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

КУРСОВАЯ РАБОТА

Лабораторная работа 1
по дисциплине: «Алгоритмы и структура данных»

Выполнил
студент гр.в3530904/10022 Тертень Д.В.

Руководитель
старший преподаватель. Фёдоров С.А.

«23 »мая 2022 г.

Санкт-Петербург
2022

Содержание

Введние

Постановка задачи:
Дан список группы в виде:

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

Иванова 1985 М
Петрова 1983 Ж
```

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

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

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 и внутренних процедур головной программы;
3. реализовать задание с использованием структуры массивов, файлов записей и модулей;
4. реализовать задание с использованием стркутуры массивов, файлов записей, модулей и хвостовой рекурсии;
5. реализовать задание с использованием модулей, хвостовой рекурсии, однонаправленных списков неизвестной длины;
6. провести анализ на регулярный доступ к памяти;
7. провести анализ на векторизацию кода;
8. провести сравнительный анализ реализаций.

Во всех вышеизложенных заданиях необходимо также использовать регулярное программирование.

Основная часть

Глава 1. Реализация и анализ применения различных структур данных

1.1 Массив строк

Ниже продемонстрировано объявление массивов строк, кол-во элементов в данных массивах равно количеству пользователей, длина каждой строки равна длинне слова:

```
integer, parameter                  :: USER_AMOUNT = 14,
        LN_LEN = 15, G_LEN = 1, BY_LEN = 4

character(:), allocatable      :: input_file, output_file, format
integer                             :: In, Out, IO, i

character(LN_LEN, kind=CH_)    :: Last_Names(USER_AMOUNT) = ""
character(G_LEN, kind=CH_)     :: Gender(USER_AMOUNT) = ""
integer(BY_LEN)                     :: birthYear(USER_AMOUNT) = 0
```

Для обработанных данных были объявлены переменные для хранения индексов самого пожилого ммужчины и самой молодой женщины, а так же переменные для хранения временных данных:

```
integer                                         :: indexBoy(1), indexGirl(1)
character(LN_LEN, kind=CH_), allocatable :: Last_Names_Boys(:),
Last_Names_Girls(:)
logical, allocatable                            :: Is_A_Boy(:), Is_A_Girl(:)
integer, allocatable                            :: BoysBirthYear(:),
    GirlsBirthYear(:), Boys_Pos(:), Girls_Pos(:)
integer, parameter          :: INDEXES(*) = [(i, i = 1, USER_AMOUNT)]
```

Заполнение массива мальчиков и массива девочек соответственно:

```
Is_A_Boy        = Gender == MALE
Boys_Amount     = Count(Is_A_Boy)

Boys_Pos    = Pack(INDEXES, Is_A_Boy)
allocate (Last_Names_Boys(Boys_Amount), BoysBirthYear(Boys_Amount))
do concurrent (i = 1:Boys_Amount)
    Last_Names_Boys(i)  = Last_Names(Boys_Pos(i))
    BoysBirthYear(i)  = birthYear(Boys_Pos(i))
end do

Is_A_Girl       = .not. Is_A_Boy
Girls_Amount    = USER_AMOUNT - Boys_Amount
```

```
Girls_Pos    = Pack(INDEXES, Is_A_Girl)

allocate (Last_Names_Girls(Girls_Amount), GirlsBirthYear(Girls_Amount))
do concurrent (i = 1:Girls_Amount)
    Last_Names_Girls(i)  = Last_Names(Girls_Pos(i))
    GirlsBirthYear(i)  = birthYear(Girls_Pos(i))
end do
```

Нахождение индексов происходит встроенными функциями MINLOC/MAXLOC:

```
indexBoy = MINLOC(BoysBirthYear, DIM=1)
indexGirl = MAXLOC(GirlsBirthYear, DIM=1)
```

1.2 Массив символов

Ниже продемонтрировано объявление массивов символов:

```
integer, parameter                          :: USER_AMOUNT = 14,
    LN_LEN = 15, G_LEN = 1
character(kind=CH_), parameter              :: MALE = Char(1052, CH_)

character(:), allocatable                   :: input_file, output_file
integer                                     :: birthYear(USER_AMOUNT)
character(LN_LEN, kind=CH_)                 :: Last_Names(USER_AMOUNT)
character(G_LEN, kind=CH_)                  :: Gender(USER_AMOUNT)
```

Для обработанных данных были объявлены перемнные для хранения индексов результата:

```
integer                                     :: indexBoy(1), indexGirl(1)
```

Подпрограмма нахождения индексов самого пожилого мужчины и самой молодой женщины:

```
subroutine Handle_User_List(birthYear, Gender, indexOfBoy, indexOfGirl)
    character(G_LEN, kind=CH_)              :: Gender(USER_AMOUNT)
    integer                                 :: birthYear(USER_AMOUNT),
        indexOfBoy(1), indexOfGirl(1)

    intent (in)          birthYear, Gender
    intent (out)         indexOfBoy, indexOfGirl

    indexOfBoy      = MINLOC(birthYear, MASK = (Gender == MALE), DIM=1)
    indexOfGirl     = MAXLOC(birthYear, MASK = (.not. Gender == MALE), DIM=1)
end subroutine Handle_User_List
```

1.3 Структура массивов

Объявление массива структур:

```
integer, parameter :: USER_AMOUNT    = 14
integer, parameter :: LN_LEN         = 15
integer, parameter :: G_LEN          = 1

type Users
    character(LN_LEN, kind=CH_)              :: Last_Names
    character(G_LEN, kind=CH_)               :: Gender
    integer                                  :: birthYear
end type Users
```

Был выбран массив структур, так как выгодно значения данного поля расположить сплошными данными в одном массиве для регулярного доступа к памяти. Реализовано формирование двоичного файла записей из входного файла:

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

      integer                     :: In, Out, IO, i
      character(:), allocatable   :: format

      character(LN_LEN, kind=CH_)              :: Last_Names(USER_AMOUNT)
      character(G_LEN, kind=CH_)               :: Gender(USER_AMOUNT)
      integer                                  :: birthYear(USER_AMOUNT)

      open (file=Input_File, encoding=E_, newunit=In)
         format = "(a, i4, a)"
         read (In, format, iostat=IO) (Last_Names(i), &
         birthYear(i), Gender(i), i=1, USER_AMOUNT)
         call Handle_IO_status(IO, "init arrays with formatted file")
      close (In)

      open (file=Data_File, form="unformatted", newunit=Out, access="stream")
         write (Out, iostat=IO) Last_Names, birthYear, Gender
         call Handle_IO_status(IO, "creating unformatted file by index")
      close (Out)
end subroutine Create_Data_File
```

И функция чтения данных из двоичного файла в структуры массивов:

```
function Read_Users_List(Data_File) result(Usrs)
      type(Users)                 :: Usrs(USER_AMOUNT)
      character(*), intent(in)    :: Data_File
      integer                     :: In, IO

      open (file=Data_File, form="unformatted", newunit=In, access="stream")
```

```
        read (In, iostat=IO) Usrs%Last_Names, Usrs%birthYear, Usrs%Gender
        call Handle_IO_status(IO, "init objects")
    close (In)
end function Read_Users_List
```

Реализовано нахождение нужных элементов с помощью чистой функции и их упаковывание:

```
pure function Handle_Users_List(Usrs) result(Usrs_Res)
    type(Users)                 :: Usrs(:)
    type(Users)                 :: Usrs_Res(2)
    integer                     :: indexOfBoy(1), indexOfGirl(1)
    character(kind=CH_)         :: MALE

    intent (in)                 Usrs

    MALE = Char(1052, CH_)
    indexOfBoy      = MINLOC(Usrs%birthYear ,MASK=(Usrs%Gender==MALE))
    indexOfGirl     = MAXLOC(Usrs%birthYear ,MASK=(.not. Usrs%Gender==MALE))

    Usrs_Res(:1) = Usrs(indexOfBoy)
    Usrs_Res(2:2) = Usrs(indexOfGirl)
end function Handle_Users_List
```

1.4 Хвостовая рекурсия

Объявляем массив структур:

```
integer, parameter :: USER_AMOUNT    = 14
integer, parameter :: LN_LEN         = 15
integer, parameter :: G_LEN          = 1

type Users
    character(LN_LEN, kind=CH_)             :: Last_Names
    character(G_LEN, kind=CH_)              :: Gender
    integer                                 :: birthYear
end type Users
```

Реализуем функцию нахождение индексов самого пожилого мужчины и самой молодой женщины

```
function Handle_Users_List(Usrs) result(Usrs_Res)
    type(Users)                 :: Usrs(:)
    type(Users)                 :: Usrs_Res(2)
    integer                     :: indexOfBoy(1), indexOfGirl(1)
    character(kind=CH_)         :: MALE
```

```
        intent (in)                 Usrs

        MALE = Char(1052, CH_)
        indexOfBoy = MINLOC(Usrs%birthYear ,MASK=(Usrs%Gender==MALE))
        indexOfGirl = MAXLOC(Usrs%birthYear ,MASK=(.not. Usrs%Gender==MALE))

        Usrs_Res(:1) = Usrs(indexOfBoy)
        Usrs_Res(2:2) = Usrs(indexOfGirl)
end function Handle_Users_List
```

Запись в файл реализована с помощью рекурсивных процедур

```
subroutine Output_Users_List(Output_File, Usrs, List_name, Position)
        character(*), intent(in)    :: Output_File, List_name, Position
        type(Users), intent(in)     :: Usrs(:)

        integer                     :: Out

        open (file=Output_File, encoding=E_, position=Position, newunit=Out)
              write (Out, "(a)") List_name
              call Output_Users_Rec(Out, Usrs, 1)
        close (Out)
end subroutine Output_Users_List

recursive subroutine Output_Users_Rec(Out, Usrs, start_index)
        type(Users)                 :: Usrs(:)
        integer                     :: Out, IO, start_index
        character(:), allocatable   :: format

        intent (in)                    Out, Usrs, start_index

        format = "(a, i4, a)"
        if (start_index <= Size(Usrs)) then
           write (Out, format, iostat=IO) Usrs(start_index)%Last_Names,
              Usrs(start_index)%birthYear, Usrs(start_index)%Gender
           call Handle_IO_status(IO, "writing users")
           call Output_Users_Rec(Out, Usrs, start_index + 1)
        end if
end subroutine Output_Users_Rec
```

1.5 Динамический однонаправленный список

Объявляем структуру узла списка

```
integer, parameter :: LN_LEN          = 15
integer, parameter :: G_LEN           = 1

type User
    character(LN_LEN, kind=CH_)                 :: Last_Names  = ""
    character(G_LEN, kind=CH_)                  :: Gender      = ""
    integer                                     :: birthYear   = 0
    type(User), pointer                         :: next        => Null()
end type User
```

Обход списка и нахождение индексов реализовано с помощью хвостовой рекурсии:

```
pure recursive subroutine Delete_unnecessary_girls(current, next)
      type(User), pointer                       :: current, tmp, next

      if (Associated(next)) then
         if ( current%birthYear < next%birthYear) then
            tmp => current

            current => next
            deallocate(tmp)
         end if
         call Delete_unnecessary_girls(current, next%next)
      else
         current%next => Null()
      end if
end subroutine Delete_unnecessary_girls

pure recursive subroutine Delete_unnecessary_boys(current, next)
      type(User), pointer                       :: current, tmp, next

      if (Associated(next)) then
         if ( current%birthYear > next%birthYear) then
            tmp => current

            current => next
            deallocate(tmp)
         end if
         call Delete_unnecessary_boys(current, next%next)
      else
         current%next => Null()
      end if
end subroutine Delete_unnecessary_boys
```

Глава 2. Сравнение реализаций

В таблице 1 приведен сравнительный анализ различных реализаций лабораторной работы:

-	**Массив строк**	**Массив символов**	**Массив структур**	**Динамический список**
Сплошные данные	Да	Да	Да	Нет
Регулярный доступ	При обработке	При обработке	При обработке	Нет
Векторизация	Нет	Нет	Нет	Нет
Потенциальная векторизация	При обработке	При обработке	При обработке	Нет

Таблица 1: Сравнение реализаций

Исходя из анализа для выполнения подходят все варианты, кроме динамических списков, как самых медленных для обработки, и неэффективных, если мы заранее знаем размер списка. Среди остальных вариантов наиболее предпочтительна структура массивов, позволяющие сгруппировать данные в одну строкуктуру, повышая уровень абстракции. Также реализация массива строк более предпочтительна перед массивом символов, так как второе решение повышает сложность кода и требует большой компетенции от разработчика при написании и поддержке кода под современных архитектуры.

Вывод

Поставленная задача была реализована несколькими способами, среди которых бели выбраны алгоритмы, наиболее подходищие для выполнения программного кода на современных архитектурах. Наиболее подходящим решением оказалась реализация массива структур. Поставленная цель выполнена.