Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

по дисциплине: «Алгоритмы и структуры данных»

Выполнил
студент гр. в3530904/10021 Р.Н. Мирзоев

Руководитель
старший преподаватель С.А. Федоров

« 25 » мая 2022 г.

Санкт-Петербург
2022

Оглавление

Введение

Лабораторная работа

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 петербуржцев, 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

Цели

Цель работы — выбор структуры данных для решения поставленной задачи на современных микроархитектурах.

Задачи:

1. Реализовать задание с использованием массивов строк.
2. Реализовать задание с использованием массивов символов.
3. Реализовать задание с использованием массива структур или структуры массивов.
4. Реализовать задание с использованием динамического списка.
5. Провести анализ на регулярный доступ к памяти.
6. Провести анализ на векторизацию кода.
7. Провести тестирование производительности.
8. Провести сравнительный анализ реализаций.

ГЛАВА 1. "Реализация и анализ применения различных структур данных".

1.1. Реализовать задание с использованием массивов строк.

В первой лабораторной работе используется массив строк. Из преимуществ - регулярный доступ к памяти, т.к. данные сплошные, а также простота реализации. В Приложении 1 показан входной файл.

Объявление данных массива строк можно увидеть в Листинге 1:

```
 integer(I_), parameter                                    ::
CANDIDATES_AMOUNT = 5, &

SURNAME_LENGTH = 15, &
                                                              DATE_LENGTH
= 4, &

MILITARY_LENGTH = 3, &

REGISTRATION_LENGTH = 1

character(SURNAME_LENGTH, kind = CH_)                     :: surnameList(
CANDIDATES_AMOUNT) = ""
character(SURNAME_LENGTH, kind = CH_), allocatable        ::
SPBMilitarySurnameList(:), &

guestMilitarySurnameList(:)

character(DATE_LENGTH, kind = CH_)                        :: birthdayList(
CANDIDATES_AMOUNT) = ""
character(DATE_LENGTH, kind = CH_), allocatable           ::
SPBMilitaryBirthdayList(:), &

guestMilitaryBirthdayList(:)

character(kind = CH_)                                     :: genderList(
CANDIDATES_AMOUNT) = ""
character(REGISTRATION_LENGTH, kind = CH_), allocatable   ::
SPBMilitaryGenderList(:), &

guestMilitaryGenderList(:)

character(MILITARY_LENGTH, kind = CH_)                    :: isServedList(
CANDIDATES_AMOUNT) = ""
```

Листинг 1 - Инициализация массива символов

Чтение строк представлено в Листинге 2:

```
   open (file = input_file, encoding = E_, newunit = In)
      format = '(5(a, 1x))'
      read (In, format, iostat = IO) &
          (surnameList(i), birthdayList(i), isServedList(i),
   registrationList(i), genderList(i), i = 1, CANDIDATES_AMOUNT)
   close (In)

   Out = OUTPUT_UNIT
   open (Out, encoding = E_)
   select case(io)
      case(0)
      case(IOSTAT_END)
         write (Out, '(a)') "End of file has been reached while reading
   class list."
      case(1:)
         write (Out, '(a)') "Error while reading class list: ", io
      case default
         write (Out, '(a)') "Undetermined error has been reached while
   reading class list: ", io
   end select
```

Листинг 2 - Считывание данных

Основные операторы обработки данных представлены в Листинге 3.

```
!VVV direct access
   SPBMilitaryMask = registrationList == P_CHARSEt  .and. &
                     isServedList /= NOT_SERVED_CHARSET

   SPBMilitaryAmount = Count(SPBMilitaryMask)
   SPBMilitaryIndexesList = Pack(indexes, SPBMilitaryMask)
! Non direct access
 do concurrent (i = 1:guestMilitaryAmount)
      guestMilitarySurnameList(i)  = surnameList(guestMilitaryIndexesList(i)
   )
      guestMilitaryBirthdayList(i)  = birthdayList(guestMilitaryIndexesList(
   i))
      guestMilitaryGenderList = genderList(guestMilitaryIndexesList(i))
 end do
```

Листинг 3 - Код модуля Group-Process

1.2. Реализовать задание с использованием массивов символов.

Во второй лабораторной работе используется массив символов.

Объявление данных двумерного массива символов представлен на Листинге 4.

```
integer(I_), parameter                :: CANDIDATES_AMOUNT = 5, SURNAME_LENGTH
    = 15, DATE_LENGTH = 4, MILITARY_LENGTH = 3
   character(kind = CH_), parameter     :: NOT_SERVED_CHARSET(*) = [Char
   (1053, CH_), Char(1077, CH_), Char(1090, CH_)]
   character(kind = CH_), parameter     :: P_CHARSET = Char(1055, CH_)
   character(kind = CH_), parameter     :: G_CHARSET = Char(1057, CH_)
   character(:), allocatable            :: input_file, output_file

   character(kind = CH_)                :: surnameList(SURNAME_LENGTH,
   CANDIDATES_AMOUNT)  = "", &
                                           birthdayList(DATE_LENGTH,
   CANDIDATES_AMOUNT) = "", &
                                           isServedList(MILITARY_LENGTH,
   CANDIDATES_AMOUNT)  = "",&
                                           registrationList(CANDIDATES_AMOUNT
   ) = "", &
                                           genderList(CANDIDATES_AMOUNT) = ""

   character(kind = CH_), allocatable   :: SPBSurnames(:, :), guestSurnames
   (:, :)
   character(kind = CH_), allocatable   :: SPBBirthdays(:, :), guestBirthdays
   (:, :)
   character(kind = CH_), allocatable   :: SPBGenders(:), guestGenders(:)
```

Листинг 4 - Инициализация массива символов

В данной лабораторной работе важно помнить о том, что в технологии Fortran массивы хранятся по столбцам. Именно поэтому можно ошибиться, написав программу, с нерегулярным доступом к памяти. Из-за этих причин, в дальнейшем, структура массив символов будет отнесена к стуктурам с потенциальной векторизацией. Поэтому хранение фамилии - surnameList(SURNAME_LENGTH, CANDIDATES_AMOUNT) выгоднее, чем surnameList(CANDIDATES_AMOUNT, SURNAME_LENGTH), т.к. индексы хранятся по столбцам, для более эффективного доступа к памяти. Чтобы векторизовать код, требуется выравнивание данных

Пример правильного и неправильного чтения представлен в Листинге 5.

```
 subroutine readData(Input_File, surnameList, birthdayList, isServedList,
  registrationList, genderList)
     character(*)                     :: Input_File
     character(kind = CH_)            :: surnameList(:, :), birthdayList(:,
   :), isServedList(:, :), &
                                         registrationList(:), genderList(:)

```

```
6       intent (in)                         :: Input_File
7       intent (out)                        :: surnameList, birthdayList,
    isServedList, registrationList, genderList
8
9       integer In, IO, i
10      character(:), allocatable           :: format
11
12      open (file = Input_File, encoding = E_, newunit = In)
13         format = '(' // SURNAME_LENGTH // 'a1, 1x, ' // DATE_LENGTH // 'a1,
     1x, ' // &
14           MILITARY_LENGTH // 'a1, 1x, '//'a1, 1x, '//'a1, 1x)'
15
16        ! VVV direct access
17         read (In, format, iostat = IO) &
18              (surnameList(:, i), birthdayList(:, i), isServedList(:, i),
    registrationList(i), &
19              genderList(i), i = 1, CANDIDATES_AMOUNT)
20
21        ! XXX non-direct access
22        read (In, format, iostat = IO) &
23              (surnameList(i, :), birthdayList(i, :), isServedList(i, :),
    registrationList(i), &
24              genderList(i), i = 1, CANDIDATES_AMOUNT)
25
26         call Handle_IO_status(IO, "reading class list")
27      close (In)
28  end subroutine readData
```

Листинг 5 - Считывание данных

Основные операторы обработки данных представлены в Листинге 6. В данном листинге Векторизация кода присутствует, а также происходит регулярный доступ к памяти.

```
1   do concurrent (i = 1:militaryAmount)
2        ! direct access
3         filteredSurnameList(:, i)  = surnameList(:,militaryIndexes(i))
4         filteredBirthdayList(:, i)  = birthdayList(:,militaryIndexes(i))
5         filteredGenderList(i)= genderList(militaryIndexes(i))
6      end do
7
```

Листинг 6 - Код модуля Group-Process

1.3. Реализовать задание с использованием массива структур или структуры массивов.

В данном разделе используется массив структур для более удобной работы с данными, Объявление структуры представлено на Листинге 7.

```
implicit none
 integer, parameter :: CANDIDATES_AMOUNT = 5
 integer, parameter :: SURNAME_LENGTH  = 15
 integer, parameter :: DATE_LENGTH = 4
 integer, parameter :: MILITARY_LENGTH = 3

 type CandidateType
    character(SURNAME_LENGTH, kind=CH_)             :: surname           =
  ""
    character(DATE_LENGTH, kind=CH_)                :: birthDate         =
  ""
    character(MILITARY_LENGTH, kind=CH_)            :: isServedCharset   =
  ""
    character(kind=CH_)                             :: registrationCharset
   = ""
    character(kind=CH_)                             :: genderChar        =
  ""
 end type CandidateType
```

Листинг 7 - Инициализация структуры массивов

Формирование двоичного файла и его чтение представлено в Листинге 8:

```
subroutine createDataFile(Input_File, Data_File)
    character(*), intent(in)   :: Input_File, data_file

    type(CandidateType)        :: candidate
    integer                    :: In, Out, IO, i, recl
    character(:), allocatable  :: format

    open (file=Input_File, encoding=E_, newunit=In)
    recl = (SURNAME_LENGTH + DATE_LENGTH + MILITARY_LENGTH  + 1 + 1)*CH_
    open (file=Data_File, form='unformatted', newunit=Out, access='direct'
 , recl=recl)
       format = '(5(a, 1x))'
       do i = 1, CANDIDATES_AMOUNT
          read (In, format, iostat=IO) candidate
          call Handle_IO_status(IO, "reading formatted class list, line "
 // i)
          write (Out, iostat=IO, rec=i) candidate
          call Handle_IO_status(IO, "creating unformatted file with class
 list, record " // i)
       end do
```

```
18       close (In)
19       close (Out)
20    end subroutine createDataFile
21
22    function readClassList(Data_File) result(candidateList)
23       type(CandidateType)              :: candidateList(CANDIDATES_AMOUNT)
24       character(*), intent(in)         :: Data_File
25
26       integer In, IO, recl
27
28       recl = ((SURNAME_LENGTH + DATE_LENGTH + MILITARY_LENGTH+ 1 + 1)*CH_) *
     CANDIDATES_AMOUNT
29       open (file=Data_File, form='unformatted', newunit=In, access='direct',
     recl=recl)
30          read (In, iostat=IO, rec=1) candidateList
31          call Handle_IO_status(IO, "reading unformatted class list")
32       close (In)
33    end function readClassList
```

Листинг 8 - Чтение из файла

Основные операторы обработки данных представлены в Листинге 9. В данном листинге Векторизация кода не присутствует.

```
1     pure subroutine lexComparer(candidateList)
2       type(CandidateType), intent(inout)  :: candidateList(:)
3       integer                             :: i, j
4       ! Non-direct access
5       do i = Size(candidateList), 2, -1
6          do j = 1, i - 1
7             if (candidateList(j+1)%surname < candidateList(j)%surname) &
8                candidateList([j+1, j]) = candidateList([j, j+1])
9          end do
10      end do
11    end subroutine lexComparer
```

Листинг 9 - Код модуля Group-Process

1.4. Реализовать задание с использованием динамического списка.

В данном разделе используется в качестве типа данных односвязный список. Объявление данных двумерного массива символов представлен на Листинге 10.

```
integer, parameter :: CANDIDATES_AMOUNT = 5
integer, parameter :: SURNAME_LENGTH = 15
integer, parameter :: DATE_LENGTH = 4
integer, parameter :: MILITARY_LENGTH = 3


type CandidateType
   character(SURNAME_LENGTH, kind=CH_)          :: surname = ""
   character(DATE_LENGTH, kind=CH_)             :: birthDate = ""
   character(MILITARY_LENGTH, kind=CH_)         :: isServedCharset = ""
   character(kind=CH_)                          :: registrationCharset =
""
   character(kind=CH_)                          :: genderChar = ""
   type(CandidateType), pointer                 :: next => Null()
end type CandidateType
```

Листинг 10 - Инициализация списка

Чтение строк представлено в Листинге 11:

```
subroutine readClassList(Input_File) result(Class_List)
   type(CandidateType), pointer     :: Class_List
   character(*), intent(in)         :: Input_File
   integer  In

   open (file=Input_File, encoding=E_, newunit=In)
      Class_List => readCandidate(In)
   close (In)
end function readClassList


recursive function readCandidate(In) result(candidate)
   type(CandidateType), pointer     :: candidate
   integer, intent(in)              :: In
   integer  IO
   character(:), allocatable        :: format

   allocate (candidate)
   format = '('//CANDIDATES_AMOUNT//'(a, 1x))'
   read (In, format, iostat=IO) &
         candidate%surname, candidate%birthDate, candidate%
isServedCharset, &
         candidate%registrationCharset, candidate%genderChar

   call Handle_IO_status(IO, "reading line from file")
```

```
25      if (IO == 0) then
26          candidate%next => readCandidate(In)
27      else
28         deallocate (candidate)
29         nullify (candidate)
30      end if
31   end function readCandidate
```

Листинг 11 - Чтение данных в список

Основные операторы обработки данных представлены в Листинге 12. В данном листинге Векторизация кода не присутствует.

```
1    pure recursive subroutine getListReg(candidateList, categoryList,
     registrationCharset, NOT_SERVED_CHARSET, categoryAmount)
2       type(CandidateType), intent(in)         :: candidateList
3       type(CandidateType), pointer            :: categoryList
4       character(*, kind = CH_), intent(in)    :: registrationCharset,
     NOT_SERVED_CHARSET
5       integer(I_), intent(inout)              :: categoryAmount
6       ! Non direct access
7       if (candidateList%registrationCharset == registrationCharset &
8          .and. candidateList%isServedCharset /= NOT_SERVED_CHARSET) then
9
10         categoryAmount = categoryAmount+1
11         allocate (categoryList, source = candidateList)
12
13         if (Associated(candidateList%next)) &
14            call getListReg(candidateList%next, categoryList%next,
     registrationCharset, NOT_SERVED_CHARSET, categoryAmount)
15         else if (Associated(candidateList%next)) then
16            call getListReg(candidateList%next, categoryList,
     registrationCharset, NOT_SERVED_CHARSET, categoryAmount)
17         else
18            categoryList => Null()
19      end if
20   end subroutine getListReg
21
22   pure recursive subroutine lexComparer(candidate, N)
23      type(CandidateType), pointer, intent(inout)  :: candidate
24      integer, intent(in)                          :: N
25
26      if (N >= 2) then
27         call dropDown(candidate, 1, N-1)
28         call lexComparer(candidate, N-1)
29   end if
30   end subroutine lexComparer
31
32   pure recursive subroutine dropDown(candidate, j, N)
```

```
33      type(CandidateType), pointer            :: candidate
34      integer, intent(in)                     :: j, N
35
36      if (candidate%next%surname < candidate%surname) &
37         call swapFromCurrent(candidate)
38
39      if (j < N) &
40         call dropDown(candidate%next, j+1, N)
41   end subroutine dropDown
42
43   pure subroutine swapFromCurrent(Current)
44      type(CandidateType), pointer  :: Current
45      type(CandidateType), pointer  :: tmpCandidate
46
47      tmpCandidate => Current%next
48      Current%next => Current%next%next
49      tmpCandidate%next => Current
50      Current => tmpCandidate
51   end subroutine swapFromCurrent
```

Листинг 12 - Код модуля Group-Process

Глава 2. "Сравнение реализаций".

В ходе выполнения лабораторной работы 1 было использовано 4 разных подхода для выполнения работы, которые отличались между собой структурами данных (1.1 - массив строк, 1.2 - массив символов, 1.3 - структура 1.5 - массивов, список) и реализуемыми модулями. Ниже в Таблице 1 представлено сравнение реализаций лабораторной работы 1.

Таблица 1 - Сравнение реализаций

	сплошные данные	регулярный доступ	векторизация	потенциальная векторизация
Массив строк	+	+	+	-
Массив символов	-/+	+	-/+	+
Массив структур	-	+	+	-
Динамический список	-	-	-	-

Глава 3. "Тестирование производительности".

Каждая из пяти лабораторных работ была протестирована на производительность. Для каждого приложения запускалось 3 теста и бралось его медианное значение в миллисекундах. Для этого была использована встроенная функция CPU_TIME, которая возвращает значение, представляющее затраченное процессорное время в секундах. Исходная конфигурация указана в таблице 2. Результаты исследования в таблице 3.

Пример использования указан в листинге 13:

```
program test_cpu_time
    real :: start, finish
    call cpu_time(start)
        ! put code to test here
    call cpu_time(finish)
    print '("Time = ",f6.3," seconds.")',finish-start
end program test_cpu_time
```

Листинг 13 - Функция CPU_TIME

Таблица 2 - исходная конфигурация:

Процессор	AMD Ryzen 5 3600 6-core
ОЗУ	16 Гб
ОС	FEDORA 36

Таблица 3 - Тестирование производительности

Стуктура / Кол-во входных значений	10	100	1000	10 000
Массив строк	0.0001	0.0003	0.00025	0.0364
Массив символов	0.0002	0.0005	0.00037	0.1411
Массив структур (без хвостовой рекурсии)	0.0002	Invalid memory	Invalid memory	Invalid memory
Массив структур (с хвостовой рекурсией)	0.0002	0.0006	0.00038	0.0417
Динамический список	0.0002	0.0008	0.00062	0.1459

Вывод: Если новые бизнес-требования к приложению не будут появляться и единственные изменения, которые будут происходить - это увеличение количества входных данных, тогда оптимальной структурой для данной задачи можно считать массив строк. Если же требования могут в дальнейшем расширяться, тогда стоит присмотреться к массиву структур.

Заключение

В данной работе была написана программа сортировки в алфавитном порядке по отдельности списка петербуржцев и гостей Санкт-Петербурга, служивших в армии
Также, в ходе работы были выполнены ряд задач :

1. Реализовать задание с использованием массивов строк.
2. Реализовать задание с использованием массивов символов.
3. Реализовать задание с использованием массива структур или структуры массивов.
4. Реализовать задание с использованием динамического списка.
5. Провести анализ на регулярный доступ к памяти.
6. Провести анализ на векторизацию кода.
7. Провести тестирование производительности.
8. Провести сравнительный анализ реализаций.

И в ходе выполнения данных задач была определена наиболее подходящая структура данных для лабораторной работы - массив структур.

Приложение

Приложение 1. Исходный список

Дудиков	1997	Да	П	М
Тихонов	1988	Да	С	М
Садовникова	1997	Нет	П	Ж
Степин	1998	Нет	С	М
Воробъёва	1990	Да	П	Ж