$		$a_{(i+1)j}+a_{(i+1)(j+1)}$	$a_{(i+1)(j+1)}+a_{(i+1)(j+2)}$

Число операций для сложения двух таких сечений (N-1)x(N-1). При векторизации сложение будет вестись со столбцам. Для инструкций AVX-512 за одну операцию будет суммироваться не одна, а 16 пар операндов, поэтому сложность этого этапа тоже составит $(N-1)x(N-1)/16 \sim N^2/16+O(N)$.

В результате такой алгоритм сможет задействовать векторизацию и потребует на архи-

текторах AVX-512 порядка $N^2/8+O(N)$ операций по сравнению с прямым алгоритмом, который потребует $3N^2+O(N)$ операций.

7.24 Обход массива проводить по столбцам. Также см. 7.20.

Упаковка двумерного массива (Pack) по маске при таком *достаточно регулярном* обращении к элементам массива *неэффективна*. Хотя эту маску и можно получить без проведения вычислений над индексами. Например, для k == 3 маска будет иметь вид:

1	8	15	22
2	9	16	23
3	10	17	24
4	11	18	25
5	12	19	26
6	13	20	27
7	14	21	28
2	**3**	2	2

Потребуется построить массив:

2	5	8	11	14	17	20	23	26
2		3		2		2		

Лучше обойтись *присваиванием* сечений массива, не проводя вычислений маски и дорогостоящей упаковки с помощью Pack *по произвольной* маске. Число нужных элементов в каждом нечётном столбце и в каждом чётном столбце *заранее* известно.

Для этого нужно разместить выходной массив с нужным числом элементов. Как во всякой задаче с *регулярным обращением* к данным, число элементов лучше определять *аналитически*, а *не алгоритмически*.

Во-первых, в каждом столбце нужные элементы (кратные k) идут с шагом k.

Во-вторых, номер столбца определяет, начиная с какой строки в нём начинают идти нужные элементы с этим шагом. В примере они идут в столбце 1 – со строки 2, в столбце 2 – со строки 1, в столбце 3 – со строки 3, а в столбце 4 – опять со строки 2. Эта зависимость устанавливается довольно просто.

В 1-ом столбце сумма индексов равна i_1+1, где i_1 – это номер строки элемента, а 1 – это, собственно, номер столбца. Для первого элемента, сумма индексов которого в 1-ом столбце кратна k, имеем: $(i_1+1)\vdots k \Rightarrow i_1+1=k \Rightarrow i_1=k-1$. Для примера это $i_i=3–1=2$, т. е. во 2-ой строке.

1	8	15	22
2	9	16	23
3	10	17	24
4	11		25
5	12	19	26
6	13	20	27
7	14	21	28

Для 2-ого столбца сумма индексов равна i_2+2, и для первого элемента в нём, сумма индексов которого кратна k, имеем: $(i_2+2)\vdots k \Rightarrow i_2+2=k \Rightarrow i_2=k-2$. Для примера это $i_2=3–2=1$,

т. е. во 1-ой строке.

1	8	15	22
2	9	16	23
3	10	17	24
4	11	18	25
5	12	19	26
6	13	20	27
7	14	21	28

Обобщая, можно сказать, что чем меньше номер столбца (*из первых k столбцов*), тем позже начинает идти в нём нужный элемент. Тогда становится видно, что для первого элемента в j-ом столбце, сумма индексов которого кратна k, имеем: $(i_j+k)\vdots k \Rightarrow i_j = k - j \bmod k$, где *mod* – операция взятия остатка от деления. Для примера это $(i_j+3)\vdots 3 \Rightarrow i_j = 3 - j \bmod 3$. Индекс первого нужного элемента в столбце совпадает у каждого k-ого столбца. Для примера у 1-ого и 4-ого столбца нужные элементы идут, начиная со 2-ой строки.

На основании этого полезно будет создать массив начальных индексов для каждого столбца Initial(k). Его индексы разумно будет пронумеровать от 0 до (k-1), чтобы легко обращаться к ним в будущем, беря остаток от деления номера нужного столбца j на k:

Initial(j mod k)	3	2	1
j (№ столбца)	3	1	2
	6	4	5
j mod k	0	1	2

Итак, массив Initilal(0:k-1) будет хранить номера строк *первых* нужных элементов в каждом столбце j: Initial(j mod k) = k – j mod k. Для *эффективного* построения этого массива с использованием векторизации можно заметить, что:

- для j == 1..(k-1): j mod k == j, поэтому $Initial(j \bmod k) = k - j \bmod k \Leftrightarrow Initial(j) = k - j$;
- для j == k: j mod k == 0, поэтому $Initial(j \bmod k) = k - j \bmod k \Leftrightarrow Initial(0) = k$.

Осталось сформировать массив *количества нужных элементов* в каждом столбце Quantity(k). Этот массив также будет иметь длину всего k, т. к. число нужных элементов в каждом столбце будет повторяться каждые k столбцов:

1	8	15	22
2	9	16	23
3	10	17	24
4	11		25
5	12	19	26
6	13	20	27
7	14	21	28

К этому массиву тоже будет лучше обращаться через остаток от деления номера текущего столбца j на k. Раз в каждом столбце N элементов и нужные элементы в столбце j встречаются с шагом k, начиная с известного номера строки в нём Initial(j mod k), то число нужных элементов в j-ом столбце будет равно (N-Initial(j mod k))/k (используется *целочисленная* операция деления). На основании этого можно построить массив Quantity(0:k-1): Quantity(j

mod k) = (N-Initial(j mod k))/k. Как и выше, для эффективного построения этого массива с использованием векторизации можно заметить, что:

- для j == 1..(k–1): j mod k == j, поэтому

  $$Quantity(j\,mod\,k)=(N-Initial(j\,mod\,k))/k \Leftrightarrow Quantity(j)=(N-k+j)/k=(N+j)/k-1$$

- для j == k: j mod k == 0, поэтому $Quantity(j\,mod\,k)=(N-Initial(j\,mod\,k))/k \Leftrightarrow Quantity(0)=(N-k)/k=N/k-1$.

После того, как эффективно построены массив номеров строк первых нужных элементов в каждом столбце Initial(0:k-1) и массив количества нужных элементов в каждом столбце Quantity(0:k-1), можно организовать цикл, на каждой итерации которого в результирующий массив будет присваиваться Quantity(j mod k) элементов j-ого столбца массива, начиная со строки Initial(j mod k) с шагом k. На каждой итерации необходимо будет пересчитывать начальное first и конечное last положение сечения в результирующем массиве в зависимости от числа присваиваемых элементов:

```
do j = 1, N
   i = j mod k
   First = ...
   Last = ...
   B(First:Last) = A(Initial(i)::k, j)
end do
```

Подумайте, как провести присваивание с помощью неявных циклов (см. 7.20).

7.25

7.26 Использовать сечения двумерного массива.

7.27 На каждой итерации do concurrent присваивать к сечению из i-ого и (i+1)-ого столбца сечение в виде (i+1)-ого и i-ого столбца.

7.28 Использовать do concurrent.

7.29 Регулярно переставлять строки при сортировке *неэффективно* при размещении матрицы в памяти по столбцам. Матрицу A лучше хранить и обрабатывать в транспонированном виде, чтобы проходя по строкам, читать элементы *в порядке их расположения* в памяти.

Сформировать массив номеров строк. Проводить сортировку первых элементов строк, переставляя при этом строки матрицы и номера в массиве. Использовать сортировку выбором (MaxLoc), для которой характерно меньшее число перестановок элементов. См. указания к сортировке.

7.30 Any, Reshape, Pack, MatMul, Count.

7.31 Сформировать массив номеров. Проводить сортировку по первым элементам столбцов, переставляя при этом столбцы матрицы и номера в сформированном массиве. Использовать сортировку выбором (MaxLoc), для которой характерно меньшее число перестановок элементов. См. указания к сортировке.

7.32 Использовать do concurrent для внешнего цикла по строкам. Ходить по самим строкам при сортировке при размещении матрицы в памяти по столбцам *неэффективно*. Матрицу B лучше хранить и обрабатывать в транспонированном виде, чтобы проходя по строкам, читать элементы в порядке их расположения в памяти.

а-б Если не использовать хранение в транспонированном виде, то перед сортировкой сформировать массив из элементов строки. Эффективно отсортировать его и присвоить к строке.

в-г Вычислять модуль элементов каждый раз *неэффективно*.

Можно перед сортировкой сформировать: массив из модулей элементов строки AbsS(:) и массив *номеров элементов* в строке Indexes(:). Сортируя первый из них, отсортировать и второй. После сортировки присвоить к строке ту же строку с отсортированным порядком элементов, используя массив индексов как вектор индексов: AbsS(:) = B(i, Indexes()), B(i, :) = AbsS или при хранении в транспонированном виде AbsS(:) = B(Indexes(), i), B(:, i) = B(Indexes(), i). См. указания к сортировке.

Такой подход требует использования *нерегулярного* обращения к массиву – B(i, Indexes()). Для того, чтобы избежать этого, перед сортировкой лучше сформировать *только* массив из модулей элементов строки AbsA(:). Сортируя первый из них, сортировать и строку.

7.33 Регулярно переставлять строки при сортировке *неэффективно* при размещении матрицы в памяти по столбцам. Матрицу A лучше хранить и обрабатывать в транспонированном виде, чтобы проходя по строкам, читать элементы *в порядке их расположения* в памяти.

Сформировать массив наибольших элементов строк (MaxVal с параметром dim). Проводить сортировку этого массива, переставляя при этом и строки матрицы. Использовать сортировку выбором (MaxLoc), для которой характерно меньшее число перестановок элементов, которыми в данном случае являются целые строки. См. указания к сортировке.

7.34 На входе матрица квадратная. Сперва сформировать массив из номеров максимальных элементов *в каждой* строке (MaxLoc с параметром dim), чтобы эта операция могла быть векторизована. В цикле с независимыми итерациями и условием (do concurrent *с условием*) провести замену диагонального элемента, если его *номер* в строке *отличается* от максимального.

7.35 Регулярно переставлять строки при сортировке *неэффективно* при размещении матрицы в памяти по столбцам. Матрицу A лучше *хранить и обрабатывать* в транспонированном виде, чтобы проходя по строкам, читать элементы *в порядке их расположения* в памяти.

Для сумм элементов строк завести отдельный массив (Sum с параметром dim). Проводить сортировку этого массива, переставляя при этом и строки матрицы. Использовать сортировку выбором (MaxLoc), для которой характерно меньшее число перестановок элементов, которыми в данном случае являются целые строки. См. указания к сортировке.

7.36 Размер массива должен быть кратен 5. В результующем двумерном массиве должно быть 5 строк. Reshape.

7.37 Reshape.

7.39 Конструктор массивов, неявный цикл, Pack.

7.40 Использовать маску не эффективно. В неявном цикле на каждой его итерации упаковывать элементы i-ого и i+1-ого столбца (нечётного и чётного).

7.41 Конструктор массива или Reshape.

7.42 Sum.

7.44 MaxVal с параметром dim. Неважно, какой по счёту элемент брать, если их несколько.

7.45 MatMul.

7.46 Искать расстояние *не требуется* – достаточно оценивать сумму квадратов. Sum и MaxLoc.

7.47 Do concurrent, MaxVal.

7.49 Использовать маску при таком регулярном обращении к данным неэффективно – проводить присваивание четырёх разделённых частей матрицы с помощью сечений.

7.50 Использование маски напрасно при таком *регулярном обращении* к данным. Формировать новую матрицу, перенеся в неё сперва верхнюю треугольную матрицу, а потом нижнюю треугольную матрицу. Треугольные матрицы обходить в порядке расположения элементов в памяти. Использовать do concurrent.

7.51 Конструктор массивов, неявный цикл по строкам, Pack.

7.52 Использовать конструкцию do concurrent (сделать итерации независимыми друг от друга). Sum.

7.53 Использовать конструкцию do concurrent (сделать итерации независимыми друг от друга – позиции обнуляемых строк в столбце зависят от номера столбца). Обнуление проводить по столбцам.

3.3.16 Допуск к упражнениям § VIII

1. Собрать и запустить опорные упражнения 8.пример-1, 8.пример-2, 8.22.
2. Разобраться в их исходном коде.
3. Ответить на вопросы преподавателя по исходному коду.

3.3.17 Указания к § VIII (не весь)

Все процедуры, предлагаемые для разработки, должны быть чистыми – иметь квалификатор pure. Они не должны использовать встроенные функции по работе с массивами или сечения.

8.1 Можно сразу реализовать процедуру умножения на лету транспонируемой матрицы на вектор *без операции* транспонирования.

8.4 Вызывать подпрограмму в цикле do concurrent.

8.6 Нет указаний.

8.7 Использовать конструкцию do concurrent (сделать итерации независимыми друг от друга). Обнуление проводить по столбцам.

8.8 Предусмотреть, что M может быть чётным или нечётным.

8.9 Transpose, MatMul.

8.10 Do concurrent по строкам.

8.12 Массивы заполнять в цикле с гарантировано независимыми итерациями (do concurrent).

8.15 Обработку данных проводите так, как данные располагаются в памяти.

8.18 Вычислите все значения X. Подставьте его в созданную *элементную чистую* функцию (elemental).

8.16 Обходиться одним циклом – проверять сразу элементы главной и побочной диагонали.

8.21 Реализовать **чистую элементную** функцию f(x). Разница между членами:

$$q_{n+1}=\frac{s_{n+1}}{s_n}=\frac{(2(n+1))!\,x^{(2(n+1)+1)}}{4^{n+1}((n+1)!)^2(2(n+1)+1)}\,\frac{4^n(n!)^2(2n+1)}{(2n)!\,x^{(2n+1)}}=¿$$

$$¿\,\frac{(2n+2)!\,x^{(2n+3)}}{4^{n+1}((n+1)!)^2(2n+3)}\,\frac{4^n(n!)^2(2n+1)}{(2n)!\,x^{(2n+1)}}=x^2\frac{(2n+1)^2}{2(n+1)(2n+3)}=x^2\frac{(n+0.5)^2}{(n+1)(n+1.5)}$$

Члены нумеруются с нуля: **n = 0, 1, 2**, … Так, например, для **n = 2** множитель составляет (если использовать *предпоследнее* равенство последней формулы):

$q_3=q_{2+1}=x^2\frac{5^2}{6\cdot 7}$, что и видно из ряда.

Упростить функцию q(x), сократив на f(x). Реализовать **чистую элементную** функцию q(x). Формируете массив X и вызываете на нём функцию q(X).

8.23 Функция вычисления интеграла должна принимать функцию вычисления подынтегральной функции и параметры p и q. Функция вычисления подынтегральной функции должна принимать вектор X и параметры p и q.

8.26 Проверку на значение нормы меньшее 1 выполнять внутри функции и в таком случае возвращать скорректированную норму.

Глава 4. Лабораторные работы

При выполнении лабораторных работ, создавая своё *произведение*, необходимо следовать дисциплине программирования, описанной в главе 1. При сдаче принимается исходный код ПО и результаты его работы.

Для студентов, обучающихся на базе профессионального образования, шестая лабораторная работа является необязательной.

§ 4.1 Общие указания к лабораторным работам

1. Лабораторная работа выполняется в виде программного проекта (см. § 1.1 Организация проектов).
2. Все исходные данные вводятся из входного форматированного файла.
3. Все входные и выходные форматированные файлы имеют формат UTF-8.
4. Текст во входных и выходных файлах написан на русском языке, сели другого не указано в задании.
5. В выходной форматированный файл отдельными процедурами выводится сперва все введённые данные, а затем полученный результат.
6. При разработке кода с модулями должны быть:
    1. один модуль ввода/вывода с определением *входных и выходных* структур данных;
    2. второй модуль обработки данных, содержащий *только чистые* процедуры.
7. Необходимо проверить разработанные чистые процедуры на различных тестах, в том числе вырожденных.

§ 4.2 Советы к лабораторным работам

Если ваш компилятор не поддерживает в исходном коде неименованные строковые константы в кодировке UTF-8 (gfortran: 45179), то при необходимости сравнения символа строки с определённым символом используется код этого символа (коды **и десятичном формате:** unicode-table.com):

```
if (string(1:1) == Char(1052, CH_) ! Сравнение 1-ого символа с «М».
DA = Char(1044, CH_) // Char(1072, CH_) ! Запись в константу DA слова «Да».
```

§ 4.3 Лабораторная работа № 1. Разработка структур данных

4.3.1 Общая часть задания

1. Одна и та же информация из входного файла вводится по-разному в оперативную память с целью освоения работы с различными структурами данных — задание выполняется в виде 5 отдельных программных проектов, где необходимо использовать:

	Проект лабораторной работы № 1				
Средства	1	2	3	4	5
массивы строк	+				

	Проект лабораторной работы № 1				
Средства	1	2	3	4	5
массивы символов		+			
внутренние процедуры головной программы		+			
массивы структур			+	+	
файлы записей[1]			+	+	
модули			+	+	+
хвостовая рекурсия				+	+
однонаправленные списки заранее неизвестной длины[2]					+
регулярное программирование	+	+	+	+	+

2. Необходимо прочитать список известной длины из не менее чем 12 строк (кроме проекта № 5). Данные в одной строке имеют заданный формат и отделяются друг от друга дополнительным пробелом.

4.3.2 Опорный вариант лабораторной работы № 1

Дан список группы с результатам сессии и с не вычисленным средним баллом в виде:

```
ФАМИЛИЯ  ИНИЦИАЛЫ ПОЛ     РЕЗУЛЬТАТЫ_СЕССИИ СРЕДНИЙ_БАЛЛ
15 симв. 5 симв.  1 симв. 5 симв.           3 симв.
```

Пример входного файла:

```
Дудиков         Д. Р. М 43453 0.0
Тихонов         Л. П. М 55353 0.0
Садовникова     П. О. Ж 43543 0.0
Степин          К. Д. М 55445 0.0
Воробъёва       Е. Р. Ж 44353 0.0
```

Рассчитать средний балл для каждого из учащихся. Отсортировать по убыванию среднего балла списки юношей и девушек по отдельности.

Пример выходного файла:

```
Успеваемость юношей:
Степин          К. Д. М 55445 4.60
Тихонов         Л. П. М 55353 4.20
Дудиков         Д. Р. М 43453 3.80
Успеваемость девушек:
Воробъёва       Е. Р. Ж 44353 3.80
Садовникова     П. О. Ж 43543 3.80
```

4.3.3 Индивидуальные задания

1. Дан список группы в виде:

```
ФАМИЛИЯ  ПОЛ     РЕЗУЛЬТАТЫ СЕССИИ
15 симв. 1 симв. 4 симв.
```

Пример входного файла:

```
Иванов          М 4455
Петрова         Ж 3554
```

1 Файл записей формируется отдельной процедурой из форматированного файла *запись за записью*, а затем из него происходит чтение *одной записи* — массива структур.

2 Каждая строка исходного форматированного файла рассматривается как элемент списка, файл с неизвестным числом строк читается до конца. Длину списка можно запомнить при чтении, если только она используется при сортировке.

Определить лидеров среди мужчин и женщин по успеваемости и их средний балл (у мужчин и женщин отдельно). Пример выходного файла

```
Лидеры по успеваемости:
Иванов          4455  ср. балл 4.50
Петрова         3554  ср. балл 4.30


Средний балл среди мужчин: 4.2
Средний балл среди женщин: 4.1
```

Указание. Сформировать сперва массивы для мужчин и женщин, а потом с ними работать. Для обработки данных обоих полов повторно использовать одни и те же процедуры.

2. Дан список группы в виде:

```
ФАМИЛИЯ          И. О.   ПОЛ     ПРОПИСКА СРЕДНИЙ_БАЛЛ

15 симв.         5 симв. 1 симв. 1 симв.  4 симв.
```

Пример входного файла (в графе прописки буква П стоит у петербурцев, С — у гостей Санкт-Петербурга):

```
Иванов          И. И. М П 4.35
```

Отсортировать по убыванию среднего балла по отдельности списки петербуржцев и гостей Санкт-Петербурга. Пример выходного файла:

```
Петербуржцы:
Иванов          И. И. М 4.35
Барабашкин      И. И. М 4.32

Гости города:
Петров          И. И. М 4.35
Петрыкин        И. И. М 4.32
```

3. Дан список группы в виде:

```
ФАМИЛИЯ          И. О.   ПОЛ     ГОД РОЖДЕНИЯ

15 сив.          5 симв. 1 симв. 4 симв.
```

Пример входного файла:

```
Иванов          И. И. М 1995
```

Определить средний возраст юношей в группе. Примеры выходного файла:

```
Средний возраст юношей в группе: 21 год
Средний возраст юношей в группе: 22 года

Средний возраст юношей в группе: 25 лет
```

4. Дан список группы в виде:

```
ФАМИЛИЯ  ГОД РОЖДЕНИЯ СЛУЖБА В АРМИИ ПРОПИСКА   ПОЛ

15 симв. 4 симв.      3 симв.        1 симв.    1 симв.
```

Пример входного файла (в графе прописки буква П стоит у петербурцев, с — у гостей Санкт-Петербурга):

```
Иванов          1992 нет П М

Петров          1993 да  С М
```

Отсортировать в алфавитном порядке по отдельности списки петербуржцев и гостей Санкт-Петербурга, служивших в армии. Пример выходного файла:

```
Служившие петербуржцы:
Барабашкин      1992 М
Служившие гости города:
Петров          1993 М
```

5. Дан список сотрудников научно-исследовательской лаборатории в виде:

```
ФАМИЛИЯ  ДОЛЖНОСТЬ
15 симв. 15 симв.
```

Пример входного файла:

```
Иванов          техник
```

Определить число одинаковых должностей. Пример выходного файла:

```
ведущий инженер - 2
старший инженер - 3
инженер         - 8
техник          - 2
```

Указание. Завести логический массив длиной N. При обработке должности очередного сотрудника сразу найти все такие должности и пометить в логическом массиве, где они встречаются. Должность следующего сотрудника обрабатывать, только если она прежде не встречалась (смотреть в логический массив). Использовать Count с маской, Any. См. упражнение 5.16, 7.25.

6. Дан список товаров с указанием цены вида:

```
НАИМЕНОВАНИЕ ВЕЩИ ЦЕНА    ВАЛЮТА
20 симв.          5 симв. 5 симв.
```

Пример входного файла:

```
Стол                2500  руб.
```

Определить три самых дорогих товара. Пример выходного файла:

```
Самые дорогие товары:
1. Диван                - 12000 руб
2. Стул                 - 9000 руб.
3. Тумба                - 6000 руб.
```

Указание. Не следует сортировать весь список – нужно найти только первые три элемента. Необходимо сформировать массив/динамический список, хранящий три самых дорогих товара.

7. Дан список группы в виде:

```
ФАМИЛИЯ         И. О.
15 сив.         5 симв.
```

Пример входного файла:

```
Иванов          И. И.
```

Отсортировать список по алфавиту с учетом инициалов методом сравнения «один со всеми». Пример выходного файла:

```
Иванов          И. А.
Иванов          И. И.
Иванова         И. И.
```

Указание. При проверке равенства массивов символов использовать встроенную функцию All.

8. Дан список владельцев телефонов в виде:

```
ФАМИЛИЯ  ТЕЛЕФОН
15 симв. 10 симв.
```

Пример входного файла:

```
Петров          9111634576
```

```
Фёдоров          9111635687
```

Отсортировать этот список в порядке убывания номеров телефонов, используя метод вставок. Пример выходного файла:

```
Фёдоров          9111635687
Петров           9111634576
```

9. Дан список группы в виде:

```
ФАМИЛИЯ          И. О.   ГОД РОЖДЕНИЯ
15 сив.          5 симв. 4 симв.
```

Пример входного файла:

```
Иванов           И. И. 1994
Петров           Д. Т. 1992
```

Выделить первого по алфавиту и самого молодого. Пример выходного файла:

```
Первый по алфавиту:
Иванов           И. И. 1994
Самый молодой:
Петров           Д. Т. 1992
```

10. Дан список товаров в виде:

```
НАИМЕНОВАНИЕ ЦЕНА    ВАЛЮТА  ПРОЦЕНТ ИЗНОСА
10 симв.     5 симв. 5 симв. 3 симв.
```

Пример входного файла:

```
Стол                2500  руб. 45
```

Используя процент износа, сформировать продажную комиссионную цену. Пример выходного файла:

```
Комиссионная цена вещей:
Стол                1375 руб.
```

11. Дан список сотрудников научно-исследовательской лаборатории в виде:

```
ФАМИЛИЯ  ДОЛЖНОСТЬ
15 симв. 15 симв.
```

Пример входного файла:

```
Иванов           техник
```

Отсортировать список в порядке повышения должности от «техника» до «вед. инженера». Пример выходного файла:

```
Иванов           техник
Петров           старший инженер
```

Указание. Для сравнения должностей завести логическую функцию, которая будет возвращать результат сравнения двух должностей. Должности: техник, инженер, старший инженер, ведущий инженер, главный инженер. Должности можно поместить в массив для организации сравнения.

12. Дан список группы в виде:

```
ФАМИЛИЯ  И. О.   ГОД РОЖДЕНИЯ ПРОПИСКА  ПОЛ
15 симв. 5 симв. 4 симв.      1 симв.   1 симв.
```

Пример входного файла (в графе прописки буква П стоит у петербуржцев, С — у гостей Санкт-Петербурга):

```
Иванов          И. И. 1995 П М
Петрова         Х. Л. 1994 С Ж
```

Выделить из них трёх наиболее молодых петербуржцев мужчин. Пример выходного файла:

```
Иванов          И. И. 1995
Галкин          В. И. 1997
Потапкин        Е. З. 1997
```

Указание. Не следует сортировать весь список – нужно найти только первые три элемента.

13. Дан список группы в виде:

```
ФАМИЛИЯ  ИМЯ      ОТЧЕСТВО
15 симв. 10 симв. 15 симв.
```

Пример входного файла:

```
Безруков        Сергей     Викторович
Лебедев         Пётр       Станиславович
```

Определить число встречающихся имен. Пример выходного файла:

```
Сергей     - 2
Пётр       - 1
```

14. Дан список владельцев телефонов в виде:

```
ФАМИЛИЯ  ТЕЛЕФОН
15 симв. 10 симв.
```

Пример входного файла:

```
Петров          9111634576
Фёдоров         9111635687
Петров          9111634573
```

Найти строку с первой по алфавиту фамилией и с наименьшим номером телефона. Пример выходного файла:

```
Петров          9111634573
```

15. Дан список группы в виде:

```
ФАМИЛИЯ  ИМЯ      ОТЧЕСТВО
15 симв. 10 симв. 15 симв.
```

Пример входного файла:

```
Безруков        Сергей     Викторович
Лебедев         Пётр       Станиславович
Глухих          Сергей     Алексеевич
```

Удалить из списка всех, у которых имена совпадают, кроме первого. Пример выходного файла:

```
Безруков        Сергей     Викторович
Лебедев         Пётр       Станиславович
```

Указание. Работать с исходным массивом. При каждом нахождении повторяющегося имени будет неэффективным тут же удалять его и сдвигать все остальные элементы в массиве, потому что дальше это имя опять может встретиться.

Будет неэффективным также нахождение сперва всех мест в массиве, где находится текущее имя, а потом удаление его оттуда сдвигом остальных элементов, потому что среди остальных имён могут оказаться другие повторяющиеся, которые всё равно в итоге придётся удалять.

Необходимо сперва сформировать массив-маску для всех уникальных имён. При этом не стоит проверять имя, если оно уже встречалось. Затем по маске упаковать исходные мас-

сивы (Pack).

При работе с динамическим списком обработку проводить двумя рекурсивными процедурами – в первой рекурсии берётся очередное имя, во второй оно удаляется дальше по списку.

16. Дан список группы в виде:

```
ФАМИЛИЯ  ГОД РОЖДЕНИЯ ПОЛ
15 симв. 4 симв.      1 симв.
```

Пример входного файла:

```
Иванов          1985 М
Петрова         1983 Ж
```

Найти самого пожилого мужчину и самую молодую женщину. Пример выходного файла:

```
Самый пожилой мужчина:
Иванов          1985
Самая молодая женщина:
Петрова         1983 Ж
```

17. Дан список группы в виде:

```
ФАМИЛИЯ  ПОЛ     РЕЗУЛЬТАТЫ СЕССИИ
15 симв. 1 симв. 4 симв.
```

Пример входного файла:

```
Иванов          М 4455
Петрова         Ж 3554
```

Отсортировать по убыванию среднего балла мужчин и женщин по отдельности. Пример выходного файла:

```
Мужчины:
Иванов          М 4455 4.50

Женщины:
Петрова         Ж 3554 4.25
```

18. Дан список группы в виде:

```
ФАМИЛИЯ  И. О.   ПОЛ     ГОД РОЖДЕНИЯ ПОЛ
15 симв. 5 симв. 1 симв. 1 симв.
```

Пример входного файла:

```
Иванов          И. Л. М 1985
Петрова         Д. О. Ж 1983
```

Сформировать отсортированные по убыванию возраста списки мужчин и женщин. Использовать для сортировки метод “выбором”. Пример выходного файла:

```
Мужчины:
Иванов          И. Л. М 1985

Женщины:
Петрова         Д. О. Ж 1983
```

19. Дан список сотрудников лаборатории в виде:

```
ФАМИЛИЯ  ПРОФЕССИЯ
15 симв. 10 симв.
```

Пример входного файла:

```
Иванов          повар
Петрова         фармацевт
```

Сформировать список профессий с указанием их количества. Пример выходного файла:

```
повар      - 1
```

```
фармацевт  - 1
```

20. В первом входном файле дан список товаров в виде:

```
НАИМЕНОВАНИЕ ЦЕНА    ВАЛЮТА
10 симв.     5 симв. 5 симв.
```

Пример первого входного файла:

```
стол                9500  руб.
бра                 1000  руб.
диван              12000  руб.
люстра               500  руб.
```

Во втором входном файле дан список обновлённых товаров в виде:

```
ЦЕНА    ВАЛЮТА
5 симв. 5 симв.
```

В конце списка приводится массив из значений T и F (истина и ложь), указывающий у каких товаров в исходном списке была изменена цена. Пример второго входного файла:

```
1200  руб.
 450  руб.
```

F T F T

Сформировать обновлённый список товаров. Пример выходного файла:

```
стол                9500  руб.
бра                 1200  руб.
диван              12000  руб.
люстра               450  руб.
```

§ 4.4 Лабораторная работа № 2. Однонаправленные списки

4.4.1 Общая часть задания

Задание выполняется в виде программного проекта из двух модулей, в котором необходимо использовать *однонаправленные списки* неизвестной заранее длины. Списки необходимо обрабатывать чистой хвостовой рекурсией **при всех** операциях с ними. При возможности применяется регулярное программирование. Смотрите § 4.1 Общие указания к лабораторным работам.

4.4.2 Опорный вариант лабораторной работы № 2

Во входном файле F1 находится исходный код программы на Fortran, а в файле F2 — тот же код, но с добавлением некоторых строк. Сформировать файл из новых строк, пометив их в начале как «++ ».

Пример входного файла F1:

```
module environment
   use ISO_Fortran_Env

   implicit none

   integer, parameter      :: I_ = int16
   integer, parameter      :: C_ = R_
   character(*), parameter :: E_ = "UTF-8"

   interface operator (//)
      module procedure IntPlusString
```

```
   end interface

contains

   pure function IntPlusString(int, str) result(res)
      integer, intent(in)                                  :: int
      character(*), intent(in)                             :: str
      character(len(str)+Floor(Log10(Real(int, real64)))+1) :: res

      write (res,'(i0, a)') int, str
   end function IntPlusString

end module environment
```

Пример входного файла F2:

```
module environment
   use ISO_Fortran_Env

   implicit none

   integer, parameter      :: I_ = int16
   integer, parameter      :: R_ = real32
   integer, parameter      :: C_ = R_
   integer, parameter      :: CH_ = Selected_Char_Kind("ISO_10646")
   character(*), parameter :: E_ = "UTF-8"

   interface operator (//)
      module procedure IntPlusString
      module procedure StringPlusInt
   end interface

contains

   pure function IntPlusString(int, str) result(res)
      integer, intent(in)                                  :: int
      character(*), intent(in)                             :: str
      character(len(str)+Floor(Log10(Real(int, real64)))+1) :: res

      write (res,'(i0, a)') int, str
   end function IntPlusString

   pure function StringPlusInt(str, int) result(res)
      character(*), intent(in)                              :: str
      integer, intent(in)                                   :: int
      character(len(str)+Floor(Log10(Real(int, real64)))+1)  :: res

      write (res,'(a, i0)') str, int
   end function StringPlusInt

end module environment

! Нечто.
```

Пример выходного файла:

```
++    integer, parameter      :: R_ = real32
++    integer, parameter      :: CH_ = Selected_Char_Kind("ISO_10646")
++       module procedure StringPlusInt
++    pure function StringPlusInt(str, int) result(res)
++       character(*), intent(in)                              :: str
++       integer, intent(in)                                   :: int
++       character(len(str)+Floor(Log10(Real(int, real64)))+1)  :: res
++
```

```
++       write (res,'(a, i0)') str, int
++    end function StringPlusInt
++
++
++ ! Нечто.
```

4.4.3 Индивидуальные задания

1. Разработать чистую подпрограмму сдвига части строк в данном тексте влево или вправо на N символов, начиная со строки N1 до строки N2. Значения N, N1, N2, а также направление сдвига («left», «right») задаются во втором входном файле.

   Пример первого входного файла:

   ```
   1
   12
   123
   1234
   12345
   ```

   Пример второго входного файла:

   ```
   2 3 4 L
   ```

   Пример выходного файла:

   ```
   1

   3
   34
   12345
   ```

   Указание. Элементом списка является строка. Для хранения строки лучше использовать размещаемый массив символов (EOShift).

2. Разработать чистую функцию проверки того, что данная строка A(M) состоит только из символов заданной строки B(L).

   Указание. Элементом списка является символ строки. Возвращать номер символа, не найденного в строке B(L). Иначе возвращаем M+1.

3. Разработать чистую функцию формирования новой строки C(L+M) с помощью вставки заданной строки B(L) в строку A(M) после k-го символа (L≤10, M≤10, 0≤k≤M).

   Указание. Элементом списка является символ строки.

4. Разработать чистую подпрограмму контекстного поиска и замены заданного слова S1 на слово S2 всюду в тексте.

   Указание. Элементом списка является слово исходного файла, хранимое в размещаемой строке. Символ конца строки (перевод каретки) хранить в виде отдельного элемента списка, содержащего всего один символ, возвращаемый функцией New_line(CH_). Слова читать во временную строку str, используя оператор

```
read (In, '(a)', advance='no', size=size, iostat=io) str
```

   где size — число реально прочитанных символов. Вы дойдёте до конца строки, когда io == IOSTAT_EOR (константа, обозначающая чтение конца записи).

5. Разработать чистую подпрограмму перемещения в тексте группы строк со строки First по строку Last после K-ой строки. Значения First, Last, K задаются во втором входном файле. Правильными данными считается ситуация , когда First≤Last ,а K не принадлежит интервалу [First, Last].

Указание. Элементом списка является строка.

6. Разработать чистую подпрограмму Top установки окна на начало текста и чистую подпрограмму Bottom установки на конец данного текста. Размер окна N (число строк в окне) задаётся во втором входном файле.

   Указание. Элементом списка является строка.

7. Разработать чистую подпрограмму удаления из текста группы строк с номерами от начальной First до конечной Last.

   Указание. Элементом списка является строка. Удаление проводить в созданном динамическом списке.

8. Разработать чистую подпрограмму вставки в строке на N-ое место M символов и чистую подпрограмму удаления в строке с N-го места M символов. После каждой вставки или удаления выводить обновленную строку. Режим ввода или удаления, а также число символов задаются во входном файле.

   Пример входного файла:

   ```
   Just for what?
   D 10 5
   I 10 fun!
   ```

   Пример выходного файла:

   ```
   Just for what?
   D 10 5
   Just for_
   I 10 fun!
   Just for fun!
   ```

   Указание. Элементом списка является символ строки. Головная программа должна состоять из вызова процедур: чтение строки, вывод строки, а затем в цикле: чтение команды, выполнение команды, вывод строки.

9. Разработать чистую подпрограмму копирования группы строк данного текста от начальной строки First до конечной Last и вставки их после M-ой строки. Правильному набору вводимых данных соответствует ситуация, когда 0≤First≤Last и M не входит в диапазон [First, Last].

   Указание. Элементом списка является строка.

10. Разработать чистую подпрограмму контекстной замены каждого отдельного вхождения подстроки B(L) в строку A(M) на L символов '*'. Если подстроки B(L) нет внутри A(M), либо ее принципиально нельзя выделить из A(M), то оставлять строку A без изменений.

    Указание. Элементом списка является символ строки.

11. Разработать чистую подпрограмму пролистывания заданного текста. Размер листа N и направления пролистывания задаются во втором входном файле.

    Пример первого входного файла:

    ```
    1
    2
    3
    4
    5
    ```

    Пример второго входного файла:

    ```
    2
    ```

```
F
F
B
```

Пример выходного файла:

```
Исходный файл:
1
2
3
4
5

Размер листа:
2

Пролистывание:
1
2

F
2
3

F
3
4

B
2
3
```

Указание. Элементом списка является строка.

12. Разработать чистую функцию поиска подстроки B(L) в строке A(M).

    Указание. Элементом списка является символ строки. Функция возвращает номер позиции и 0 в случае ненахождения вхождения.

13. Разработать чистую функцию выделения подстроки B(L) из строки A(M), начиная с её K-ого символа. Выделять столько символов, сколько возможно, вплоть до ни одного.

    Указание. Элементом списка является символ строки.

14. Разработать чистую процедуру сортировки строк заданного текста по убыванию длины строки. Использовать сортировку вставками.

    Указание. Элементом списка является строка. Применять функцию Len_trim.

15. Разработать чистую подпрограмму “центрирования” строк заданного текста, каждая из которых имеет длину L≤40 символов и строки имеют разное число пробелов слева.

    Указание. Элементом списка является размещаемая строка.

16. Разработать чистую подпрограмму сортировки списка имен файлов с расширениями по имени и чистую подпрограмму сортировки того же списка по расширению. Использовать метод сортировки выбором.

    Указание. Элемент списка содержит название файла и его расширение. Применять функцию Index для поиска точки после предварительного прочтения всего имени файла.

17. Разработать чистую подпрограмму удаления из списка файлов имён с заданным расширением.

Пример входного файла:

```
obj
main.f90
main.obj
group.f90
group.obj
```

Пример выходного файла:

```
Исходный список:
obj
main.f90
main.obj
group.f90
group.obj

Полученный список:
main.f90
group.f90
```

Указание. Элемент списка содержит название файла и его расширение. Применять функцию Index для поиска точки после предварительного прочтения всего имени файла.

18. Разработать чистую подпрограмму поиска в списке файлов имён по заданной маске. Например, если вводится строка wi*.* , то необходимо оставить имена файлов с любыми расширениями, но содержащие в качестве первых букв имени буквы wi.

    Указания.

- Элемент списка содержит название файла и его расширение. Применять функцию Index для поиска точки после предварительного прочтения всего имени файла.
- В разрабатываемой подпрограмме лучше сперва получить список, удовлетворяющий части маски по имени, а потом из него — список, удовлетворяющий части маски по расширению.
- Проверку соответствия части маски (по имени или по расширению) можно проводить, проверяя сперва соответствие начала, а потом соответствие конца этой части. Для этого каждую часть маски можно хранить как две строки. Например, маска wi*.* представляется такими двумя частями: («wi», «») и («», «»). Тогда при проверке части маски по имени нужно найти все файлы, первые два символа которых совпадают с «wi», а последние — не важны (длина «» равна 0), а при проверке части маски по расширению нужно найти все файлы, первые и последние символы которых не важны (длина «» и «» равна 0).

19. Дан список имен файлов с расширениями, расположенных по алфавиту, и список ранее удаленных файлов в произвольном порядке. Разработать чистую подпрограмму восстановления в первом списке имен файлов из второго списка, вставляя имена из последнего в нужное место по алфавиту.

    Указание. Элемент списка содержит название файла и его расширение. Применять функцию Index для поиска точки после предварительного прочтения всего имени файла.

20. Даны два списка имён файлов с расширениями. Разработать чистую подпрограмму добавления в первый список имён файлов имён файлов из второго списка таким образом, чтобы в случае нахождения такого файла у него бы менялось расширение на *.bak, а новый добавлялся за ним.

Указание. Элемент списка содержит название файла и его расширение. Применять функцию Index для поиска точки после предварительного прочтения всего имени файла.

§ 4.5 Лабораторная работа № 3

4.5.1 Общая часть задания

Задание выполняется в виде программного проекта из двух модулей, в котором необходимо использовать *динамические списки* неизвестной заранее длины. Списки необходимо обрабатывать *чистой хвостовой рекурсией* **при всех** операциях с ними. При возможности применяется регулярное программирование. Смотрите § 4.1 Общие указания к лабораторным работам.

4.5.2 Индивидуальные задания

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.

Указание. Элементом списка является слово. Также в элементе списка присутствуют две переменные-ссылки – next и next_alph. Сортировать список *после* чтения не эффективно – лучше сразу вставлять его на нужное место. Для этого на список организуется две переменные-ссылки-головы – head и head_alph. При чтении очередного слова из файла оно помещается *как обычно* в новый элемент переменной-ссылки next (в самом начале этой переменной-ссылкой будет являться head). Потом *этот же* элемент вставляется *на нужное место*, начиная с головы head_alph, двигаясь по полю next_alph.

§ 4.6 Лабораторная работа № 4

4.6.1 Общая часть задания

Задание выполняется в виде программного проекта из двух модулей, в котором необходимо использовать *динамические списки* неизвестной заранее длины. Списки необходимо обрабатывать *чистой хвостовой рекурсией* **при всех** операциях с ними. При возможности применяется регулярное программирование. Смотрите § 4.1 Общие указания к лабораторным работам.

4.6.2 Индивидуальные задания

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.

Указание. Элементом списка является пара значений – коэффициент a_n и степень n. При вычислении производной использовать схему Горнера, чтобы не проводить возведение в степень каждый раз.

16.